Detecting Grammatical Errors with Treebank-Induced, Probabilistic Parsers

Joachim Wagner

Magister Artium

A dissertation submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy

to the

Dublin City University

School of Computing

Supervisors: Dr. Jennifer Foster
Prof. Josef van Genabith

January 2012

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed _____

(Joachim Wagner)

Student ID 53154541

Date January 2012

Contents

Joachim Wagner

Detecting Grammatical Errors with Treebank-Induced, Probabilistic Parsers

Abstract

Today's grammar checkers often use hand-crafted rule systems that define acceptable language. The development of such rule systems is labour-intensive and has to be repeated for each language. At the same time, grammars automatically induced from syntactically annotated corpora (treebanks) are successfully employed in other applications, for example text understanding and machine translation. At first glance, treebank-induced grammars seem to be unsuitable for grammar checking as they massively over-generate and fail to reject ungrammatical input due to their high robustness.

We present three new methods for judging the grammaticality of a sentence with probabilistic, treebank-induced grammars, demonstrating that such grammars can be successfully applied to automatically judge the grammaticality of an input string. Our best-performing method exploits the differences between parse results for grammars trained on grammatical and ungrammatical treebanks. The second approach builds an estimator of the probability of the most likely parse using grammatical training data that has previously been parsed and annotated with parse probabilities. If the estimated probability of an input sentence (whose grammaticality is to be judged by the system) is higher by a certain amount than the actual parse probability, the sentence is flagged as ungrammatical. The third approach extracts discriminative parse tree fragments in the form of CFG rules from parsed grammatical and ungrammatical corpora and trains a binary classifier to distinguish grammatical from ungrammatical sentences.

The three approaches are evaluated on a large test set of grammatical and ungrammatical sentences. The ungrammatical test set is generated automatically by inserting common grammatical errors into the British National Corpus. The results are compared to two traditional approaches, one that uses a hand-crafted, discriminative grammar, the XLE ParGram English LFG, and one based on part-of-speech n-grams. In addition, the baseline methods and the new methods are combined in a machine learning-based framework, yielding further improvements.

Acknowledgements

I would like to thank my current and past supervisors Jennifer Foster, Josef van Genabith and Monica Ward for all their support throughout my PhD studies. It was Josef who repeatedly suggested to me the basic idea of Chapter 4 and who convinced me to abandon research on using NLP methods to build CALL exercises in favour of grammatical error detection, the subject of this thesis. This work would not have been undertaken without his persistence. Also vitally important was Jennifer's role as collaborator and, for the past four years, leading supervisor. She gradually took over the latter role while Josef had to focus on his duties as director of the then newly established Centre for Next Generation Localisation (CNGL). I am grateful for Jennifer and Josef's advice and for their valuable comments on various drafts of this thesis. I would also like to thank Monica for encouragement and feedback during the first three years of my PhD. Further valuable feedback was given by the examiners Stefan Evert (Technische Universität Darmstadt), Michael Gamon (Microsoft Research) and Darragh O'Brien (School of Computing, DCU). I thank them for their comments, which have greatly improved the final version of this thesis.

This research has been supported by two scholarships and three travel grants. I would like to express my gratitude to the Irish Research Council for Science, Technology and Engineering (IRCSET) for supporting the first three years of my studies with an "embark" scholarship under Basic Research Grant SC/02/298 ("Integrating Techniques from Computational Linguistics into Computer-Assisted Language Learning") and the Association for Computational Linguistics (ACL) and the Australian Research Council's Network in Human Communication Science (ARC HCSNet) for a scholarship to attend its 2006 summer school "Advanced Program in Natural Language Processing" in Melbourne. I also thank the Office of the Vice-President for Research (OVPR), the School of Computing and the Centre for Next Generation Localisation (Science Foundation Ireland Grant 07/CE/I1142) for supporting me to attend conferences.

The training data and the artificial test data used in this research is based on the British National Corpus (BNC), distributed by Oxford University Computing Services on behalf of the BNC Consortium. I thank Djamé Seddah for his help and guidance developing the tools to parse the BNC and other corpora on the computers of the SFI/HEA Irish Centre for High-End Computing (ICHEC). I wish to acknowledge ICHEC for the provision of computational facilities and support. I am also very grateful to James Hunter of Gonzaga University for providing Jennifer and me with the spoken error corpus.

I would like to thank all my co-authors I had during the time of my PhD research. They were a pleasure to work with and broadened my view for research outside the narrow scope of my thesis. I would also like to thank the many colleagues, fellow students and research group members who helped me in various ways, engaged in interesting discussions or were good friends, and the many friends I met outside the lab during my PhD studies, including members of DCU Postgraduate Society, DCU Judo Club and Portmarnock Judo Club. Furthermore, I would like to thank my fiancée Lorraine Murphy for her great support, love and patience with me during the writing of this thesis.

Finally, I thank my family, for everything.

List of Tables

List of Figures

List of Abbreviations and Acronyms

AG	Agreement (error)
AI	Artificial Intelligence
ALEK	Assessing LExical Knowledge
ANN	Approximate Nearest Neighbour
API	Application Programming Interface
APP	Actual Parse Probability
ASCII	American Standard Code for Information Interchange
AUC	Area Under the Curve
Base64	An encoding scheme for binary data
BLEU	BiLingual Evaluation Understudy, an evaluation measure
BitPar	A parser using bit vector operations (IMS Stuttgart)
BNC	British National Corpus
BootCaT	Bootstrap Corpora And Terms
Bot.	Bottom
BZip2	A block-sorting file compressor
CALL	Computer-Assisted Language Learning
CFG	Context-Free Grammar
CL	Computational Linguistics
CLC	Cambridge Learner Corpus
CLEC	Chinese Learner English Corpus
CLEF	Cross-Language Evaluation Forum
CNGL	Centre for Next Generation Localisation
CPU	Central Processing Unit
CSSC	Context-Sensitive Spelling Correction
CYK	Cocke-Younger-Kasami algorithm (also CKY)
DCU	Dublin City University
DOP	Data-Oriented Parsing
DPP	Distorted Parse Probability
DT	Decision Tree

EER	Equal-Error-Rate
EM	Expectation Maximisation, an algorithm
EPP	Estimated Parse Probability
ERG	The English Resource Grammar (LinGO lab at Stanford University)
ESC	Escape (control character)
ESL	English as a Second Language
ESOL	English for Speakers of Other Languages
EuroParl	European Parliament, corpora derived from the proceedings
EW	Extra word (error)
FO	Factoring Out
GB	Gigabyte, $1{,}000^3$ or $1{,}024^3$ bytes depending on context
GHz	Gigahertz, unit of frequency
GIMP	The GNU Image Manipulation Program
Gr.	Grammatical
HPSG	Head-driven Phrase Structure Grammar
HTER	Human-targeted Translation Error Rate
HTML	HyperText Markup Language
I-CALL	Intelligent CALL (using NLP)
ICLE	International Corpus of Learner English
ID	Identifier or Identification
IELTS	International English Language Testing System
IMS	Institut für Maschinelle Sprachverarbeitung (at Universität Stuttgart)
IR	Information Retrieval
JPU	Janus Pannonius University, now part of University of Pécs, Hungary
L1	First Language (native language)
L2	Second Language (any language learned after the L1)
LA	Leaf-Ancestor
LA	Latent Annotation
LFG	Lexical-Functional Grammar
LM	Language Model

LoPar	A parser from the IMS
LP	Labelled Precision
LR	Labelled Recall
MAP	Maximum A Posteriori
MED	Maximum Entropy Discrimination
MI	Mutual Information
MIME	Multipurpose Internet Mail Extensions
ML	Machine Learning
MSE	Mean Square Error
MT	Machine Translation
MW	Missing word (error)
NAACL	North American Chapter of the Association for Computational Linguistics
NE	Normalisation Exponent
NLG	Natural Language Generation
NLP	Natural Language Processing
OCR	Optical Character Recognition
OOV	Out Of Vocabulary
OT	Optimality Theory
PA	Parent Annotation
PAD	Padding
ParGram	Parallel Grammar, a project
PCFG	Probabilistic Context-Free Grammar
PELCRA	Polish and English Language Corpora for Research and Applications
PLUR	Plural
POS	Part Of Speech
PTB	Penn II Treebank
RAM	Random Access Memory, main memory of a computer
RASP	Robust Accurate Statistical Parsing
RF	Rule Features
ROC	Receiver Operating Characteristic

RW	Real-word spelling (error)
SD	Standard Deviation
SGML	Standard Generalized Markup Language
SING	Singular
SL	Sentence Length
SLA	Second Language Acquisition
SMT	Statistical Machine Translation
SRILM	The Stanford Research Institute Language Modeling toolkit
SST	Standard Speaking Test
SVM	Support Vector Machine
TG	Trigram
TG	Test Grammatical
TOEFL	Test Of English as a Foreign Language
TSNLP	Test Suites for Natural Language Processing
TU	Test Ungrammatical
URL	Uniform Resource Locator
UTF-8	UCS Transformation Format – 8-bit, a character encoding for Unicode
VF	Verb Form (error)
VPP	Vanilla Parse Probability
W	Weight
Weka	Waikato Environment for Knowledge Analysis
WordNet	A lexical database of English (Princeton University)
WSD	Word Sense Disambiguation
WSJ	Wall Street Journal
XLE	The Xerox Linguistics Environment
XML	Extensible Markup Language

Chapter 1

Introduction

This thesis is concerned with the task of automatic *grammaticality judgements*, i. e. detecting whether or not a sentence contains a grammatical error, using probabilistic parsing with treebank-induced grammars. A classifier capable of distinguishing between syntactically well-formed and syntactically ill-formed sentences has the potential to be useful in a wide range of applications:

- Grammar checking is the most obvious application and an application that helps many people, both native speakers and L2 writers, to improve the quality of their written communication. A growing number of people self-publish and want to achieve high quality, e. g. on blogs or small business websites. Also, many employees carry out reporting and minutes taking duties, exposing their language skills to co-workers. As researchers with a strong interest and background in language and grammar, we tend to focus on the shortcomings of current grammar checkers and to overlook their usefulness to many people (Dale, 2004). Nevertheless, improvements in the underlying technology are likely to be welcomed.

- Computer-assisted language learning (CALL) tries to help language learners to develop their language skills, e. g. with computer-mediated communication and electronic delivery of course materials. As regards automatic grammaticality judgements, one could envisage, for example, the use of such judgements in automatic essay grading and as a first step towards diagnosing an error and providing appropriate feedback in a language tutoring system. For advanced learners, it might also

be helpful to use automatic grammaticality judgements to point the learner towards an error without indicating its precise nature.

- Automatic grammaticality judgements have applications in NLP, e. g. for evaluating the output of natural language generation and machine translation systems.

Section 2.1.6 of Chapter 2 lists more possible applications.

In the area of computer-assisted language learning (CALL), the use of parsing technology often faces scepticism as systems using traditional, hand-crafted grammars fell short of expectations (Borin, 2002; Nerbonne, 2002). However, today's dominant parsing technology uses a different type of grammar: grammars that have been automatically induced from treebanks, i. e. text annotated with syntactic structures. Given sufficiently large treebanks, such grammars tend to be highly robust to unexpected input and achieve wide coverage of unrestricted text (van Genabith, 2006). The robustness also covers grammatical errors. Almost all input is parsed into a (more or less plausible) parse tree, meaning that parsability cannot be used as a criterion for grammaticality.

In this thesis, we present three new methods for judging the grammaticality of a sentence with probabilistic, treebank-induced grammars, demonstrating that such grammars can be successfully applied to automatically judge the grammaticality of an input string. Our best-performing method exploits the differences between parse results for grammars trained on grammatical and ungrammatical treebanks. This method combines well with n-gram and deep grammar-based approaches, as well as combinations thereof, in a machine learning-based framework. In addition, voting classifiers are proposed to tune the accuracy trade-off between finding all errors and not overflagging grammatical sentences as ungrammatical.

1.1 Scope of Work

This section outlines the scope of our work, characterising the task we address, our objectives and the focus of our research.

Task Judging the grammaticality of a sentence is a basic task that either has to be carried out before further checking for the type or location of particular grammatical errors or it

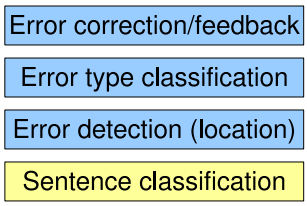


Figure 1.1: Problem setting

will be a by-product of any grammar checking. Gamon and Leacock (2010) describe binary grammaticality judgements as "a baseline task that any error detection and correction system needs to address." Figure 1.1 positions the binary classification of a sentence as either grammatical or ungrammatical below the tasks of locating errors, identifying the type of error and providing feedback to the user, e. g. in the form of corrections. We discuss the task in relation to previous work in Section 2.1.3 of Chapter 2.

Grammaticality Grammaticality is difficult to define. A formal definition as success to parse given a grammar of the language is not practical as available grammars either reject too many sentences in edited, high quality text or accept too many sentences that clearly contain grammatical errors — see Sections 5.1 and 5.3 of Chapter 5. Grammatical errors may or may not break communication, add uncertainty, be noticed or prompt for correction. In our work, we take a practical approach and rely on the judgements found in various annotated error corpora. As a rule of thumb, constructions are treated as ungrammatical if they require a correction in order to be acceptable to a broad audience of educated native speakers or if a language teacher thinks they need to be flagged, excluding content changes. For pointers to the literature on grammaticality, see Section 2.1.1 of Chapter 2.

Choice of Parser We focus on using probabilistic parsing with treebank-induced grammars. A number of advantages of probabilistic, treebank-induced grammars make them attractive:

- Each parse is assigned a probability which we hypothesise may, in some way, reflect grammaticality. (This is our research hypothesis.)

- Software for grammar induction (training) and parsing, treebanks and "off-the-shelf" grammars are available.
- These parsers are robust (which includes robustness to unexpected but still grammatical constructions).
- Efficient parsing algorithms are used (fairly low computational costs despite the large number of rules extracted from treebanks).
- New grammars can be created at relatively low costs (given a treebank).
- Ambiguous analyses (alternative trees) are inherently disambiguated.
- Additional data (trees) is easily integrated as new grammars can be induced from an augmented or modified treebank.
- Such parsers can be automatically adapted to new domains with self-training.

Target Language Our current experiments are restricted to English as the target language for the following two reasons: *(a)* English is a highly relevant language for grammar checking as many people have to write high quality text in English regardless of the native language. *(b)* Our experiments require various resources such as treebank grammars and error corpora. While the availability of error corpora has improved over the last years (Albert et al., 2009; Boyd, 2010; Rozovskaya and Roth, 2010a), the English error corpus of Foster (2005) was crucial in starting our research and remains a corner-stone as it informs our error model.

Methodology To train our machine learning-based error detectors, we use a large corpus of well-formed sentences and an equally-sized corpus of ill-formed sentences. To obtain the ill-formed sentences, we automatically introduce errors into the sentences in the original well-formed corpus, employing an error model grounded in an analysis of authentic errors (Foster, 2005). Our automatic grammaticality judgements are tested on various types of test data including *(a)* synthetic ungrammatical sentences, created in the same way as the ungrammatical sentences in the training set, *(b)* authentic error data produced by advanced learners of English, e. g. sentences from the International Corpus of Learner

English (Granger, 1993), and *(c)* transcribed spoken sentences produced by learners of English of varying levels. Testing the method on the artificially produced ungrammatical sentences allows us to gauge the efficacy of our machine learning features, while testing the method on real learner data also provides information on potential gaps in our error model.

1.2 Contributions

This section gives an overview of the contributions to error detection research we make with this thesis.

1.2.1 Data, Training and Evaluation

Artificial Error Corpus Based on an analysis of authentic errors (Foster, 2005), we insert errors into a 100 million word corpus, creating a large artificial error corpus that we use to train and develop our methods (Section 3.3 of Chapter 3). This corpus is first used outside DCU by Rethmeier (2011).

Annotation of Authentic Error Corpora We are fortunate to collaborate with Jennifer Foster, a native speaker of English, who annotated four authentic error corpora with grammaticality judgements and/or error type information for this work.

Evaluation Measures Given the wide range of error densities to be expected in different applications and even within applications (consider, for example, language learners at different proficieny levels using a grammar checker), evaluation measures that depend on error density or misclassification costs are not suitable for the development of basic methods for detecting grammatical errors. We think that the strengths of each method need to be explored and documented rather than simply discarding methods if they fall short of the performance of another method on a single scale. We propose to measure accuracy separately on grammatical and ungrammatical test data.

Tuning the Accuracy Tradeoff Some classifiers, e. g. maximum entropy classifiers and support vector machines, provide a parameter for tuning it between high accuracy

on grammatical sentences (avoiding overflagging) and high accuracy on ungrammatical sentences (detecting most errors). In the absence of such a parameter, we propose to train multiple classifiers and to combine them in a voting scheme.

Finding Optimal Classifiers Since we use two evaluation measures, we can arrive at a situation where classifiers are ordered differently by them. We show how, in the case of accuracy, a small number of classifiers can be identified in a set of classifiers that cover the performance (as measured by accuracy on grammatical and ungrammatical data) of all remaining classifiers.

1.2.2 Error Detection with Probabilistic Parsing

As pointed out at the start of this chapter, probabilistic, treebank-induced grammars derived from large treebank resources differ from hand-crafted grammars in that they parse almost any input, irrespective of whether it is grammatical or not. We propose and evaluate three new error detection methods that use probabilistic parsing with treebank-induced grammars.

Distorted Treebank Our best-performing method for using probabilistic parsing for error detection induces two grammars from a vanilla treebank and a distorted copy of the vanilla treebank. The distorted treebank is automatically derived from a vanilla treebank by inserting artificial errors and by making minimal adjustments to the corresponding parse trees. Input is parsed with both grammars and a grammaticality judgement is based on a comparison of the parse results obtained with the different grammars.

APP/EPP Method We introduce a method that judges the grammaticality of sentences by comparing the parse probability assigned by a probabilistic, treebank-induced grammar to a reference probability obtained from a data-driven model that uses only positive, i. e. grammatical, data and does not make any assumptions about the types of errors.

(P)CFG Pruning Given the observation in the literature that treebank-induced probabilistic context-free grammars (PCFGs) derived from large treebank resources parse almost

any input and that coverage of grammatical language remains high if rules with a low frequency in the treebank are removed from the grammar, one may speculate that these rare rules are contributing towards the robustness to grammatical errors and that this can be exploited to detect ungrammatical input. We prune different PCFGs induced from a treebank to test this hypothesis.

1.2.3 Baseline Methods and Combined Methods

We implement two baseline methods: the precision grammar approach uses a large handcrafted grammar that aims to only describe grammatical constructions. If parsing fails the sentence is classified as ungrammatical. The second baseline method flags patterns of part-of-speech n-grams as ungrammatical if they occur below a certain number of times in a reference corpus of correct English.

We demonstrate that combining information from the above baseline methods and our distorted treebank method, i. e. information from a variety of linguistic sources, is helpful. In particular, incorporating information from probabilistic parsing can lead to significant improvements.

1.3 Outline of Chapters

Chapter 2 Research on methods for detecting grammatical errors in English text using probabilistic parsing draws from a wide range of fields including computational linguistics, computer-assisted language learning and machine learning. Chapter 2 aims to provide an overview of the relevant fields and of the main concepts of data-driven error detection.

Chapter 3 Statistical methods including those we will employ rely heavily on data for training and evaluation. The first half of Chapter 3 presents the corpora we use, including preprocessing, part-of-speech tagging and error annotation. The second part of Chapter 3 deals with evaluation. We discuss the implications of evaluating with accuracy on two test sets (grammatical and ungrammatical test data) both on development and final testing.

Chapter 4 The APP/EPP method for using the parse probability assigned by a probabilistic, treebank-induced grammar (APP) for grammaticality judgements occupies a full chapter as it requires the development of a new data-driven model that we call the EPP model. In addition, Chapter 4 provides empirical evidence that parse probabilities reflect grammaticality and provides necessary background on probabilistic parsing.

Chapter 5 We evaluate four basic methods for automatically evaluating the grammaticality of a sentence that share the property that they do not rely on machine learning to set their parameters: *(a)* parsing with a hand-crafted precision grammar, *(b)* flagging unattested or rare part-of-speech n-grams, *(c)* pruning rare rules from PCFGs in an attempt to make them more discriminative, and *(d)* the distorted treebank method that compares parse results with vanilla and error-distorted treebank grammars.

Chapter 6 Three methods of Chapter 5 (leaving out the unsuccessful PCFG pruning method) are developed further using machine learning in Chapter 6. First, we extract features for learning and show the effect of machine learning on each basic method. Then we combine feature sets, building stronger methods. Finally, we propose to tune the accuracy trade-off of machine learning-based methods in a voting scheme.

Chapter 7 We compare results of Chapters 4 to 6, expand the evaluation to a breakdown by error type and test selected methods on real learner data. We observe varied performance and draw conclusions for future work involving our methods.

Chapter 8 The final chapter summarises the contributions, main results and lessons learned. We discuss implications for future work on error detection methods, not limited to the methods we present.

Chapter 2

Previous Work

Research on methods for detecting grammatical errors in English text using probabilistic parsing draws from a wide range of fields including computational linguistics, computer-assisted language learning and machine learning. Consequently, researchers come from different backgrounds and publish in different journals and at different conferences. This chapter aims to provide an overview of the relevant fields and of the main concepts of data-driven error detection.

Previous parsing-based work in the area of grammar checking focuses on techniques that are activated once parsing has revealed that a sentence is ungrammatical with respect to a given hand-crafted grammar. In contrast, our research explores new methods for making grammaticality judgements. Therefore, the literature on grammar checking is not quite as relevant to probabilistic error detection as one would expect. A short overview of the traditional techniques should be sufficient to note the marked difference between the methods developed in this thesis and previous parsing-based approaches.

In Section 2.1, we start with an overview that briefly reviews each field and points to more detailed discussions in Chapters 2 to 7. Section 2.2 introduces the main ideas of data-driven methods to grammatical error detection: the types of patterns that are used to identify errors and how these patterns are learned from training data. More concretely, Section 2.3 describes methods targeting three major error types that have received much attention in the literature on data-driven error detection. We continue in Section 2.4 with an overview of approaches to parsing ill-formed input, i. e. techniques that have

been developed to analyse grammatical errors or to make parsing more robust. Finally, Section 2.5 concludes with a summary of the chapter.

2.1 Diversity of Relevant Fields: an Overview

This section gives an overview of relevant fields and points to sections of the thesis containing more details and, where appropriate, discussion of the literature.

2.1.1 Grammaticality Judgements

Since the task we address with our research is to automatically decide whether a sentence is grammatical or not, the various aspects of grammaticality judgements are important. In Section 3.2.2 of Chapter 3, we point to some literature on inter-annotator agreement for this task which turns out to be surprisingly low, not only for trained language teachers but also for expert linguists. A possible explanation for the low level of agreement is gradience in grammaticality: in contrast to Chomsky's hypothesis (Chomsky, 1957), no clear-cut separation of grammatical and ungrammatical strings is assumed and grammaticality is graded (Gisbert Fanselow and Schlesewsky, 2006). Alternatively, the notion of grammaticality can be rejected completely. A full issue of a major journal is devoted to the question of whether "grammar without grammaticality" is possible (Sampson, 2007).

In our research, we take a practical approach and rely on the judgements found in various annotated error corpora (Section 3.2 of Chapter 3). As a rule of thumb, constructions are treated as ungrammatical if they require a correction in order to be acceptable to a broad audience of educated native speakers or if a language teacher thinks they need to be flagged. This may be less strict than, for example, Rosén and de Smedt (2007) who reject a coordination of unlike constituents in a sentence produced by a professional translator, speculating that a "really careful copy editor" would have changed it. We even go so far to use the British National Corpus (Section 3.1 of Chapter 3) as grammatical data even though it clearly does contain some material that falls inside the above rule of thumb for ungrammatical constructions.[1] Even less strict is Boyd (2010) who only flags sentences as

[1] For example, *What would be **the the** consequences of it. [BNC JT8.408]*

ungrammatical if there is no hypothetical context in which they could be used.[2]

Automatic grammaticality judgements can be made in various ways. Chapters 4 to 6 add new methods to the pool of methods and discuss related work. Data-driven methods (Section 2.2) are usually seen as approximations that improve as the amount of training data supplied is increased. Certain phenomena may remain outside the scope of a model even with unlimited training data, e. g. long distance dependencies with n-gram models (Section 5.2 of Chapter 5). In contrast, grammar writers in the Chomskyan tradition aim to find a finite set of rules that cover exactly all grammatical constructions. Figure 4.1 and the first paragraph of Section 4.1 of Chapter 4 illustrate this idea within the generative grammar framework. More concretely, Section 5.1 of Chapter 5 exemplifies this approach with the ParGram English LFG grammar.

2.1.2 Frequency, Probability and Parsing

The term "grammaticality judgement" is also associated with research undertaken in psycholinguistics and cognitive science. Subjects are asked to assess sentences specifically designed by experimenters to infer new knowledge of, for example, the structure of language or how humans process language. One of the aspects studied is the role of the frequency of structures which is linked to our research question of whether probabilistic models can be used for grammaticality judgements. Section 4.4.1 of Chapter 4 points to some literature.

The frequencies of structures are also considered outside psycholinguistics: the frequencies of grammar rules in parsed corpora are used to inform the rule probabilities of probabilistic grammars. In Section 5.3 of Chapter 5, we experiment with removing rules with low frequency from a grammar in order to use the grammar for making grammaticality judgements and discuss related work.

Since we try to use generative parse probabilities for making automatic grammaticality judgements, probabilistic parsing must be discussed. The key idea in probabilistic parsing is to assign a likelihood to each parse tree in a generative parsing model in or-

[2] Note that this definition *(a)* relies on the annotator to be able to find suitable contexts and *(b)* requires that certain contexts are excluded, e. g. contexts that talk about language and errors or contexts that give words new meaning.

der to disambiguate between competing analyses.[3] Section 4.1 of Chapter 4 provides an overview. Related probabilistic models are language models as they describe the likelihood of sentences or strings instead of parse trees.

An important resource for probabilistic parsing are treebanks, i. e. corpora manually or semi-automatically annotated with parse trees. Treebanks provide higher quality data for inferring grammar rules and setting rule probabilities than fully automatically parsed corpora. Throughout our experiments, we use the Wall Street Journal section of the Penn II Treebank (PTB) which has been manually annotated with phrase structure trees (Marcus et al., 1994). This treebank is the basis for the probabilistic grammars we use in our experiments and we also use it in other ways, e. g. to induce language models that mirror the lexical coverage of the grammars (Section 4.5.6 of Chapter 4). We talk about treebanks in many places throughout Chapters 2 to 5.

In order to use parse probabilities for automatic grammaticality judgements, one has to be able to obtain them, i. e. to parse any given input. Therefore, robustness is important. As we will discuss in Chapters 4 and 5, PTB-induced grammars are inherently robust.[4] For other (often hand-crafted) grammars, various robustness techniques have been developed that explicitly deal with input that is outside the coverage of a core grammar and can therefore be used to detect grammatical errors with respect to this core grammar — see Section 2.4 below.

2.1.3 Task and Methods

As illustrated in Chapter 1 with Figure 1.1, research on grammatical error detection can address a number of different tasks, among others: identifying that there is an error, locating the error, determining the type of error, making or assisting in making corrections and providing feedback for language learning. Research focusing on any of these tasks may

[3] It is also possible to disambiguate between parses with discriminative models. A common type of discriminative models is probabilistic in the sense that, for each sentence of the training data, the probabilities of its parse trees sum to one. In contrast, in a generative model, the probabilities sum to the probability of the sentence to be generated, which can be a very small probability on its own. We limit our current research to generative parsing models as the generative probability of the best parse approximates the probability of the sentence and therefore potentially says more about the sentence than a discriminative probability which focuses on ranking the parse tree in respect to competing parse trees.

[4] As long as unseen words are assigned some POS (other than the POSs for punctuation, quotes, brackets, symbols and interjections), even a basic PTB-induced PCFG will parse any input (Charniak, 1996).

contain relevant ideas for the task we address. However, results are difficult to interpret and to compare across tasks. In practice, research also differs in other ways, e. g. the area of application, evaluation measures, corpora being used and the range of error types addressed (see also the discussion in Section 2.2.1 below), and does not always easily fit categories as those given in Figure 1.1.

Correspondingly, the methods employed are diverse as well: an overview of the most important methods is given in Section 2.2.4 below.

2.1.4 Grammatical Errors

Error detection methods differ as to the types of grammatical errors they address and whether they are specifically designed for these error types. On the one end, there are methods that focus on a single error type, e. g. article errors, — see Section 2.3. On the other end, methods are anticipation-free, i. e. these methods should detect any deviation from grammatical language and are implemented without a knowledge resource describing possible errors.[5] In between are, for example, methods that learn error patterns from an error corpus. While machine learning methods try to generalise from seen examples, it is difficult to generalise to new error types and it can be argued that such methods anticipate the errors they have seen in their error corpus. Independently of error anticipation, a method may show inherent strengths or weaknesses in detecting errors of particular types.

Even before a unit of text, for example a document, a sentence or an individual word, is considered, there is a priori information that an error detection system should include in its judgement: what is the expected error density and how likely is each error type based on the type of text, the circumstances of writing, the native language (L1) of the L2 learner,[6] the learner level, the immediate textual context and what is known about the writer. For example, Wang and Garigliano (1992) discuss the importance of modelling L1 influence based on data they collected.

A systematic classification of error types, in particular if it is hierarchical, is called an error taxonomy. Classification criteria can be, for example, the edit operations needed for

[5] Klenner and Visser (2003) also use anticipation freeness to distinguish the mal-rules approach from constraint relaxation for error diagnosis, see Section 2.4.

[6] L2 refers to any language being learned after the first language.

correction or the part-of-speech involved — see Section 3.2.1 of Chapter 3.

2.1.5 Corpora, Error Corpora and Annotation

Broadly speaking, corpora serve three purposes in error detection research: manual inspection of the types of errors to expect (e. g. the corpus analysis summarised in Section 3.2.1 of Chapter 3), automatic training of detection systems (Sections 2.2 and 2.3 below) and evaluation (Sections 3.4 to 3.7 of Chapter 3). To make clear that the corpora intentionally contain (grammatical) errors, either authentic or artificially created, the term "error corpus" is used.

However, error detection research often also employs corpora of grammatical texts, e. g. standard corpora also used in other areas of NLP that typically consist of newspaper text. One re-occurring theme is to use the web as a corpus: for example, Ringlstetter et al. (2006) retrieve web documents found with a web search engine and seed keywords to build domain-specific corpora. They then remove documents containing orthographic errors with a method that does not simply reject words unknown to the system but relies on a list of possible misspellings that is automatically generated from a dictionary of correct words. Gamon et al. (2008) retrieve examples of correct language use from a web search engine to help the user to select a correction from a list of candidate corrections. Instead of compiling a corpus from web documents, some work directly uses the statistics provided by web search engines, e. g. Elghafari et al. (2010) use the number of web pages the Yahoo search engine reports for 7-grams centred around potential preposition errors to rank candidate corrections and the original input, backing off to lower order n-grams if no matches are found.[7] Kilgarriff (2007) comments critically on using such statistics. Reproducable and more reliable results are possible with the Web 1T 5-gram corpus from Google.[8] Gamon and Leacock (2010) compare this resource to Bing and Google search engine statistics in the task of preposition and article error detection and find that each resource has strengths and weaknesses. Tetreault and Chodorow (2009) show that patterns of language learner errors can be mined from the web — see Section 3.2 of Chapter 3.

Error corpora become more useful if they are annotated with information on each er-

[7] See Sections 2.3.2 and 2.3.3 for more previous work using web search for error detection.
[8] Available through the Linguistic Data Consortium, University of Pennsylvania.

ror, on the writer or speaker, possible corrections etc. Recently, a number of annotated error corpora have been presented (Albert et al., 2009; Boyd, 2010; Rozovskaya and Roth, 2010a; Dahlmeier and Ng, 2011b). Error type annotation requires an error taxonomy. Tetreault et al. (2010b) show that untrained raters can be effectively employed to annotate error corpora. In addition to error-specific annotation, error corpora can contain annotation familiar from corpora of grammatical language. However, some adaptations may be necessary, e. g. Díaz-Negrillo et al. (2010) argue that part-of-speech annotation of learner corpora requires a different tag set than native speaker material.

2.1.6 Applications and Related Fields

As indicated in Chapter 1, there are many areas of application for error detection methods, in particular if we consider that most automatic classification methods can be adapted to give some kind of confidence score that can be used for a graded ranking of sentences instead of a binary classification. In the case of grammaticality judgements, this gives us a grammaticality score that can be used to grade the well-formedness or fluency of candidate sentences, e. g. translation candidates generated by a machine translation system. The relevance of application areas is two-fold: firstly, they provide a motivation to develop error detection systems apart from advancing scientific knowledge. Secondly, surveying existing methods of these areas may give us new ideas for error detection research. We see the following application areas:

- **Grammar checkers for native speakers and language learners, e. g. in a word processor:** in addition to error detection on the sentence level, a more precise location of errors and the generation of candidate corrections are desirable. A stand-alone sentence classifier can still be useful to control overflagging, i. e. the reporting of errors for grammatical structures, in a second-stage error detection component.

- **Error detection in CALL exercises or tutoring systems:** in CALL, the range of possible applications is wider but so can be the demands on error categorisation and feedback generation.

- **Essay grading:** Error detection can be used to estimate the number of errors in

text, e. g. essays, even if individual errors are not detected as reliably as desired for grammar checkers or CALL applications. An estimation of the error rate complements other measures such as semantic similarity to a reference essay (Duwairi, 2006) and semantic coherence (Bestgen et al., 2010). According to Lonsdale and Strong-Krause (2003), traditional parsers have not been successful in essay grading due to insufficient robustness and computational complexity. They therefore propose to use a more lightweight parsing system, a shallow dependency parser, improve its coverage and error anticipation and evaluate the parser in the task of grading essays on a corpus of human-rated L2 essays.

- **Sentence ranking in such areas as machine translation, natural language generation, optical character recognition and automatic speech recognition:** see Section 4.4.4 of Chapter 4 and our note on the work of Carter and Monz (2009) in Section 4.2.

- **Post-editing and evaluation of machine translation output:** Stymne and Ahrenberg (2010) show that corrections proposed by a Swedish grammar checker improve machine translation output and that system evaluation metrics based on this grammar checker give a different ranking of machine translation systems than the standard evaluation metric (BLEU). Some research on evaluation measures for machine translation output uses similar methods and casts the problem as a task similar to judging the grammaticality of a sentence: Corston-Oliver et al. (2001) build a classifier that distinguishes between human translations (presumably grammatical) and machine translation output (often ungrammatical) with an accuracy of nearly 83% (equal number of grammatical and ungrammatical test sentencens).

- **Augmentative and alternative communication:** Minnen et al. (2000) see an application of their research on article generation in helping users with motor impairments to complete keyboard and other slower device input. They observe that articles are often omitted, degrading quality of subsequent text-to-speech synthesis.

In addition to learning from methods employed in application areas, there are fields of research that are relevant to error detection as they share methods or address a similar

task:

- **Readability measures:** readability measures are proposed for selecting learning material, e. g. Heilman et al. (2008), and for use in authoring tools, e. g. vor der Brück et al. (2008). In this area, we also came across the interesting idea of combining a binary classifier with a robust sorting algorithm to rank a list of texts (Tanaka-Ishii et al., 2010). This idea could be adapted for error detection to rank sentences by grammaticality and to only flag the most suspicious sentences, e. g. if a human reviewer has only limited time.

- **Spelling error correction:** while spelling error detection usually relies on a simple dictionary lookup,[9] spelling error correction methods can be interesting for grammatical error detection using the candidate correction approach — see Section 2.2.4 For example, Elmi and Evens (1998) present efficient methods for scanning a lexicon for spelling variants with multiple character errors (edit operations).

- **Statistical machine translation (SMT) as an error detection method:** Brockett et al. (2006) detect and correct errors using SMT trained on a parallel error corpus, i. e. ill-formed sentences with corrections. Hermet and Désilets (2009) propose a different model that translates text produced by an L2 learner into their native language (L1) only to translate it back to L2. The assumption is that errors of the L2 learner are caused by L1 interference and that such errors can be repaired by an L2-L1-L2 translation cycle as the system falls back to word-for-word translations for erroneous parts. While they do not find any statistically significant difference of the round-trip model to a basic candidate correction approach with web search counts (see Section 2.2.4), a hybrid method combining the two methods outperforms these methods. Dahlmeier and Ng (2011a) directly populate the phrase table of a statistical machine translation system with real word spelling variants, homophones and synonyms of words, and paraphrases for collocation candidates in an error correction system for collocation errors.

[9] Spelling error detection is far from trivial though: a lexicon always has limited coverage and a lexicon entry may match the input even though a different word was intended — see Section 2.3.1 on real-word spelling errors.

A better understanding of how the L2-L2 and L2-L1-L2 translation methods achieve their error detection performance may lead to new error detection methods.[10]

- **Information retrieval and question answering:** Pinchak et al. (2009) transform pairwise preference decisions of a binary classifier into a ranking of candidate answers to a question.[11] The classifier predicts which answer of two answers is more appropriate based on the difference of their feature vectors and Pinchak et al. (2009) then, using methods of Joachims (2002),[12] turn the pairwise decisions into a ranking.

- **Other areas:** classification tasks on the sentence, paragraph or document level can be found in many areas of NLP, for example sentiment analysis and authorship identification. Even research outside of NLP may give ideas for error detection research, e. g. see the pointers to deviation and outlier detection in Section 4.4.2 of Chapter 4.

2.2 Data-Driven Methods

We distinguish error detection systems which make use of hand-crafted rules to describe well-formed and ill-formed structures from purely data-driven systems which use various means to automatically derive an error detection system from corpus data.[13] Here, however, we focus on data-driven error detection systems and we attempt to categorise these methods according to the nature of the task, the type of features or patterns that are automatically extracted from the data and the type of data used.

2.2.1 The Nature of the Task

Our work is most closely related to that of Andersen (2007), Okanohara and Tsujii (2007) and Sun et al. (2007), since all three are concerned with the task of automatic grammat-

[10] Machine translation is also used to generate erroneous test data — see Section 3.3.1 of Chapter 3.

[11] More precisely, Pinchak et al. (2009) deal with answer types.

[12] Joachims (2002) presents a method for improving the ranking of search results in an information retrieval system using the selections of the user as feedback.

[13] Bender et al. (2004), for example, describe a hand-crafted system in which input sentences are parsed with a broad-coverage, generative grammar of English which aims to describe only well-formed structures. If the sentence cannot be parsed with this grammar, an attempt is made to parse it with mal-rules which describe particular error types. In Chapter 5, we evaluate a broad-coverage, hand-crafted, generative grammar in the task of classifying sentences as either grammatical or ungrammatical and then integrate it into our data-driven method in Chapter 6.

icality judgements, i. e. classifying a sentence as either grammatical or ungrammatical. Other error detection research focuses on identifying and possibly also correcting errors of one particular type, e. g. errors involving

- articles (see Section 2.3.2),
- prepositions (see Section 2.3.3) and
- real-word spelling errors (see Section 2.3.1), as well as
- verbs (Gamon et al., 2008; Lee and Seneff, 2008).

2.2.2 The Nature of the Pattern

There are many possible features of a sentence to which an error detection pattern can refer. If we are to acquire patterns automatically from corpora, we have to restrict their type in order to make the extraction process tractable. Often, automatically acquired error patterns are limited to word or part-of-speech (POS) sequences of a certain length (Golding and Schabes, 1996; Verberne, 2002). For example, the sequence of three POS tags "determiner determiner noun" might indicate an error in English while the sequence "determiner adjective noun" does not. The choice of features limits the types of errors that can potentially be detected.[14] In the example above, the POS information does not handle agreement phenomena between determiner and noun. This gap can be filled by extending the POS tag set such that singular and plural determiners and nouns are distinguished. In some work, closed class words are not reduced to their POS tags, effectively augmenting the POS tag set to be as fine-grained as possible for prepositions, pronouns, etc.

In NLP, sequences of n words or POSs are called n-grams. They are widely used for error detection — an overview is given in Section 5.2.1 in Chapter 5 — but are not the only type of patterns in previous work: Sun et al. (2007) extend n-grams to non-continuous sequential patterns allowing arbitrary gaps between words. In addition, patterns are collected for all n. Sjöbergh (2006) uses sequences of chunk types, for example "NP-VC-PP". The parse trees returned by a statistical parser are used by Lee and Seneff (2008) to detect verb form errors, and by Wong and Dras (2010) to detect five major error types.

[14] The extraction method and the corpus also impose limits, see Section 2.2.3 for the latter.

Patterns extracted from parse trees are not restricted to the local context provided by n-grams. A third example is the proof of concept system of Gojenola and Oronoz (2000) who use syntactic error patterns to detect errors in date expressions. The work of Malmsten and Klasen (2005) is another example. It is interesting for us as it uses Charniak's parser[15] which we also use in our experiments in Chapters 4 to 7. They hand-craft error patterns for parse trees that this parser outputs: firstly, they write patterns that detect target errors in a set of ungrammatical sentences. Then they make the patterns more specific to reduce overflagging in a larger corpus of grammatical sentences. The system has 100% accuracy on a randomly chosen test document but, unfortunately, the size of this document is not reported.

Patterns can also be augmented with linguistic features of the sentence in which they occur, for example the overall density of function words. Section 6.4.1 of Chapter 6 discusses previous work on automatic error detection that combines heterogeneous feature sets, including linguistic features. We will think of such features as being part of the patterns, e. g. a pattern "determiner determiner noun 5 17" could mean that the POS trigram "determiner determiner noun" is present in a sentence containing 5 function words and 17 words in total. Of course, this makes the patterns very specific and unlikely to match new input. However, machine learning methods are designed for the task of generalising from training data to new unseen data.

To use POS information or even full parse trees in error patterns, the reference corpus and test sentences have to be annotated with the required information. Automatic POS tagging and parsing introduce annotation errors and may have a bias towards annotating grammatical structures even if the input is ungrammatical as such systems are built assuming that input will be grammatical. It seems plausible that annotation errors will have negative effects on the detection accuracy for both grammatical and ungrammatical sentences.

[15] Neither the version of the parser nor the year of the respective publication is given.

2.2.3 The Nature of the Data

Data is central to data-driven methods. This section explains the categorisation of data for error detection systems into positive and negative data, illustrates the use of data with basic methods and points to further literature. More background on data gathering for error detection research can be found in the first three sections of Chapter 3 which describe the data sets that we use for our research throughout Chapters 4 to 7.

In this section, our examples focus on simple n-grams as patterns as these are easiest to understand, but the same principles hold for more complex patterns, e. g. patterns extracted from parse trees.

Positive Reference Data Only

Grammatical text is available in vast quantities for many languages, for example in news, parliamentary proceedings and free electronic books.[16] The simplest method of using such text corpora for grammatical error detection is to treat every pattern that is not attested in the corpus as an indicator of an error. For POS trigrams, for example, the list of all possible trigrams is manageable and simply needs to be ticked off while reading the corpus sequentially.[17] For POS n-grams with higher n, or for other types of patterns, more sophisticated indexing methods have to be employed.

The type of pattern also determines how much data is needed in order to reach good coverage of grammatical language with the basic "attested patterns are acceptable" approach. Raw token n-grams are particularly data hungry as each n-gram has to occur verbatim in the positive reference data[18] in order to be accepted as grammatical. Any n-gram that contains a new word or a new combination of attested words will be flagged as ungrammatical.

A number of methods are available to break the close relationship of data size and language coverage:

[16] For example, Project Gutenberg stores over 25,000 books (270 GB of text) as of July 2008. Assuming 1/4 is English, this is 90 times the British National Corpus described in Section 3.1 of Chapter 3.

[17] The trigram table would only need $50^3 = 125{,}000$ bits (less than 16 KB) of memory assuming a POS tag set containing 50 tags.

[18] We use the term "reference data" to refer to data that we use to look up basic statistical properties, e. g. the frequency of patterns. From a machine learning point of view, reference data is training data as it contributes to the model we are learning. We reserve the term "training data" for data that is used to set core model parameters, e. g. n giving the size of n-grams used.

- We can impose more strict criteria on the patterns, e. g. more errors will be detected (but also more grammatical sentences will be rejected), if more than one occurrence in the reference data is required for a pattern to be acceptable. This stricter criterion can be necessary for various reasons: firstly, the reference corpus may in fact contain ungrammatical language. Secondly, there may be patterns that sometimes occur in grammatical sentences but are more likely to be caused by an error. Thirdly, tagging errors can distort the reference corpus. These problems grow with the size of the reference corpus and can be counteracted by a higher frequency threshold for acceptable patterns.

- We can accept some unseen patterns by generalising patterns. This can be motivated by the observation that patterns that are unattested in the reference corpus can still occur in correct sentences. Therefore, it is desirable to generalise from the seen patterns. Bigert and Knutsson (2002) propose a similarity measure on POS n-grams to extend the set of acceptable n-grams. Gamon et al. (2008), Tetreault and Chodorow (2008a) and De Felice and Pulman (2008) exploit machine learning using word, POS and parser features to learn a model of correct usage from positive data only, and then compare actual usage to the learned model.

- Finally, candidate corrections can be used to make error detection independent of the absolute frequency of patterns in positive reference data. Both methods above (imposing a higher frequency threshold and accepting patterns that are similar to an attested pattern) still suffer from the basic problem that adding more data in order to improve the coverage of grammatical language has the side effect of deteriorating the detection of errors as error patterns (or patterns similar to error patterns) also occur in positive data. Instead of using the absolute frequency of a pattern, Gamon et al. (2008) and Lee and Seneff (2008) measure the frequency relative to candidate corrections proposed by a separate component of the error detection system. If a candidate correction's patterns are more frequent than the input's patterns, the original input is assumed to be erroneous. Therefore, a pattern is not simply accepted as grammatical if it (or a similar pattern) has been (frequently) observed in positive reference data. It has to be at least as frequent as the patterns of all candidate

corrections — see also Section 2.2.4 below.

A priori information on how likely it is that certain errors are made or that the input is correct can be included into the candidate correction approach using the noisy channel model that effectively adds an error-dependent threshold that has to be crossed before a candidate correction is considered to be the intended string. Again, see Section 2.2.4 below for more details on this method.

While all these methods only use positive reference data, it should be noted that a small amount of negative data, i. e. an error corpus, is necessary to tune the system parameters (type of patterns, frequency thresholds etc.) before applying the final system to unseen test data.

Adding Negative Reference Data

Negative reference data consists of a corpus of (mostly) ungrammatical sentences, optionally annotated with the location and type of errors — see also Section 3.2 of Chapter 3. If there is error annotation in the negative reference data, patterns indicative of errors can be extracted more reliably. The presence of a pattern in negative reference data reinforces the information gained from the absence of the same pattern in positive reference data. Han et al. (2010) confirm this positive effect of adding negative data. A basic method therefore simply flags all patterns as ungrammatical that appear in the negative data but do not in the positive data. For example, Ringlstetter et al. (2006) generate artificial negative patterns (misspelled words in their case) and filter these with a list of positive patterns. As with positive data, this method can be extended by looking at the frequencies of patterns. The frequency ratio between positive and negative reference data is a possible measure of the discriminativeness of a pattern (Sun et al., 2007).

As in the case of using only positive reference data, it is possible to generalise to patterns that cannot be found in any of the positive and negative reference data sets. Hand-crafted similarity measures are not used here, to our knowledge. Instead, machine learning methods are applied to automatically induce a classifier that discriminates between grammatical and ungrammatical patterns based on some features of the pattern, for example by Andersen (2007); Okanohara and Tsujii (2007); Sun et al. (2007) and in

our work in Chapter 6, partly published in Wagner et al. (2007a).

As large amounts of authentic negative data is difficult to come by, it is often artificially generated by replicating errors observed in authentic data — see Section 3.3 of Chapter 3.

2.2.4 Models and Learning

Section 2.2.3 above briefly described basic error detection methods using the frequency of patterns in positive and negative reference data, pattern similarity and automatically induced classifiers (machine learning). This section gives more background on the candidate correction approach, the noisy channel model and machine learning.

Confusion Sets and Candidate Corrections

The idea of the candidate correction approach to error detection is to choose the most likely candidate from a set of possible corrections, including the original input, and to flag the input as erroneous if the input is not chosen. Two problems need to be addressed: *(a)* generating candidate corrections and *(b)* choosing among them. An important solution to the generation problem, initially proposed for real-word spelling errors (Section 2.3.1 below), is to draw up sets of words that are frequently confused, e. g. from observations in error corpora or using lexical resources. Golding and Schabes (1996) compile 18 confusion sets, mostly taken from a dictionary appendix "Words Commonly Confused". Given an input word, it can be replaced by any other word of a confusion set in which it appears. Golding and Schabes (1996) keep the confusion set disjoint. Therefore, to choose a candidate, a disambiguation model can be trained on grammatical training data for each confusion set. Leacock et al. (2010) point out that such methods are similar to word sense disambiguation[19] where each word is associated with a set of word senses and the task is to predict the correct word sense based on contextual clues.

The following methods develop the confusion set idea further: Rozovskaya and Roth (2010c) include the information on which item of the confusion set was chosen by the writer in the prediction process. They argue that it is often safest to replicate the writer's decision as the accuracy of disambiguation methods is often lower than the accuracy

[19] See McCarthy (2009) for an overview.

of the writer's choices. Therefore, the correction module should only intervene if it is highly confident to have found an error. Rozovskaya and Roth (2010c) conclude that an evaluation of the selection accuracy in the confusion set disambiguation task says little about error detection and correction. Evaluation has to focus on the actual task. The baseline of the detection and correction tasks is to report no error, not to propose the most frequent preposition, article etc. Rozovskaya and Roth (2010b) further improve the method by limiting confusion sets to confusions observed in an error corpus with the L1 of the writer.

To select a correction from a confusion set, usually a classifier is trained for each set as each set poses a different selection problem. However, confusion sets can follow patterns, e. g. easily confused inflectional or derivational word forms as in the work of Stehouwer and van den Bosch (2009) who focus on Dutch words ending in *d* and *dt*. Here, confusion sets follow patterns like the Dutch verb forms $\{word, wordt\}$ and $\{houd, houdt\}$. A single "monolithic" classifier can be trained that makes the morphological decision independently from the base form. One advantage is that more training material is available as the union of all training sets of the individual confusion set classifiers can be used. Stehouwer and van den Bosch (2009) go one step further and also combine the monolithic and individual classifier approaches with an ensemble classifier that consults the individual classifiers for some confusion sets and backs off to the monolithic classifier for all remaining confusion sets. While the classifiers are optimised for accuracy of selection, Stehouwer and van den Bosch (2009) also evaluate error detection accuracy and interestingly find a different ranking of system configurations.

A choice among a set of candidate corrections can also be made independently of confusion sets and other candidate generation methods: for example, Gamon et al. (2008), Lee and Seneff (2008) and Elghafari et al. (2010) use frequency information obtained from reference corpora or web search results and van Zaanen (1999) and Lee and Seneff (2006) use probabilistic parsing models.

Noisy Channel Model

The basic confusion set or candidate correction approach postulates an error if a correction is preferred (by a classifier) or more likely (according to a probabilistic model) than the word(s) chosen by the writer, not taking into account that a less likely input sentence may well be grammatical, for example if multiple sentences are possible or similar in likeness. As we have seen in the previous section, this limitation can be addressed by including the input candidate as a feature in a classifier framework, given that an annotated error corpus is available for training. However, often, no or very limited error data is available. Artificial error data (Section 3.3 of Chapter 3) can alleviate the problem but a prerequisite for generating artificial error data is an error model (Section 3.3.2).

The noisy channel model is a tool that allows us to combine probabilistic models for error insertion and (grammatical) sentence generation: we want to find the most likely correction S' given the input S, i. e. we want to maximise $\mathrm{P}(S'|S)$. Using Bayes' Rule, this can be rewritten as $\mathrm{P}(S') \times \mathrm{P}(S|S')/\mathrm{P}(S)$ where $\mathrm{P}(S')$ is the probability of generating sentence S', $\mathrm{P}(S|S')$ is the probability of erroneously writing S when S' is the intended grammatical sentence, and $\mathrm{P}(S)$ is the probability of generating sentence S. Note that $\mathrm{P}(S)$ is not needed for finding an S' that maximises the term. In terms of the noisy channel model, the correction S' is the source which is distorted by noise into S according to the probability distribution $\mathrm{P}(S|S')$. At the end of the channel, we see S and would like to restore S'. In less technical terms, the noisy channel model adjusts the score of each candidate correction by a penalty according to the error insertion model.

Despite the noisy channel model's plausibility, it has rarely been used in error detection research. Mays et al. (1991) apply the noisy channel model to candidate correction disambiguation. They use a trigram language model for the sentence generation model $\mathrm{P}(S')$ and a simple two-valued function for $\mathrm{P}(S|S')$ that assigns a high probability α to the case $S = S'$ (no correction needed) and distributes $1 - \alpha$ over the remaining candidates. This method is re-evaluated by Wilcox-O'Hearn et al. (2008) on WSJ data with artificial real-word spelling errors. Since they experiment with different error densities for the test data and also vary other parameters for testing and training, the results span large ranges on precision, recall and f-score measures, for example 0.200 to 0.709 for f-score of the error

correction task. Their main finding is that increasing the vocabulary size of the trigram model improves results.

A uniform probability distribution over candidate corrections other than the input does not make full use of the noisy channel model. It effectively means that only a threshold is added by which a candidate correction has to be more likely than the input in order to be proposed as a correction (or an error to be detected). Such a model is present in other work not referring to the noisy channel model, e. g. Golding and Schabes (1996) introduce an ad-hoc threshold to get more favourable results in a comparison with a word processor's error correction module.

Machine Learning

Starting with Section 2.2.2, the previous sections gave an idea of key concepts of data-driven error detection: choosing pattern types, using frequency information observed in data and generalising to unseen patterns with manually designed pattern similarity measures. Machine learning concerns itself with automating the building of models that generalise from the observed patterns to all possible patterns. A large collection of methods is available in the machine learning literature. For an introduction, see for example Mitchell (1997), Witten and Frank (2000) or Marsland (2009). Three methods are discussed in Chapters 4 and 6:

- k-nearest neighbour (Section 4.5.1 of Chapter 4),
- decision trees (Section 6.1 of Chapter 6), and
- classifier combination (Section 6.5.2 of Chapter 6).

A fourth method, maximum entropy classification, is mentioned in Section 2.3.2 below.

In addition to being a tool for generalising from training data, the subfield of machine learning concerning itself with deviation or anomaly detection (Chandola et al., 2009) is relevant to error detection — see also Section 4.4.2 of Chapter 4.

2.3 Methods Targeting Individual Error Types

Most research on error detection and correction using statistical data-driven methods focuses on one type of errors, e. g. preposition errors. Important methods are developed in this setting and while we give an overview of these methods in Sections 2.1 and 2.2 above, the field of research can be understood better if broken down by error type. This section gives an overview of error detection methods for the three most-often addressed error types: context-sensitive or real-word spelling errors, article errors and preposition errors.

It should be noted that systems addressing only some error types do only partially address our task of detecting any deviation from grammatical language, and may have unpredictable behaviour when confronted with unanticipated errors. In principle, a collection of error detection systems each addressing a different error type can be combined into a general error detection system by flagging any input sentence as ungrammatical if one of the component systems reports an error. However, there is the problem that overflagging can be cumulative: say we have 5 systems that each flag 10% of grammatical sentences as erroneous. Depending on the correlation of overflagging of the systems, up to 50% of grammatical sentences will be flagged by the combined system that flags any sentence flagged by one of the five systems.[20] Overflagging could be countered by reducing the sensitivity of each component system but this would lead to lower detection accuracy than expected from the results reported for the individual methods.

Lee (2009) also points out that it is insufficient to handle error types in isolation in previous work. However, he focuses on robustness of error detection methods to neighbouring errors which often will be of a different type than the target error. The higher the error density, the more critical robustness becomes.

The three error types discussed in Sections 2.3.1 to 2.3.3 below are not the only error types discussed in the literature. The following list shows the diversity of possible error types. We do not restrict this list to error types addressed in isolation. Furthermore, error types in the list may overlap as different error taxonomies are used, and a single list item

[20] In this example, overflagging above $1-(1-0.1)^5 = 40.951\%$ is only possible with a negative correlation of systems' decisions.

may cover multiple, closely related error types.

- Collocations are particularly error-prone for advanced L2 learners: Nesselhauf (2005) shows a negative correlation between the number of years spent learning English and collocation use. Yi et al. (2008) include collocation errors in the evaluation of a web search-based error detection and correction system. Leacock et al. (2010) devote a chapter to collocation error detection and correction. Dahlmeier and Ng (2011a) correct collocation errors of L2 learners with "L1-induced paraphrases".

- Gamon and Leacock (2010) report that content word choice errors are the most frequent error type in the Cambridge Learner Corpus, an annotated error corpus of English language proficiency test (ESOL) essays (Nicholls, 1999). In addition, learner data often contains unidiomatic but still acceptable word combinations.

- Brockett et al. (2006) observe that mass noun and countable noun confusion errors are common among L2 learners and propose to use machine translation to correct such errors — see also Section 2.1 above. There is research on noun countability prediction in other areas of NLP, see for example Nagata et al. (2006) who improve predictions using the assumption that the countability of nouns does not change within a discourse. Noun countability also is an important factor for article selection (Section 2.3.2 below).

- Stehouwer and van den Bosch (2009) address erroneous word forms such as adjective-adverb confusions (*broad* vs. *broadly*) and wrong pronouns (*he* vs. *him*). Adjective-noun confusions are among the eight error types Gamon et al. (2008) target.

- False friends or false cognates are words that look or sound alike in the learner's L1 and L2 but differ substantially in meaning (Wagner, 2004). They can pose problems to L2 learners both in reading and writing. Amaral et al. (2011) categorises this error type together with "false translations chosen from bilingual dictionaries" as word choice errors.

- Lee and Seneff (2006) cover confusions across different part-of-speech, e. g. *on* (a preposition) vs. *an* (an article). To our knowledge, other work only considers confu-

sions within each part-of-speech, e. g. confusions of *on* with other prepositions. More precisely, Lee and Seneff (2006) allow any sequence of articles, auxiliaries, modals and prepositions to be inserted, deleted or substituted.

- In addition, Lee and Seneff (2006) include alternative inflections of nouns and verbs in the word lattice they use to search for the best candidate corrections — see Figure 2.1 in Section 2.3.2. Gamon et al. (2008) more specifically distinguish gerund/infinitive confusions, auxiliary verb errors, over-regularised verb inflections and noun pluralisation errors. Lee and Seneff (2008) discuss sub-types of verb form errors in terms of semantic and syntactic criteria.

- Accent errors are outside the focus of most research on error detection methods as most prototype implementations use English as the target language.[21] L2 learners of languages such as French can have difficulties with accents, especially if their L1 does not use accents. Amaral and Meurers (2009) observe accent errors in an I-CALL system for Portuguese and disambiguate between candidate corrections with a word lattice-based parser. Yarowsky (1994) restores accents in Spanish and French with an early implementation of the confusion set idea (Section 2.2.4).

- Nagata et al. (2010) automatically detect missing sentence-final punctuation in L2 English. A high density of misspelled words and errors in capitalisation make the problem hard, even for human readers. Nagata et al. (2010) train a binary classifier to predict whether a candidate sentence needs to be split into two (or more) sentences.

- Errors can be characterised by the edit operation required for correction, e. g. missing word and extra word errors. In Section 3.2.1 of Chapter 3, we point to previous work supporting error categorisation by edit operations.

- Yi et al. (2008) observe that 30.8% of sentences in an English L2 learner corpus are unintelligible, i. e. their native speaker annotator was unable to form a hypothesis as to what the learner intended to write and how it could be adequately corrected. It is difficult to pin-point this error type. If the distortion is caused by L1 influence, an

[21] In English, accent errors are probably best handled as spelling errors and with the inclusion of the few acceptable words like "naïve" in the lexicon.

annotator familiar with the L1 may be able to identify the intended meaning or the L2-L1-L2 round trip translation method of Hermet and Désilets (2009) may reveal the meaning.

2.3.1 Context-sensitive or Real-word Spelling Errors

At first sight, spelling correction has nothing to do with grammar checking. However, not all spelling errors result in non-words. Kukich (1992) summarises seven studies and concludes that between 25% and 40% of all spelling errors are real-word spelling errors, i. e. misspellings that produce an unintended existing word. These real-word spelling errors cannot be detected or corrected in isolation. Context is necessary to provide syntactic, semantic, or even pragmatic constraints as hints to what is intended. Spell-checkers exploiting at least the first two levels of information could handle sentences like the following adapted from Kukich (1992):

(2.1) *See you in five* ***minuets****.*

(2.2) *I need to* ***notified*** *the bank [of this problem].*

(2.3) *I can't pick him* ***cup*** *cuz he might be working late.*

(2.4) *He tries to* ***fine*** *out.*

Sentence (2.1) can be considered grammatical[22] given that *minuets* is a plural noun. However, its semantics are odd as dances are usually not used as a measure of time. The example sentence 2.1 shows that other types of information, e. g. a restricted vocabulary from an L2 curriculum or statistics of collocations, might be useful in error detection as well.

The task of detecting and correcting such spelling errors is often referred to as Context-Sensitive Spelling Correction (CSSC), hinting at the fact that correction methods have to take context into account. The shorter name "real-word spelling error" is also used and we will use the latter. Depending on the definition, error types covered can range from confusions between words that are similar on the character level (short edit distance, starting with the same letter) and that sound alike (near-homophone confusions) to word usage

[22] See Section 2.1.1 on the notion of grammaticality.

errors such as confusions between *amount* and *number* (Golding and Schabes, 1996).[23] Sometimes, a definition of real-word spelling errors includes extraneous, missing or misplaced spaces that do not result in non-words, e. g. ***Dot he*** *right thing* or *She is up to* ***here yes***.

As discussed in Section 2.2.4, disambiguation within confusion sets is often used for real-word spelling error detection (and correction). Pedler (2007) gives an overview and compiles a large corpus of real-word spelling errors made by dyslexic students. She disambiguates confusion sets using a two-stage architecture first using part-of-speech (POS) information and falling back to semantic information if the best candidate correction and the input word have the same POS.

In general, low-level NLP methods and frequency information are used: POS n-grams and phrase boundary detection are used by Bigert and Knutsson (2002), finite state automata by Hashemi (2003) and chunking into non-overlapping phrases by Sjöbergh (2006). Parsing is rarely used, with Elmi and Evens (1998) being one of the few exceptions. Not referring to any parser in particular, Bigert (2004) argues that language coverage is too low, grammar development too difficult and detection of multiple errors hardly possible for parsing to be worthwhile.

2.3.2 Article Errors

Article errors naturally lend themselves to the confusion set approach as there is a clearly defined small set of possible corrections: the indefinite articles *a/an*, the definite article *the* and the null element representing the absence of any article.[24, 25] Despite the limited choice, article selection is difficult both for L2 learners and automatic error detection systems. Han et al. (2006) find that TOEFL essays written by native speakers of Chinese, Japanese and Russian have article errors in 12% to 15% of all noun phrases. They discuss

[23] In our work, we do not include word usage errors into real-word spelling errors as doing so would open the error type to any word substitution errors such as preposition errors.

[24] The distinction between the two indefinite articles receives no attention in the error detection research known to us with the exception of Andersen (2007). However, this type of error is challenging to detect as the phonetic level, not the orthographic level, determines the choice and it is not always clear how a word is pronounced. Knight and Chander (1994) give the example *a NATO grant* vs. *an NIH grant*.

[25] Often, determiner error detection is taken synonymously for article error detection. Technically, however, determiners are a bigger class of words including quantifiers, demonstratives and possessives, e. g. *some*, *these* and *her*. A rare example of research on other determiners is Andersen (2007) who targets *this/these* determiner confusions among other error types.

in detail factors that affect article selection and give examples, e. g. a prepositional phrase can allow an uncountable noun to select an article as in *a knowledge of Spanish*. The investigation of Kiss et al. (2010) who try to shed light on the patterns of article omissions in prepositional phrases with singular count nouns, e. g. *at school*, suggests that article selection touches explanatory gaps in linguistics and is not just a problem of NLP. According to them, state-of-the-art grammars would wrongly demand an article in phrases like the above example. They investigate factors that trigger article omission by inspecting classifiers that have been trained to predict article usage in prepositional phrases. In order to test many candidate factors, Kiss et al. (2010) focus on annotating the training data with detailed linguistic information, e. g. preposition and noun senses.

Challenges

For a machine, there are three challenges: firstly, one of the three classes of the classification task is much more frequent than the other two classes: depending on the corpus used, between 70% and 72% of noun phrases[26] start without an article while an indefinite article appears in only 7.8% to 9.4% of cases (Han et al., 2006; Gamon et al., 2008). This can lead to a bias towards classifying instances with the majority class. Also, the skewed class distribution requires considerably more training data than for a uniform distribution to provide the same number of examples for the smallest class (indefinite articles). Addressing this issue, both Han et al. (2006) and Gamon et al. (2008), further expanded by Gamon (2010), improve results by splitting the task into a presence/absence classification followed by a (binary) article choice classification.

Secondly, local context is not sufficient to predict article selection: as an upper bound for the task, Knight and Chander (1994) report human performance of 79% to 80% accuracy if only a limited context consisting of the head noun and its pre-modifiers is provided. (If given full context, the human judges are reported to reach 94% to 96% accuracy.) Finally, world knowledge is needed, e. g. a discourse element can have been introduced indirectly or be familiar to the audience. Again, Han et al. (2006) give examples.

[26] It is assumed that noun phrases can be identified reliably and that noun phrases are the only error site for article errors. This assumption reduces the number of training items (as other potential error sites are not considered) and the computational costs of training. Without this assumption, the fraction of training items that predict the absence of articles would be even higher.

Early Work on Article Generation

Article selection is a major source of errors in machine translation output, especially when translating from languages without articles and/or number markings, and efforts have been made to improve the output either with rules inside the machine translation system or in an automatic post-editing module (Knight and Chander, 1994). Initially, rules and lexicon entries have been hand-written. Minnen et al. (2000) and Han et al. (2006) point to such work. To our knowledge, Knight and Chander (1994) are the first to automatically extract rules for article selection from corpora. Minnen et al. (2000) extend this work using more features of the training data, testing two machine learning methods and providing a more detailed evaluation.

Machine Learning Methods

Maximum entropy classifiers have become popular for the automatic disambiguation of confusion sets (Gamon, 2010). These classifiers are based on a probabilistic model of the training data that is motivated with the information theoretic definition of entropy and constructed using constraint optimisation theory (Berger et al., 1996; Rosenfeld, 1996; Ratnaparkhi, 1997). Interestingly, the resulting models are simple: a log-linear combination of feature values is normalised to ensure a proper probability distribution. As the normalisation term is constant for each input, maximum entropy classifiers effectively score candidate classes with a log-linear combination established in training.

Maximum entropy classifiers efficiently handle large feature sets. Therefore, features can encode the presence of specific words at specific positions relative to the candidate article and other information, e. g. Han et al. (2006) use around 390,000 features (details below). It would be intractable to directly optimise the correspondingly high number of parameters (one weight for each feature) for accuracy of the classification or even accuracy of error detection. Nevertheless, Andersen (2006) finds comparable accuracy of results with Maximum Entropy, Naïve Bayes and Balanced Winnow learning methods.

Han et al. (2004) present a maximum entropy classifier approach to article error prediction. They train on local contexts of token and part-of-speech tags extracted from a 31.5 million word subcorpus of the MetaMetrics text corpus, a 500 million word corpus

with English fiction, non-fiction and textbooks with a wide range of reading levels. They produce a learning curve showing that training data has to grow by an order of magnitude to achieve two percentage points of improvement. With 6 million noun phrases for training, they reach 88% accuracy in the article prediction task. The work is further expanded by Han et al. (2006) who evaluate this classifier in the article error detection task using 668 TOEFL essay for testing and 8 million well-formed noun phrases for training. (In 2004, the 8 million noun phrases were split 3:1 into training and test data in a 4-fold cross-validation setup.) Two annotators add article-specific annotation to all noun phrases in the TOEFL essays.[27] As the classifier produces many false positives and agrees with the human annotators much less than the two annotators with each other, Han et al. (2006) add a binary classifier for presence of articles but only gain a small improvement. They observe that wrong corrections are often only preferred over the input by a small margin and modify the binary classifier to only flag input for which the classifier is highly confident that the presence or absence of an article needs to be changed, essentially implementing a basic noisy channel model. Their work discusses possible reasons for poor performance, e. g. for *a-the* confusions.

Other machine learning methods have been used: Knight and Chander (1994) apply decision trees (see also Section 6.1 of Chapter 6) in the confusion set approach (excluding the null element, i. e. only definiteness is predicted). Due to computational costs, they train individual decision trees for the 1,600 most frequent head nouns and back off to predicting the most frequent article *the* if the noun is not covered. Effectively, this creates an overall decision tree with a top-level 1,601-way split. Knight and Chander (1994) further reduce computational costs by pruning the feature set (initially 30,000 features), approximating the splitting criterion (see Section 6.1 of Chapter 6) and indexing the training data (400,000 items).

Gamon et al. (2008) train decision trees with 75,000 features on a corpus of 1.56 million sentences and do not mention any efficiency issues. The choice of implementation seems to be crucial though as Andersen (2006) reports too high computational costs training with one decision tree implementation while a second implementation finishes training quickly.

[27] Test items that the human annotators could not judge or that have a misspelling in the head noun are removed from the evaluation.

Gamon et al. (2008) follow the confusion set approach predicting article and preposition selection using positive training data. They split the task into two steps: presence/absence classification and choice classification. Furthermore, candidate choices are filtered with a combination of language model score and confidence score of the first-stage classifiers. Results are reported both for the disambiguation task on positive data and the correction suggestions for L2 text.

Finally, k-nearest neighbour memory-based learning (see also Section 4.5.1 of Chapter 4) has been applied to article selection: Minnen et al. (2000) test IB1, a k-nearest neighbour implementation, with two different distance measures and three values of k and compare the performance to IGTREE, a decision tree variant that speeds up tree induction by globally pre-ordering features by their information gain (Daelemans et al., 1997, 1999). All six k-nearest neighbour methods outperform the decision tree method by a small margin (0.1 to 0.7 percentage points in selection accuracy).

Feature Sets for Article Prediction

Minnen et al. (2000) extract head information, part-of-speech tags, constituent labels and functional tags from noun phrases in the Penn Treebank, a corpus of manually parsed sentences. Additional information on countability and semantic class of the head noun is taken from a lexical resource. In their discussion, they identify that part-of-speech information distinguishing between pre-determiners and determiners gives the classifier information that would not be available in applications. For example, if the determiner *the* is deleted from the sentence

(2.5) *He ate half* ***the*** *cake.*

the pre-determiner *half* will still be tagged as a pre-determiner as Minnen et al. (2000) use the original gold tags from the treebank. The classifiers can then easily predict that an article must follow rather than a noun. However, in a more realistic setting, *half* is likely to be tagged as a determiner as it precedes a noun.

Han et al. (2006) use a chunker to identify noun phrases in well-formed text for training. Features are the concatenation of words and optionally part-of-speech of the noun phrase

in question (excluding the target determiner) and individual words to the left, right and start of the noun phrase.

De Felice and Pulman (2008) apply maximum entropy classifiers trained on positive data in a basic confusion set approach to article and preposition error detection. They include syntactic and semantic features of the context extracted using the RASP parser (Briscoe and Carroll, 2002; Briscoe et al., 2006) and WordNet (Miller et al., 1990; Fellbaum, 1998).

Using Web Search Frequencies

An alternative approach to extracting features and training machine learning methods is to rely on simple statistics of text matches, e. g. n-gram frequencies, and to compensate for data sparseness using large corpora that would be impractical or unavailable to use in any of the machine learning approaches.

Yi et al. (2008) detect article and collocation errors with web search frequency information. For article error detection, candidate corrections are generated (as in the confusion set approach) and are sent to a web search engine together with words chosen from the context of the candidate error site. Initially, a clause query is used. If the number of search results is too low, they back off to chunk queries and word level queries. To rank the candidate corrections, they normalise the web frequency by query length as shorter queries are likely to receive higher frequencies. In addition, a preference is given to the input sentence: the input will not be flagged as erroneous not only if the input is more frequent (after normalisation) than any candidate correction but also if its frequency is relatively close to the highest frequency of all candidate queries.

Gamon and Leacock (2010) evaluate different configurations of the web frequency approach by measuring how often corrections are ranked higher than the queries containing article or preposition errors.[28] They consider web search frequencies obtained using the Bing and Google APIs and the (offline) Google 5-gram resource and investigate web query formulation strategies, aiming for a system that can detect errors with a single web query

[28] Even though this task is described as distinguishing between learner errors and corrections, this is not our sentence classification task as their prediction is relative to another sentence: they predict which sentence of a pair of learner error and correction is the correction. This task is relevant for selecting a candidate correction in the confusion set and candidate correction approach.

per candidate correction. Gamon and Leacock (2010) find that Google 5-gram frequencies give better results than web search page frequencies if cases where both error and correction receive zero frequency are excluded from the evaluation. Otherwise, Google web search frequencies achieve the highest accuracy. A final system should therefore combine both. For only two of the six error sub-types (missing, extra and wrong article or preposition), the query formulation with linguistic knowledge (head information) proved useful. The best query formulation depends on the correction edit operation and error type.

Training on Negative Data

Izumi et al. (2004) identify 45 error types and present a prototype that targets 13 of these error types. Results for article errors are reported as this was the best performing error type. Initially, they train on an error corpus with 16,837 sentences of transcribed L2 speech and get disappointing results. They then experiment with adding more training material: *(a)* the corrections of the error corpus, *(b)* additional transcriptions of native speaker interviews similar to the L2 data and, finally, *(c)* artificial error data mirroring error patterns observed in the authentic error data. In each step, Izumi et al. (2004) observe improvements.

While Han et al. (2006) train their maximum entropy classifier on grammatical data, they use negative data to set a parameter of their system: the confidence threshold is set to achieve 90% precision on TOEFL essays.

Gamon (2010) replaces the language model filter of Gamon et al. (2008) with a meta-classifier that is trained on error-annotated learner data to review the primary classifiers' decisions to flag and correct article or preposition errors at potential error sites. This approach is interesting as the meta-classifier only deals with a small number of features and can thus be trained with a limited amount of learner data, while the primary classifiers can be trained on large corpora of grammatical language. The language model score of Gamon et al. (2008) is kept as one of the input features to the meta-classifier. Gamon (2010) uses maximum entropy classifiers for the primary classifiers: a presence/absence classifier and a choice classifier for each error type (article and preposition errors). The meta-classifier is a decision tree. It does not directly pick a candidate correction but

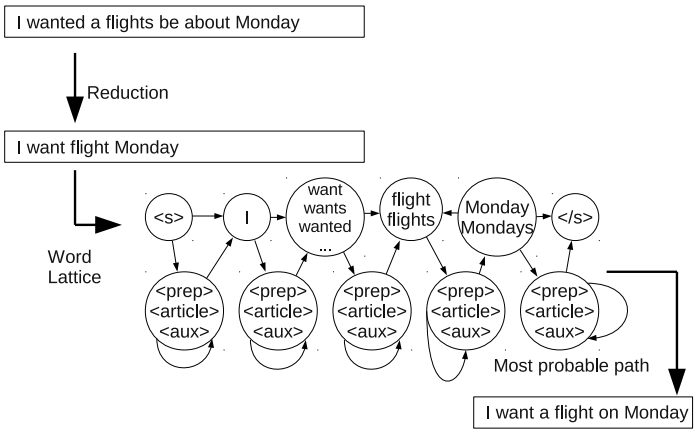


Figure 2.1: Lee and Seneff (2006)'s word lattice approach to candidate correction generation

makes a binary decision about corrections proposed by the primary classifiers. The meta-classifier also makes it easy to trade precision for recall. Results for the error correction task are plotted as precision-recall curves for varying thresholds on the class probability estimated by the classifiers and, in case of the language model, the log-likelihood ratio of input and candidate correction. Surprisingly, the language model outperforms the primary maximum entropy classifiers. The meta-classifier gives the best results. How much learner data is needed for the meta-classifier in order to outperform the (second best performing) language model method depends on the error type: preposition errors require 10 times more data than article errors. Finally, an error analysis of the system is presented with examples of overflagging and wrong correction suggestions.

Candidate Selection with Probabilistic Parsing

Lee and Seneff (2006) address five error types (article, preposition, auxiliaries, verb inflection and noun inflection errors) simultaneously, allow any number of insertions, deletions and substitutions for the first three categories of words, e. g. a confusion of *be a for* with *before*, and rank candidate corrections with a probabilistic parser. The large number of candidate corrections is handled with a word lattice as shown in Figure 2.1. In principle, a probabilistic parser could be developed that can parse word lattices directly (Weber et al., 1997; Hall, 2005). Instead, Lee and Seneff (2006) generate a k-best list of candidate corrections with an n-gram model constrained to the word lattice, parse the k candidates

and select the sentence with the highest parse score.[29]

2.3.3 Preposition Errors

From the point of view of the confusion set approach, the most striking difference between article and preposition error detection is the size of the confusion set: there are over 100 prepositions in English (Leacock et al., 2010). As with articles, local context can be insufficient to predict the preposition found in grammatical test data. If a sentence is judged without discourse context, groups of prepositions like {*above, behind, below, next to, in front of*} and {*with, without*} can hardly be disambiguated. Also, there can be choices that make no difference in meaning: Baldwin et al. (2009) give the example *book on NLP* vs. *book about NLP*. Therefore, preposition error detection and correction demands that not just the most likely preposition is predicted but that the choice of the writer or L2 learner is considered as well, e. g. through application of the noisy channel model.

The field is as diverse as research on article errors:[30] Chodorow et al. (2007) train maximum entropy classifiers with contextual features including word, lemma, part-of-speech, phrase and head of phrase information extracted with a heuristic chunker and a part-of-speech tagger. Like Han et al. (2004), they observe that many classification errors occur when the probabilities of the first and second choice of the maximum entropy model are close and they improve performance in the preposition correction task by adding a confidence threshold. The classifier is used to check the preposition selection only, not for presence/absence decisions. Two types of extra preposition errors are addressed with hand-written rules. De Felice and Pulman (2007) use a voted perceptron machine learning algorithm with features extracted from the output of a parser. Instead of following the common candidate correction approach, they train their classifier to predict whether a preposition is correct in its context.

Hermet et al. (2008) use Yahoo web search statistics for preposition error detection and correction in French. Distinctly, they do not use a single confusion set of all preposi-

[29] No details are given whether this is the probability of the best parse, the probability of the sentence according to the parsing model or some other score.

[30] We omit De Felice and Pulman (2008), Gamon et al. (2008) and Gamon and Leacock (2010) who also cover article errors and have been discussed in Section 2.3.2 above.

tions but select candidate corrections from "minimal" lists of those prepositions that are usually confused with the input preposition. Tetreault and Chodorow (2008a) extend the work of Chodorow et al. (2007) with Google n-gram features, lexical resources and feature combination. Post-processing filters inhibit correction suggestions like *with/without* confusions and apply a threshold on the probability estimates of the classifier to reduce overflagging. Furthermore, they argue that multiple human raters should be used to annotate test data as preposition choice is unreliable. Tetreault and Chodorow (2008b) discuss their observations on human judgement experiments in more detail.

Bergsma et al. (2009) use Google n-grams with a sliding window over the potential error site. Only preposition substitution errors are considered (no missing or extra preposition errors). Two scoring models for candidate corrections are tested: a linear support vector machine and the product of all n-gram frequencies (for each candidate correction). Hermet and Désilets (2009) use L2-L1-L2 round-trip machine translation to address preposition errors — see Section 2.1.3. Lee (2009) investigates the usefulness of linguistic features for preposition selection with a memory-based learning algorithm. He finds that prepositional phrase attachment information is very useful in the task.

Dickinson et al. (2010) target Korean particle errors, which have functions similar to English prepositions. They acquire training data from the web with the BootCaT tool (see Baroni and Bernardini (2004) and Appendix B.8) using seed words from the vocabulary of the L2 learners targeted and develop a feature set to train a memory-based classifier to predict correct particle use. Han et al. (2010) train a maximum entropy classifier to predict preposition corrections on an error-annotated L2 learner corpus. The input preposition is given as an input feature so that the classifier can learn what confusions are typical and when to trust the learner input. They find that this classifier outperforms a similar classifier trained on positive data alone, even if the training data is five times bigger than the learner data. Tetreault et al. (2010a) add 14 types of features extracted from parse trees to the system of Tetreault and Chodorow (2008a). The parse trees are automatically obtained using the Stanford parser. While the system is trained on positive data for preposition prediction, the evaluation includes both the preposition prediction task (on positive test data) and the error detection and correction task (on L2 learner data). The

improvements are in the order of four percentage points accuracy for the prediction task but, possibly due to the small size of the L2 test data, not statistically significant in the second task. Elghafari et al. (2010) experiment with web search frequency information in the preposition selection task. They try different search window sizes (n-gram order) and back-off strategies and report competative results — see also Section 2.1.5.

Additional references for preposition error detection research can be found in the overview table of Leacock et al. (2010).

2.4 Parsing Ill-Formed Input

Douglas and Dale (1992) point out that there are basically two different aims that systems can have when processing erroneous or out-of-coverage sentences: one is to extract the intended meaning. In many applications like dialogue systems, question answering and information retrieval, rejecting sentences that cannot be fully processed would degrade usability or performance considerably. Such systems should exploit whatever clues are available in order to gain some information from ill-formed or out-of-coverage input. In the case of parsing, a syntactic structure should be produced that resembles the structure of the intended sentence. The second possible aim is to provide some error feedback, for example a correction. To do this, the system has to become aware of the fact that there is an error to be reported, locate and identify the error and generate one or more corrections.

Grammar-Independent Approaches One approach to dealing with an ill-formed sentence is to search for a minimal modification of the sentence that makes it well-formed. Since only the parsing algorithm is extended, any grammar can be used with this approach. Early implementations only iteratively considered modifications at the right end of the longest portion parsed so far and were prone to get stuck in local optima. Mellish (1989) adds rules to a chart parser to do a global search for the "most plausible explanation." Kato (1994) extends the work using A-star search.[31]

[31] Another grammar-independent solution is to simply collect all fragments that can be parsed.

Inherently Robust Parsing Some parsers happen to be quite good at parsing ill-formed input although they have not been designed to do so. This is true for many large treebank-based probabilistic parsers. Their grammars are too general to reject ungrammatical input. Foster (2004) evaluates Charniak's probabilistic parser to illustrate a new evaluation method and reports that the parser does well on erroneous input. The parser's probabilistic model seems to enable it to find plausible analyses that are close to gold-standard analyses of corrected versions of the sentences. Foster (2005) extends this experiment to Collins' parser and finds similar results.

Mal-rules and Constraint Relaxation The third approach focuses on extending a (hand-crafted) precision grammar. If parsing fails, the grammar is expanded to cover ungrammatical constructions. This is often implemented in terms of a two-stage parsing model. For these systems, the first-stage grammar has to be carefully designed to distinguish grammatical from ungrammatical sentences and the diagnosis of grammatical errors is usually tightly integrated into the parsing process. The second stage of the parsing process extends the coverage of the grammar to ungrammatical sentences. There are two ways of expanding a grammar in the literature: specific rules, called mal-rules, that facilitate the parsing of a specific grammar error can be added to the grammar or existing rules can be rendered less specific by relaxing the context in which they can be applied.

For example, Bender et al. (2004) add hand-crafted mal-rules to a generative grammar, the English Resource Grammar (ERG), in order to obtain a parse tree that contains a description of the error and then use a process they call "aligned generation" to propose a correction. Earlier work uses a single-stage parsing model. Schneider and McCoy (1998) show how to design mal-rules that do not flag grammatical input and address efficiency issues.

The second method relaxes feature constraints of a grammar (Reuer, 2003). This works with grammars that use features instead of a rich set of categories to describe what phrases can be combined, for example Lexical Functional Grammar (LFG, see Section 5.1 of Chapter 5).

2.5 Summary

This chapter discussed various topics relevant to grammar checking, pointed to application areas and described important concepts. Errors cannot simply be divided into spelling and grammar errors as spelling errors can result in real words that often render a sentence ungrammatical and nearly always do not fit in semantically. The candidate correction approach to error detection and correction is widely used and recent work focuses on article and preposition errors. Probabilistic parsing models have been used to rank candidate corrections and the parse results (parse trees) can be used to extract additional features in automatic error classification.

Chapter 3

Datasets and Metrics

Statistical methods including those we will employ in the following chapters rely heavily on data. Model parameters are estimated based on the frequency of events in training data and models are evaluated on test data. Sections 3.1 to 3.3 discuss the data sets we use, starting with the British National Corpus, a large corpus of (for the most part) grammatical language, then describing various corpora with authentic grammatical errors (Section 3.2), and finishing with a large artificial parallel error corpus. All corpora are automatically annotated with part-of-speech tags and phrase structure trees. This annotation provides the basis for the events that our statistical methods count, e. g. part-of-speech n-grams in one of the methods of Chapter 5 and parse probabilities in Chapter 4. Since we experiment with various grammars, we do not describe the particulars of annotation in this chapter but postpone the description to the discussion of the respective experiments (Chapters 4 and 5).

The second part of this chapter deals with the evaluation of methods for classifying sentences as either grammatical or ungrammatical. Given the wide range of error densities to be expected in different applications and even within applications (consider, for example, language learners at different proficieny levels using the same grammar checker), evaluation measures that depend on the error density of the test data are not suitable here. Section 3.4 reviews the basic measures derived from the confusion matrix of binary classifiers and motivates our choice of reporting two numbers, namely accuracy on grammatical and ungrammatical test data. As more than one evaluation score is used, there is no clear

ordering of classifiers and we often cannot say which classifier in a set of classifiers is "best". However, we can in fact do better than just excluding classifiers that have lower accuracy on both scales (accuracy on grammatical and ungrammatical data): Section 3.5 provides a tool for eliminating "inferior" classifiers from a set of classifiers. This tool will not only be used in summative evaluation but also during training, i. e. to select the parameters of a method. In Sections 3.6 and 3.7, we discuss the implications of our two-dimensional evaluation measure on pooling results of cross-validation runs and on statistical significance testing. Finally, Section 3.8 suggests changes to the data sets for future experiments and points to areas of future research on evaluation with two scales.

3.1 The British National Corpus

The British National Corpus (BNC) is a one hundred million word corpus of written and spoken English from a variety of sources (Burnard, 2000). The BNC is a balanced corpus and is designed to be a representative sample of British English from the late twentieth century. Written text comprises 90% of the BNC: 75% of this is non-fiction. The written text is taken from newspapers, published and unpublished letters, school and university essays, academic journals and novels. The spoken component of the BNC consists of transcriptions of spontaneous unscripted dialogue with participants of various ages, regions and social classes, and transcriptions of more formal speech, e. g. business meetings, speeches or radio shows. The BNC is automatically tagged for part-of-speech using the CLAWS4 tagger (Garside et al., 1987). A two million word subset has been manually tagged using a richer tag set. The corpus is encoded in SGML, with metadata expressed at the document (e. g. document source, genre, id) and sentence (e. g. sentence id) level.

3.1.1 Preprocessing

> *"Cleaning is a low-level, unglamorous task, yet crucial: The better it is done, the better the outcomes. All further layers of linguistic processing depend on the cleanliness of the data."*
>
> (Kilgarriff, 2007, p.149)

This section documents the preprocessing carried out for the BNC and the process of parsing it with the first-stage parser of the Charniak and Johnson (2005) reranking parser.[1] Hereafter, we will refer to this parser as Charniak's parser.[2] The following description expands and adapts details provided by Wagner et al. (2007b) who use the same BNC data to show that a treebank-based Lexical Functional Grammar parsing architecture previously developed at Dublin City University (Cahill et al., 2002, 2004; Burke, 2006) adapts well to new domains.

The British National Corpus (BNC) is in a very different format than that expected by treebank-trained parsers. Some basic preprocessing is necessary in order to parse the BNC with these parsers. Adaptations carried out to more closely match the Penn II Treebank (PTB)[3] encoding conventions can be expected to improve the parse results because the number of unknown tokens for the parser is reduced. This includes SGML entities, soft hyphens, quotes, currency symbols and spelling differences between American and British English.

Extraction of Sentences

In the original BNC, sentences are marked with an `<s>` tag. We extract a total of 6,228,111 sentences — see Appendix Section A.1 for details.

While processing the BNC SGML files, various tags present in the BNC were exploited to annotate sentences with additional information, for example whether they belong to headers, list items, spoken utterances, poems, etc. A BNC tag that needs special attention is the `<gap>` tag. It marks omissions due to anonymisation and replaces various non-textual material including formulae and figures. Gaps in sentences are likely to break grammaticality and therefore result in sentences that are more difficult for the parser to analyse correctly. To facilitate parsing, we automatically re-inserted text for gaps according to Table 3.1. The gap substitutions are recorded and are recoverable. In total, 51,827 gap substitutions were performed in 38,452 sentences, i. e. 0.617% of all extracted BNC sentences.

[1] We use the June 2006 version.

[2] See also Section 4.1.1 and in particular Footnote 5 of Chapter 4.

[3] We use the Wall Street Journal part of the PTB Marcus et al. (1994). See also Section 2.1.2 of Chapter 2.

Gap Description	Substitution String
last or full name	Jon1234es
list of names	Jon1234es , Smi1234th and Mur1234phy
date	29/12/1970
list of dates	29/12/1970 , 30/12/1970 and 31/12/1970
list of countries	Germ1234any , Ire1234land and Spa1234in
address	11234 Sun1234set Avenue
name and address	Mur1234phy , 11234 Sun1234set Avenue
telephone number	0123/4561234
number	1231234
formula	1231234

Table 3.1: Substitutions of anonymisation gaps: 1234 is replaced by a random number drawn from an exponential distribution. The inserted numbers prevent an abnormal frequency distribution while still being readily identifiable as artificial, and their position in the middle of each token is intended to reduce effects on named entity recognition.

UTF-8 Encoding of SGML Entities

Character encodings do not receive much attention in mainstream NLP. An exception is Buchholz and Green (2006) who discuss character encoding and SGML problems with a number of treebanks. The BNC uses a large number of SGML entities to represent special characters, symbols, fractions, typographic quotes etc. A PTB-trained parser will treat such entities as unknown tokens and often interpret them as adjectives, nouns or foreign material. As a first step to normalising the encoding, we map the SGML entities to the UTF-8 character encoding, which is a superset of the ASCII encoding used in the PTB. The mapping was manually created based on the description in the file `bncents.dtd` included in the BNC distribution and Unicode character code charts[4] and other web resources.[5] The conversion immediately resolves certain ASCII characters that are represented by an SGML entity in the BNC, for example, the dollar sign. For special UTF-8 characters, however, UTF-8 serves more as an intermediate format that helps us to keep as much information as possible and at the same time to visualise the intended symbols in text editors. After conversion, 1,255,316 (20.156%) BNC sentences contain non-ASCII characters. Further quote and currency conversions (see below) reduce this number to 45,828 sentences (0.736%).

[4] `http://www.unicode.org/charts/` accessed during 2005 and 2006

[5] We thank Grzegorz Chrupała for help with this work.

Disambiguation of Soft Hyphens

Inspection of the frequency table of special characters reveals that soft hyphens occur in the BNC: in fact, 4,190 sentences (0.067%, 4,235 tokens, 3,878 types) contain soft hyphens. According to the ASCII standard, soft hyphens are hyphens inserted by pagination processes at the end of a line. However, in practice, they are often used to mark permissible hyphenation points, obligatory hyphens and as bullet points in lists.[6] As many NLP components are not equipped to handle soft hyphens and consequently will treat tokens containing soft hyphens as unseen tokens, we replace them with the following simple strategy: we create three candidate substitutions (deletion, space, normal hyphen) and vote based on the frequency of the respective tokens and bigrams in the BNC.[7] Manual evaluation of this strategy on 100 randomly extracted instances showed 6 clear errors and 12 unclear cases.

Normalisation of Quotes

The PTB uses (and therefore PTB-trained parsers expect) sequences of two single left or right quotes to represent left and right quotes. In most cases, distinct quotes in the BNC can be easily converted to PTB-style. However, some sections of the BNC use neutral quotes. Very rarely, single quotes are used as well. In order to achieve optimal results, a conversion is necessary. We disambiguate neutral quotes by replacing them with alternating left and right quotes. Existing unambiguous quotes are respected, so that a neutral quote after a left quote becomes a right quote. Single quotes are not changed as there would be a high risk of accidently damaging apostrophes. We test the effect of this and three simpler quote replacement strategies on PTB data: we replace all quotes in WSJ section 23 of the PTB with neutral quotes, disambiguate these quotes, parse the new text with Charniak's parser and measure the bracketing f-score. Table 3.2 confirms that the correct usage of left and right quotes affects parse results. While our replacement strategy gives an F-score close to the one obtained with the original quotes, simpler strategies deteriorate parse results considerably. The total number of BNC sentences containing

[6] The correct usage is controversial – compare for instance the Wikipedia article on hyphens and the detailed discussion on the web-page `http://www.cs.tut.fi/~jkorpela/shy.html`

[7] Appendix Table A.2 shows some examples.

F-score	Quote replacement strategy
89.73%	original quotes from treebank (oracle)
89.65%	alternating left and right quotes (per sentence)
89.08%	convert all quotes to right quotes
88.72%	randomly chose left or right quote
88.41%	convert all quotes to left quotes

Table 3.2: Bracketing F-scores with Charniak's parser and different quote replacement strategies evaluated against WSJ section 23 (POS tags adjusted accordingly)

ambiguous neutral double quotes is 68,020 (1.092%).

Currency and Other Normalisations

The PTB uses individual tokens for currency and number, for example `US$` 2,000, while the BNC amalgamates them into a single token. Furthermore, the pound sign is the dominant currency symbol in the BNC while the PTB does not provide much training data for it.[8] A substitution with the dollar sign provides more reliable statistics for the parser. Therefore, we map pound, yen and euro symbols to the dollar sign and, in a second step, insert a token boundary after each dollar sign to separate a possibly attached amount. In contrast to Wagner et al. (2007b), we do not restore the original symbols after parsing as our research is not concerned with the detection of incorrect usage of these symbols. A total of 69,459 BNC sentences (1.115%) contain currency symbols.

Additionally, dashes are replaced by PTB-style sequences of minus signs (short "en" dashes with one, long "em" dashes with two). Horizontal ellipsis is replaced by three full stops. Many fractions are represented by single entities in the BNC, and consequently mapped to single characters in Unicode (if possible) in our first preprocessing step, e.g. frac23 and U+2154 for two-thirds. The common fractions 1/4, 1/2, and 3/4 are re-written with normal numbers and a forward slash as they appear in the PTB. Prime and double prime are encoded as single and double (neutral) quotes. The multiplication sign is replaced by 'x'. The bullet and micro signs that are quite frequent in the BNC are not replaced because we could not find suitable examples in the PTB.

[8] The pound sign is represented by the # sign in the PTB, see `http://www.ldc.upenn.edu/Catalog/docs/treebank2/cl93.html`, and appears 142 times in WSJ sections 2 to 21 of the PTB. The dollar sign appears 7,374 times in the same WSJ text.

Shuffling the Sentences

There are two reasons why we randomise the order of sentences: *(a)* spreading the effect of processing errors, e. g. the parser failing or misbehaving for a batch of sentences, and *(b)* making training and test data more similar. Matching training and test data is important for machine learning methods to achieve best results: we split data into 90% training and 10% test data and if the data was sorted by text type and domain, the mismatch could be big, e. g. news articles vs. transcribed spoken text.[9] Instead of using (pseudo-) random numbers, we sort all sentences by a sort key derived from the BNC sentence ID with a cryptographic hash function.[10] This yields a reproducible, pseudo-random permutation that can be reproduced more easily than the behaviour of some programming language specific pseudo random number generator.

Translation to American English

The varcon package (`http://wordlist.sf.net`) is used to translate the BNC to American English. This is expected to improve parse results as the parsers are trained on American English. Reviewing the source code and vocabulary file, the varcon translation process is only a matter of different spelling and word substitutions. Word order and tokenisation are not changed. The total number of BNC sentences that are changed by varcon is 333,745 (5.359%).

3.2 Error Corpora and Learner Corpora

A corpus of authentic ungrammatical language, an error corpus, is essential to realistically test error detection systems and will also be most useful during their development. Unfortunately, access to error corpora is a challenge for researchers in the field. There are proprietary error corpora, e. g.

- the Test of English as a Foreign Language (TOEFL) language test data used in experiments by Han et al. (2006); Tetreault and Chodorow (2009); Tetreault et al.

[9] How well a system adjusts to a new domain is, of course, also an important question — see for example future work Section 4.7.6 of Chapter 4.

[10] We use the MIME Base64 representation of the MD5 hash of the three character BNC filename, followed by a forward slash and the value of the n attribute of the BNC <s> tag.

(2010a),

- the ESOL language test data (part of the Cambridge Learner Corpus, CLC) used by Gamon and Leacock (2010); Gamon (2010, 2011),
- the Chinese Learner English Corpus (CLEC) used by (Gamon et al., 2008),
- the Standard Speaking Test (SST) speech corpus of Japanese learner English presented by Izumi et al. (2004), and
- the International Corpus of Learner English (ICLE) (Granger, 1993)

However, such corpora either have only recently become available, i. e. after we started our experiments, and/or are prohibitively expensive.[11] Researchers have been forced to become highly creative to circumvent the data problem, e. g. Okanohara and Tsujii (2007) sample a probabilistic n-gram model in order to generate pseudo-negative examples. We also use artificial error data during development. However, we base the creation of the artificial data on an analysis of an authentic error corpus (see Section 3.3 below) and we test and compare final methods on authentic data in Chapter 7.

Generally, an error corpus is a corpus that was built with the intention to provide examples of errors. This presupposes a norm that can be violated and contrasts with corpora that simply aim to describe language, e. g. a part of the BNC shows idiosyncrasies of Yorkshire English (Mitton et al., 2007). Any type of errors such as collocation errors or style errors can be included in an error corpus. For our purposes, we focus on grammatical errors (including real-word spelling errors) and crucially require annotation to mark sentences that contain at least one error.

A learner corpus is a collection of written text or speech produced by non-native learners of a language, usually in a learning context (Izumi et al., 2004; Granger, 1993).[12] A learner corpus is an error corpus, though it can also be seen as a corpus describing the interlanguage of language learners (Selinker, 1972). Learner corpora may also differ from general error corpora in regard to the annotators and the aim of annotation: linguists or

[11] See also the list of learner corpora compiled by Leacock et al. (2010) and the references in Section 2.1.5 of Chapter 2.

[12] For example, contrast this with a corpus of customer support queries written in English but originating from a non-English speaking country.

error detection researchers who scan text for errors (Becker et al., 1999; Foster, 2005) vs. teachers who mark a text to help their students improve their language skills or to grade them. Learner corpora are particularly useful in the study of second language acquisition since they provide insight into the difficulties faced by native speakers of a particular language when attempting to learn the corpus language (L1 influence). The more general form of error corpus is unconcerned with whether an error reflects linguistic competence or performance, it merely records that an error has occurred. In the following, we describe the authentic error corpora we use including a number of learner copora.

A new type of error corpus is used by Tetreault and Chodorow (2009). They derive information about ungrammatical patterns from web search statistics contrasting frequency information from different geographical regions. The underlying assumption is that a relevant fraction of the English text published on the web from a geographical region is written by native speakers of one of the languages spoken in that region, e. g. French web pages containing English text are more likely to be written by French native speakers than general web pages.

3.2.1 Foster's Parallel Error Corpus

Foster's parallel error corpus (Foster and Vogel, 2004a,b; Foster, 2005) is a collection of 923 ungrammatical English sentences with aligned corrections. The sentences are taken from newspapers, academic papers, e-mails and web forum posts written by native and non-native speakers. The errors were corrected in context at the time they were encountered.

Foster's error corpus is fundamentally different from a learner corpus because, although it contains competence errors which occur due to a lack of knowledge of a particular structure, many of the errors are in fact performance slips. Some error types are particularly associated with performance slips, e. g. real-word spelling errors. Nevertheless, the effects of language transfer from the writer's mother tongue are clear in some examples, e. g. the missing word error in

(3.1) *I **am psychologist**.*

Since this corpus is the basis for the procedure for creating an artificial error corpus in Section 3.3, we look at its error types in more detail. Foster (2005)'s analysis of the

Error Type	**Examples**
Missing Word	*I'm not sure what I'm* ***up tomorrow****.* *I* ***am psychologist****.* *She* ***didn't to*** *face him.*
Extra Word	*Why* ***is do*** *they appear on this particular section?* *Is our youth really* ***in in*** *such a state of disrepair?* *Do you ever go and visit* ***the any*** *of them?*
Real Word	*Yoga brings peace and vitality to* ***you*** *life.* *We can order* ***then*** *directly from the web.* *I love* ***then*** *both.*
Agreement	*I* ***awaits*** *your response.* *The first of* ***these scientist*** *begin in January.* *The* ***contrasts was*** *startling.*
Verb Form	*Brent would often* ***became*** *stunned by resentment.* *I* ***having*** *mostly been moving flat.* *Want to* ***saving*** *money?*

Table 3.3: Sentences from Foster error corpus (Foster, 2005)

word-level edit operations used to correct grammatical errors in the 20,000 word corpus shows the following frequency ordering:

1. substitution (48% of all errors),
2. insertion (24%),
3. deletion (17%) and
4. combinations of the above three edit operations (11%).

Among the grammatical errors which could be corrected by substituting one word for another (48% of total), the most common errors are real-word spelling errors (20%), agreement errors (9%) and errors in verb form (5%). The five error classes shown in Table 3.3 account for 75% of all errors. Foster (2007b) points to previous studies (Stemberger, 1982; Nicholls, 1999; Hashemi, 2007) that confirm the frequency ordering of edit operations and support the chosen error taxonomy, concluding that the error corpus is sufficiently broad to cover important error types that are likely to occur in applications.

Foster's error corpus also contains instances of *covert* errors (James, 1998) or errors which result in structurally well-formed sentences with interpretations different to the intended ones. An example is the sentence

(3.2) *We can order **then** directly from the web.*

Because the errors in the corpus were observed in their discourse context, it was clear that a real-word spelling error had been produced and that the intended sentence was, in fact,

(3.3) *We can order **them** directly from the web.*

Obviously, these kinds of sentences will pose a particular problem for our classifiers which process sentences in isolation. A similar point is made by Andersen (2007).[13]

The following types of sentences were *not* included in the error corpus, but were recorded:

- Unintelligible cases: if the sentence cannot be understood then it cannot be corrected. Foster (2005) excludes such sentences for the task of evaluating robust parsing as no human gold parse can be assigned.
- Ambiguous cases: if the sentence had more than one interpretation, a correction could not be confidently supplied. These were rare because the discourse context was always available and usually contained enough information to disambiguate.
- Doubtful cases: sentences which sounded odd or infelicitous to a native speaker but were not technically incorrect.

3.2.2 Learner Corpora

An important area for applications of grammatical error detection methods is second language learning. Therefore, we test our methods on learner data in addition to the small held-out section of 44 sentences of Foster's parallel error corpus. We aggregate a number of learner corpora as each corpus is fairly small:

1. Essays produced by advanced learners of English (608 sentences) (Granger, 1993; Horváth, 1999; PELCRA, 2004)
2. Transcribed spoken language produced by learners of English of all levels (4602 sentences)[14]

[13] See also Section 3.3.3 on covert errors.

[14] We are very grateful to James Hunter from Gonzaga University for providing us with this data.

3. Sentences containing mass noun errors produced by Chinese learners of English and a corrected version of these sentences (123 × 2 sentences) (Brockett et al., 2006)[15]

Advanced Learner Essays

These essays were produced by advanced learners of English with Hungarian, Polish, Bulgarian or Czech as their mother tongue, including 289 sentences from the ICLE corpus. One annotator read through these essays and attempted to judge each sentence as either grammatical or ungrammatical. The grammaticality judgement task is not straightforward for native speakers, with high levels of inter-annotator disagreement (Snow and Meijer, 1976; James, 1998; Tetreault and Chodorow, 2008b; Rozovskaya and Roth, 2010a). Because of this and because only one annotator was available, we excluded from our test set those sentences for which the annotator was not confident in her judgement. These "questionable" sentences are often syntactically well-formed but contain words which would not be used by a native speaker in the same context, and hence would be likely to be corrected by a language teacher. Some examples are:

(3.4) *I became **even devoted** to the British.*

(3.5) *The **very first look** of the streets shows something else.*

(3.6) *Today the role of the family **extremely increases**.*

Spoken Language Corpus

This corpus contains transcribed spoken sentences which were produced by learners of English of all levels (beginner, low-intermediate, intermediate, advanced). The speakers' L1s come from the following set: Amharic, Arabic, Cantonese, French, Icelandic, Indonesian, Italian, Japanese, Korean, Mandarin, Portuguese, Russian, Spanish, Thai, Ukrainian and Vietnamese. The sentences were produced in a classroom setting and transcribed by the teacher. The transcriptions were verified by the students.

Wagner et al. (2009) examined a 499-sentence subset of this corpus, correcting the sentences to produce grammatical data. 56 of these 499 sentences were found to be gram-

[15] Available to download from `http://research.microsoft.com/research/downloads`

matically well-formed (either covert errors or questionable). Of the remaining 443 sentences which were corrected, 253 contained more than one grammatical error. The 190 sentences containing just one error were classified according to the manner in which they were corrected (insert/delete/substitute):

- 23 sentences contain an extra word (the most common of which is a preposition);
- 39 sentences contain a missing word error, with almost half of these being missing determiners;
- 66 sentences were corrected by substituting one word for another, with agreement errors as the most common subtype.

The remaining 62 sentences contain errors which are corrected by applying more than one correction, e. g. the sentence

(3.7) *She is one of* ***reason*** *I became interested in English*,

which was corrected by changing the number of the noun *reason* and inserting the determiner *the* before the noun.

Mass Noun Error Corpus

The 123 sentences in this corpus were encountered in online documents produced by Chinese learners of English. Each sentence contains an error involving a mass noun, e. g.

(3.8) *I learnt* ***a few*** *knowledge about the Internet.*

Brockett et al. (2006) corrected the 123 sentences, resulting in a parallel corpus containing 246 sentences.

3.3 Artificial Error Creation

The compilation of an authentic error corpus is a time-consuming process. It is not enough to merely collect a body of sentences, the grammaticality of each sentence must also be judged in order to determine whether an error has occurred. If an error has occurred, it

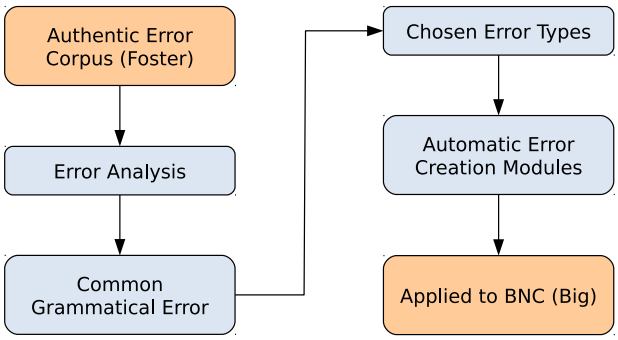


Figure 3.1: Steps of the design of the artificial error corpus

then must be classified according to some error taxonomy. However, in order to train the parameters of statistical methods, we need a large error corpus with tens of thousands of ungrammatical sentences as training data and we do not have a suitably large corpus of naturally occurring erroneous data at our disposal.

We artificially generate a large error corpus by automatically distorting BNC sentences. In order to ensure that this distortion process is realistic, it has been designed to replicate the errors found in the corpus of ungrammatical sentences (Foster, 2005) described in Section 3.2.1. While this means that the errors are not new, they are provided in new contexts. Figure 3.1 illustrates the steps of the design. A first version of the error creation procedure is described by Foster (2007a), Foster (2007b) and Wagner et al. (2007a). The following description is based on Wagner et al. (2009). Foster and Andersen (2009) add more flexibility to the procedure and evaluate the usefulness of the artificial data.

3.3.1 Related Work

The use of artificial error data is not new: Bigert (2004), expanded by Bigert et al. (2005), for example, automatically introduce spelling errors into texts, and use these in spelling error detection and parser robustness evaluation. Okanohara and Tsujii (2007) generate ill-formed sentences (they use the term "pseudo-negative examples") using an n-gram language model and they then train a discriminative language model to tell the difference between these pseudo-negative examples and well-formed sentences. Post (2011) apply the same technique to training classifiers with features extracted from parse results obtained with a tree substitution grammar.

Smith and Eisner (2005a,b) automatically generate ill-formed sentences by transposing or removing words within well-formed sentences. These ill-formed sentences are employed in a unsupervised learning technique called contrastive estimation which is used for part-of-speech tagging and dependency grammar induction.

Lee et al. (2007) use machine translation output as ungrammatical training data for a support vector machine (SVM) classifier, i. e. the source material is in a different language and the imperfections of the translation process are exploited to obtain ungrammatical text. Grammatical training data is provided in the form of (human) reference translations.

Rozovskaya and Roth (2010c) focus on replicating the frequency distribution of article errors for three different L1s in order to generate training data for classifiers that detect and correct such errors.

Dickinson (2010) targets Russian, a morphologically rich language, as the L2. Consequently, he focuses on automatically generating morphological errors. Morphemes are combined randomly subject to constraints obtained from an error analysis.

3.3.2 Procedure and Probabilistic Error Modelling

The automatic error creation procedure accepts as input a part-of-speech-tagged sentence[16] and outputs a deviant version of the input sentence. The automatic error creation procedure is informed by the error analysis of Foster's parallel error corpus, see Section 3.2.1 above for a summary. For each input sentence, the error creation procedure attempts to produce five kinds of ungrammatical sentence, each exhibiting a different grammatical error. The five error types are the five most frequent error types in Foster's error corpus, i. e. errors involving a missing word, errors involving an extra word, verb form errors, agreement errors and real-word spelling errors.

For each error type, we try to mimic properties of the errors observed in Foster's error corpus. Properties include the context in which the errors occur and the choice of new words inserted. Of course, the error model is constrained by what can be implemented easily and we also try to avoid covert errors that we cannot expect to be detectable if the context of the sentence is not given.

[16] We use the pre-terminals of the reranked parse results of Wagner et al. (2007b) covering 6,218,273 (99.842%) of all BNC sentences.

Missing Word Errors

The automatic error creation procedure creates missing word errors by deleting a word from a sentence. The likelihood of a word being deleted will be determined by its part-of-speech tag. In Foster's error corpus, 98% of the missing word errors involve the omission of the following parts of speech (ordered by decreasing frequency):

- determiner (28%)
- verb (23%)
- preposition (21%)
- pronoun (10%)
- noun (7%)
- infinitival marker "to" (7%)
- conjunction (2%)

Adjectives and adverbs are not deleted by the procedure because their omission is likely to result in a well-formed sentence.[17] A sentence with a missing word error will not be produced if the sentence contains none of the above part-of-speech tags or if the sentence contains just one word.

Extra Word Errors

Approximately two thirds of the extra word errors are created by duplicating a randomly selected token in the input sentence or by inserting a word directly after another word with the same part-of-speech tag, e. g.

(3.9) *A hand touched **his his** shoulder.*

(3.10) *Why **is do** they appear in this section?*

Adjectives are the only exception because their duplication will tend not to result in an ungrammatical structure (*the long long road*). The remaining extra word errors are created by inserting a random token at a random point in the input sentence.

(3.11) *Resist **than** him , she told herself.*

[17] One exception is a noun phrase containing a list of coordinated adjectives, e. g. *the green, white and [orange] tricolour.*

is ↔ if	is ↔ in	is ↔ it	is ↔ as	is ↔ us
is ↔ its	is ↔ his	if ↔ in	if ↔ it	if ↔ of
in ↔ it	in ↔ an	in ↔ on	it ↔ its	it ↔ at

Table 3.4: Some English real-word spelling errors

Real-word Spelling Errors

A list of real-word spelling errors involving commonly occurring function words (prepositions, auxiliary verbs and pronouns) is used to insert errors of this type. Table 3.4 shows a sample of 15 real-word spelling errors involving function words related to the words *is*, *it*, *in* and *if*.

Agreement and Verb Form Errors

Agreement and verb form errors are created by searching the input sentence for likely candidates, randomly selecting one of them and then replacing it with its opposite number form or a different verb form. For a morphologically sparse language such as English, the error creation procedure is at its least productive for agreement errors. It is not possible, for example, to change the *number* of the noun, verb or determiner in the sentence

(3.12) *The man walked home*,

such that a syntactically ill-formed structure is created.

3.3.3 The Problem of Covert Errors

The tendency of the error creation procedure to produce covert errors was estimated by carrying out the following small experiment: sentences were randomly extracted from the BNC and the error creation procedure applied to them. 500 of the resulting sentences (the first 100 for each error type) were then manually inspected to see if the sentence structures were grammatical (Foster, 2007b). The percentage of superficially well-formed structures that are inadvertently produced for each error type and an example of each one are shown below (average 8%):

(3.13) Agreement Errors, 7%

Mary's staff ***include*** *Jones,Smith and Murphy* ↔

Mary's staff ***includes*** *Jones,Smith and Murphy*

(3.14)Real-Word Spelling Errors, 10%

And ***then****?* ↔ *And* ***them****?*

(3.15)Missing Word Errors, 13%

She steered ***Melissa*** *round a corner* ↔ *She steered round a corner*

(3.16)Extra Word Errors, 5%

She made no effort to check her tears ↔ *She made no effort to check* ***in*** *her tears*

(3.17)Verb Form Errors, 6%

There was no ***turning*** *back* ↔ *There was no* ***turned*** *back*

The occurrence of these grammatical sentences in the artificial error corpus can be reduced by fine-tuning the error creation procedure or by using a finely grained part-of-speech tag set to tag the input corpus. For example, if verb subcategorization frames were available to the error creation procedure, it would know that the verb *steer* can be used intransitively (Example 3.15) or that the verb *check* can be used with the particle *in* (Example 3.16). This idea is taken up again in Section 3.8.3.

It is questionable, though, that covert errors should be completely eliminated, since they are are a natural linguistic phenomenon which occur in error corpora containing real errors. Without context, whether an error is recognised as a covert error just depends on the effort we make to find a suitable context in which the sentence is grammatical.

3.3.4 Constructing a Parallel Error Corpus

From the BNC preprocessed as in Section 3.1.1, the artificial error creation procedure tries to insert each of the five error types into each input sentence as described in Section 3.3.2 above. It produces

- 3,074,956 sentences containing an agreement error,
- 3,466,581 sentences containing a verb form error,

- 5,357,314 sentences containing a real-word spelling error,
- 5,794,153 sentences containing a missing word error and
- 6,218,273 sentences (100% of the input) containing an extra word error.

Instead of merging these sets, we build a parallel error corpus that aligns each grammatical sentence with exactly one ungrammatical sentence.[18] We hope that the parallelism will help statistical methods to learn meaningful discriminators rather than picking up random content differences between grammatical and ungrammatical training sentences.

Error types are assigned in a round robin fashion and a grammatical sentence is skipped if the error creation procedure does not provide an ungrammatical sentence with the desired error type. If we tried alternative error types the procedure probably would yield a higher number of ungrammatical sentences but this would come at the risk of side effects: e. g. sentences with properties that prevent them from receiving one particular error type, say agreement errors, would be over-represented in other error types. Table 3.5 illustrates the alignment procedure. The resulting parallel error corpus contains 4,409,265 grammatical sentences and 881,853 ungrammatical sentences for each error type ($5 \times 881{,}853 = 4{,}409{,}265$).

3.4 Evaluation Metric

Previous work on error detection is dominated by the use of precision and recall for evaluation. However, we will show below that precision depends on the error density of the test data and it is difficult to estimate results for other error densities if results are reported only for a particular error density. Therefore, we report accuracy values separately for ungrammatical and grammatical test data. The first value says how many of the ungrammatical sentences are detected. The second value says how many grammatical sentences pass the classifier without being erroneously flagged as ungrammatical. The confusion matrix of a classifier can easily be derived from these two values and any given

[18] Furthermore, merging these sets would over-represent missing word and extra word errors. Note that the output of the error creation procedure reflects the feasibility of inserting each of the five error types, not the distribution of the five error types in Foster's error corpus.

Seq.	BNC ID	Gr.	MW	EW	RW	AG	VF
1	HAB.131	✓	✓/✓	✓/—	✓/—	✓/—	—/—
2	EWC.1520	✓	✓/—	✓/—	✓/—	✓/✓	✓/—
3	ECH.0009	✓	✓/—	✓/—	✓/—	—/—	✓/✓
4	K5M.06304	✓	✓/—	✓/—	✓/✓	—/—	✓/—
5	A3P.165	✓	✓/—	✓/✓	✓/—	✓/—	—/—
6	EBS.1451	✓	✓/✓	✓/—	✓/—	✓/—	✓/—
7	A1D.092	✓	✓/—	✓/—	✓/—	✓/—	✓/✓
8	CDK.2010	✓	✓/—	✓/—	✓/—	✓/✓	✓/—
9	GVP.1149	✓	✓/—	✓/—	✓/✓	✓/—	✓/—
10	HL9.0207	✓	✓/—	✓/✓	✓/—	✓/—	—/—
11	BMS.0081	—	✓/—	✓/—	—/—	—/—	✓/—
12	KCT.13944	—	—/—	✓/—	—/—	—/—	—/—
13	B1X.1131	✓	✓/—	✓/—	✓/✓	✓/—	✓/—
14	ABW.0892	✓	✓/—	✓/✓	✓/—	✓/—	✓/—
15	B1G.1633	—	✓/—	✓/—	✓/—	✓/—	—/—
16	GX4.101	✓	✓/—	✓/—	✓/—	✓/—	✓/✓
17	CL2.1939	—	✓/—	✓/—	✓/—	—/—	—/—
18	HJ1.19176	—	✓/—	✓/—	—/—	—/—	—/—
19	HGV.0426	✓	✓/—	✓/—	✓/—	✓/✓	—/—
20	GVH.1660	—	—/—	✓/—	—/—	—/—	—/—
21	HHU.007	✓	✓/✓	✓/—	✓/—	—/—	✓/—
22	K5H.1046	✓	✓/—	✓/✓	✓/—	✓/—	—/—
23	H61.2134	✓	✓/—	✓/—	✓/✓	—/—	✓/—
24	FR1.2489	✓	✓/—	✓/—	✓/—	✓/✓	✓/—
25	GV0.1483	✓	✓/—	✓/—	✓/—	✓/—	✓/✓
26	G37.0508	✓	✓/✓	✓/—	—/—	—/—	—/—
27	HSB.0184	✓	✓/—	✓/✓	✓/—	✓/—	—/—
28	G5K.0460	—	✓/—	✓/—	✓/—	—/—	—/—
29	G19.0676	✓	✓/—	✓/—	✓/—	✓/—	✓/✓
30	G0R.1233	✓	✓/—	✓/—	✓/—	✓/✓	—/—
31	AT4.1958	✓	✓/—	✓/—	✓/✓	—/—	—/—
32	CHW.0250	✓	✓/✓	✓/—	✓/—	—/—	—/—

Table 3.5: Alignment of sentences in the artificial parallel error corpus: for each block of 5 alignments, the order of the 5 error types is randomised before seeing the input sentences, e. g. sentence 15 (B1G.1633) is not used since the third error type in this block (sentences 11–21) is the verb form error type and our error creation procedure cannot insert a verb form error into this sentence. The "Gr." column marks grammatical sentences included in the final parallel error corpus. For each error type column, the first check mark says whether the error creation procedure was able to insert an error and the second check mark gives the alignment to the grammatical input sentence.

	Classified as G	Classified as U	Total
Actual G	141,151	58,849	200,000
Actual U	73,322	126,678	200,000
Total	214,473	185,527	400,000

Table 3.6: Example of a confusion matrix (G = grammatical, U = ungrammatical)

	Classified as G	Classified as U	Total
Actual G	tn	fp	$tn + fp$
Actual U	fn	tp	$tp + fn$
Total	$tn + fn$	$tp + fp$	all

Table 3.7: True positives and other labels for confusion matrix entries (positive being defined as ungrammatical)

error density. Consequently, precision and recall for this error density can be calculated if desired.

3.4.1 The Confusion Matrix

A confusion matrix of a classifier shows for each of the correct class labels the frequency distribution of class labels assigned by the classifier. In our case the class labels are grammatical (G) and ungrammatical (U). Table 3.6 shows an example. The four cells can be described with the terms true positives (tp), false positives (fp), true negatives (tn) and false negatives (fn). However, the notion of positive and negative is subjective. Ungrammatical sentences can either be seen as negative or as positive depending on whether one focuses on the desired error-free text or on the classification task of detecting errors. We treat ungrammatical sentences as positive, see Table 3.7.

3.4.2 Evaluation Measures

From a confusion matrix, a number of evaluation measures can be derived (Manning and Schütze, 1999; Fawcett, 2006):

- Precision is the fraction of all items classified as positives that are true positives, i. e. $pr = tp/(tp + fp)$. Since this ratio spans multiple rows of the confusion matrix, it is linked to the error density $(tp + fn)/all$.

- Recall (also sensitivity, true positive rate or hit rate) is the fraction of all positive items that are detected as positive, i. e. $re = tp/(tp + fn)$.

- F-score is the harmonic mean of precision and recall, i. e. $f = 2 \times pr \times re/(pr + re)$.

- Fallout (also false positive rate or false alarm rate) is the fraction of grammatical (negative) sentences that are wrongly classified as ungrammatical (positive), i. e. $fo = fp/(tn + fp)$. In other words, this is the over-flagging rate.

- Accuracy on ungrammatical data is the fraction of ungrammatical sentences that are correctly classified as ungrammatical, i. e. $au = tp/(tp + fn)$ which is identical to recall.

- Accuracy on grammatical data is the fraction of grammatical sentences that are correctly classified as grammatical, i. e. $ag = tn/(tn + fp)$ which is identical to 1 - fallout and is also called specificity.

For the example confusion matrix in Table 3.6, the respective measurements are 68.28% precision, 63.34% recall, 65.72% f-score, 29.42% fallout, 63.34% accuracy on ungrammatical data and 70.58% accuracy on grammatical data.

3.4.3 Dependency of Precision on Error Density

Changing the error density means that the ratio of the number of grammatical sentences and the number of ungrammatical sentences shifts. In a confusion matrix as shown as in Table 3.6, we expect each row to be scaled by some factor, e. g. if the error density is dropped to 25% with the same total of 400,000 test sentences in Table 3.6 we would have 300,000 grammatical and 100,000 ungrammatical sentences, the numbers in the first row would multiply by 1.5 and the numbers in the second row would divide by 2.[19]

Precision $pr = tp/(tp + fp)$ is a ratio of values within a column of the confusion matrix, see also Table 3.7. As the rows are scaled, this ratio changes. Consequently, precision depends on the error density in the test data.[20] The same is true for f-score

[19] Of course, these are expectation values. The confusion matrix depends on the test data and its entries can only be integers.

[20] In the example of reducing the error density from 50% to 25% for the confusion matrix in Table 3.6, precision would drop from 68.28% to 41.78%.

which depends on precision.

3.4.4 Accuracy Graph and Areas of Direct Improvement

Accuracy on grammatical data ($tn/(tn+fp)$) describes the first row of the confusion matrix and accuracy on ungrammatical data ($tp/(tp+fn)$) the second row. A classifier is fully described by these two numbers and the numbers are independent from the error density of the test data. While the second number is often reported as recall, the first number is rarely available. We found two exceptions: Golding and Schabes (1996) evaluate their real-word spelling error detection system on two test sets, one with correct text and the other a copy of the first set with target words of the system replaced by other words of the respective confusion set used by the system. Pedler (2007) reports absolute numbers for all confusion matrix entries (and the number of correctly detected but wrongly corrected errors).

The pair of accuracy on grammatical and ungrammatical data can be represented by a point in the plane.[21] We choose the x-axis for accuracy on ungrammatical data. Figure 3.2 shows an accuracy point (0.6334, 0.7058) in the accuracy plane as a small black diamond where two shaded rectangular areas meet. The area to the bottom left of the accuracy point is the area of (direct or clear) degradation. It represents the set of accuracy points of all classifiers that are inferior on both scales.[22] Accordingly, the area to the top right is occupied by classifiers that are superior to the classifier shown in the graph. We call this the area of (direct or clear) improvement. Using these areas, we can easily compare classifiers in a graph. However, classifiers can fall into the undecided areas which do not allow us to say which classifier is better. In the following section, we will provide a tool to narrow these areas.

[21] This is, of course, possible with any two measurements, e. g. precision and recall. See also Section 3.5.3 for ROC analysis which plots true positive rate over false positive rate.

[22] In this definition, we ignore the fact that classifiers with very low accuracy on both scales can be trivially transformed into a well-performing classifier by negating its decisions. In practice, learning methods like decision tree induction and hill-climbing do not produce such classifiers as long as training and test data do not contradict each other or are not very small.

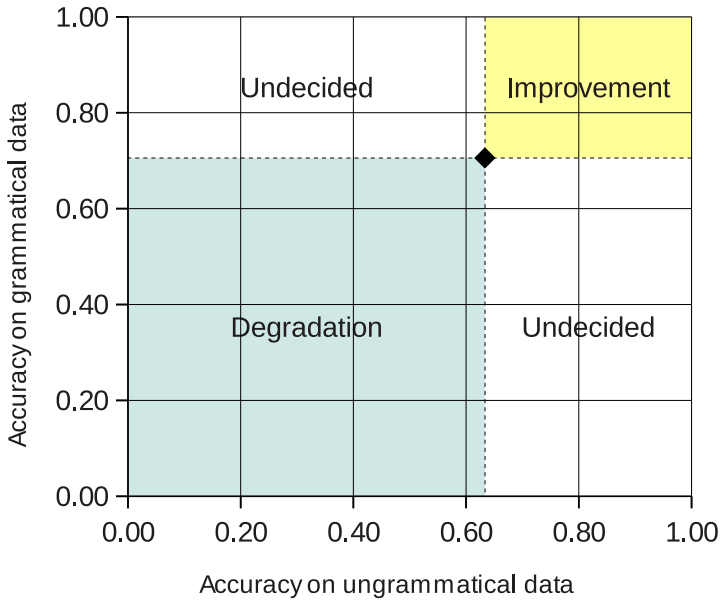


Figure 3.2: Accuracy graph for a classifier with 63.34% accuracy on ungrammatical data and 70.58% accuracy on grammatical data and the regions of direct improvement and degradation

3.5 Classifier Interpolation and Indirect Improvements

Two classifiers can be interpolated by randomly choosing one of the classifiers each time an item needs to be classified. It is easy to see that accuracy is linearly interpolated between the accuracies of the classifiers to be interpolated: first of all, assuming that the test data is not changed, each entry in the confusion matrix is linearly interpolated. The expectation value for an entry c is $c = r \times c_1 + (1 - r) \times c_2$ where r is the probability of choosing the first classifier and c_1 and c_2 are the respective values for the two classifiers. The row totals $tn + fp$ and $tp + fn$ do not change as they refer to the amount of grammatical and ungrammatical test data. These totals appear in the divisors of the calculation of accuracy. Since the dividends are just tp and tn respectively, accuracy is also linearly interpolated, i. e. the expectation values for accuracy on grammatical and ungrammatical data are $ag = r \times ag_1 + (1-r) \times ag_2$ and $au = r \times au_1 + (1-r) \times au_2$. In the accuracy graph, this means that the accuracy point of an interpolated classifier lies on the connecting line between the two classifiers that are used in the interpolation. If we vary the probability

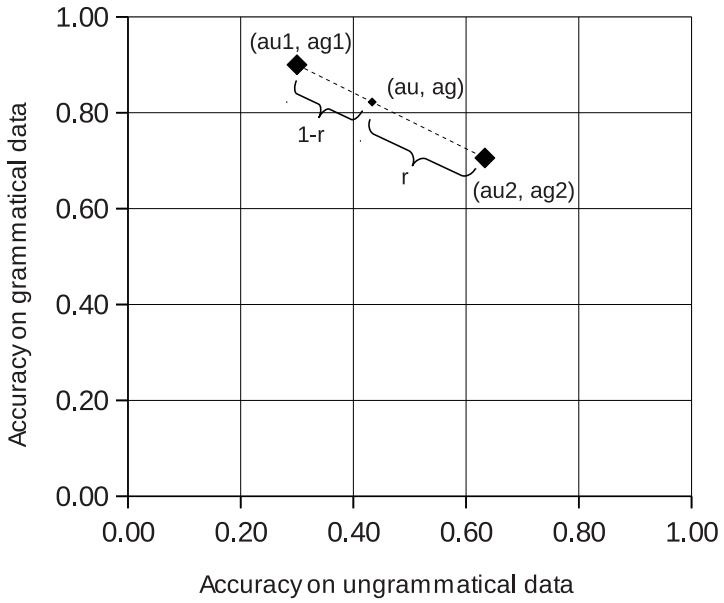


Figure 3.3: Linear interpolation of classifiers in the accuracy plane

r between 0 and 1, we draw the full line segment between the two points as shown in Figure 3.3.

3.5.1 Areas of Indirect Improvements in the Accuracy Graph

The two trivial classifiers that either classify all items as grammatical or ungrammatical are always available for interpolation with a given classifier. In an accuracy graph, the trivial classifiers occupy the upper-left and lower-right corner of the accuracy plane. Varying the parameter r (see previous section), we get two line segments connecting the two corners via the accuracy point of the given classifier as shown in Figure 3.4. Any point below the curve defined by these two line segments is in the area of (direct) degradation for some parameter r, i. e. we can build a classifier that is superior on both accuracy scales. Therefore, we can expand the area of degradation to an area of indirect degradation. This expanded area of degradation is shaded in Figure 3.4.

If we extend the line segments beyond the accuracy point of the given classifier as shown with dotted lines in Figure 3.4, we see that there is also an expanded area of improvement: for any point above the dotted lines, one of the two interpolation lines to

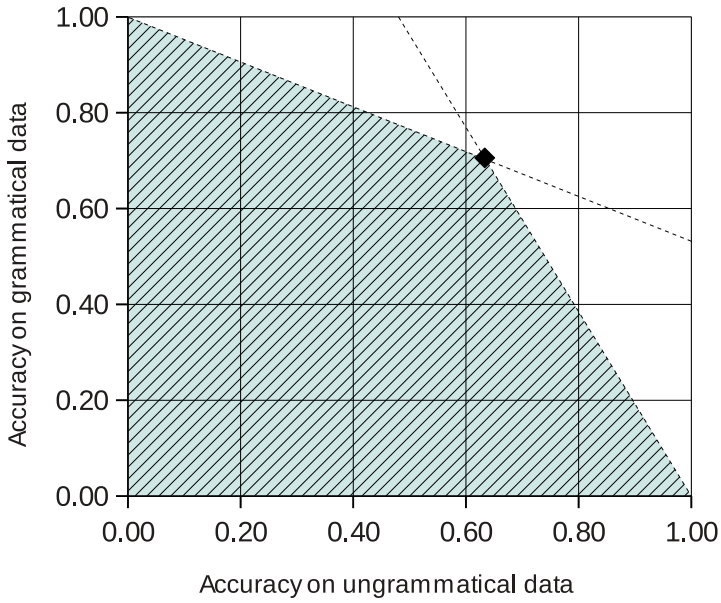


Figure 3.4: Interpolation with the trivial classifiers

the trivial classifiers must pass through the area of (direct) improvement. Figure 3.5 shows the areas of direct and indirect improvement and degradation for a classifier with accuracy point (0.62, 0.71). In comparison to Figure 3.2, the undecided areas are smaller meaning that more classifiers can be ordered in their performance independently from the error density.

3.5.2 Convex Hull of Classifiers

In the previous section, we have used classifier interpolation only with the trivial classifiers. However, consider a method that gives us a number of classifiers, e. g. with a discrete parameter inherent to the method. We can interpolate between these classifiers as well as interpolating them with the trivial classifiers. This gives rise to many line segments as shown in Figure 3.6. However, any line that is below (or to the left of) a sequence of other line segments represents inferior classifiers as for each of its points there is a superior classifier on another line segment. If we exclude all these line segments, we are left with the

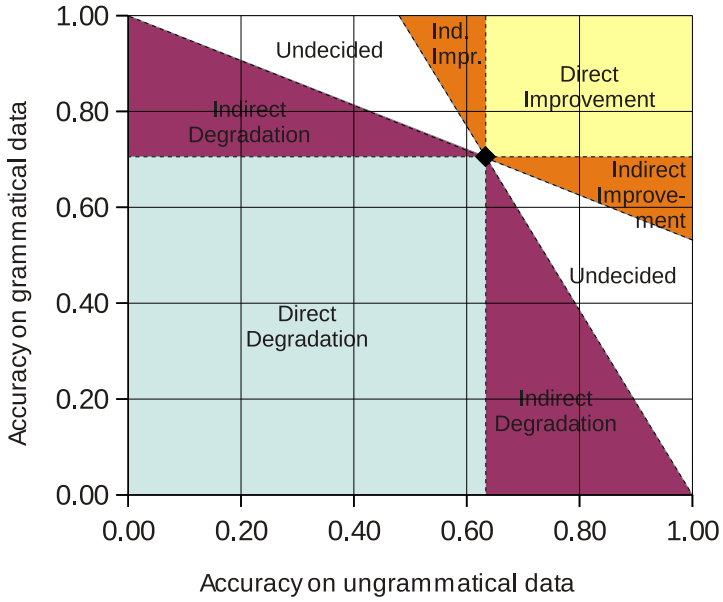


Figure 3.5: Regions of direct and indirect improvement and degradation in the accuracy plane

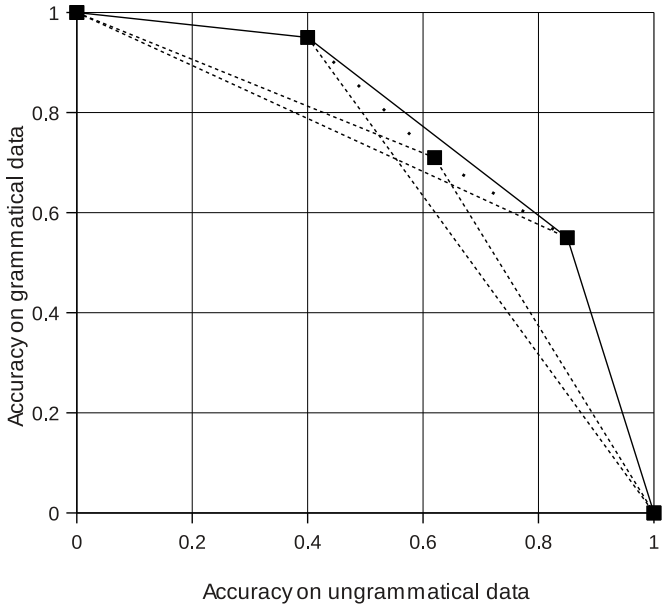


Figure 3.6: Interpolating multiple classifiers: interpolation lines between the non-trivial classifiers are dotted, the interpolations with trivial classifiers are shown with dashed lines and the upper part of the convex hull is shown solid

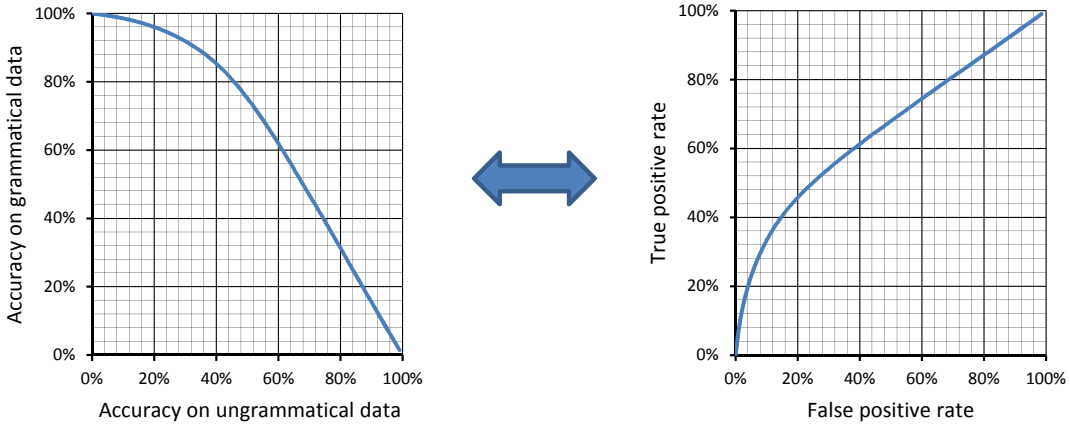


Figure 3.7: Rotating an accuracy curve 90 degrees counter clockwise gives the respective ROC curve (example showing data from Figure 5.10 of Chapter 5); note that false positve rate = 1 - accuracy on grammatical data and true positive rate = accuracy on ungrammatical data

upper-right half of the convex hull of the given classifiers including the trivial classifiers.[23] This hull can be efficiently calculated, see Section 5.2.2 in Chapter 5.

3.5.3 Related Work: ROC Analysis

A receiver operating characterics (ROC) graph plots the true positive rate of one or more classifiers over the false positive rate. As a tool for comparing and choosing classifiers, it has been developed within signal detecting theory, was picked up for medical diagnostics and only recently entered machine learning (Fawcett, 2006). Recalling the relationships of evaluation measures (Section 3.4.2), an ROC graph shows the same information as an accuracy graph and can be obtained by rotating the accuracy graph 90 degrees counter clockwise as illustrated in Figure 3.7.

Provost and Fawcett (2001) motivate the use of ROC analysis for selecting classifiers: often, "key parameters of the target environment are not known", requiring adaptation at run time. ROC graphs are independent of the class distribution in the test data and the costs of misclassification. Provost and Fawcett (2001) derive the ROC convex hull method, which corresponds to the convex hull method in the accuracy plane, with the help of iso-performance lines of objective functions that are linear in true positive and false positive

[23] In geometry, the convex hull of a set of points is the smallest convex set containing all the given points. However, in less formal terms, the interior is often excluded, yielding the tightest convex closed curve or hull surrounding the points.

rate, such as overall accuracy. Moving iso-performance lines from the upper-left corner towards any ROC curve shows that classifiers inside a concavity are inferior to the two or more classifiers on the line segment closing the concavity. Furthermore, they describe how classifiers with any performance on the connecting line segments can be constructed with stochastic interpolation. However, there is also earlier work on the ROC convex hull, e. g. Srinivasan (1999) extends the ROC convex hull to n-way classification.

Fawcett (2006) provides an introduction to ROC analysis starting with the basics of the confusion matrix and derived evaluation measures (see also Section 3.4.2). Like Provost and Fawcett (2001), he covers iso-performance lines, ROC convex hull, area under the curve (AUC) and classifier interpolation. Furthermore, averaging ROC curves, confidence intervals and multi-class ROC are discussed. Davis and Goadrich (2006) discuss the relationship to precision-recall (PR) curves. They prove that a ROC curve dominates another ROC curve if and only if the corresponding PR curves stand in the same dominance relationship. They introduce the "achievable PR curve" which corresponds to the ROC convex hull. Interestingly, the area under the PR curve is not necessarily maximised by the achievable PR curve. (For the ROC convex hull, AUC is maximised.) Barreno et al. (2008) improve ROC curves by combining binary classifiers with Boolean operations. They search the space of possible Boolean rules, e. g. the AND operator that requires that all n basic classifiers vote "1" in order for the ensemble output to be "1". Such combinations can result in classifiers outside the ROC convex hull of the basic classifiers and therefore can further expand the AUC.[24]

An alternative derivation of the ROC convex hull is given by Flach (2010): he focuses on the list of test items ranked by the classifier's scores. (It is assumed that individual classifiers are built by applying thresholds to these scores.) The ROC curve can be drawn from the ranked list by moving up for positive test items and to the right for negative test items. If multiple items have the same score, the intermediate steps are replaced with a diagonal line as no threshold could separate the test item list at these points. A concavity

[24]Note that consulting different basic classifiers does not incur additional run time if the classifiers only apply different thesholds to the same score as the score only needs to be computed once. The latter is usually the case in ROC analysis and explains why classifier combination techniques are seen as an extension of stochastic classifier interpolation (which only consults one of the component classifiers at a time).

of the ROC curve can therefore be removed by combining particluar scores into a bin, effectively discretising the scores. The ROC convex hull is the discretisation of scores achieving highest AUC.

3.6 Pooling Cross-validation Results

If we used a one-dimensional evaluation measure, averaging cross-validation results would be trivial as each cross-validation run would only elect one optimal classifier. With our two-dimensional evaluation measure, however, we can (and usually will) get a sequence of classifiers and corresponding accuracy points per cross-validation run. As we have seen in Section 3.5, the accuracy points can be connected to an accuracy curve which will be a convex hull if the sequence of classifiers has been fully optimised. It is not clear how such accuracy curves should be combined into an overall average curve. We choose a practical approach and average accuracy points of classifiers with comparable parameters. For example, in Chapter 4, we average classifiers with the same parameter C. (These classifiers have also other parameters which are estimated from training data but the parameter C is an important parameter controlling the accuracy trade-off in Chapter 4.) We take up this discussion again in Section 3.8.4.

3.7 Statistical Significance

The core idea behind statistical significance testing of the difference between two results is to calculate the probability of getting the observed difference or a larger difference under the null-hypthosis which says that both results were drawn from the same distribution. In a parametric test, e. g. using a multivariate Gaussian distribution, the magnitude of a difference is defined by the probability density function of the chosen distribution. However, we prefer a randomised test (also called an exact test) as we would like to avoid making assumptions about the distribution of results and how the magnitude of an accuracy difference should be measured. In the following, we propose a solution based on the idea of measuring the overlap of cross-validation result sets.

3.7.1 One Dimensional Case

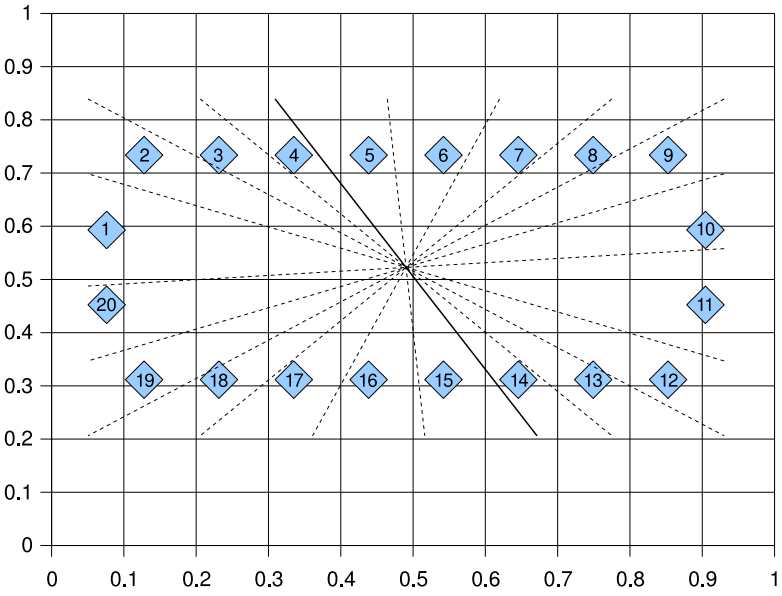
With 10-fold cross-validation, we have 10 results for each of the two methods that are to be tested for statistically significant differences. Under the null-hypothesis, these 20 results are drawn from the same distribution. Let us assume that the results for the first of the two methods to be compared are drawn first and let us consider the case that each individual result for the first method is below (inferior) to all 10 results drawn for the second method. It is highly unlikely though that the first 10 items fall below the second 10 items by chance: since each result has a 50% chance of being in the upper half, the probability of exactly this outcome is $0.5^{20} \approx 0.000001$. For the calculation of the significance level, we have to consider this event and all other events with the same or higher difference of results. The only other outcome for which the results separate into non-overlapping sets is that the first 10 items fall above the second 10 items. Again, the probability for this is 0.5^{20} under the null-hypothesis. Therefore, the p-value for the criterion of non-overlapping cross-validation results (with 10 runs) is $p = 2 \times 0.5^{20} = 0.5^{19} \approx 0.000002$ (two events).[25]

3.7.2 Moving to Two Dimensions

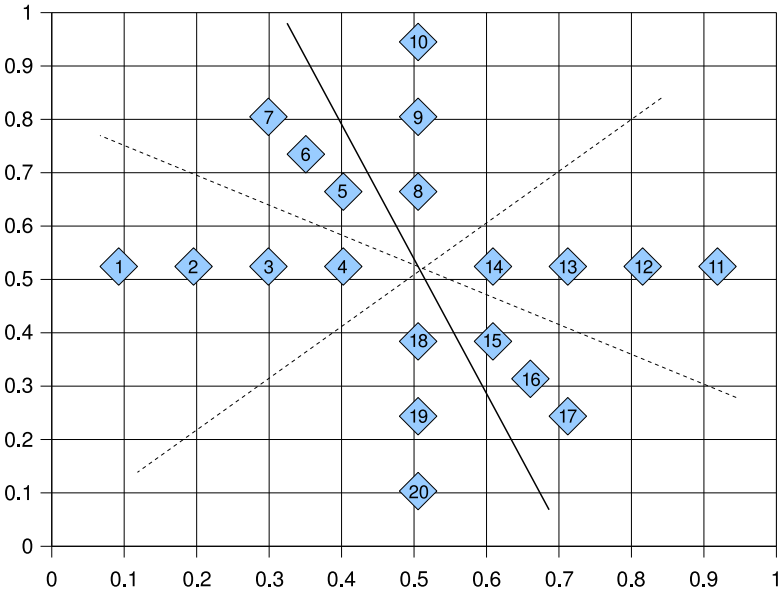
We would like to use the same criterion of non-overlapping cross-validation results discussed above with our two-dimensional evaluation metric (accuracy on grammatical and ungrammatical data in our experiments). Two questions arise: *(a)* what do we mean with overlapping results in two dimensions and *(b)* what is the significance level (p-value) with this criterion? Considering that a random configuration of points can always be separated by a curve such that the first 10 points are on one side of the curve (except for the rare event that two points coincide), we choose linear separability.[26] The p-value increases over the p-value of the one-dimensional case as the orientation of the separating line is not pre-defined and a higher number of re-orderings of the observed results are separable: Figure 3.8a shows the extreme case of accuracy points in a circle: 20 out of the 2^{20} configurations (assigning each point to one of the two result sets) are linearly separable, leading

[25] Note that this p-value is independent of the test set size. If we have insufficient test data, the variance of results will be high, results will overlap due to the high variance and the significance test will fail. As more data is provided, the cross-validation runs become more consistent.

[26] This is identical to the criterion of non-overlapping convex hulls: any overlap implies that the sets cannot be separated linearly and two separable sets will have separate hulls.



(a) circular configuration



(b) starlike configuration

Figure 3.8: Linear separability of two 10-fold cross-validation result sets in the accuracy plane; one of the possible separation lines is shown solid

to an upper bound for the p-value of $20 \times 0.5^{20} \approx 0.00002$. We think that the actual p-value is very close to this bound as we construct configurations with a lower number of possible separations by aligning accuracy points on lines as in Figure 3.8b.

3.8 Future Work

This chapter described the data sets and measures we will use for training and evaluation of methods for the automatic detection of sentences containing grammatical errors. We characterised the test sets including an artificial error corpus derived from the BNC and discussed the simultaneous evaluation on two scales, namely accuracy on grammatical and ungrammatical data. Classifier interpolation narrows the areas in the accuracy graph for which it cannot be decided whether there is an improvement or a degradation (the decision depending on the error density). The convex hull of interpolated classifiers allows us to eliminate inferior classifiers from a set of classifiers, a tool that will be useful in the training of our methods. We end the chapter with some suggestions for how the work presented here could be extended.

3.8.1 Selecting only Cleanest BNC Data

In favour of using as much data as possible and in contrast to Wagner et al. (2007a), we keep transcribed speech, poetry, headings, list items and sentences with anonymisation gaps or non-textual material in the BNC. Future experiments should either exclude such problematic text or train separate classifiers for these text types and provide an evaluation broken down by text type. Using text type information during testing can be justified with the likely availability of this information in applications. Finally, sentences with non-word spelling errors could be excluded and sentence-initial bullet points be removed.

3.8.2 Adding Additional Error Corpora

Since carrying out the experiments presented in Chapters 4 to 7, some new learner corpora have become available.[27] Evaluation could also be extended to other types of artificial error

[27] For example, a number of new learner corpora have been presented at the Fifth Corpus Linguistics Conference, `http://www.liv.ac.uk/english/CL2009/`, and at the Fifth Workshop on Innovative Use of NLP for Building Educational Applications (at NAACL 2010).

corpora, e. g. n-gram language model and machine translation output (see Section 3.3.1).

3.8.3 More Detailed Error Modelling

The error creation procedure could be expanded not only to support more error types, e. g. preposition and article substitution errors that are frequent in learner data,[28] but also to model errors more closely to the errors observed in authentic data. Additional contextual information of the error could be used, e. g. the position of errors (relative to start and end of the sentence or relative to phrase boundaries) — see also Foster and Andersen (2009). Automatically extracted subcategorisation frames (O'Donovan et al., 2004; Chrupała and van Genabith, 2007) could be used to avoid inserting missing word or extra word errors in contexts that would lead to covert errrors (Section 3.3.3).

3.8.4 A Revised Method for Averaging Accuracy Curves

Currently, we average the results for each parameter setting of a method over the cross-validation runs. However, parameter settings may not be optimal in all cross-validation runs and results may vary largely for one parameter setting. An alternative could be to build classifiers with specific accuracy trade-offs in each cross-validation run (using classifier interpolation if necessary) and to average over these. The accuracy trade-off could be parameterised with the angle $\arctan(ag/au)$ where ag and au are accuracy on grammatical and ungrammatical data.

[28] Note that missing and extra preposition and article errors are part of our missing and extra word error types.

Chapter 4

Detecting Ungrammatical Sentences by their Deviation from Estimated Parse Probabilities: the APP/EPP Method

In this chapter, we introduce a new method for detecting ungrammatical sentences. The method uses the parse probability of the best parse of a sentence under consideration as an indicator of its grammaticality. We overcome the problem that this probability is influenced predominately by factors such as sentence length and lexical choice by comparing the parse probability of the input sentence to the probability expected for grammatical sentences of the same type as the input sentence. In our simplest implementation of the APP/EPP method, a reference corpus of grammatical language provides the probability estimate: sentences similar to the input sentence are retrieved from the corpus and their average probability is used as the estimated parse probability (EPP). If the actual parse probability (APP) of the input sentence is significantly lower than the EPP, the sentence will be classified as ungrammatical.

Sections 4.1 to 4.5 motivate and describe the method. Section 4.1 briefly reviews the basics of probabilistic grammars required to understand the method. Section 4.2 presents observations on a parallel error corpus in order to substantiate the claim that

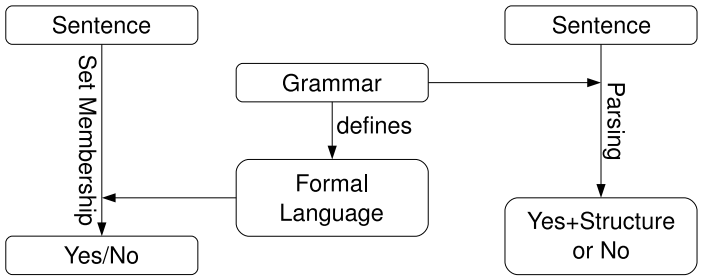


Figure 4.1: Grammaticality and formal languages

parse probabilities to some extent reflect grammaticality. The general idea underlying our method is described in Section 4.3. Research questions specific to the method are formulated. Section 4.4 discusses related work. Section 4.5 presents a series of models employed in the method. Section 4.6 reports the experiments carried out. The chapter finishes with conclusions and open questions for future research.

4.1 Parse Probabilities and Treebank-induced Grammars

In theory, parsing provides a grammaticality judgement as shown in Figure 4.1. Whether or not a sentence is grammatical is determined by its parsability with respect to a grammar of the language in question. A grammar together with a notion of derivation defines how strings can be derived from a start symbol and a set of rules. The set of all derivable strings identifies a formal language which will coincide with the set of all grammatical sentences of a natural language once all the rules describing the syntactic structures of the language have been found and coded in the grammar. Overgeneration, i. e. failure to reject an ungrammatical string, will be interpreted as a flaw in the grammar that needs to be addressed. This architecture dominates manual grammar writing, it is a key concept in generative linguistics and is strongly influenced by Chomsky's view on language (Chomsky, 1957).

Unfortunately, writing grammars for natural languages in this way is very difficult and time-consuming and as yet nobody has actually succeeded in providing a truly complete grammar for any natural language (Bod, 1992b, p. 26). Hand-written grammars usually achieve only partial coverage, i. e. a parser will reject sentences that are judged grammatical by most native speakers. On the other hand, there are data-driven methods that

automatically derive a big grammar from training corpora. They tend to generalise too much and produce grammars that accept nearly any sequence of words as grammatical. Data-driven grammars are useful in applications that assume well-formed input and require an analysis of the input.[1] To select a plausible parse among all possible parses — with a large grammar there could be billions of parses — a probabilistic model is usually added to the grammar. The combination of overgeneration and probabilistic parse selection results in high robustness to errors and broad language coverage. These are very desirable properties in many practical applications. For grammar checkers, however, it is generally assumed that data-driven parsers are less useful as they fail to reject ungrammatical strings.

4.1.1 Generative Probabilistic Parsing Models

A generative probabilistic parsing model assigns probabilities to parse trees. Starting from the start symbol that is the root of each parse tree, each production step, i. e. adding daughter nodes to a node, is often viewed as an independent event that can occur with a certain probability. Therefore, the total probability of a parse tree generated from the start symbol is simply the product of all production probabilities.[2] While plain PCFG models estimate the probabilities based on the frequency of the productions in parse-annotated training data, more sophisticated models condition the probabilities on context, for example through parent annotation and head-lexicalisation (Johnson, 1998; Collins, 1999), or by combining arbitrary features in a log-linear model (Johnson et al., 1999; Charniak, 2000). Horizontal Markovisation can even assign high probabilities to productions that have not been observed in the training data at all. Klein and Manning (2003) gain additional improvements by adding other linguistically motivated annotations. In contrast, Petrov et al. (2006) automatically uncover *latent annotations* on non-terminal symbols by iteratively splitting and merging non-terminal symbols and guiding the split-

[1] To some extent, data-driven grammars are also robust to errors and produce parse trees similar to the ones found for corresponding grammatical sentences and it is estimated that they would be similar to gold standard parses if such test data was available (Foster, 2005, pp. 159–161, 189–198; Foster 2007).

[2] Here, it is assumed that the notion of derivation restricts the order in which symbols can be expanded so that there is only one derivation for each parse tree. Otherwise, the sum of the probabilities over all derivations leading to the parse tree has to be used. A well-known example of a parsing model that has to consider all derivations is the DOP model of Bod (1992a).

#	Sentence (Tokenised)	Tokens	Nodes	P(Parse)	Log.
1	*John loves Mary .*	4	$5+8$	5.7×10^{-25}	-55.8
2	*She buys shares .*	4	$5+8$	2.0×10^{-14}	-31.5
3	*She buys radiators .*	4	$5+8$	3.7×10^{-21}	-47.0
4	*She likes shares .*	4	$5+8$	2.7×10^{-15}	-33.5
5	*She likes radiators .*	4	$5+8$	3.4×10^{-21}	-47.1
6	*John loves Mary who buys shares .*	7	$11+14$	1.1×10^{-38}	-87.4
7	*John loves Mary who buys radiators .*	7	$11+14$	2.0×10^{-45}	-102.9

Table 4.1: Probability of the best parse for 7 made-up sentences

and-merge decisions by the objective to maximise the probability of the treebank. The principle that the parse probability is the product of the probabilities of events which correspond to the nodes in the parse tree stays the same though.[3]

Probabilistic parsing searches for the most likely parse tree that has the same yield as the input sentence. Note that this parse probability is not the probability of the parse tree being the "correct" parse tree for the input sentence, where "correct parse tree" refers to the tree a human annotator would choose. It makes sense though to assume that the parse tree that is most likely generated is also most likely "correct" if the probabilities of the individual productions are based on statistics of manually annotated treebanks.[4]

While technical details of how the probabilities are conditioned and estimated can be found in the literature, a set of example sentences and their parse probabilities will be given here. Table 4.1 shows a simple example sentence and some variations. The sentences were parsed with the first-stage parser of Charniak and Johnson (2005)'s reranking parser as of June 2006 trained on Penn Treebank WSJ data (Marcus et al., 1994), hereafter referred to as Charniak's parser.[5] The first observation is that parse probabilities are very small even for simple, short sentences. If we replace the noun in sentence 2 by a

[3]Of course, to give the parse probability of a tree without extra annotation, the probabilities of all trees that reduce to the same unannotated tree have to be cumulated.

[4]Alternative selection criteria have been proposed in the literature: Goodman (1996) maximises the expected recall. Carroll and Rooth (1998) mention a criterion which sums the probabilities of all possible parses for chunks. In addition, there are a number of methods that rerank a list of n-best parses (n-best according to the parse probability): Collins (2000) trains a discriminative model to rerank parses. Another method approximates the minimum risk criterion by evaluating each candidate parse on the remaining $n-1$ parses, e. g. Titov and Henderson (2006).

[5]Charniak and Johnson (2005) extend the generative parser of Charniak (2000) to n-best parsing and combine it with a discriminative re-ranker, but in this chapter we only use the best parse according to the generative model. Also note that we mean the complete generative parser with "first-stage parser", not the coarse-grained CFG parser that precedes the main parsing step for efficiency.

less frequent noun (according to the training data of the parser) as in sentence 3, the parse probability drops by 7 orders of magnitude.[6] This decrement indicates that lexical information has a large impact on the probabilities. However, in sentences 4 and 5 which exhibit the same noun variation with a different verb, the probability ratio is different (only 6 orders of magnitude) to the one observed in sentences 2 and 3. The context seems to play a role as well. Sentences 6 and 7 combine 2 of the first 3 sentences. The resulting parse probabilities are very close to the products of their components.[7]

The right-most column of Table 4.1 shows the natural logarithm of the parse probability. Instead of calculating the ratio of probabilities, it is easier to look at the differences on the logarithmic scale. For example, there are 15.5 points between sentences 2 and 3 and also between 6 and 7. The table also gives the sentence length in tokens and the number of nodes in the most likely parse tree broken down by internal nodes and pre-terminal nodes with their leaves (2 times the number of tokens). Combining sentences 1 and 2 to sentence 6 only requires one additional internal node in the parse tree. We will pay particular attention to the relationship between sentence length, number of nodes and parse probabilities in the experiments below.

4.2 Do Parse Probabilities Reflect Grammaticality?

The error detection method presented in this chapter is based on the assumption that parse probabilities reflect grammaticality. Naively, one would expect that a probabilistic parser assigns lower probabilities to ungrammatical strings than to grammatical strings because it has been trained on grammatical strings only. However, language is highly productive and most grammatical strings will be new to the parser as well. In addition, the assumption is only weakly supported by previous research (see Section 4.4) which mostly found only a small or no advantage in using parse probabilities for error detection or for ranking machine translation output. Therefore, before we build yet another system based on parse probabilities in the following sections, this section provides evidence that the assumption that grammaticality is reflected in parse probabilities is in fact justified.

[6] The noun "shares" was chosen because it is one of the most frequent nouns in the WSJ training data of the parser, while the noun "radiators" is unknown to the parser.

[7] Note that adding the exponents is equivalent to multiplying the numbers: $b^{x+y} = b^x \times b^y$.

The objective of the experiment reported here is to determine whether and to what extent ungrammatical sentences behave differently from grammatical sentences as regards their parse probabilities. We study two types of corpora:

- two parallel error corpora that consist of authentic ungrammatical sentences and manual corrections, and
- a parallel error corpus that consists of authentic grammatical sentences and automatically induced errors.

The parallel corpora allow us to compare pairs of sentences that have the same or very similar lexical content and differ only with respect to their grammaticality.[8] A corpus with automatically induced errors is included because such a corpus is much larger than the available authentic data and also because it sheds some light on the suitability of artificial errors for evaluation.

Previous work exploring parse probabilities as a ranking criterion for n-best machine translation or other generation output has only limited relevance to this section as the effect of grammaticality on parse probabilities is usually not reported — see Section 4.4.4 for a discussion. We noticed one exception: Carter and Monz (2009) report the fraction of reference translations that receive a higher parse probability than respective machine translation output and the average logarithmic parse probability for the two types of sentences, showing that parse probabilities can be used to distinguish machine translation output from human reference translations.

4.2.1 Parallel Corpora with Authentic Errors

The first parallel error corpus we analyse is from Foster (2005). The corpus contains 1,132 pairs comprising an authentic ungrammatical sentence and a correction each. (See Chapter 3 for details on the corpus.) Figure 4.2 shows how corrections affect the probability of the best parse obtained with Charniak's parser. For ranges of 4 points on the logarithmic scale, the bars depict how many sentence pairs have a probability ratio within the respective range. For example, there are 48 pairs (5th bar from left) for which the

[8] Basic unedited learner data is less suitable as differences between the grammatical and ungrammatical subset can happen for various reasons, e. g. sentence length, vocabulary and proficiency level.

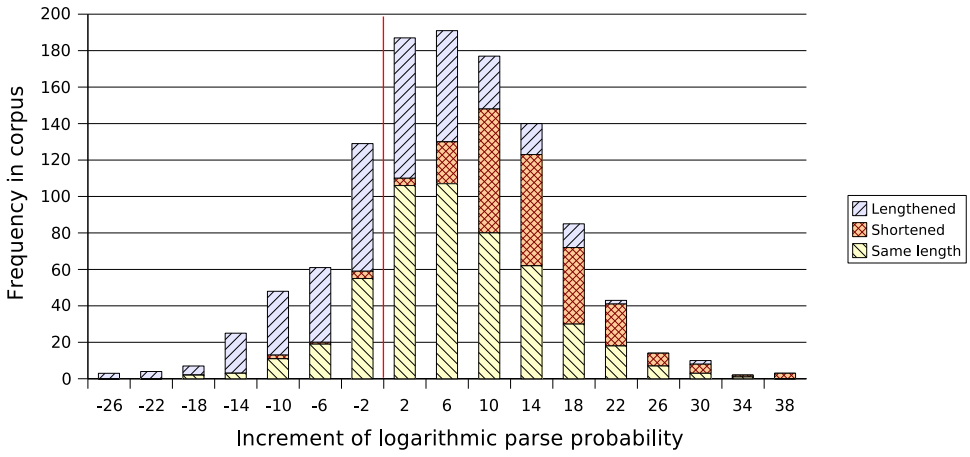


Figure 4.2: Effect of correcting erroneous sentences (Foster corpus) on the probability of the best parse. Each bar is broken down by whether and how the correction changed the sentence length in tokens. A bar labelled x covers ratios from e^{x-2} to e^{x+2} (exclusive).

correction has a parse probability which is between 8 and 12 points lower than the parse probability of its erroneous original, or, in other words, for which the probability ratio is between $e^{-12} \approx 0.000006$ and $e^{-8} \approx 0.000335$.[9] The overall distribution looks somewhat discouraging with its wide range of possible effects grammatical errors can have on parse probabilities. However, 853 pairs show a higher probability for the correction vs. 279 pairs which do not. This is a ratio of 3.06:1. If we focus on corrections that do not change sentence length (note that this excludes certain error types completely), the picture becomes even more favourable with 414 vs. 90 pairs, a ratio of 4.60:1. Ungrammatical sentences do often receive lower parse probabilities than their corrections.

The second corpus is a learner corpus of transcribed spoken utterances of students learning English. Wagner et al. (2009) manually corrected 500 erroneous sentences producing a parallel error corpus which we call Gonzaga 500 — again, see Chapter 3 for details on this corpus.[10] Figure 4.3 shows a picture similar to the Foster corpus. The peak for the range from $e^0 = 1$ to $e^4 \approx 54.6$ is much more pronounced. Overall, 348 sentence pairs show an increased parse probability for the corrected sentence, 152 do not. This is a ratio of 2.29:1 which is smaller than for the Foster corpus (3.06:1). For sentences that stay the same length the ratio is 154 to 34, or 4.53:1, for this corpus which is almost identical to the

[9] The next range boundaries used in the graphs are $e^{-4} \approx 0.018$, $e^0 = 1$, $e^4 \approx 54.6$, $e^8 \approx 2981$ etc.

[10] It is permissible to use test data here because we do this after the experiments of this chapter have been conducted. Only the results of the Foster corpus above have influenced design decisions, if at all.

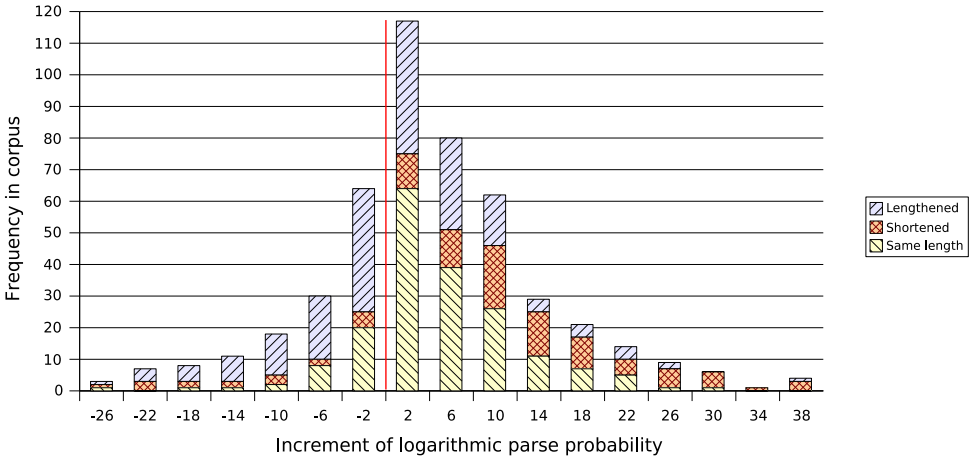


Figure 4.3: Effect of correcting erroneous sentences (Gonzaga 500 corpus) on the probability of the best parse. Each bar is broken down by whether and how the correction changed the sentence length in tokens. A bar labelled x covers ratios from e^{x-2} to e^{x+2} (exclusive).

Foster corpus (4.60:1). It is clear from these observations that grammatical errors have some negative effect on the probability assigned to the best parse. However, the figures show that factors other than grammaticality have a strong effect on parse probabilities as well.

4.2.2 A Parallel Corpus with Artificial Errors

How do the observations of the previous section using authentic errors translate to the artificial parallel error corpus we created from BNC data? We extract 199,600 sentence pairs from the first cross-validation set that we use in our experiments in Section 4.6.[11] Figure 4.4 shows what happens to the parse probability of a BNC sentence when Foster's error creation procedure (Section 3.3 of Chapter 3) inserts an error. In order to keep the orientation of the graph as before, we change the sign by looking at decrements instead of increments. Also, we swap the shadings for shortened and lengthened sentences. Clearly, the distribution is wider and moved to the right. The peak is at the bar labelled 10. Accordingly, the ratio of the number of sentence pairs above and below the zero line is much higher than before (overall 32,111 to $167,489 = 5.22$, for same length only 8,537 to 111,171 = 13.02), adding further support to the assumption that grammaticality affects

[11] 400 of 200,000 pairs could not be included due to sentence length restrictions and five parse failures.

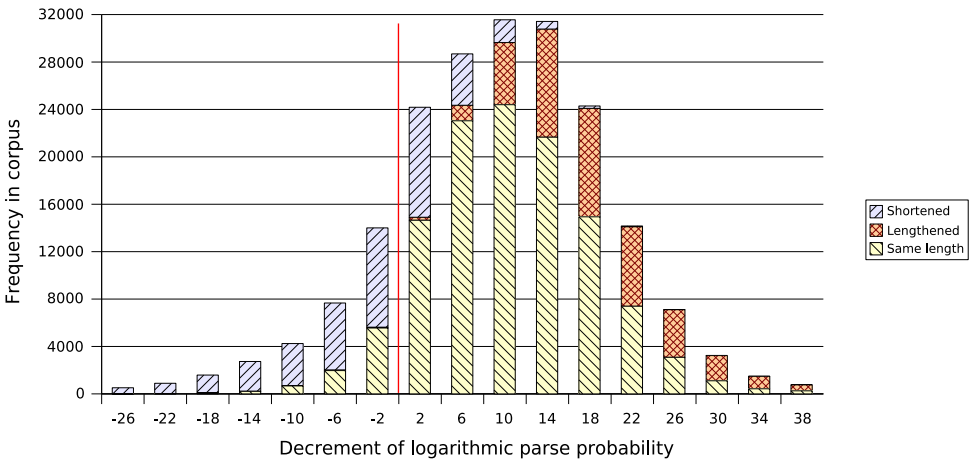


Figure 4.4: Effect of inserting errors into BNC sentences on the probability of the best parse. Each bar is broken down by whether and how the error creation procedure changed the sentence length in tokens. A bar labelled x covers ratios from e^{x-2} to e^{x+2} (exclusive).

parse probabilities, but also suggesting that our artificial errors might be easier to detect with probabilistic parsing than authentic errors. This does not necessarily mean that the error creation procedure inserts errors in an unnatural way, for example at odd positions. The differences may well be due to the fact that the automatic error creation procedure only mimics the five main error types observed in Foster's corpus of authentic errors.

4.2.3 Conclusion

The experiments in Sections 4.2.1 and 4.2.2 show that roughly 3/4 of ungrammatical sentences have a lower parse probability than their corrections or grammatical counterparts. Depending on the type of data, the average effect of grammaticality on parse probability is up to 14 points on the logarithmic scale. While this is less than the effect of lexical content observed in Section 4.1.1, the effect is clearly visible and we proceed to explore the idea of detecting ungrammatical sentences by their deviation from estimated parse probabilities for grammatical sentences of the same type.

Wagner and Foster (2009) summarise the findings of this section and add a brief discussion of the effect of certain error types.

4.3 General Idea and Research Questions

Unlike in the experiments of Section 4.2, in the task of judging the grammaticality of a sentence we will not have access to a correction (otherwise the problem would already have been solved). Therefore, the parse probability of a correction is not available as a reference point for comparison in order to judge grammaticality. Other means are needed to derive such a reference point from the input sentence. In this chapter, we propose to use a probability estimate as a reference probability that accurately estimates parse probability of grammatical sentences but fails to model grammaticality and therefore overestimates the probability of ungrammatical sentences. We show how such a model can be built.

The introduction to this chapter exemplified the APP/EPP method through the idea of comparing the sentence under consideration to grammatical sentences of the same "type". This stems from the data-driven model that we chose as the core component of the implementation of our method. It retrieves grammatical reference sentences from a large corpus and uses their parse probabilities as a reference point. However, our method is not restricted to such a data-driven model and neither is our implementation because it contains a component that re-adjusts the estimated parse probabilities according to other models as described in Section 4.5.8. In this section, we define the APP/EPP method, state the properties the estimation model for parse probabilities must have, refine the research questions, derive evaluation measures and discuss related work.

4.3.1 Brief Description of the APP/EPP Method

Central to the APP/EPP method is a model that provides the reference probability or estimated parse probability (EPP) with which to compare the actual parse probability (APP) of an input sentence under consideration. Our aim is to judge whether the input sentence is grammatical. The EPP is an estimate of what the parse probability of the input sentence would be if it was correct. One possible way of approximating this probability is to use probabilities of sentences that are similar to the input sentence in terms of length, lexical items etc. If the APP is significantly lower than the EPP, we flag the input sentence as ungrammatical. Therefore, the EPP model must

- map input sentences to parse probabilities without actually parsing them,

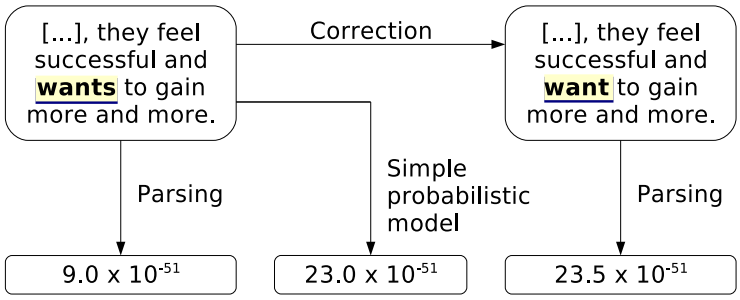


Figure 4.5: Using a simplified model to estimate the parse probability of a corrected version of the input sentence

- make sufficiently good predictions for grammatical input, but
- fail to penalise grammatical errors in the input as we need to estimate the parse probability we would expect if the sentence was grammatical.

If presented with an ungrammatical sentence, the EPP model will systematically overestimate the parse probability because it assumes that the input sentence is grammatical, i. e. it needs to be insensitive to ungrammatical aspects of it. A very simple model is, for example, an exponential function of the sentence length. More complex models will be built in the course of this chapter. They will use machine learning methods to exploit various features of sentences. Since the aim is to predict the probabilities accurately for grammatical sentences, training data contains only grammatical sentences.

Figure 4.5 relates the proposed method to the observations made in Section 4.2. Suppose the ungrammatical sentence in the left box is given as input. We would like to compare its parse probability with the parse probability of a corrected version shown on the right of Figure 4.5. However, the correction is unavailable. This gap is filled by the proposed probabilistic model that outputs a probability similar to the parse probability of the (unavailable) correction. We then compare the two probabilities in order to judge the grammaticality of the input sentence.

The ratio of estimated parse probability (EPP) and actual parse probability (APP) is used as an indicator of grammaticality. Assuming an ideal EPP model, an APP/EPP ratio close to 1.0 means that there is no error in the input sentence. If the ratio is considerably smaller than 1.0, then there must be a grammatical error. The lower APP reflects an error that the simpler EPP model fails to detect and therefore outputs a higher EPP. Ratios

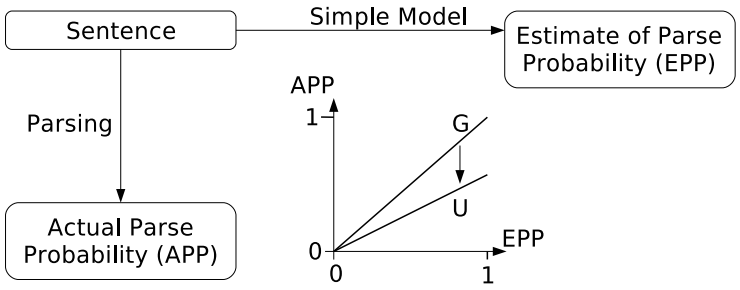


Figure 4.6: Estimated and actual parse probability: if the desired model can be built, grammatical sentences (G) will fall on the diagonal of the APP/EPP graph and ungrammatical sentence (U) will fall below it.

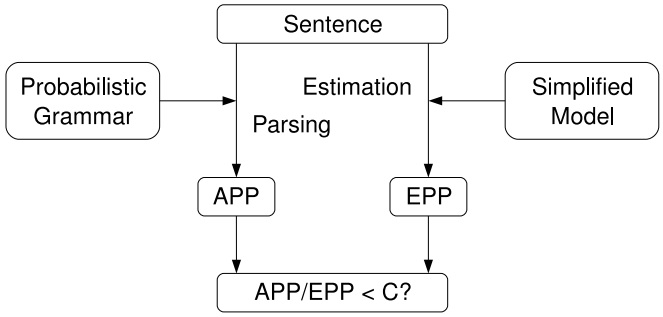


Figure 4.7: Proposed architecture for the detection of ungrammatical sentences

greater than 1.0 can only occur due to imperfections in the EPP model.[12]

Figure 4.6 illustrates the relationship between APP and EPP. If the EPP prediction model is perfect, plotting APP over EPP for grammatical sentences will give the bisecting line (G) because both probabilities are identical for each sentence (APP/EPP = 1.0). Ungrammatical sentences (U), however, will fall below this line because EPP is not affected as much as APP by grammatical errors.[13]

[12]The observation in Section 4.2 that up to 1/4 of ungrammatical sentences receive higher parse probabilities than their grammatical counterparts is consistent with our picture: the effect of grammaticality becomes manifest in the location of the centre of the distribution between 2 and 10 logarithmic points of increments in parse probability. The width of the distribution can be explained by the effect of other factors like sentence length and lexical choice. The EPP model accounts for exactly these factors. Recall that we do not aim to model the parse probability of actual corrections of an input sentence, but the parse probability of the input sentence as if it was grammatical.

[13]Figure 4.6 also explains why the ratio APP/EPP is proposed as an indicator and not the difference APP-EPP. Due to the multiplicative nature of probabilistic parsing models, the probabilities get arbitrarily small for longer and longer sentences. Long sentences are concentrated near the origin on the graph in Figure 4.6. Since the probabilities cannot be smaller than 0, also the difference must get arbitrarily small and therefore the magnitude of the difference does not say much about the grammaticality. In contrast, the slope of the lines stays constant and the ratio APP/EPP tells us whether we are closer to the line representing grammatical data or the line for ungrammatical data.

A binary classification can be derived from APP/EPP ratios by simply applying a threshold C, see Figure 4.7. Different trade-offs between high accuracy on grammatical data (few grammatical sentences are erroneously flagged) and high accuracy on ungrammatical data (most errors are detected) can be made. The classification threshold C can be visualised in Figure 4.6 as a line through origin with slope C. This line must run between the two lines representing grammatical (G) and ungrammatical sentences (U) and should be at some distance to both of them in order to account for noise. The observations of Section 4.2, in particular the peak of the distribution in Figure 4.4, suggest that the threshold C should be somewhere between $e^{-12} \approx 6.1 \times 10^{-6}$ and 1.

4.3.2 A More Abstract Description of the APP/EPP Method

The main idea of the APP/EPP method is not restricted to probabilistic parsing. Two models are needed of which one reflects grammaticality more strongly than the other. The weaker model approximates the behaviour of the more complex model for grammatical sentences, but fails to do so for ungrammatical input because it does not detect deviations from correct language use, or at least not to the same extent as the stronger model. The two models' outputs can be probabilities, real numbers, vectors or something else. An indicator of grammaticality is constructed as a function of the output of the two models. The function can be the ratio of two numbers, their difference or some other function — see the discussion in Footnote 13 for criteria in the example of parse probabilities. For a binary classification into grammatical and ungrammatical sentences, a simple threshold can be applied to the indicator value.

4.3.3 Research Questions

If we simply used different off-the-shelf APP and EPP models, e. g. Charniak's parser for APP and a trigram language model for EPP, naturally we would expect systematic differences between the two models even for grammatical sentences. Clearly, such a combination of models would not be useful for our method as we would like APP and EPP to agree on grammatical input. Therefore, the EPP model should be designed to be similar to the APP model for grammatical sentences and, a priori, it is not entirely clear whether

it is possible to do so without also copying the behaviour for ungrammatical sentences. The investigation in this chapter is limited to the special case that the more complex APP model is provided by Charniak's probabilistic parser.[14] Therefore, the main research questions for this chapter are

1. Can we build a simple probabilistic EPP model that produces good estimates of the probability output of Charniak's parser?

2. Can we do this without penalising ungrammatical input in the same way as the APP model does?

The former can be measured by the mean square error (MSE) of the prediction on a grammatical test corpus on the logarithmic scale.[15] The prediction error has to be smaller than the effect of grammatical errors in order for the APP/EPP method to work well.

The second question is more difficult to address. In principle, the answer is yes if the average APP/EPP ratio for ungrammatical test data stays below 1. However, in order for the ratio to perform well in a classifier (as illustrated in Figure 4.7), the average ratio has to be sufficiently small in relation to the noise. Figure 4.8 shows the type of distributions of APP/EPP ratios for grammatical and ungrammatical data we hope to obtain with our EPP models. Assuming that the EPP model is optimised for good estimates on grammatical data, the distribution for grammatical data is centred around 0[16] and its standard deviation σ_{G} is the square root of the mean square error (which happens to be the evaluation metric we use to address the first research question above). For ungrammatical data, the distribution and its mean μ_{U} move to the left in the graph because the EPP values are systematically greater than the APP values. We also expect the distribution to be wider than for grammatical data because different error types have different effects on parse probabilities. Therefore, σ_{U} is wider in our illustrative example

[14] This parser was chosen for various reasons: (a) it was the best-performing constituent parser on the standard WSJ test data at the time, (b) it is robust when used with the shipped PTB-induced grammar, it parses almost any input, (c) it is faster than the other constituent parsers available at that time and, most importantly, (d) it outputs parse probabilities (though only in n-best mode).

[15] In principle we could measure the mean square error on raw probabilities, but then we would miss small but significant errors for longer sentences that have very small parse probabilities. See also Footnote 13 on why we use APP/EPP ratios, i. e. log-differences, and not differences between raw values.

[16] If the mean μ_{G} was not 0, a better EPP model could be built by subtracting the mean from the output of the previous model. For other objective functions, μ_{G} may systematically deviate from 0.

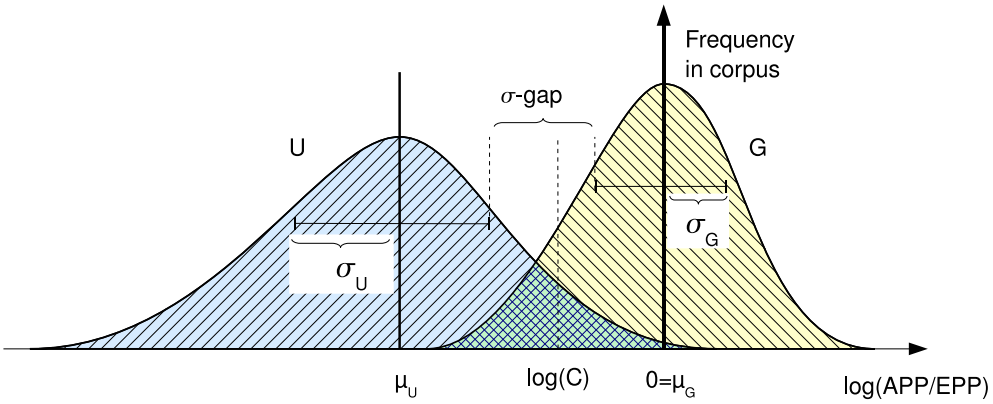


Figure 4.8: How to measure the behaviour of the EPP model on ungrammatical data (fictional results for illustrative purpose only): μ is the mean, σ the standard deviation or square root of the variance, G and U stand for grammatical and ungrammatical data.

in Figure 4.8. The classifier will apply a threshold C to the APP/EPP ratios, represented by a dashed line at log(C) in Figure 4.8. Any sentence to the left of this line is flagged as ungrammatical. We propose to use the size of the gap between the two σ-intervals, i. e. $(\mu_{\mathrm{G}}-\sigma_{\mathrm{G}})-(\mu_{\mathrm{U}}+\sigma_{\mathrm{U}})$, as a measure of how insensitive the EPP model is to grammaticality and also as an indicator of how well a classifier may do while staying independent of the choice of threshold C.[17]

4.4 Relation to Previous Work

Previous data-driven approaches to detecting grammatical errors focus on automatically learning patterns of POS, lexical items and phrase categories that are indicative of grammatical errors (Section 2.2.2 of Chapter 2). In cases where parse probabilities are added to the feature set, e. g. in Lee et al. (2007) and Sun et al. (2007), the observed improvements in accuracy are small and the training data includes negative examples, i. e. ungrammatical sentences.

Lee and Seneff (2006) detect and correct ungrammatical sentences by comparing the parse score[18] of a possibly ill-formed input sentence to the parse scores of candidate corrections which are generated by arbitrarily deleting, inserting and substituting articles,

[17]The σ-gap would be suitable as an objective function or evaluation measure for any kind of binary classifier that internally uses a graded score. A question for future work is the bias of this objective.

[18]It is not clear whether this is the best parse score or the generative probability of the sentence.

prepositions and auxiliaries and changing the inflection of verbs and nouns. Unlike our APP/EPP method which cannot tell the location of errors, let alone propose a correction, Lee and Seneff (2006) inherently obtain a correction from the winning candidate string. However, their system is only tested on the restricted domain of flight enquiries. They relate their approach to natural language generation and propose to try other reranking models as future work, e. g. discriminative parse tree rerankers.

4.4.1 Psycholinguistics

Previous work studying the suitability of probabilistic parsing for making grammaticality judgements only considers direct qualities of the probabilistic model like the absolute probability of a sentence or its best parse, e. g. Koontz-Garboden and Jaeger (2003) measure the correlation of the frequency ratios of competing surface realisations with human acceptability judgements. Crocker and Keller (2006) provide a good discussion of the literature, but there seems to be, to our knowledge, only one piece of work that uses probabilistic parsing in a more sophisticated manner: Hale (2003) calculates the information-theoretic load of words in sentences assuming that they were generated according to a probabilistic grammar and then finds that these values are good predictors for observed reading time and other measures of cognitive load. However, psycholinguistic research seems to focus on borderline cases which show graded acceptability by human judges rather than clear-cut cases that we are interested in here and that can often easily be classified as either grammatical or ungrammatical by humans.

4.4.2 Deviation Detection

Holst et al. (2004) show that the idea of modelling "normal" behaviour in order to detect unusual or undesired events is in fact employed in areas such as network intrusion detection, billing fraud detection and ship movement monitoring. Here, the likelihood of a new event is calculated according to previous observations and a given model space. A low (log) likelihood is interpreted to mean that the event is abnormal and should be flagged. The underlying assumption is that the likelihood correlates well with whether an event should be flagged. This, of course, depends on the model. For example, in Sec-

tion 4.2 we show that parse probabilities as an approximation of a parsing-based language model (probability sum over all possible trees) drop only by a few orders of magnitude if confronted with ungrammatical input while they vary considerably more with sentence length.[19]

4.4.3 Discriminative Language Models

Discriminative language models map input strings to a score much like generative probabilistic models, but are trained to rank lists of candidate strings. Training data typically contains lists of competing candidates. For example, Roark et al. (2007) train a global conditional log-linear model on word lattices of a baseline speech recogniser in which the word sequence with the lowest word error rate (according to a reference transcription) is marked as the one that the discriminative model should bring to the top of the list. It should be noted that in this example the word error rate of the overall system is only optimised indirectly: the objective function employed in the training phase maximises the probability of the training data as in generative language models. What makes the model discriminative is the conditioning on the list of candidates. The probabilities of all candidates are required to sum to 1 for each training list. Under this constraint, the probability of a string that should be ranked highest can only be increased if the probabilities of other strings on the same list are decreased.

Most recently, Carter and Monz (2010) train a discriminative reranker for machine translation n-best lists with syntactic features taken from Collins and Koo (2005) who rerank parse trees. An application to error detection would require lists of one grammatical sentence and $n-1$ ungrammatical sentences, either related to the grammatical sentence like in the candidate correction approach (Section 2.2.4 of Chapter 2) or possibly unrelated. To the best of our knowledge, this has not been tried yet.

However, in a broader sense of the term *discriminative*, any classifier trained to distinguish two classes can be described as being discriminative. For example, Okanohara and Tsujii (2007) call their linear classifier that assigns negative scores to ungrammatical

[19]This does not undermine the APP/EPP method. The correlation between likelihood and grammaticality is extremely small, too small for a naive approach, but present. The APP/EPP method is designed to exploit the correlation despite its small size.

and positive scores to grammatical sentences a discriminative language model. Essentially, they take an online learning algorithm and train on a large corpus of BNC sentences and artificial ungrammatical sentences. Considerable effort is made to tackle efficiency issues which arise because individual n-grams are included as features. To us, an interesting aspect of their work is that negative training data is generated by an ordinary n-gram language model. It is assumed that either all possible errors are covered by the output of the generator or that uncovered errors can easily be detected as ungrammatical by the n-gram language model that generated the negative training data.[20]

The log-difference of two language models is applied to speech recognition by Stolcke et al. (2000) who call this approach "anti-language model". One n-gram language model is trained on reference transcriptions as usual while the second model is trained on 500-best recognition hypotheses. The anti-language model is intended as a corrective that penalises strings that are likely to be erroneous and is combined with the first language model in a log-linear model. The optimal weight of the anti-language model for minimal word error rate is negative, as expected. Their work can be described as a variant of the APP/EPP method proposed in this chapter as it implements an EPP model with the anti-language model and calculates APP/EPP ratios where the APP model is the language model trained on reference transcriptions. However, probabilistic parsing is not used in their approach.

4.4.4 Machine Translation

Machine translation is another area where probabilistic parsing and n-gram language models are combined in a log-linear model. Och and Ney (2002) do not impose restrictions on the scaling factors of their log-linear model, so the log-difference of two models can be expressed with the help of a negative factor. However, Och et al. (2004) find no improvement when adding parse probability features. Actually, they observe that the parser assigns higher parse probabilities to (imperfect) machine translation output than

[20] Recently, the idea of training with n-gram model output has been re-used: Post (2011) trains a linear classifier with features extracted from parse tree derivations obtained with a tree substitution grammar to discriminate between grammatical text and n-gram model output. Lavergne et al. (2011) also train classifiers to distinguish between LM output and natural text, but for a different purpose: to automatically detect web spam that should be excluded from a search index.

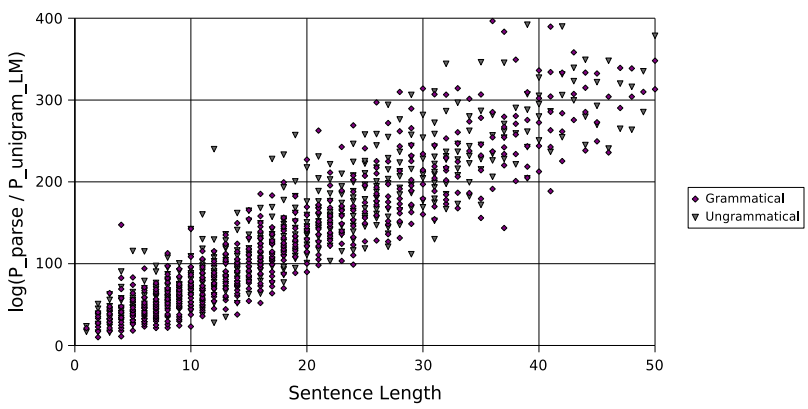


Figure 4.9: APP/EPP ratios for using SRILM's unigram language model for EPP and Charniak's parser for APP

to the respective reference translations produced by human translators. Och et al. (2004) speculate that this is caused by a preference for frequent tokens and try to eliminate lexical probabilities "by dividing the parser probability by the word unigram probability." Again, this can be seen as a simple APP/EPP model.[21] The idea is abandoned, though, because it "did not yield improvements" (Och et al., 2004, page 165, section 6.1).

Post and Gildea (2008) combine the probability of the best parse according to a treebank-induced lexicalised probabilistic parser and the probability output of an unsupervised dependency parser in order to select the output of a machine translation system. In other work combining multiple parsers for machine translation (Mutton et al., 2007; Zwarts and Dras, 2008) only one of the component systems provide a probability.[22]

4.4.5 Bayes' Decision Rule

We came across a formula resembling the decision rule APP/EPP $<$ C in the work of Jaakkola et al. (2000) and Jebara (2004) who combine discriminative and generative models with a "maximum entropy discrimination (MED) framework". Closer inspection

[21] For our purpose of error detection with a threshold C, such a model is not suitable though. As can be seen in Figure 4.9, the ratio of parse probability to word unigram probability strongly correlates with sentence length. Furthermore, the vertical variance in the graph shows that the probability ratio depends on at least one more factor apart from sentence length and grammaticality. Applying a constant threshold (or sentence length-dependent threshold) to this probability ratio cannot yield a useful classifier. See also Section 4.7.9 on future work.

[22] The other parser matrics used include the number of fragments (in a dependency parse) and the number of deletions necessary to make the input parsable (using a link parser which differs from a dependency parser in that links are undirected).

showed that the formula is a special form of Bayes' Decision Rule: in the usually stated general case, we have a set H of hypotheses or labels and want to find the hypothesis $\hat{h} \in H$ that has the highest probability given the data or observation D:

$$\hat{h} = \arg\max_{h \in H} \mathrm{P}(h|D) \tag{4.1}$$

$$= \arg\max_{h \in H} \frac{\mathrm{P}(D|h) \times \mathrm{P}(h)}{\mathrm{P}(D)} \tag{4.2}$$

$$= \arg\max_{h \in H} \mathrm{P}(D|h) \times \mathrm{P}(h) \tag{4.3}$$

In the case $H = \{h_1, h_2\}$, this simplifies to a comparison of two terms:

$$\hat{h} = \begin{cases} h_1 & \text{if} \quad \mathrm{P}(D|h_2) \times \mathrm{P}(h_2) < \mathrm{P}(D|h_1) \times \mathrm{P}(h_1) \\ h_2 & \text{otherwise} \end{cases} \tag{4.4}$$

This is turn can be re-written as

$$= \begin{cases} h_1 & \text{if} \quad \mathrm{P}(D|h_2)/\mathrm{P}(D|h_1) < \mathrm{P}(h_1)/\mathrm{P}(h_2) \\ h_2 & \text{otherwise} \end{cases} \tag{4.5}$$

which resembles our APP/EPP model if we set APP = $\mathrm{P}(D|h_2)$, EPP = $\mathrm{P}(D|h_1)$ and C = $\mathrm{P}(h_1)/\mathrm{P}(h_2)$ and interpret h_1 to mean that the sentence D is ungrammatical and h_2 to mean that the sentence D is grammatical. However, in our APP/EPP model, neither APP nor EPP are probability distributions over all sentences. The probability mass of APP is below 1 as it only accounts for the best (most likely) parse for each sentence. An EPP model, on the other hand, assigns the parse probability of a grammatical sentence to any ungrammatical sentences, effectively multiplying the probability mass many times. In the case of a k-nearest neighbour model, it is easy to see that the EPP model is not a probability model: the probabilities assigned to ungrammatical sentences have a lower bound given by the smallest probability in the training set and there is an infinite number of ungrammatical sentences. Therefore, such an EPP model will accumulate an infinite probability mass. For the Bayes's rule, the two models would have to be generative models for grammatical and ungrammatical sentences respectively.

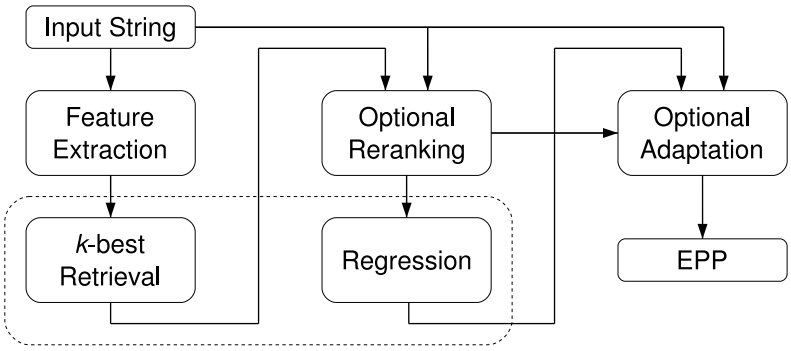


Figure 4.10: Components of our EPP model

4.5 A Model to Estimate Parse Probabilities

The EPP model is central to the APP/EPP method. Its role is to accurately predict the parse probability of grammatical sentences while overestimating parse probabilities of ungrammatical input due to a lower sensitivity to grammatical errors than the APP model. In this section, we describe the EPP model that we chose for testing the APP/EPP method on our data set.

Figure 4.10 shows the components of our EPP model. The dashed box highlights the components of the core prediction model that is discussed in Section 4.5.1. Sections 4.5.2 to 4.5.7 go through the features of input sentences the core model uses to make its predictions. The core model is modified by two optional components that (from right to left in Figure 4.10) use language modelling and n-gram sentence similarity and are described in Sections 4.5.8 and 4.5.9 respectively.

An additional EPP model is evaluated outside this chapter: Section 5.4 presents an EPP model that is not based on the idea of learning from the parse probability of grammatical sentences. Instead, the model parses the input with a probabilistic error grammar.

4.5.1 k-Nearest Neighbour Learning

The k-nearest neighbour (k-NN) machine learning method retrieves the k items that are most similar to the input item from the training data and uses these to make a prediction for the input item. Often, (dis-)similarity is measured using the Euclidean distance in vector space and the target value is simply the average of the retrieved items (or the most

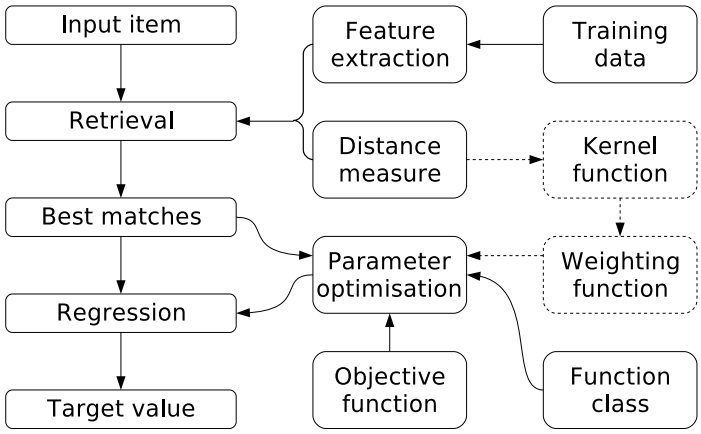


Figure 4.11: Components of the k-nearest neighbour machine learning method (optional components are shown with dashed lines)

frequent class in the case of classification problems). The k-NN method is appealing due to its simplicity, easy implementation and single parameter k. Training on large data sets is fast because all that needs to be done is to fill a multi-dimensional search structure with the training items.[23] Also, Daelemans et al. (1999) provide evidence that k-NN is suitable for many NLP tasks. It can handle local exceptions well because it does not attempt to fit all data into a global model.

However, as always, the devil is in the detail. If the training items are mapped into vector space without much thought, the Euclidean distance will probably fail to identify the training items best suitable for predicting the target value of the input item because large differences in less relevant features might dominate small but relevant differences in other features. In the experiments reported in Section 4.6, each feature is mapped to its own axis and the axes are scaled to adjust the weights of each feature. Statistical independence of the features is assumed in order to be able to optimise the vector space representation in this manner.

The second detail that can make the k-NN method more complex (but also more powerful) is how the retrieved k training items are used to make a prediction of the target value. The idea is to build a simple local model that can be computed quickly from the k items. Figure 4.11 shows required and optional components for a k-NN learner.

[23] In contrast, the duration of experiments with support vector machines grows quickly with the size of the training data.

The function class determines what type of functions are considered for the local model. The simplest possible class is the class of all constant functions. Parameter optimisation searches for the best function in the function class to describe the k items and is then used to make a prediction for the input item. An optimisation criterion (or objective function) defines what we mean by best function. Usually, this is the mean square error. For constant functions and using the mean square error criterion, the best constant can be calculated as the arithmetic mean of the k target values. However, we will optimise the mean square error of logarithmic values and therefore we have to use the geometric mean instead of the arithmetic mean. Sometimes, the function class is extended to linear or even higher order polynomial functions. The coefficients can then be found with linear regression. The optimisation criterion for finding the best local model does not have to treat all k items equally. Mitchell (1997)'s presentation of the k-NN method suggests that weighting of the k items with (a function of) their distance is very popular.

Thirdly, there are efficiency issues if the number of training items is very large. Mitchell (1997) mentions kd-trees as an indexing structure. Mount (2005) points out that exact methods for finding the k nearest neighbours often have runtime or space requirements that grow exponentially with the number of dimensions. Therefore, we use an approximate nearest neighbour (ANN) algorithm[24] in our experiments.

4.5.2 Sentence Length

The first two features are very simple: the length of the sentence measured in tokens (including words and punctuation) and in characters. We have seen in Section 4.1.1 that sentence length has a strong effect on parse probabilities. Plotting logarithmic parse probabilities over number of tokens as in Figure 4.12 shows that the effect of sentence length is multiple orders of magnitude bigger than the effect of grammaticality observed in Section 4.2 (1,000 vs. 10 points on the logarithmic scale). Therefore, we can expect sentence length measured in tokens to be important in our EPP model. Since the number of characters can also easily be extracted from the corpus, we include this length measurement as well. Together with the number of tokens, it indirectly provides the average token

[24] `http://www.cs.umd.edu/~mount/ANN/` webpage by David M. Mount and Sunil Arya accessed in 2006, 2008 and April 2009.

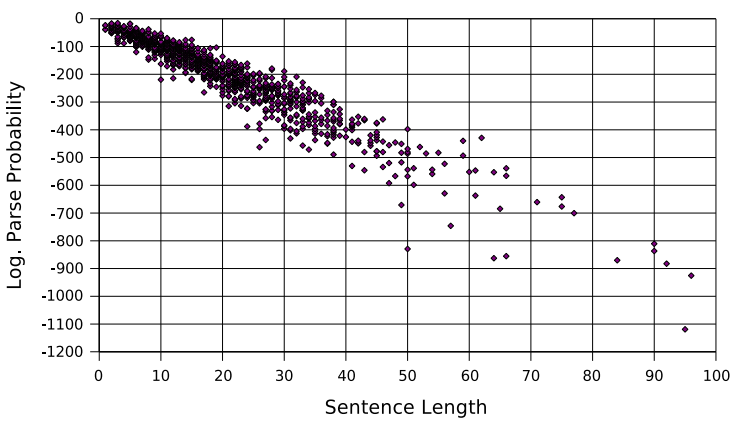


Figure 4.12: Scatter plot of logarithmic parse probability over sentence length measured in tokens for 1,000 random BNC sentences

length to the EPP model which is an indicator for whether the sentence is dominated by function words and simple content words, or by long words that are very specific, rare and consequently may lower the parse probability.

4.5.3 Tree Height and Number of Nodes

Features of the actual parse result can be expected to improve the predictions considerably. However, we have to be careful to only use features that do not expose the lower parse probability of ungrammatical sentences to the computation of the EPP. Figure 4.13 shows that the number of non-terminal nodes is often not affected by correcting a sentence, in particular if the sentence length is not changed: 73.2% of same-length corrections did not change the number of non-terminal nodes. This means that lower parse probabilities of ungrammatical sentences are often caused by the application of less likely rules as opposed to a higher number of rules. Therefore, it is safe to add the number of nodes of the parse tree and a related number, the height of the tree, as features to the k-NN model.

Figure 4.14 shows how features of parser output are integrated into k-NN learning. In the example, only sentence length and tree height are used. The sentence length 4 is directly extracted from the input sentence. In addition, the sentence is parsed. However, we ignore most of the parser's output. Of course, if we were just trying to build a good predictor, this would not make any sense. It is important to keep in mind that the EPP model is supposed to overestimate the parse probability for ungrammatical sentences,

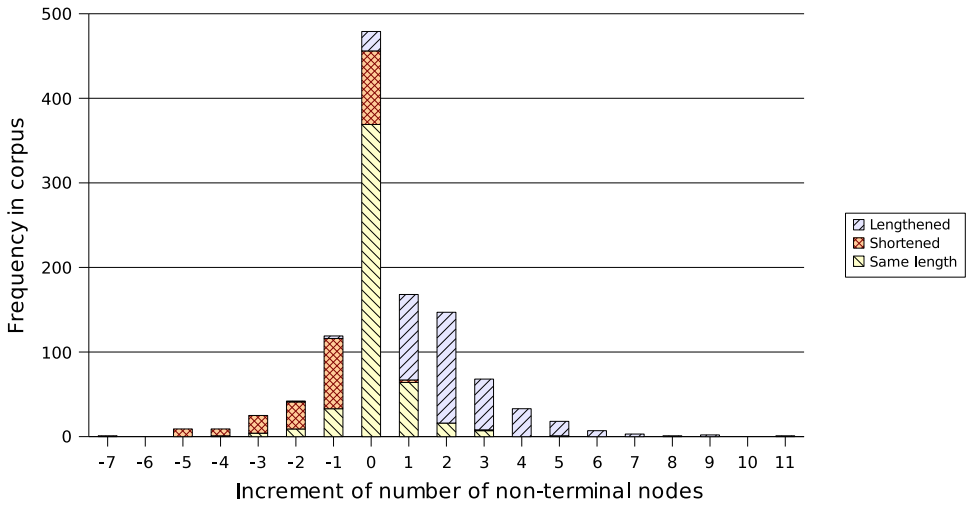


Figure 4.13: Effect of correcting erroneous sentences (Foster corpus) on the number of non-terminal nodes of the best parse. Each bar is broken down by whether and how the correction changed the sentence length in tokens.

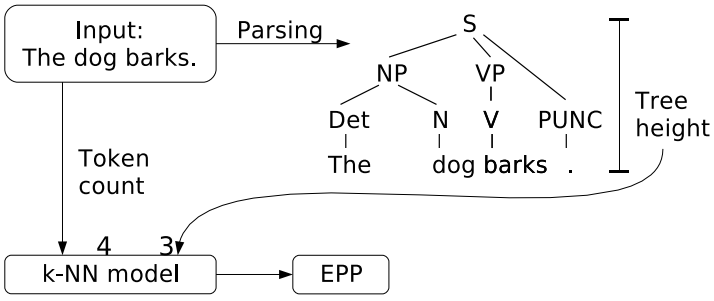


Figure 4.14: Adding tree height to the k-NN model

Trigram	Frequency	Per Sentence	Rank	Early Experiments
'the'	1,154,184	2.82	2	Yes
' , '	678,322	1.66	4	Yes
' of'	482,565	1.18	9	Yes
'and'	395,244	0.96	15	—
'ed '	360,617	0.88	18	—
'ing'	346,876	0.85	20	Yes
'for'	230,051	0.56	42	Yes
' a '	217,685	0.53	44	—
'ope'	132,121	0.32	90	Yes

Table 4.2: Character trigrams chosen as k-NN features and their frequency in the EuroParl data

effectively ignoring ungrammatical aspects of the input sentence, if present. Therefore, we only use generic features of the parse tree. In Figure 4.14 this is the tree height 3. The k training sentences closest to the query point (4,3) are then retrieved by the k-NN model and used to estimate the parse probability. The feature "number of non-terminal nodes" is added in the same way.

4.5.4 Character Trigrams

Table 4.1 in Section 4.1.1 shows that lexical choice (replacing the noun "shares" with the noun "radiators") has a strong effect on parse probabilities. As a first step towards capturing lexical information, the frequencies of nine manually chosen character trigrams are included in the set of features.[25] Trigrams are easy to count and they can cover short function words, suffixes and arbitrary substrings. Table 4.2 lists the trigrams that were picked. The trigrams were chosen among the most frequent trigrams in a subset of the EuroParl corpus, used in early experiments (see Appendix B). The rank by frequency in this corpus is given in the 4th column of Table 4.2. Among the top 10 trigrams, the 2nd, 4th and 9th trigrams ('the', ' , ' and ' of') seem promising for predicting parse probabilities. Additional trigrams were selected to include a conjunction, verb suffixes, another preposition and determiner, and, as a basis for comparison, an unmarked trigram that appears in words such as 'open', 'property' and 'hope'. The right-most column marks

[25] A large number of trigram features (as used in discriminative language models) would significantly slow down k-NN retrieval.

the six trigrams that were used in the early experiments described in Appendix B. For the final experiments in this chapter, we decided to include three more trigrams based on the observation that trigram features improved the initial EPP models considerably. The features added to the k-NN method are the frequencies of the nine trigrams in the input sentence. Average numbers for the EuroParl corpus used in early experiments are given in the middle column of Table 4.2.

4.5.5 POS Frequencies

While sentence length measured in tokens clearly is one of the most important factors affecting parse probabilities,[26] we observed that adding optional punctuation or quotes to a sentence decreases its parse probability only mildly compared to adding content words, for example adjectives, even if the words we add are frequent words which one would therefore naively expect to have high probabilities. Therefore, we expect the number of (non-word) symbols appearing in a sentence to be a useful feature. Seeing that, for example, a pair of quotes around a noun phrase does not lower parse probabilities much while they reinforce the bracketing of the parse, we speculate that function words behave similarly, i. e. that their effect on parse probability is small because there is only a small number of possible structural configurations and terminal expansions to choose from, meaning that probability mass will concentrate on a few choices. In contrast, content words will require terminal rules with a very wide probability distribution. The number of function words and the number of content words appearing in a sentence are therefore also added to the set of features. In general, the frequency histogram over all POS tags may be useful to select sentences "of the same type" as the input sentence. However, since the k-NN method cannot deal well with large feature sets, we have to collapse POS tags into classes. In the following, we describe the procedure.

Training and test sentences are POS tagged with the pre-terminal symbols of the best parse trees chosen by Charniak's parser.[27] We reduce the tag set by collapsing

[26] We expect only sentence length measured in characters to be a competitor for importance. Since Section 4.5.2 gives a good reason to include both length measures as features, we see no need to determine which one correlates the most with parse probability.

[27] We fall back to the IMS Tree Tagger (Schmid, 1994) if parsing fails or the sentence is longer than 100 tokens. Small differences in how verbs and punctuation are tagged have been accounted for.

proper nouns to nouns and by ignoring number, and, in the case of adjectives, degree of comparison. Verbs and auxiliaries are reduced to V. For each sentence we extract a frequency vector recording the number of times each of the nine tags NN, IN, DT, full-stop, comma, JJ, CC, TO and V, other symbols, other content words and other function words appears in the sentence. In addition to these 12 POS frequency features, we add a 13th and 14th POS feature: the frequency ratio of content words to function words and the number of consecutive duplicates. The latter feature is expected to expose many extra word errors and is optionally included in early experiments (see Appendix B) to test its (presumably negative) effect on the EPP model and the APP/EPP classifier.

4.5.6 N-gram Language Model Probability

As shown in Section 4.1.1, lexical items have a huge impact on parse probabilities. We add n-gram language model (LM) probabilities as features in order to retrieve reference sentences in the k-NN method with lexical probabilities similar to the input item. We focus on small n ($n = 1, ..., 3$) because higher order LMs are likely to reflect grammaticality.

A unigram LM assumes independence of each token and therefore simply multiplies the probability of each token that appears in the string. The individual probabilities are estimated from frequencies in a corpus relative to the total number of tokens while reserving some probability mass for unseen tokens. In higher order LMs, probabilities are conditioned on the preceding token(s). We use the SRILM toolkit[28] (Stolcke, 2002) with default options, i. e. Gold-Turing smoothing, and the same corpus the parser has been trained on: the Penn Treebank WSJ sections 2 to 21. We extract seven features for each sentence: unigram, bigram and trigram language model probabilities, the respective information-theoretic perplexity values[29] and the number of out-of-vocabulary tokens, i. e. tokens that were not seen in the training data.[30]

Sometimes n-gram language models are normalised by sentence length, but we expect that raw generative probabilities will perform better in EPP models because actual parse probabilities (APP) are also not normalised by sentence length. Figure 4.15 compares

[28] `http://www.speech.sri.com/projects/srilm/` accessed 2009-04-18

[29] The SRILM documentation does not give any details, for example whether the perplexity is normalised by sentence length.

[30] SRILM command `ngram -order 3 -lm` *LM-file* `-ppl` *input-file* `-debug 1`

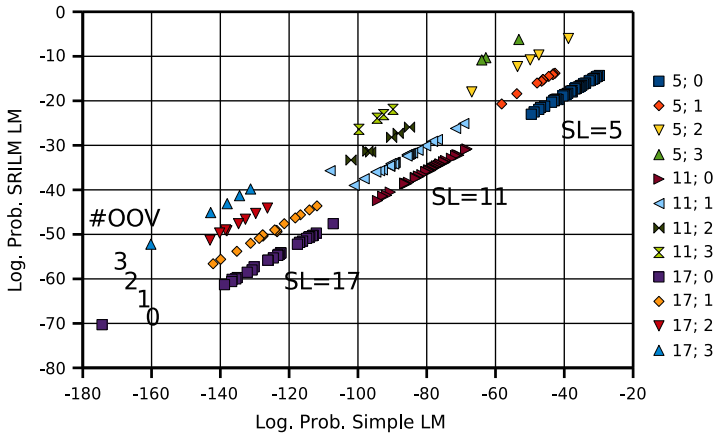


Figure 4.15: Effect of smoothing on unigram language model probabilities: SRILM's Gold-Turing smoothing vs. naive smoothing on a subset of 2,000 BNC sentences with sentence length 5, 11 or 17 and number of out-of-vocabulary (OOV) tokens up to 3.

the SRILM output to an LM we implemented for early experiments (see Appendix B) that uses a naive simple smoothing method (allocating the probability mass $1/(N+1)$ for out-of-vocabulary tokens). The fact that points for sentences with the same number of out-of-vocabulary (OOV) tokens fall on one line confirms that SRILM does not normalise the probabilities by sentence length. The figure also shows the effect of OOV tokens and smoothing: SRILM's smoothing boosts the probability of sentences containing OOVs while the naive smoothing used in our early experiments penalises such sentences. This observation shows that the number of OOVs is important if we want to predict the probability output of one model with the output of another model that uses different smoothing. Therefore, we add the number of OOVs as a k-NN feature so that the k-NN model can learn to adjust probabilities depending on the number of OOVs.

In addition to the SRILM features, we also keep three models as features that we used in early experiments: the simple unigram LM mentioned above, the linear interpolation of a unigram and a bigram LM with $\lambda = 0.5$, and finally a unigram LM in which tokens are annotated by their POS. An LM on POS tags alone is not considered because POS information is already present through the features of Section 4.5.5.

4.5.7 Terminal Rule Probability

A comparison between how LM probabilities for POS-tagged words are calculated and the probabilities of terminal rules of a basic PCFG shows that there is a difference in the way the probabilities are conditioned: for example, ignoring smoothing, the POS-tagged unigram LM probability of the token `the`, tagged `DT`, would be P(`the` tagged `DT`) = count(`the` tagged `DT`)/N where N is the total number of tokens while the rule `DT->the` would have the probability P(`the` | `DT`) = count(`the` tagged `DT`)/count(`DT`). A model that accounts for this, i. e. a model that accurately describes the contribution of lexical choice to the parse probabilities of a basic PCFG, might be more helpful than ordinary LMs as a feature in an EPP model for Charniak's parser because the more complex probability model of the parser is still related to basic PCFG models.

Ideally, we would like to model the terminal rule probabilities of Charniak's parser which is a head-lexicalised probabilistic parser that conditions the probability of terminal symbols on the pre-terminal symbol and its mother's head. For example, the probability of the rule r = `DT->the` within the fragment (`NP` (`DT the`) (`NN house`)) is modelled as P(`the` | cat(r) = `DT` and head(mother(r)) = `house`). We would like to use the product of the probabilities of all terminal rules appearing in a parse tree as a feature in our k-NN model. However, the output of Charniak's parser is not head-annotated. Therefore, we drop the head-lexicalisation part of the conditioning, i. e. we revert to a PCFG model.[31]

4.5.8 Factoring out Lexical Probabilities

Our EPP model is not a pure k-NN model as we extend the k-NN model with two components. We address the top-right adaptation component of Figure 4.10 (page 99) first because it is related to the LM and terminal rule probability features discussed above.

If we could accurately calculate the probability of terminal rules as described in Section 4.5.7, we would only need predictions for the probability of the upper half of the parse tree (above the pre-terminals) in order to output the product of the probabilities of the two halves as an EPP. This would relieve the k-NN model of the burden of predicting

[31] An alternative approach would be to apply head-finding heuristics to the parser output. Depending on how often the heads found this way coincide with the parser's decision, a model that is conditioned on these heads may be more accurate (see Section 4.7.10 on future work).

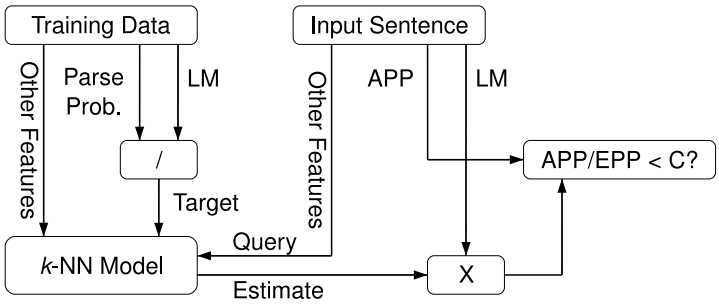


Figure 4.16: Factoring out LM probabilities from the k-NN model

lexical probabilities and training data would become less sparse because lexical features can be ignored or at least weighted low in the k-NN model, reducing noise. Therefore, this method may lead to better overall predictions of parse probabilities by the EPP model. We propose to apply this scheme with the terminal rule probabilities of Section 4.5.7 that are based on a PCFG approximation and also with the language models of Section 4.5.6 as alternative models of the lexical contribution to the parse probability of a sentence.

Figure 4.16 illustrates the idea. The calculation of EPP values is split into two factors (incoming arrows of the box marked with a multiplication sign): one factor is given by the LM or terminal rule model and the other factor is provided by a new k-NN model. Training data for this new model is generated by dividing actual parse probabilities by the first factor (left side of Figure 4.16). For example, consider a k-NN model with $k = 1$ and with a single training item that has parse probability p_1 and LM probability q_1. We train our k-NN model with the target value p_1/q_1. To calculate an EPP value for an evaluation item, we calculate its LM probability q_2, retrieve the prediction p_1/q_1 from the k-NN model and output $q_2 \times p_1/q_1$. Another way of describing this is that we use the k-NN method to predict what factor is needed to adjust an LM probability to the parse probability.

How much has to be changed in order to evaluate this method, i. e. to obtain values for mean square error and classification accuracy? The square error of APP $= p_2$ and EPP $= q_2 \times p_1/q_1$ on logarithmic scale can be rewritten in terms of the logarithmic square error

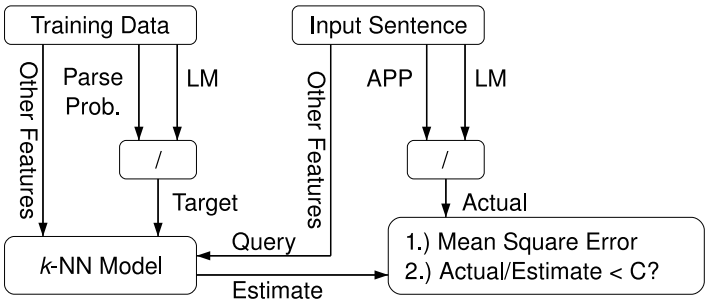


Figure 4.17: Experimental setup for factoring out LM probabilities

on the intermediate p/q terms:

$$\mathrm{E} = [\ln(q_2 \times p_1/q_1) - \ln(p_2)]^2 \tag{4.6}$$

$$= [\ln(q_2) + \ln(p_1) - \ln(q_1) - \ln(p_2)]^2 \tag{4.7}$$

$$= [\ln(p_1) - \ln(q_1) - (\ln(p_2) - \ln(q_2))]^2 \tag{4.8}$$

$$= [\ln(p_1/q_1) - \ln(p_2/q_2)]^2 \tag{4.9}$$

This means that we can get the mean square error for this method without any changes to the experimental setup except for replacing p by p/q, i. e. APP by APP/LM, in both training and test data. The ratio

$$\mathrm{APP/EPP} = p_2/\frac{q_2 \times p_1}{q_1} = \frac{p_2 \times q_1}{p_1 \times q_2} = \frac{p_2}{q_2} \times \frac{q_1}{p_1} = \frac{p_2}{q_2}/\frac{p_1}{q_1} = \text{actual/estimate} \tag{4.10}$$

can also be calculated as the ratios of the p/q values. The classifier that compares this value to a constant C does not require any adaptation, either. The revised architecture is shown in Figure 4.17.

Unfortunately, we do not have any accurate model of the lexical contribution to parse probabilities. However, the seven models presented in Sections 4.5.6 and 4.5.7 describe important aspects and the target model may lay somewhere in between. Therefore, we learn a log-linear combination of these models from training data, i. e. the LM value used in the factoring out method is given by

$$\mathrm{LM} = \mathrm{LM}_1^{w_1} \times \mathrm{LM}_2^{w_2} \times ... \times \mathrm{LM}_7^{w_7} \tag{4.11}$$

where LM_1 to LM_7 are the seven models (naive unigram, PCFG terminal rules, interpolated LM, POS-tagged unigram, SRILM unigram, bigram and trigram) and w_1 to w_7 are the respective weights. If all weights are 0, LM will be 1 and we fall back to the EPP model purely based on k-NN.

4.5.9 Re-ranking of k-NN results with BLEU Score String Similarity

The reference sentences returned by the k-NN method have very little in common with the input sentence on the surface level. Of course, the similarities lie in the features that we selected in Sections 4.5.2 to 4.5.7. It has therefore been suggested to experiment with string similarity measures, for example BLEU (Papineni et al., 2002) which is familiar from machine translation evaluation, to retrieve sentences that are more similar to the input sentence to the human eye. Unfortunately, implementing the BLEU measure as a Euclidean distance in vector space is not trivial. Also, evaluating the measure on all training items for each input sentence is not feasible due to the size of our corpus. As an approximate solution, we retrieve k_1 nearest neighbours with our previous vector space features, rerank the results with the final similarity measure and then pick the top k_2 sentences. Experiments will determine how big k_1 needs to be.

BLEU measures the weighted geometric mean over n-gram precision, i. e. the fraction of n-grams of the input sentence that are also present in the reference sentence, for $n = 1, 2, 3, 4$ with the addition of a length penalty.[32] We derive BLEU variants replacing tokens by POS (original and reduced tag sets) and frequency class (order of magnitude of frequency of token in the BNC, parameters chosen experimentally). We also test linear combinations of these measures and the logarithm of our APP/EPP grammaticality measure.

[32] In machine translation, the penalty is called a brevity penalty because the input sentence is variable and precision would favour short input sentences. In our case, we want to select reference sentences for a fixed input sentence and have to avoid selecting very long sentences that happen to cover most of our input's n-grams. The penalty formula does not change. This is only a question of naming.

4.6 Experiments with BNC Data

This section presents the experiments conducted with the same BNC-based artificial error data we use in Chapters 5 to 7. Earlier experiments conducted in 2004 and 2005 use different data sets. As these early experiments have influenced the experiments reported in Sections 4.6.2 to 4.6.3 below, we summarise the main findings from the early (2004/5) experiments in Section 4.6.1. All details, including the various corpora used for training and testing and the reasons for switching between them, can be found in Appendix B. Section 4.6.2 describes the cross-validation setup, evaluation measures and the parameter optimisation method of the new experiments. We present results in Section 4.6.3.

4.6.1 Findings of Early Experiments (2004–2005)

Our early experiments described in Appendix B show that the APP/EPP method works in principle. Due to the intervening development of both method and evaluation criteria, hard conclusions are difficult to draw. In the following, we highlight the most important findings and point to the relevant sections in Appendix B.

Evaluation Metrics and Error Density

Precision, recall and f-score are difficult to interpret for the error detection task as recall and f-score depend on the error density in the test set which is highly variable in real applications. In addition, it is unsatisfactory that a trivial classifier that always assigns one class can achieve an f-score of $2/3$ for an error density of 0.5 while an unbiased classifier can have a lower f-score even if it is better than coin flipping (Appendix Section B.5). This finding motivated the development of the two-dimensional accuracy metric introduced in Chapter 3.

Character Trigram Feature

The first 6 character trigram features described in Section 4.5.4 can improve the EPP model considerably (in terms of the mean square error on grammatical test data) if the frequency values are normalised by sentence length in characters (Appendix Section B.4.1).

Language Model and Terminal Rule Probabilities

The first of the early experiments (Appendix Sections B.1 to B.3) were conducted with LoPar (Schmid, 2000) and an English PCFG which allows us to extract exact probabilities for the terminal rules. Factoring out these probabilities (as described in Section 4.5.8) yields a large improvement of the EPP model in terms of mean square error and accuracy on grammatical data over the baseline model only using sentence length and tree height (Appendix Section B.2.4).

With Charniak's parser, Appendix Section B.4.1 also shows large improvements when we add PTB-trained terminal rule probabilities or a token unigram language model to the k-NN feature set (Sections 4.5.6 and 4.5.7). The factoring out method also shows large improvements on its own, but does not perform as well as using the probabilities as k-NN features. Combining all three methods (using terminal rule probabilities as features, using LM probabilities as features and factoring out unigram LM probabilities) does not significantly improve results over combining the first two features (Appendix Section B.4.1).

k-Nearest Neighbour Parameters

Experimental results in Appendix Section B.3 suggest that it is counterproductive to apply weighting functions (Section 4.5.1) because the density of data points varies by multiple order of magnitudes (consider for example the distribution by sentence length) and the target values are very noisy. Also, this means that we need high values for k and large training corpora. Applying linear regression (Section 4.5.1) to the k retrieved data items only improved results marginally.

Reranking with BLEU Score

The standard configuration of the BLEU measure (Section 4.5.9) in machine translation is to use n-grams up to $n = 4$, but we often do not find any 4-gram matches between retrieved sentences and the test sentence, i. e. all scores are 0. A small improvement of the mean square error is observed if we rerank with a modified BLEU measure that is restricted to unigrams and bigrams only and large value of k_1, the number of sentences retrieved with the k_1-NN model. It is interesting though that results do not degrade. BLEU

somehow avoids moving unsuitable reference sentences to the top of the list (Appendix Section B.7). Due to the computational costs, we decided to not use BLEU reranking in the new experiments reported in Sections 4.6.2 and 4.6.3 below.

4.6.2 Recent Experiments: Experimental Setup

Data Sets

In all our experiments, we use 10-fold cross-validation, i. e. we split the BNC-based artificial error corpus (see Section 3.3 of Chapter 3) into 10 parts and repeat the experiment 10 times with a different test set each. Within a cross-validation run, 8 sets are used as training data or reference corpus for the k-NN model. The 9th set is split into 2 development sets: 20% are used to tune the parameters of the k-NN model. 75% are occasionally used to monitor the progress of the parameter training and to detect over-fitting. The remaining 5% are not (yet) used.

Evaluation Measures

Three evaluation measures are employed both for final evaluation and as objective functions during parameter optimisation:

1. mean square error of predictions of logarithmic parse probabilities for grammatical test data,
2. the σ-gap defined in Section 4.3.3, and
3. accuracy of the classifier APP/EPP $<$ C with C chosen such that the accuracy is identical on grammatical and ungrammatical test data.

Parameter Optimisation

For the experiments reported in this chapter, we do not repeat the evaluation of a sequence of EPP models with increasing number of features as in Appendix B, but directly train the weights of all 37 features,[33] the seven weights for the "factoring out" method (Sec-

[33] We exclude the "POS duplicate count" feature (Section 4.5.5)

tion 4.5.8), and normalisation exponents for character trigram frequencies (Section 4.5.4) and POS frequencies (Section 4.5.5), and finally the value for k of the k-NN model.

We start with randomly chosen parameters and hill-climb towards better parameters, i. e. we iteratively try new parameters within a certain radius from the currently best parameters and move to them if they are better, where better is defined by the objective function which is one of the three evaluation measures listed above.

Search Radius

With a high number of dimensions, it is relevant whether we choose perturbation vectors from a hypercube or hypersphere. The length of the diagonal of an n-dimensional unit hypercube is $\sqrt{n}$, hence vectors chosen from a hypercube may be too long, or, in other words, change the parameters too much. On the other hand, random vectors from a hypersphere will often have very small coordinates as the volume of a hypersphere relative to the enclosing hypercube goes to 0 with $n \to \inf$. The latter fact also makes it difficult to efficiently generate uniformly distributed random vectors from a hypersphere. We address these problems by randomly generating perturbation vectors from four different distributions (probability in brackets):

- the uniform distribution within the hypercube with coordinates from -1 to 1 (8%),
- the above hypercube radially compressed into the hypersphere with radius 1, i. e. the density increases towards the diagonals (12%),
- a non-uniform distribution within the above hypercube which sets most coordinates to 0 and some coordinates uniformly between -1 and 1; exactly one coordinate is picked with 50% probability, two with 25% probability, three with 12.5% etc. ; the order of coordinates is random (40%),
- a non-uniform distribution within the hypersphere with radius 1; the first coordinate is chosen uniformly from the range -1 to 1; the range for the next coordinate depends on the length of the vector so far and is set such that the hypersphere cannot be left; note that this procedure is fairly likely to produce a point close to the surface of the hypersphere; again, the coordinates are randomly reordered (15%), and finally

- a distribution that we obtain by drawing a random vector each from the above four distributions and returning a weighted average (25%).

The perturbation vectors with coordinates between -1 and 1 are scaled to accommodate the hill-climb as described below.

Multiple Shots and Search Scale Decay

Hill-climbing is a very simply optimisation method but with it comes the risk of getting stuck in a local optimum. The chances of finding good parameters can be improved by repeating the hill-climb with different initial parameters. Each repetition is called a shot. We run 20 shots for each cross-validation run and objective function. The first 10 shots use a constant search scale of 5, i. e. coordinates of perturbation vectors can be up to ± 5. In addition, we run 5 shots with linearly decreasing scale. The scales start with radii of 8, 4, 2, 1 and 0.5 respectively and decrease towards 0 over 1,000 steps. The final 5 shots start with the same search scales as the linearly decreasing shots, but the decay is exponential such that a scale of 0.01 is reached after 1,000 steps.

Parameter Scales

Weights are on a logarithmic scale so that doubling a weight always requires a hill-climbing step of the same length ($ln(2) \approx 0.69$) which does not depend on the absolute weight. k is on 1/10 scale so that a hill-climbing step of length 1 along the k coordinate increases or decreases k by 10.

Parameter Boundaries

We clip all parameters so that certain bounds cannot be exceeded. The weights are limited to the range from -7 to 6 on logarithmic scale which approximately corresponds to the range from 0.0009 to 403 which should be large enough to differentiate features and small enough to be able to recover from extreme weights encountered during the hill-climb. Weights of the factoring out method and the normalisation exponents are restricted to the range -1 to 2 as they are expected to be between 0 and 1. k is limited to values

between 1 and 80. If a vector component is clipped, its value will be set to the minimum or maximum of the range, whichever is closest.

Initial Parameter

All initialisation ranges are 50% of the width of the clipping ranges (see above) and centred. This means that we initialise the parameter vectors randomly within the following ranges: from -0.25 to 1.25 for factoring out and normalisation exponents, from -3.75 to 2.75 for logarithmic scaling factors and from 20.75 to 60.25 for k.

Fractional Parameter k

Fractional values of k are made meaningful with a stochastic process, for example $k = 10.2$ is taken to mean that 20% of test items are evaluated with $k = 11$ and 80% with $k = 10$. This method replaces plateaus interrupted by jumps with a continuous sequence of linear segments. We hope that this will enable the hill-climbing to optimise k even with search steps much smaller than 1.[34]

Computation

The optimisation runs on a computing cluster with 128 CPU cores where the evaluation of each parameter vector can be run in parallel. Since we have 10 cross-validation runs with 20 shots and 3 objective functions, parallelisation on the level of hill-climbing steps is sufficient. Each parameter evaluation can run as a sequential process. We run until no improvements are measurable or 1,000 steps are reached. We may also stop certain shots (across all cross-validations runs) if certain search scales or decay functions are inferior according to measurements on the first development set.

4.6.3 Recent Experiments: Results

In the following, we present observations from the optimisation process and results of the final model that results from this process.

[34] The trick will not work if the slope is too small in order to produce noticeable differences in the evaluation on the development set.

Init. Scale	Decay	0	50	100	200	400	800
0.5	linear	58.11%	59.04%	60.03%	60.30%	60.43%	60.45%
1	linear	55.83%	58.93%	59.86%	60.19%	60.27%	60.29%
2	linear	54.67%	58.69%	59.40%	60.37%	60.69%	61.04%
4	linear	57.35%	60.25%	60.34%	60.36%	60.39%	60.39%
8	linear	55.72%	58.67%	59.48%	60.42%	60.60%	60.76%
0.5	exponential	52.92%	56.87%	59.07%	59.67%	59.73%	59.73%
1	exponential	58.35%	59.94%	60.15%	60.26%	60.34%	60.34%
2	exponential	58.37%	60.05%	60.05%	60.60%	60.69%	60.71%
4	exponential	57.26%	60.22%	60.42%	60.67%	60.97%	61.01%
8	exponential	57.62%	60.49%	60.57%	60.85%	60.89%	60.93%
5	none	58.18%	60.44%	61.53%	61.58%	61.60%	61.91%
5	none	55.83%	60.46%	60.70%	60.95%	61.03%	61.20%
5	none	57.94%	60.27%	60.78%	61.34%	61.56%	61.56%
5	none	57.02%	59.96%	60.61%	61.53%	62.13%	62.17%
5	none	54.77%	59.28%	60.31%	61.01%	61.10%	61.35%

Table 4.3: Effect of search scale on hill-climbing: average accuracy over cross-validation runs at the start and after 50, 100, 200, 400 and 800 hill-climbing steps measured on the first development set; 5 of the 10 shots with constant scale are also shown for comparison

Effect of Search Scale on Hill-Climbing

Table 4.3 shows the optimisation progress for different search scales and decay functions in the example of using the third objective function measuring accuracy and defined in Section 4.6.2. With a linear decay function, the initial search scale has no clear effect on the optimisation. Within the first 100 hill-climbing steps, scale 4 is best, followed by scale 0.5. In contrast, exponential decay seems to benefit from larger initial search scales and gives slightly better results than linear decay. Surprisingly, best results are obtained with a constant search scale. From step 200 on, the first 5 of the 10 shots with constant search scale outperform all shots with scale decay. However, there is also variation within the cross-validation runs (not shown in Table 4.3). At step 200, an exponential decay shot of the 10th cross-validation run is the best of all 20 shots for this run.[35]

[35] At this stage it was decided to re-assign computation time to the shots with constant search scale that had been suspended when we implemented the decay functions but also to continue devoting some computation time to the decay shots.

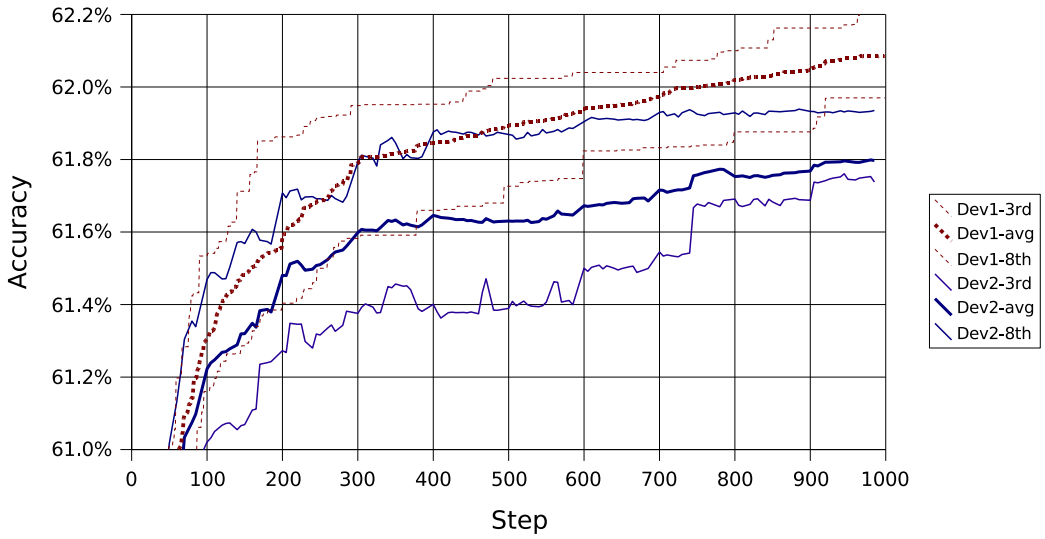


Figure 4.18: Optimisation progress: accuracy of the best shot measured on development data (dev1) and validated on unseen data (dev2)

Do we Over-fit the Training Data?

In machine learning, over-fitting occurs if a learner improves its predictions for training items at the price of making worse decisions for unseen test items. It is a possible stopping criterion (Mitchell, 1997, pp. 66–67, 108–111). Figure 4.18 plots accuracy on training data (development set 1) and unseen validation data (development set 2) over hill-climbing steps. Up to step 400, the average accuracy over all cross-validation runs increases for both data sets (bold solid and bold dashed lines). However, from step 400 to 540, accuracy slowly declines for unseen data while it continues rising on training data. If we had constantly monitored the progress, this would have been considered a sign of over-fitting meaning that training should be stopped. Luckily, we did not stop the parameter search, and results increase again from step 540.

The variance of cross-validation results is also indicated in Figure 4.18 with thin lines showing the third-best and 8th-best result of the 10 runs. The third-best curve for validation data is very flat from step 700, suggesting that we reached a limit and that the average only increases because other cross-validation runs are catching up. Continuing the optimisation beyond step 1,000 might be worthwhile.

Optimal Parameters

Optimised parameters are an experimental result as they are not set manually, but by the optimisation procedure. For each cross-validation run and objective function, we select the best shot according to the first development set at step 999. We analyse the parameters optimising for accuracy.

Tables 4.4 and 4.5 condense the 10 vectors of the cross-validation runs into minimum, average, maximum and standard deviation (square root of the variance). The minimum column has many 0 or almost 0 entries. Often, the average weight (next column) is large when the minimum is zero. This could mean that these features are redundant. A subset of them is needed, but different subsets are equally possible. The most obvious examples are the trigram and POS features counting commas which must correlate strongly.

The sentence length measured in tokens is much less important than expected. There is at least one cross-validation run that produced a model that works without this feature. It is notable that the sentence length measured in characters received a higher weight than the sentence length measured in tokens. Considering that the range of possible values of this feature is higher, its influence on the k-NN choices must be much higher than for the sentence length measured in tokens.

In general, feature weights have to be seen in relation to the values the feature can take. The language model weights seem low, but the logarithmic probabilities are a multiple of sentence length in tokens. (The factor often is between -3 and -2.) The values of the SRILM perplexity features (Section 4.5.6) are an order of magnitude bigger, but the feature weights are over two orders of magnitude smaller suggesting that these feature are less important. Still, interpreting individual weights is difficult (see Section 4.7.8).

The high variance of weights (rightmost column) means that we have not converged to a single parameter vector. Either, we found different local optima, or the objective function has a large plateau or a very flat optimum on which we perform a random walk.

Weights for Factoring out Lexical Probabilities

Table 4.5 shows the weights for the factoring out method (Section 4.5.8) and, for completeness, the normalisation exponents that are applied to the sentence length in tokens

Parameter	**Min**	**Avg**	**Max**	**Std. Dev.**
W sentence length in tokens	0.00	7.00	63.27	18.81
W sentence length in characters	0.00	19.71	94.51	27.37
W trigram 'the'	0.00	66.41	403.43	120.13
W trigram ' , '	0.00	120.65	403.43	181.53
W trigram ' of'	0.00	61.37	389.90	125.82
W trigram 'and'	0.00	22.41	136.85	43.24
W trigram 'ed '	0.00	17.02	149.19	44.14
W trigram 'ing'	0.11	95.15	403.43	157.26
W trigram 'for'	0.01	41.87	403.43	120.55
W trigram ' a '	0.00	53.93	364.42	106.22
W trigram 'ope'	0.00	40.20	326.07	97.17
W POS NN	0.00	68.66	383.75	117.36
W POS IN	0.03	58.86	333.75	103.55
W POS DT	0.00	1.75	9.88	2.96
W POS full-stop	0.00	317.08	403.43	123.62
W POS comma	0.00	43.72	382.91	114.03
W POS JJ	0.00	76.23	384.52	125.78
W POS CC	0.00	120.25	403.43	154.77
W POS TO	0.00	30.84	232.43	70.60
W POS verb	0.00	25.16	124.16	38.20
W POS other symbol	0.00	87.73	350.93	135.08
W POS other open class	0.00	60.83	310.69	119.39
W POS other closed class	0.00	154.01	403.43	163.27
W POS open/closed ratio	0.00	4.59	27.46	8.50
W naive unigram LM	0.00	25.14	208.62	61.61
W terminal rule probability	0.00	2.15	8.57	2.76
W interpolated LM	0.00	5.04	37.66	11.18
W POS-tagged LM	0.00	1.15	7.17	2.12
W number of nodes	0.00	11.11	46.10	17.18
W tree height	0.00	21.09	131.50	40.89
W SRILM unigram LM	0.00	6.80	50.94	14.88
W SRILM bigram LM	0.00	5.68	29.82	9.35
W SRILM trigram LM	0.00	5.20	14.54	5.90
W SRILM unigram perplexity	0.00	0.40	2.48	0.81
W SRILM bigram perplexity	0.00	0.06	0.49	0.14
W SRILM trigram perplexity	0.00	0.04	0.35	0.10
W number of unknown tokens	0.00	96.87	403.43	123.93
k	33.27	61.46	81.00	17.29

Table 4.4: Statistics for the weights (W) and parameter k of the best shots of the 10 cross-validations runs optimising accuracy

Parameter	Min	Avg	Max	Std. Dev.
FO naive unigram LM	-1.00	0.75	1.87	0.91
FO terminal rule probability	0.45	1.08	2.00	0.51
FO interpolated LM	-0.20	0.02	0.63	0.23
FO POS-tagged unigram LM	-0.10	0.52	1.93	0.59
FO SRILM unigram LM	-1.00	-0.48	1.79	0.86
FO SRILM bigram LM	-1.00	0.20	1.30	0.83
FO SRILM trigram LM	-0.98	-0.09	1.49	0.82
NE character trigrams	0.00	0.55	2.00	0.63
NE POS frequencies	0.00	0.51	1.16	0.49

Table 4.5: Exponents for factoring out (FO) and normalisation (NE) of the best shots of the 10 cross-validations runs optimising accuracy

and characters before POS and character trigram frequencies (features described in Sections 4.5.4 and 4.5.5) are normalised. Weights outside the expected range from 0 to 1 have often been chosen by the optimisation procedure. Negative weights mean that the parse probability is not divided but multiplied by the respective feature value. This does not necessarily mean that the final value will be smaller as there are seven factors in total. The lower range boundary -1 is often reached, suggesting that the weight constraints should be relaxed further.

Evaluation on Test Data

In each cross-validation run, we train the k-NN model on the union of 8 data sets, use one set for held-out data (split into two development sets), and the 10th set is reserved for final testing. Tables 4.6a to 4.6c show evaluation results for the 4,000,000 test sentences (400,000 per cross-validation run, 50% ungrammatical) that have not been seen during the optimisation process and for all three optimisation criteria. Naturally, the results are best when the optimisation criterion coincides with the evaluation measure. However, differences in mean square error of predictions for grammatical data and in the σ-gap measure are small between the experiments optimising for these two measures (Tables 4.6a and 4.6b). Only accuracy improves significantly when using the σ-gap optimisation criterion.

Table 4.6c shows what the EPP model can achieve if its parameters are directly trained for high accuracy. The order of the models (lowest accuracy for the model optimised for

Measure	Min	Avg	Max	Std. Dev.
MSE	160.1	163.8	167.5	2.4
Sigma gap	-21.0	-20.8	-20.5	0.2
Accuracy	60.17%	60.53%	60.80%	0.19%
Duration	1230s	1576s	1982s	282
μ_G	-0.89	-0.61	-0.42	0.16
μ_U	-6.04	-5.77	-5.55	0.16
σ^2_G	159.86	163.44	167.05	2.30
σ^2_U	169.88	173.18	176.68	2.23

(a) optimising for mean square error (MSE)

Measure	Min	Avg	Max	Std. Dev.
MSE	160.2	164.5	168.6	2.6
Sigma gap	-20.9	-20.6	-20.3	0.2
Accuracy	60.88%	61.08%	61.41%	0.14%
Duration	929s	1620s	2388s	484
μ_G	-0.69	-0.53	-0.26	0.12
μ_U	-6.16	-5.98	-5.69	0.13
σ^2_G	159.94	164.22	168.14	2.59
σ^2_U	169.82	174.13	178.56	2.85

(b) optimising for σ-gap

Measure	Min	Avg	Max	Std. Dev.
MSE	220.3	280.3	328.5	32.8
Sigma gap	-29.1	-26.7	-23.4	1.7
Accuracy	61.50%	61.81%	62.03%	0.16%
Duration	853s	1669s	2612s	561
μ_G	-0.97	-0.42	0.75	0.52
μ_U	-8.05	-7.39	-6.50	0.49
σ^2_G	219.65	279.88	328.45	32.82
σ^2_U	228.49	290.13	339.47	32.47

(c) optimising for accuracy

Table 4.6: Evaluation results on test data in 10 cross-validation runs with parameters optimised for the three different objective functions; duration does not include the time for parsing the 400,000 test sentences nor for extracting features

mean square error and highest accuracy for the accuracy model) is not surprising. However, the accuracy model improves accuracy more over the σ-gap model than the σ-gap model improves over the mean square error (MSE) model. Contrary to our expectations in Section 4.3.3, the σ-gap optimisation criterion does not separate the APP/EPP distributions adequately and seems to be a weak predictor of accuracy. Future work should investigate why the σ-gap optimisation criterion does not achieve more because an answer to this may also lead to a better optimisation criterion for training on grammatical data only (see Section 4.7.2).

Computational Costs

Tables 4.6a to 4.6c also show how long it took to evaluate the 400,000 test sentences of each cross-validation run. The time for parsing and feature extraction is not included because we processed all data in advance as each test set is required as training data in the other cross-validation runs. Compared to parsing, the computational costs of applying the final model to new data is negligible. On average, it takes $1669/400000 \approx 0.004$ seconds per sentence to retrieve the k-best reference sentences and to calculate an EPP.[36]

Training the EPP model parameters, however, is very costly. Table 4.3 and Figure 4.18 suggest that at least 50 to 100 hill-climbing steps are needed to get close to the reported performance. Ideally, 300 steps or more should be made. Assuming that 10 shots have to be run and that no cross-validation is performed (reducing costs by factor of 10), the first 100 steps correspond to 60 CPU hours[37] on a single core of an Intel Xeon E5440 CPU. However, the model parameters have to be found only once for an application, and it is a question for future work whether they have to be adapted for new domains — see Section 4.7.6.

[36] Actually, the numbers are even more favourable as we did not subtract 53 seconds loading time for test and training data, 51 seconds for building the ANN search tree and 49 seconds for calculating and printing various evaluation tables, to give the overhead observed in the first cross-validation run.

[37] Computational costs slightly increase as the model parameters improve: all 1,000 steps took 735 CPU hours (instead of 600).

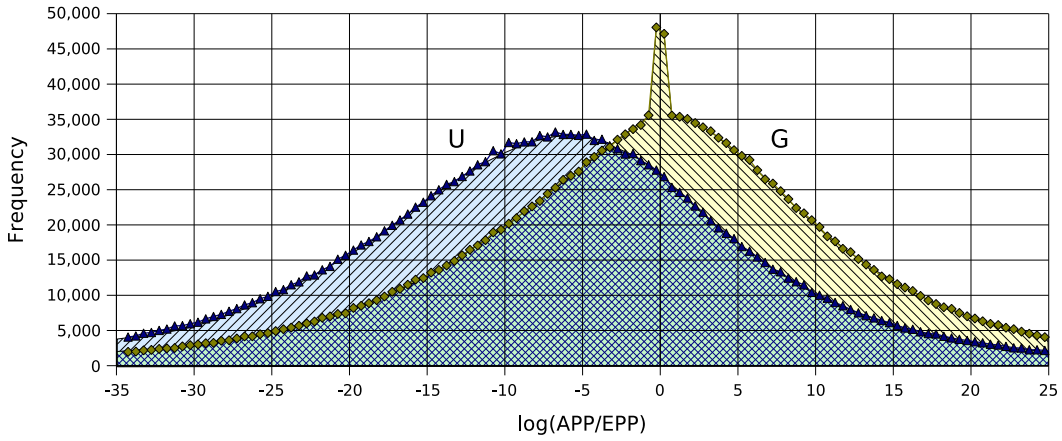


Figure 4.19: Distribution of APP/EPP values for grammatical (G) and ungrammatical (U) test data (2 million sentences each). Frequencies are measured in intervals of 0.5 points on the logarithmic scale.

APP/EPP Distribution

The last four rows of Tables 4.6a to 4.6c report statistical properties of the distribution of log(APP/EPP) for grammatical and ungrammatical data. Our expectations stated in Section 4.3.3 are confirmed: μ_G stays near 0, μ_U is in the range of the effect of grammatical errors observed in Section 4.2 and σ_G is consistently smaller than σ_U.

Figure 4.19 shows the actual distributions for the final EPP model optimised for accuracy. The overlap is the reason why classification with the APP/EPP ratio is not perfect. For ungrammatical data the log(APP/EPP) distribution is bell-shaped like a normal distribution. However, the distribution for grammatical data has a peak around 0 meaning that there are a number of sentences for which the EPP model can predict the parse probability accurately. A manual inspection suggests that this peak is caused by sentences that are frequent in the training data and also appear in the test data, for example "Oh yes.", "Is it?" "Thank you." and "Video-Taped report follows". The latter is presumably a peculiarity of a sub-corpus of the BNC, but if there are no side effects, there is nothing wrong with the EPP model being able to predict the parse probability of some sentences well. However, it is often argued that duplicates have negative effects on probabilistic models and should therefore be removed. Future work (Section 4.7.7) has to show whether this also holds for our EPP models.

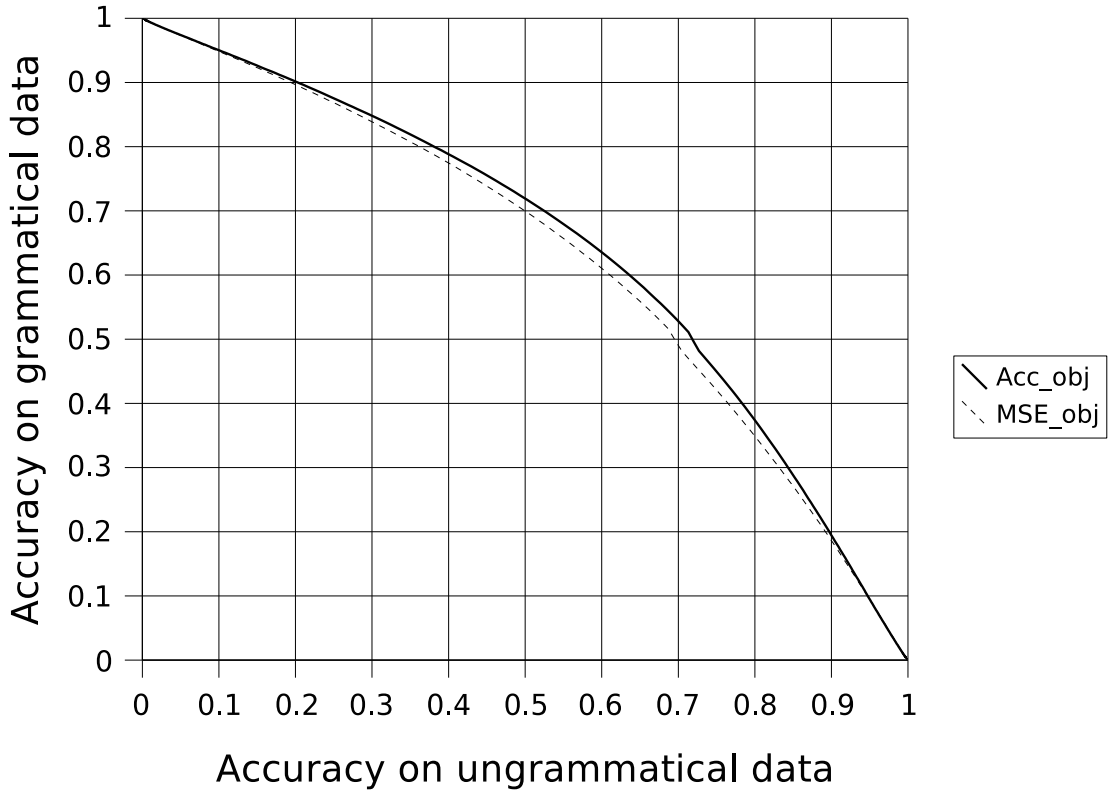


Figure 4.20: Accuracy curves for the APP/EPP method with parameter C running from e^{-70} to e^{60} — parameter optimisation with both grammatical and ungrammatical data (accuracy objective function) and with grammatical data only (mean square error objective function)

Accuracy Curve

The accuracy figures above are for the value of C that results in a classifier that has identical accuracy on grammatical and ungrammatical test data. Figure 4.20 shows the accuracy curves parameterised by C for the two final EPP models optimised with the MSE and accuracy objective functions (Sections 4.3.3 and 4.6.2). The MSE optimised model is included as it can be trained without ungrammatical development data. In contrast, the better performing accuracy model requires both grammatical and ungrammatical development data which might not be available in applications. The accuracy reported in Tables 4.6a and 4.6c can be seen at the intersection of the curve with the bisecting line (not shown). Near the ends of the curves, the two curves show no relevant difference. For example, if the accuracy on grammatical data has to be 90% we would achieve approximately 20.3%

accuracy on ungrammatical data.

4.7 Conclusions and Future Work

We have succeeded in building an EPP model that penalises ungrammatical sentences less than the actual parser while being sufficiently accurate to support a classifier that can classify 61.8% of grammatical and ungrammatical sentences correctly or, with a different accuracy trade-off, a classifier that finds 20.3% of errors while only flagging 10% of grammatical sentences. However, the predictions are still very noisy. Chapter 7 shows how the APP/EPP method performs compared to other methods and in respect to particular error types.

A clear advantage of the APP/EPP method is that it only requires a grammatical reference corpus. While a limited amount of ungrammatical data is useful for setting the threshold C on the APP/EPP ratio, the threshold can be chosen based on the desired accuracy on grammatical data alone.

The choice of reference corpus is independent from the training of the probabilistic parser. Therefore, reference corpora are not restricted to treebanks. Any raw text corpus can be used. To adapt the APP/EPP method to a new or specific domain, e. g. student essays, it should be sufficient to exchange the reference corpus without adapting the underlying parser because parse probabilities of grammatical and ungrammatical sentences should be affected by the domain change in the same way — see Section 4.7.6.

The APP/EPP method works on top of existing probabilistic parsers and requires little knowledge of the algorithms used. Only parser output and independent prediction models are used, as in Figure 4.7. No rules are added to the grammar nor does the method change the interpretation of grammar rules, e. g. in terms of relaxing constraints contained in the rules. The grammar employed must be probabilistic, cover the language well and be robust. There is no need for the grammar to reject ungrammatical input, in fact it may well parse all input, because the new method detects ungrammatical sentences based on their parse probabilities.[38]

[38] If some input is rejected by the parser, it has to be classified by some other means than the APP/EPP ratio, effectively embedding our method into a classifier cascade.

In this chapter, we only were able to scratch the surface of what could be done to improve the EPP model. Below we suggest some modifications that would be worthwhile trying and raise additional questions to be investigated.

4.7.1 Exploiting Feature Correlations with Linear Transformations

Our k-NN model applies scaling factors to the features in order to improve the relevance of the retrieved k items. The literature suggests that correlated features should be replaced by component vectors. This can be done by calculating the covariance matrix of the data and either deriving the Karhunen-Loève transformation (principle component analysis), or directly applying the Mahalanobis distance measure (Bishop, 1995, pp. 35, 310–313). These transformations are sound if one assumes that the data is distributed according to a multivariate Gaussian distribution. Since this is clearly not the case (consider for example the asymmetry in Figure 4.12), an idea might be to optimise a transformation matrix with the parameter optimisation methods used in this chapter. The starting point could be the Karhunen-Loève transformation matrix.

4.7.2 Weakness of Sigma-Gap Objective Function

An explanation for the inferior performance of the σ-gap objective function compared to optimising accuracy directly (Section 4.6.3) could help us to design a better objective function for training on grammatical and ungrammatical data that approximates the performance of the accuracy objective function and then to reduce it to an objective function that only requires grammatical data and gives better results than the mean square error objective function.

4.7.3 Adding Negative Training Data

For grammatical data the role of the EPP model is clear: it should predict the actual parse probability well. Viewing the role of the EPP model for ungrammatical data as a predictor of the parse probability of a hypothetical correction, it makes sense to add negative training data to the EPP model for which the parse probability has been replaced by the parse probability of a correction. Since our artificial error corpus is a parallel error corpus,

finding a correction, i. e. the original sentence before the error creation procedure has been applied, and its parse probability is straightforward. For example, the grammatical sentence s_G with probability p_G and the ungrammatical version s_U with probability p_U could be added to the training data as 2 pairs (s_G, p_G) and (s_U, p_G). Note that p_U is ignored.

Having such negative examples also (or only) in the development test set may be useful: it may allow us to add arbitrary features to the model even when optimising parameters with the mean square error objective function. However, if we have negative development data, it makes more sense to directly optimise accuracy.

The (s_U, p_G) pairs can potentially be problematic because the relationship between s_U and p_G might be more difficult to model than the one between s_G and p_G. Maybe certain pairs should not be added at all, for example it might be better to exclude some ungrammatical sentences based on error type or on whether p_U is actually smaller than p_G.

4.7.4 Basic PCFG Parsing

We see two ways in which basic PCFG models could be tried in the APP/EPP method: a) as an APP model (replacing Charniak's history-based probabilistic model) or b) by adding PCFG probabilities as another feature to the EPP model. On the one hand, a PCFG might be sufficiently strong to react to ungrammatical input with a lower parse probability and therefore work as an APP model. For a PCFG-based APP model, the EPP model might be easier to build than for Charniak's model because the influence of lexical items is less complex or, in case of unlexicalised parsing, minimal (only mediated through POS tags). If, on the other hand, a PCFG model is too weak, i. e. its parse probabilities do not drop when confronted with ungrammatical input, then it will be a useful addition to the EPP model.

4.7.5 Probability Mass of n-best Parses

Currently we use the probability of the best parse. Potentially, the probability mass of all n-best parses ($n > 1$) is more suitable for the APP/EPP method. It could be that

summing over multiple parses reduces noise. It could also be the case that the generative probability of a sentence which is approximated by a sum over n-best parses is easier to model than the probability of an individual parse tree. For n up to 50 we could extract these probabilities from the parse results we stored and then re-run the experiments.

4.7.6 Domain Adaptation

An important question for applications is how much effort is necessary to adapt the APP/EPP method to a new or specific domain:

1. How does the method perform on a new domain without any changes?
2. Is it sufficient for good performance to exchange the reference corpus?
3. Do we have to retrain the parameters of the k-NN model?
4. How much is gained from adapting the parser, for example with self-training on the new domain (McClosky et al., 2006; Foster et al., 2007)?

To address the first two questions, we could associate our feature vectors with the BNC domain annotation, filter training and test data accordingly and repeat the experiments.

4.7.7 Effect of Duplicate Sentences

The observations in Section 4.6.3 raise the question of whether duplicate sentences like "Thank you." and "Video-Taped report follows" that appear many times in the training data have a negative effect on k-NN-based EPP models. It is commonly assumed in NLP that duplicates must be removed before probabilistic models are built. An experiment removing duplicate sentences from the training data would be as easy to implement as the domain adaptation experiment (Section 4.7.6).

4.7.8 Features and Grammaticality

More time could be spent on investigating for each feature of our EPP model how much it contributes to the prediction and how much it reflects grammaticality. The latter could be addressed by analysing pairs of sentences of our parallel error corpus as we did for the

number of nodes in Figure 4.13. The former could be answered by measuring performance differences of classifiers derived from EPP models with and without the feature in question.

4.7.9 N-gram Language Models as EPP Models

If n-gram language models were good predictors of parse probabilities, our k-NN models would simply assign a high weight to them and retrieve k sentences with very similar parse probabilities. (There is enough training data to find good matches.) Nevertheless, it would be interesting to see how close n-gram language models are to our EPP models on our 3 evaluation measures MSE, sigma gap and accuracy. If a simple EPP model could be built that comes close to the performance of our EPP models, then, for many applications, our method might not be worth the effort. However, Figure 4.9 is discouraging as it shows that raw language models probability will not work as an EPP model and that at least sentence length and another factor (causing the vertical variance within each length column in Figure 4.9) have to be accounted for.

4.7.10 Head-Lexicalised Terminal Rule Probabilities

Early experiments in Appendix B show larger improvements when terminal rule probabilities were added as a feature for a basic PCFG parser than in this chapter for a history-based parser. This is not surprising as we still use a PCFG-style model for the terminal rule probabilities. Therefore, the EPP model might benefit from a better model. We could apply head-finding heuristics to the parser output and then use this data to build a head-lexicalised terminal rule probability model. The aim should be to reconstruct the parser's decision, not to make the best decisions.

Chapter 5

Basic Grammar and n-gram based Approaches to the Detection of Ungrammatical Sentences

In this chapter, we evaluate four basic methods for automatically evaluating the grammaticality of a sentence that share the property that they do not rely on machine learning to set their parameters. Most of these methods will be revisited in Chapter 6 where we attempt to improve them with a machine learning method. Therefore, the results of this chapter can be seen as a baseline for the next chapter.

In Section 5.1 we test how a hand-crafted wide-coverage precision grammar performs in the task of classifying a sentence as either grammatical or ungrammatical. Section 5.2 evaluates a basic n-gram method which flags POS n-grams as ungrammatical if they have a low frequency in reference data. We then try to transform a treebank-induced PCFG into something like a precision grammar by pruning rare rules in Section 5.3. The final basic method is presented in Section 5.4. It compares the parse probability of two treebank grammars, one induced from the original treebank and one from a copy of the treebank with errors inserted automatically, to classify a sentence as either grammatical or ungrammatical. We conclude in Section 5.5 with some remarks on the observed results.

5.1 Precision Grammar Judgements

In contrast to treebank-induced grammars that we use in a number of other approaches (in Sections 5.3 to 5.4 and in Chapter 4), precision grammars are designed, in the traditional generative grammar sense (Chomsky, 1957), to distinguish grammatical sentences from ungrammatical sentences. Grammar rules are kept as restrictive as possible, avoiding overgeneration by all means. In the Chomskyan view, a grammar rule that leads to the acceptance of an ungrammatical sentence must be wrong. A more pragmatic motivation for writing specific grammar rules is to avoid implausible analyses that can easily arise when a rule is applied in a different context, i. e. the aim is that all parses produced for a sentence, not just its "best" parse, are plausible and describe the ambiguity of the sentence. Ungrammatical sentences should not receive a parse tree and they often do not because the rules of the grammar are too specific to smooth over grammatical errors.[1] During grammar development, each modification is tested on a set of hand-crafted sentences that exemplify various grammatical constructions. This careful discrimination between what the grammar should and should not cover should therefore results in good grammaticality judgements. In the following, we will test this hypothesis experimentally.

5.1.1 The ParGram English LFG

In order for a precision grammar to be suitable for our experiments, its coverage must be broad enough to parse unrestricted text, and for this reason, we choose the ParGram English grammar (Riezler et al., 2002; Butt et al., 2002, 1999) which is a broad-coverage hand-crafted Lexical-Functional Grammar (LFG) (Kaplan and Bresnan, 1982; Bresnan, 2001; Dalrymple, 2001). This grammar can be used with the XLE parser engine, an efficient and robust parsing system for LFG (Kaplan and Bresnan, 1982; Maxwell and Kaplan, 1996; Kaplan et al., 2004). The system employs robustness techniques, some borrowed from Optimality Theory (OT) (Prince and Smolensky, 1993), to parse extra-grammatical input (Frank et al., 1998), but crucially still distinguishes between optimal and suboptimal solutions. While the main aim of OT is to handle syntactic and lexical

[1] An exception are precision grammars augmented with mal-rules that address particular ungrammatical constructions. Also note that most precision grammars have optional robustness features to make up for coverage limitations, e. g. fragment rules.

ambiguity by dispreferring certain analyses, it also allows for the addition of mal rules that must only be employed if there is no parse using the core grammar. Frank et al. (1998) give the example of a mal rule for violations of agreement between subject and a third person singular verb. If the XLE parser has to revert to such a rule, it marks the analysis as ungrammatical with a star. The ParGram LFGs are often described as deep grammars because they resolve non-local dependencies and therefore integrate well with meaning representations. As a constraint-based theory of grammar, LFG associates each node of a context-free phrase-structure tree with functional equations. The solutions of these equations are feature structures that describe grammatical and semantic relations.

Achieving high coverage with a hand-written grammar is difficult and a time-consuming process. The ParGram project started developing its English, German and French LFGs in 1994. The publications between 1998 and 2002 cited above suggest that it took years to reach the limits of the basic formalism before extensions (OT and probabilistic parse disambiguation models) were added. A possible alternative to the ParGram English LFG for our experiment is the Lingo English Resource Grammar (ERG) which is a precision Head-Driven Phrase Structure Grammar (HPSG) of English (Copestake and Flickinger, 2000; Pollard and Sag, 1994). We use XLE in the following experiment because it is an efficient parser that is readily available to us.

5.1.2 Related Work

There has been previous work using the ERG and the ParGram grammars in the area of computer-assisted language learning. Bender et al. (2004) use a version of the ERG containing mal-rules to parse ill-formed sentences from the SST corpus of Japanese learner English (Izumi et al., 2004). They then use the semantic representations of the ill-formed input to generate well-formed corrections. Khader et al. (2004) study whether the ParGram English LFG can be used for computer-assisted language learning by adding additional OT marks for ungrammatical constructions observed in a learner corpus. However, the evaluation is preliminary, with only 50 test items.

We are unaware of any previous work that systematically evaluates the grammaticality judgements of precision grammars for both grammatical and ungrammatical test data.

Results for grammatical data are usually reported as the coverage of the grammar, i. e. the fraction of grammatical sentences the grammar can analyse. Coverage coincides with the accuracy the grammaticality classifier would have on grammatical test data.

5.1.3 Experimental Setup

We test how the ParGram English LFG core grammar performs in the task of judging a sentence's grammaticality using our artificial test data (Section 3.3 of Chapter 3). Two classifiers are considered: X_1 and X_2. Both classifiers share that they classify sentences as grammatical that are covered by the core grammar and that sentences that have no parse or can only be parsed using robustness techniques are classified as ungrammatical. Classifier X_1 treats time-outs and out-of-memory errors as instances of the "no parse" event, i. e. such sentences are classified as ungrammatical. The second classifier X_2 treats these errors as indicating grammatical input.

The experimental setup is very simple: we parse our grammatical and ungrammatical data with the XLE parser and the ParGram English grammar[2] and count the number of sentences for which the parser resorts to robustness techniques, for which no parse is found or for which it runs out of memory or time. The (relative) frequencies of the respective subsets of these events are cumulated to calculate the accuracy of each classifier. For better comparison with the other chapters, we report results for the development sets of each run of the 10-fold cross-validation. Since there are no parameters, the model is the same in each run.

5.1.4 Results

Table 5.1 shows overall frequencies of the observed parser events and the range of frequencies in the 10 cross-validation runs broken down by grammatical and ungrammatical test data.[3] The accuracies of our classifiers X_1 and X_2 are given in the bottom rows of the table.[4] XLE's robustness features can produce a parse for almost all sentences. Within

[2] XLE command `parse-testfile` with `parse-literally` set to 1, `max_xle_scratch_storage` set to 1,000 MB, a time-out of 60 seconds and no skimming

[3] 86 of the 4 million test sentences (0.00215%) were not parsed for other reasons than reported in the table including crashes of XLE.

[4] In cases where accuracy is calculated as the sum of the relative frequencies of multiple events, the minimum and maximum accuarcy cannot be calculated by cumulating the reported individual event minima

	Grammatical			Ungrammatical		
Event	**Min**	**Avg**	**Max**	**Min**	**Avg**	**Max**
Covered	62.53%	62.82%	63.06%	42.92%	43.15%	43.41%
Robustness	32.66%	32.87%	33.12%	51.97%	52.22%	52.46%
No parse	0.23%	0.24%	0.27%	0.23%	0.24%	0.26%
Time out	0.53%	0.55%	0.57%	0.50%	0.53%	0.55%
Out of memory	3.43%	3.51%	3.58%	3.80%	3.86%	3.94%
Accuracy X_1	62.53%	62.82%	63.06%	56.59%	56.85%	57.08%
Accuracy X_2	66.64%	66.88%	67.10%	52.20%	52.46%	52.69%

Table 5.1: Coverage of the ParGram English LFG and resulting classifier accuracy: range of values (and average) over 10 cross-validation runs (disjoint test sets)

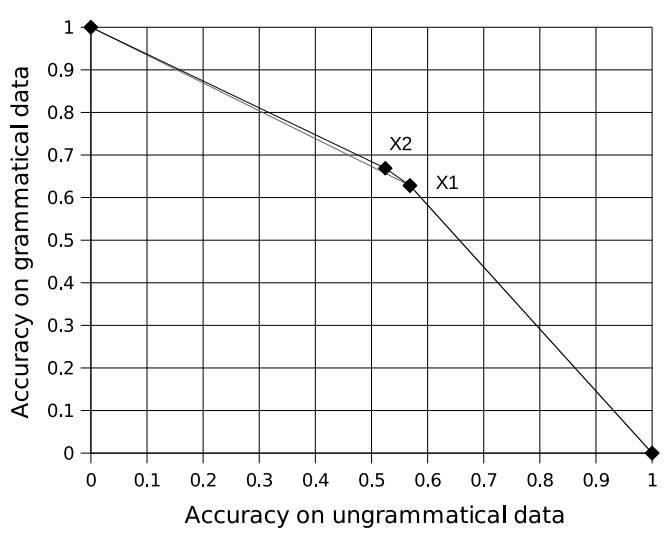


Figure 5.1: Accuracy point for the XLE-based classifier and interpolation with the 2 trivial classifiers (pass-all and flag-all)

the time and memory constraints, the parser rarely concludes that no parse is possible. Interestingly, the "no parse" event does not provide any information about whether the input sentence is grammatical since it occurs equally frequently in both test sets. Also, the differences for parser exceptions are small.[5]

The accuracy points of the two XLE-based classifier (56.85%, 62.82%) and (52.46%, 66.88%) are shown in Figure 5.1. Normally, we would show a curve in the accuracy plane

or maxima as these extremes may have been reached for different cross-validation runs.

[5]The coverage of grammatical BNC sentences increases by 4.3 percentage points to 67.1% if we exclude transcribed speech, poetry, captions and list items which amount to 23.3% of the corpus (16.7% alone is spoken). From these numbers we can calculate that only $(62.82 - 67.1 \times (1 - 0.233))/0.233 \approx 48.7\%$ of the excluded sentences (speech, poetry etc.) are covered by the grammar.

to visualise the accuracy trade-offs that can be achieved by varying parameters. However, in the case of our XLE-based classifiers, we can only choose between classifiers X_1 and X_2.[6] The figure shows the interpolation with the 2 trivial classifiers, i. e. accuracy trade-offs that result from randomly passing or flagging sentences. The classifier X_2 outperforms the interpolation of X_1 with the "pass all" classifier only by a small margin. This can be exlained with the limited discriminativeness of the parser exception events (time-out and out-of-memory). In Chapter 7, these classifiers are compared to other methods.

5.2 POS n-gram Frequency

This section evaluates a vanilla POS n-gram approach to the problem of error detection that flags as ungrammatical any n-gram of the input that is rare or absent in a reference corpus. Parameters of this approach include what we mean by rare, the POS tag set and n. We keep our experiment simple by restricting ourselves to raw frequency values and a tag set that closely follows the Penn treebank tag set.

5.2.1 Related Work

N-gram-based approaches to the problem of error detection have been proposed and implemented in various forms by Atwell (1987), Chodorow and Leacock (2000), and Bigert and Knutsson (2002) amongst others. Existing approaches are hard to compare since they are evaluated on different test sets which vary in size and error density. Furthermore, most of these approaches concentrate on one type of grammatical error only, namely, context-sensitive or real-word spelling errors.

The (to our knowledge) earliest work in this area is that of Atwell (1987) who uses a POS tagger to flag POS bigrams that are unlikely according to a reference corpus. While he speculates that the bigram frequency should be compared to how often the same POS bigram is involved in errors in an error corpus, the proposed system uses the raw frequency with an empirically established threshold to decide whether a bigram indicates an error.

[6] Another 28 genuine XLE-based classifiers with different accuracy trade-offs can be built if we explore the remaining of all $2^5 = 32$ possibile treatments of the five (exclusive) events recorded and notice that two of the 32 classifiers are the trivial classifiers flagging all or no sentences. For example, the classifier that only flags sentences as ungrammatical if the parser produces a parse using robustness techniques has the accuracy point (52.22%, 67.13%).

In addition, he speculates that "local minima" (low frequency relative to neighbouring bigrams) may be used as an indicator. In the same paper, a second, completely different approach is presented that uses the same POS tagger to consider spelling variants that have a different POS. In the example sentence *I am very* ***hit*** the POS of the spelling variant *hot/JJ* is added to the list NN-VB-VBD-VBN of possible POS tags of *hit*. If the POS tagger chooses *hit/JJ*, the word is flagged and the correction *hot* is proposed to the user. Unlike most n-gram-based approaches, Atwell's work aims to detect grammar errors in general and not just real-word spelling errors. However, a comprehensive evaluation of the approach is missing.

Disambiguating methods for confusion sets and candidate correction (Section 2.2.4 of Chapter 2) make use of n-grams, e. g. Golding (1995) builds a classifier based on a rich set of contextual features.

Bigert and Knutsson (2002) extend a basic n-gram approach by attempting to match n-grams of low frequency with similar n-grams in order to reduce overflagging. Similarity is defined by the normalised frequencies of the POS contexts in which the two POS tags to be compared can occur. Furthermore, n-grams crossing clause boundaries are not flagged and the similarity measure is adapted in the case of phrase boundaries that usually result in low frequency n-grams.

Chodorow and Leacock (2000) use a mutual information (MI) measure in addition to raw frequency of n-grams. MI measures the ratio of the observed n-gram frequency to the expected frequency according to an $(n-1)$-gram language model. If the ratio is below 1, i. e. if the n-gram occurs less often in the reference data than expected from a lower order model, the n-gram can still indicate a grammatical error even if its absolute frequency is high. Apart from this, their ALEK system employs other extensions to the basic approach, e. g. frequency counts from both generic and word-specific corpora are used in the measures and they condition the probability of the middle item of each trigram on its first and last item. The evaluation focuses on 20 specific target words for which overall recall is 19.0% and precision 77.9%. It is not reported how much each of these modifications of the vanilla method contribute to the overall performance. In particular, it remains unclear to what extent the MI measure provides useful information to the classification task. This

measure has also been used in the area of collocation detection, where it is used to detect word combinations that occur more often than expected, see for example the evaluation of collocation measures by Evert (2005) and Pecina and Schlesinger (2006).

Rather than trying to implement all of the previous n-gram approaches, we implement the basic approach which uses rare n-grams to predict grammaticality. This property is shared by all previous shallow approaches. We also test the n-gram approach on a wider class of grammatical errors.

5.2.2 Experimental Setup

We count part-of-speech (POS) n-gram frequencies for $n = 2, ..., 7$ in our reference corpus[7] comprising 2,409,265 BNC sentences — see Chapter 3 for details of how we split the corpus. Using these reference frequencies, we flag sentences of our test data as erroneous if they contain an n-gram that falls below a certain frequency threshold. Sentences with $n - 1$ or fewer tokens cannot contain any n-grams. In one version of the experiment, such sentences are always accepted as grammatical. Alternatively, we consider a variant that adds $n - 1$ padding tokens (tagged with a POS that is not part of the normal tag set) to the start and the end of each sentence (for both reference and test data), for example the sentence "Yes." would receive the tag sequence PAD PAD UH SENT PAD PAD for $n = 3$ and thus receive four trigrams.[8] The classifier therefore has three parameters: n, the frequency threshold and whether or not we use padding. As before, we measure accuracy for each of the 10 test sets of our cross-validation and report the average and ranges of values. To choose the parameters of the n-gram method, we strictly follow the cross-validation setup, i. e. in each of the 10 cross-validation runs, 9 training sets inform the decision of which parameters will be used and the choice is then evaluated on the 10th set. The role of the sets rotates from one cross-validation run to the next.

[7] The IMS TreeTagger (Schmid, 1994) was used to annotate POS information.

[8] Note that this variant of the n-gram method also changes the behaviour for sentences with n or more tokens: if, for example, sentences starting with a coordinating conjunction (CC) were ungrammatical, the first method would not be able to detect this kind of error while the second method could.

Optimal Parameter Sequence

We expect that the method will not find one best parameter but that it will offer different accuracy trade-offs. Each cross-validation run produces a sequence of optimal parameters[9] running from high accuracy on grammatical data (very little overflagging) and high accuracy on ungrammatical data (almost all errors found). A parameter choice is part of the sequence if it is not outperformed by another parameter choice or by a linear combination that interpolates 2 classifiers. Therefore, the sequence of optimal parameters together with the 2 trivial classifiers "pass-all" and "flag-all" forms a convex hull (see Chapter 3) which we can calculate with the Graham scan algorithm (Goodrich and Tamassia, 1998). In order to report overall results, we need a way to calculate an average accuracy curve from the 10 curves of the cross-validation runs. If the parameter sequences of the individual runs are mostly identical and only differ due to tiny differences leading to different points being included in the convex hulls, we can simply report average individual test results for the union of all parameter sequences. Otherwise, we will have to resort to methods outlined in Section 3.8.4 of Chapter 3. In both cases, the final accuracy curve is not guaranteed to be convex itself because test set results may differ substantially from training set results.

In order to limit the number of points on the convex hulls, i. e. the length of the optimal parameter sequence for which we will report detailed results, the n-gram frequency threshold t is not explored for all integer values. We only include thresholds $t = \lfloor 0.5 + b^x \rfloor$ between 1 and 1,000,000 where x is an integer and b some number larger than 1.[10] For example, if we set $b = \sqrt{2}$, x will run from 0 to 39 and the thresholds are the powers of 2 and intermediate numbers: 1, 2, 3, 4, 6, 8, 11, 16, 23, 32, 45, 64, ..., 524,288, 741,455. We will also measure how much accuracy we lose with this restriction by comparing results to a convex hull with a value of b considerably closer to 1 than for the one we report in detail.

[9] If there is one best parameter, then the sequence will have length 1.

[10] $t = 1,000,000$ is more than sufficient because the highest observed bigram frequency is 430,515.

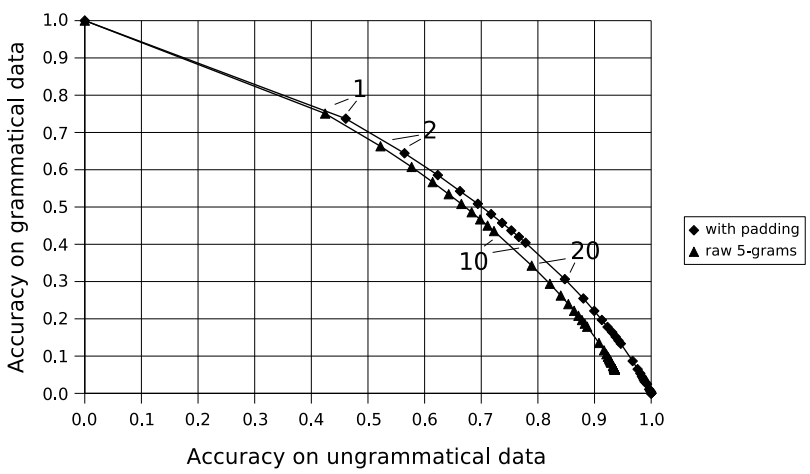


Figure 5.2: Effect of padding sentences on the accuracy of a classifier flagging rare 5-grams (frequency threshold $t = 0, 1, 2, ..., 9, 10, 20, ...90, 100, 200, ...$)

5.2.3 Results

First we look at the effect of padding for $n = 5$ which is the best choice of n for data with 50% ungrammatical sentences according to our earlier work (Wagner et al., 2007a). Figure 5.2 shows the 2 accuracy curves for the basic 5-gram method with and without padding. The top left point $(0, 1)$ shows the trivial classifier with threshold $t = 0$. For this parameter setting, all sentences are classified as grammatical because no sentence contains an n-gram with a frequency lower than 0. The next possible classifier with $t = 1$ already classifies a quarter of grammatical sentences as ungrammatical (accuracy falls to 73.69% with padding and to 75.07% without padding). The sentences flagged by this classifier contain at least one 5-gram that is unattested in the reference corpus. For increasing thresholds t, the accuracy points fall closer together and we compensate for this in this graph by increasing the stepping by a factor of 10 each time we reach a power of 10 ($t = 1, 2, 10, 20$ are labelled in Figure 5.2).

Padding is clearly advantageous. The accuracy curve is above the curve of the raw method without padding, i. e. for each point on the curve of the raw 5-gram method there is a point on the other curve that lies in the area of improvement. (See Chapter 3 for definition of the area of improvement and an explanation of linear interpolation of classifiers.) The lower curve does not reach the lower right corner $(1, 0)$, instead it approaches the point $(93.58\%, 6.39\%)$ for increasing threshold t. This is explained by the fact that 6.39%

b	$\lvert\{t\}\rvert$	**|Hull|**	A_G **for** $A_U = 0.1$	$A_G = A_U$	A_G **for** $A_U = 0.9$
$10^{1/50} \approx 1.047$	255	49.3	96.715%	60.443%	22.038%
$10^{1/20} \approx 1.122$	110	44.6	96.715%	60.443%	22.038%
$10^{1/10} \approx 1.259$	57	40.2	96.715%	60.443%	22.036%
$\sqrt{2} \approx 1.414$	39	33.0	96.712%	60.443%	22.007%
2	20	23.3	96.712%	60.365%	21.951%
10	7	14.0	96.714%	59.783%	21.576%

Table 5.2: Loss due to reduced number of considered thresholds $t = b^x$: $\lvert\{t\}\rvert$ is the number of thresholds, |Hull| the average number of points on the convex hull (10 cross-validation runs), A_G and A_U are accuracy on grammatical and ungrammatical data

of grammatical and 6.42% of ungrammatical test sentences are shorter than 5 tokens and are therefore always classified as grammatical if we do not use padding.

Optimal Parameter Sequences

Independent of these observations, we calculate the optimal parameter sequences over all possible parameters including the methods without padding, except for the threshold t which is limited to values $t = b^x$ (see experimental setup in Section 5.2.2 above). Table 5.2 shows the effect of b on the optimal parameters. The first three rows suggest that increasing the number of thresholds beyond 57 by setting b smaller than $10^{1/10}$ only adds a few points to the parameter hull and marginally improves accuracy. On the other hand, $b = \sqrt{2}$ shows a first sign of deterioration in accuracy. In the following, we use $b = 10^{1/10}$.

Table 5.3 shows for each cross-validation run which parameters are included in the optimal sequences as described on p. 140. Mostly, the runs agree, i. e. the models learned from the training data for a certain range of accuracy trade-offs are identical. 4 parameter settings appear only in some cross-validation runs. An explanation could be that the corresponding accuracy points are very close to the line connecting their neighbouring hull points and therefore small changes in the training data can cause them to fall on either side of the line. Only if a point is on the outside will it be included in the convex hull.

Figure 5.3 shows the average test set results as an accuracy curve.

Parameter (Triplet)	Run 1	2	3	4	5	6	7	8	9	10
raw, 2, 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
raw, 2, 25	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 3, 1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 3, 2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 3, 3	✓	✓	✓	✓	✓	✓	—	✓	✓	✓
padded, 3, 4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 3, 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 3, 8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 3, 10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 10	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 13	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 16	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 20	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 25	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 32	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 40	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 50	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 63	—	—	—	—	—	✓	✓	—	—	—
padded, 5, 79	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 100	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 126	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 158	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 1259	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 1000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 3162	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 4, 3981	—	—	✓	—	—	—	—	✓	—	—
padded, 4, 5012	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 5, 5012	✓	✓	—	✓	✓	✓	✓	✓	✓	✓
padded, 6, 5012	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
padded, 7, 5012	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.3: Optimal parameter sequence for each cross-validation run: "✓" means that a parameter setting is included in the sequence; parameters are padding, n and threshold t; only a subset of possible thresholds t is considered — see text.

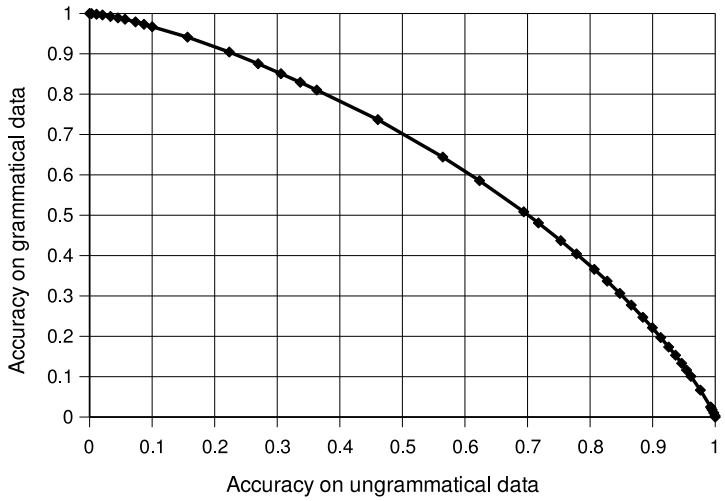


Figure 5.3: Accuracy curve of the n-gram method using the union of optimal parameter sequences of the cross-validation runs

5.3 Pruning Treebank-induced (P)CFGs

Grammars induced from sufficiently large treebanks tend to parse any input string (Charniak, 1996). This fact was one of the foundations of the APP/EPP method presented in Chapter 4 which relies on the parser to output a best parse and its probability for ungrammatical input.[11] The ability to accept almost any input is independent of the probabilistic disambiguation model which is used in parsing to select the best parse and which can also be applied to generation (Cahill and van Genabith, 2006; Hogan et al., 2008). Often the term "overgeneration" is used which can both refer to the overly large set of strings that the grammar accepts or generates and to the set of trees, i. e. the tree language.

In this section, we aim to automatically modify a treebank-induced CFG so that it stops accepting ungrammatical input while keeping the coverage of grammatical input high. Assuming that grammaticality is linked to frequency, we choose to remove rare rules from the grammar, i. e. rules that have a low frequency in the treebank from which the grammar is induced. Our observations in Chapter 4 support the assumption. In particular, we concluded in Section 4.5.3 from the effects of errors on parse probability and on the number of nodes in the parse tree that one or more rules are often replaced with rules

[11] Footnote 11 in Chapter 4 reports that only 5 of the 400,000 test sentences (50% ungrammatical) of the first cross-validation run fail despite being under the length limit of 100 tokens which we applied for efficiency.

with lower probability when a grammatical error is inserted. Psycholinguistic research also investigates the relationship between frequency and grammaticality, see Section 4.4.1 of Chapter 4.

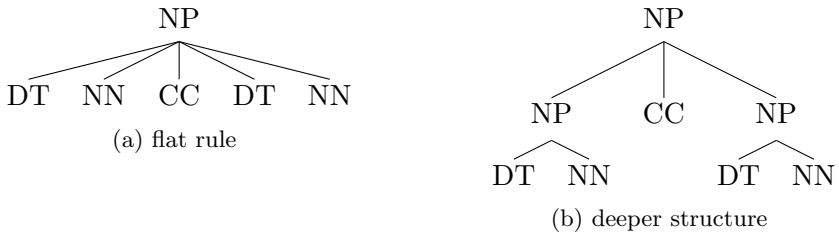
We choose to prune rules based on frequency rather than probability because, as in the n-gram method in Section 5.2, we have the intuition that rare rules may indicate errors, i. e. rules that only are attested coincidentally in the reference treebank. Rule probabilities can be low for other reasons: a competing production with very high frequency or a large number of competing productions.

5.3.1 Related Work

Charniak (1996) evaluates a treebank-induced, unlexicalised PCFG on a preliminary version of the PTB and comments on coverage and overgeneration. The grammar covers all sentences of the test set which excludes sentences longer than 40 tokens. He uses the phrase "extreme overgeneration" and argues that the grammar would accept almost all strings due to a property of prefixes of the formal language defined by the grammar. The focus of the report, however, is on the quality of the most likely parse tree assigned to sentences measured with precision, recall and accuracy of the bracketings. The main source of improvement over previous work is a modified probability model giving more weight to right bracketings ending at the last but one token of a sentence. No attempt is made to actually parse ungrammatical data.

The PCFG extracted by Charniak (1996) has 10,605 rules of which 3,943 occur more than once in the treebank training section. The reduced grammar with just the latter 3,943 rules produces very similar results as regards precision, recall and accuracy. The finding that rules with frequency 1 are not important for parsing grammatical input is encouraging for our work. If these rules are needed to parse ungrammatical input, the pruning method for detecting ungrammatical sentences will work.

Krotov et al. (1999) evaluate two pruning methods for the purpose of compacting grammars for lower computational costs: *(a)* by rule frequency and *(b)* by the ratio of the probability of the candidate rule to the probability of the (most likely) subtree that can replace the candidate rule using other rules of the grammar, e. g. the flat rule extracted



(a) flat rule

(b) deeper structure

Figure 5.4: Krotov et al.'s rule-parsing: a rule that can be replaced by a subtree is considered for elimination from the grammar

from the subtree in Figure 5.4a is compared to the deeper structure of Figure 5.4b. If the overall probability of the deeper structure is higher, the elimination of the flat rule will not change the best (most likely) parse tree as the flat structure would always be dispreferred and appear lower down in any n-best list.[12] They experiment with probability ratios between 0 (naive method that replaces all rules for which a replacement exists) and 1 (only replacing rules that are not needed for the best parse). Surprisingly, Parseval bracketing recall (Black et al., 1991) improves at first with a peak at 80% of the original grammar size. However, as the naive method is approached, performance deteriorates.[13] Coverage of the pruned grammars is not reported for this pruning method but coverage should not change given the rule replacement strategy.

For frequency-based rule pruning, Krotov et al. (1999) experiment with 11 thresholds up to a frequency threshold of 100 which reduces the number of rules by 97%. Their result table includes coverage as the percentage of rejected (no parse) sentences are reported. For small frequency thresholds, coverage is high (97.6% for a threshold of 10) even though the size of the grammar is greatly reduced (88%). However, for thresholds over 20, coverage starts falling moderately. Coverage is down at 70.7% at the highest threshold tested, 100. As to parsing performance, Parseval recall does not suffer much from frequency-based rule pruning. Labelled recall falls by 6.8 percentage points for a frequency threshold of 100.

A third method for pruning (P)CFGs is suggested by Gaizauskas (1995) in a study

[12]Krotov et al. (1999) say that the parse probablility is preserved. However, the elimination of the flat rule in the example implies a lower total number of rules with NP on the left side and therefore a higher rule probability of the rules used in the deeper structure. No details on how rule probabilities are re-estimated are given.

[13]They conducted this experiment with the PCFG that excludes rules that appear only once in order to keep computational costs low as performance and coverage of this frequency-pruned grammar is close to the vanilla grammar.

on statistical properties of the PTB. In a discussion of the distribution of rule frequencies broken down by left-side category and frequency rank, it is mentioned that 95% of rule occurrences within each left-side category can be covered with just 2,144 rules. However, no parsing performance or coverage of sentences is reported.

5.3.2 Experimental Setup

In this section, we investigate whether imposing a frequency threshold to the rules of a (P)CFG improves its discriminativeness, i. e. whether it is possible to obtain a grammar that rejects at least a relevant fraction of ungrammatical input while maintaining high coverage on grammatical data by removing rare rules from a treebank grammar.

We choose two grammars for our experiment based on an evaluation of 288 treebank grammars with different tree transformations (Cahill, 2004). The first grammar is a basic grammar with full coverage on section 23 of the PTB. This grammar is obtained by adding parent annotation (Johnson, 1998) and a root node to all treebank trees. With this grammar, an unlabelled Parseval (Black et al., 1991) f-score of 82.21% is achieved on section 23, just 0.09 percentage points below the best grammar having full coverage reported by Cahill (2004). The second grammar omits parent annotation. The f-score falls to 74.08%.[14]

Pruning the Grammar and Parsing with the Pruned Grammars

The grammar rules are annotated with frequency information, i. e. the number of times the respective rule is observed in the training section of the treebank. We filter the rule set with a frequency threshold between 1 and 1,000 deleting all rules with a frequency in the reference treebank below the threshold.

For parsing, we use BitPar, an efficient CKY-style parser that uses bit vectors to represent the chart and then builds the parse forest in a second parsing stage (Schmid, 2004). A number of adjustments to the grammar files are necessary so that BitPar does not abort with a message indicating a slip of the grammar writer, e. g. unreachable rules

[14]For comparison, the overall best grammar (excluding grammars that do not cover PTB section 23 in full) is obtained with transforming verb labels of auxiliaries to AUX in addition to parent annotation and has an f-score of 82.30%. Note that all grammars discussed here keep the PTB II functional labels.

(rules with a left side that cannot be generated) have to be deleted. One simplification can affect the results: instead of also processing the lexicon files in order to remove unreachable preterminals (POS), we add rules TOP $\rightarrow$ X to the grammars for each preterminal X that cannot be generated otherwise.[15]

Improving the Efficiency of the Experiment

Naively, we would parse all test data with each of the 1,000 pruned grammars and report coverage of grammatical and rejection of ungrammatical sentences. However, note that if a sentence cannot be parsed with a grammar pruned with a threshold, it cannot be parsed with any grammar with higher threshold because these grammars will only contain the same or fewer rules. If, on the other hand, a sentence can be parsed, it will also be parsable with any smaller pruning threshold because the rules that were used in parsing the sentence will also be present in the bigger grammars. Therefore, we start with the 500th grammar. We parse all 4 million BNC test sentences with this grammar. Then, we parse those sentences for which we did not get a parse tree with the 250th grammar and the other sentences with the 750th grammar etc. Effectively, we apply a binary search to the problem of finding for each sentence the highest threshold for which it can still be parsed. Since the range of thresholds left to be tested is halved at each step, we will only parse each sentence 10 times ($2^{10} = 1024 > 1000$) instead of 1,000 times.[16]

Parameters of the Classifier

The classifier to be evaluated has one parameter: the frequency threshold for pruning grammar rules. If a sentence's highest threshold for which it can still be parsed is higher than the threshold given by the classifier, it will be classified as grammatical. As usual, we measure accuracy of our classifiers separately on grammatical and ungrammatical test data and for each cross-validation run.

[15] Only sentences with length 1 are affected.

[16] It is also possible to compute the maximal threshold directly within a modified CYK parser, but since we have sufficient computing power at hand it was easier to just use an off-the-shelf parser.

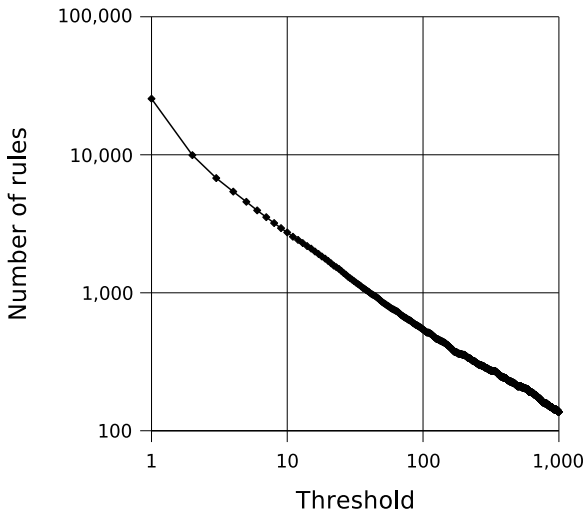


Figure 5.5: Effect of pruning rare rules of a PCFG on the number of rules

5.3.3 Results

Effect of Pruning

Pruning grammar rules with a low frequency quickly reduces the number of rules. Figure 5.5 shows the effect for the grammar without parent annotation. Both axes are on logarithmic scale, otherwise the curve would be very close to the lower left corner. We observe a ratio of 2.56 between the number of rules in the initial grammar and the grammar we obtain when we prune all rules with frequency 1. This is similar to the ratio 2.69 observed by Charniak (1996). We observe a reduction of coverage on grammatical data from 99.97% with the vanilla grammar to 99.80% already in this first pruning step.

Optimal Pruning Thresholds

We first parsed the full corpus with the grammars without parent annotation because these grammars are smaller and therefore parsing requires less resources. Still, a small number of sentences caused memory problems which rendered the affected package of 20,000 sentence useless[17] and we decided to perform the evaluation without the failed packages as opposed to isolating the problematic sentences. In order to use the same amount of test data in each cross-validation run, we further restrict the test data to 2 x 100,000 sentences per

[17]We ran the binary search within blocks of this size instead of the full corpus in order to make it easy to parallelise the process.

Parameter (Threshold)	Run 1	2	3	4	5	6	7	8	9	10
5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
6	✓	—	✓	✓	—	✓	✓	✓	✓	—
21	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
23	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
27	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
31	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
79	✓	✓	—	✓	✓	✓	✓	—	✓	✓
82	—	—	✓	—	—	—	—	✓	—	—
598	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
602	✓	✓	✓	✓	✓	—	✓	✓	✓	—
603	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
992	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 5.4: Optimal parameter sequence for each cross-validation run: "✓" means that a parameter setting is included in the sequence.

cross-validation run which is half of the test data we normally use.

Table 5.4 shows the classifiers that remain after the calculation of the optimal parameter sequence (see p. 140), i. e. all classifiers not listed are inferior to a linear combination of listed classifiers. Figure 5.6 shows the accuracy curve which is barely above the baseline of randomly guessing the grammaticality. Thresholds -1 and "inf" represent the trivial classifiers passing all or no input.

Results with Parent Annotated (P)CFG

For the experiment with the grammars with parent annotation, we exclude sentences longer than 100 tokens in order to avoid memory problems. Since long sentences are generally more difficult to classify correctly, the classifier results are not comparable to the accuracy figures above and cannot be joined in a convex hull calculation over both types of grammars. However, the results are only marginally better (51.39% vs. 51.07% accuracy for the interpolated classifiers with equal accuracy on grammatical and ungrammatical data) despite the large difference in unlabelled Parseval (Black et al., 1991) f-score of the two unpruned grammars (82.21% vs. 74.08% according to Cahill (2004)). Considering these negative results, it does not seem to be worthwhile to re-run the first experiment with the same sentence length restriction in order to calculate a convex hull over all classifiers,

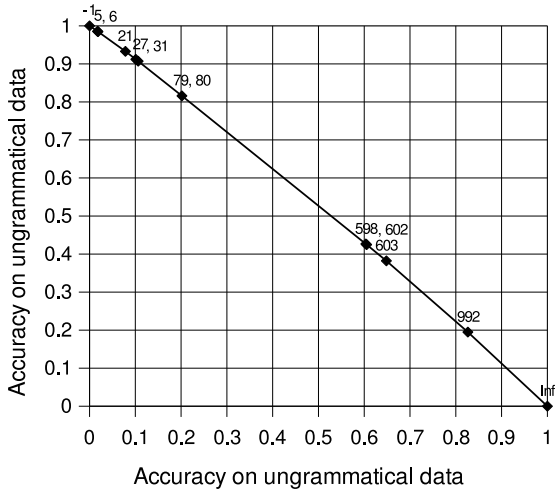


Figure 5.6: Accuracy curve of the PCFG pruning method

nor to expand the experiment to other treebank transformations.

Summary

We conclude from the negative result that rare rules are as important for parsing grammatical sentences as they are for ungrammatical input. This means that rare rules are needed for rare grammatical constructions or to fill gaps in coverage, rather than being dispensable for grammatical sentences and only allowing for ungrammatical input to be parsed.

5.3.4 Presence of Rare Rules in the Parse Tree

An alternative approach based on the assumption that rare rules are more often used for parsing ungrammatical sentences than for covering grammatical sentences is to parse the input sentence with a vanilla treebank grammar and then to flag a sentence as ungrammatical if the parse tree contains a rare rule. This may produce different results than the pruning method as a probabilistic parser may select a rare rule for the most-likely parse even if a parse avoiding the rare rule exists.[18] We do not explore this approach here as we do not expect major improvements over the pruning method and also because exper-

[18]To understand this behaviour, note that rule probabilities are conditioned on the symbol on the left side of each rule, i. e. the probability reflects the frequency within the set of rules with the same symbol on the left, while the rule frequency reflects the overall frequency distribution.

iments with Markovisation rules reported in Appendix C.3.3 give similar results to those in Section 5.3.3. Markovisation rules are a special case of rare rules that are unattested in the treebank from which the grammar is induced and only appear in the parser output due to horizontal Markovisation which allows the parser to create new productions.

5.4 Using a Distorted Treebank

In many NLP applications, it is desirable that errors including grammatical errors do not only not break the system but also do not change the system's behaviour. For a parser, this can mean that it should output a parse tree as similar as possible to the parse tree it would produce for the corrected input.[19] To build resources for evaluating such robust parsers, Foster (2007a) introduces the idea of automatically generating a treebank of ungrammatical sentences from a vanilla treebank like the PTB. The procedure works in two steps: firstly, an error is inserted into the yield of each tree using an automatic error creation procedure.[20] The parse trees are then adapted according to the edit operations involved in creating the errors with minimal changes and as close as possible to the terminals.

Figure 5.7 shows a PTB tree and four trees for ungrammatical sentences derived from it. For a missing word error, Foster (2007a) deletes a token, its pre-terminal and any parent nodes up to a node that has more than one daughter (here: NP-SBJ). Extra words need a reference token either to the left or the right that guides the attachment of the pre-terminal of the extra word. In case of repeated word errors or double syntactic function errors (or at the start or end of the sentence), it is clear which token will be the reference token. For other extra word errors, two trees are produced if the the pre-terminal to the left and the pre-terminal to the right have different parents.[21] For errors that only substitute a token in the sentence, e. g. real-word spelling errors, agreement errors and verb form errors, we only substitute the token in the parse tree and leave the pre-terminal

[19] The alternative approach would be to add detailed information on the error to the parse tree and to add support for these annotations in all components that use the parser's output.

[20] Foster (2007a) uses an early version of the error creation procedure described in Chapter 3.

[21] Foster (2007a) implements a Parseval measure that calculates scores for multiple reference gold trees and picks the reference trees with the highest f-score for overall evaluation, similarly to the BLEU score measure in machine translation that can use multiple reference translations.

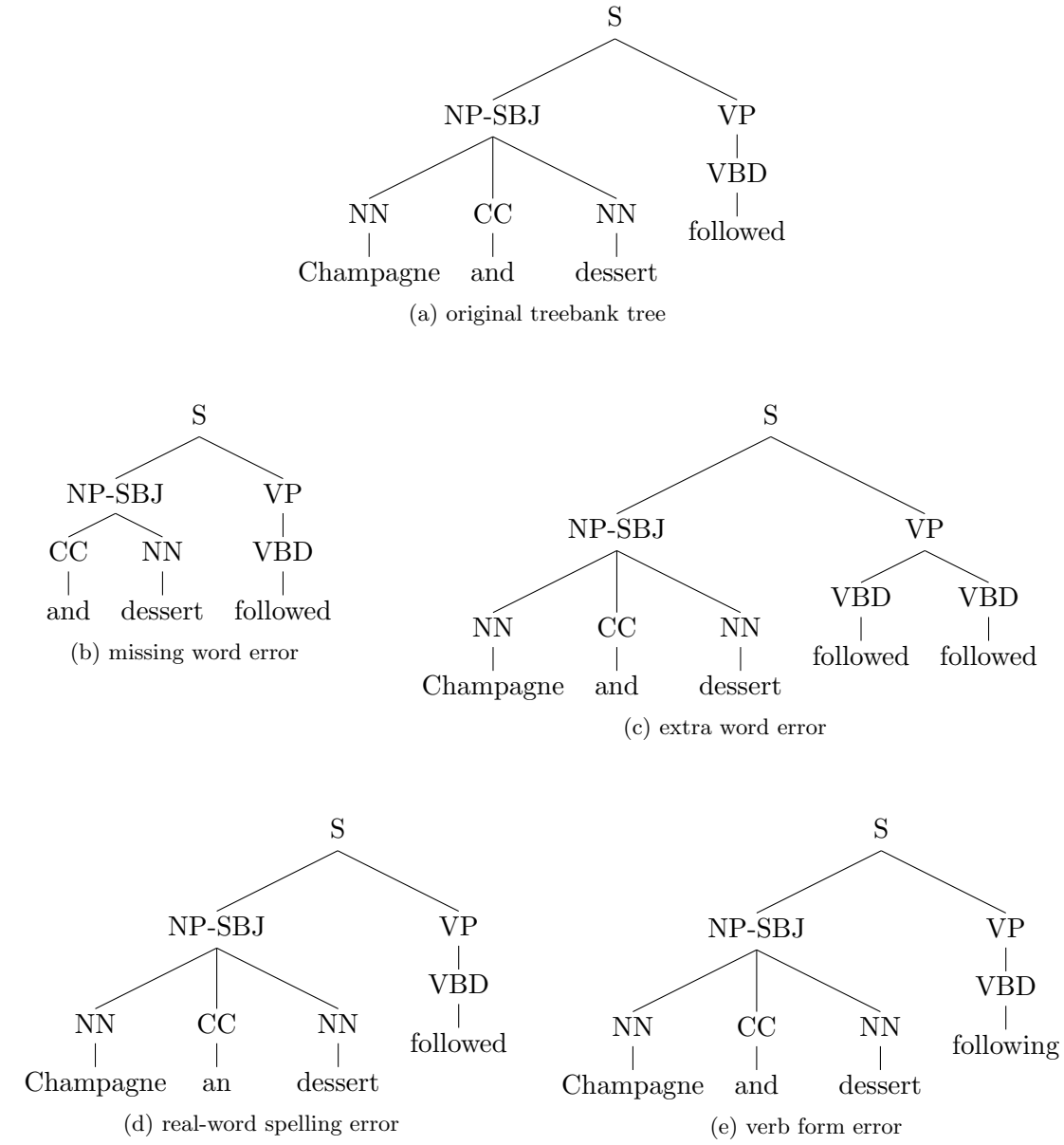


Figure 5.7: Automatic insertion of errors into treebank trees (a sentence from WSJ Section 00, top node and punctuation omitted); agreement errors cannot be created in past tense.

unchanged. It is not possible to insert an agreement error into the example sentence in Figure 5.7, but we include a general verb form error.[22] Note that the pre-terminal stays VBD, reflecting the intended meaning of the sentence.

The resulting treebank is a distorted version of the input treebank. While Foster (2007a) uses the distorted treebank for parser evaluation, Foster (2007b) extends this work and induces grammars that can analyse ungrammatical input. She find that the accuracy of parse results improves for ungrammatical input but deteriorates for grammatical sentences. Therefore, we add a classifier to decide whether the input is grammatical and then use the better suited of the two grammars, i. e. the regular grammar or the one induced from the distorted treebank (Foster et al., 2008). Two classifiers are tested: a decision tree with POS n-gram features as in Chapter 6 and a classifier that parses the input with both grammars and then picks the output with the highest parse probability. In this section, the latter classifier is evaluated on our BNC test data in the task of detecting ungrammatical sentences and we add a probability offset to the classifier in order to be able to tune the accuracy trade-off.

5.4.1 An Instance of the APP/EPP Method

While we develop the grammaticality classifier based on distorted treebank probabilities (Foster et al., 2008) independently of the APP/EPP method presented in Chapter 4, we notice in retrospect that the two methods have in common that they compare the parse probability obtained with a vanilla treebank grammar to the output of another probability model. The difference lies in the reference probability model. The distorted treebank method directly uses the parse probability with the distorted treebank grammar as a reference model (or EPP model in the terminology of Chapter 4), while the APP/EPP method as we implemented it in Chapter 4 retrieves reference sentences from a large corpus. However, in the wider sense outlined in Section 4.3.1, the distorted treebank method is an instance of the APP/EPP method.

Figure 5.8 shows the distorted treebank method in a diagram derived from an APP/EPP illustration in Chapter 4. The vanilla parse probability (VPP) corresponds to the actual

[22] Verb form errors were added to the error creation procedure later (Foster, 2007b).

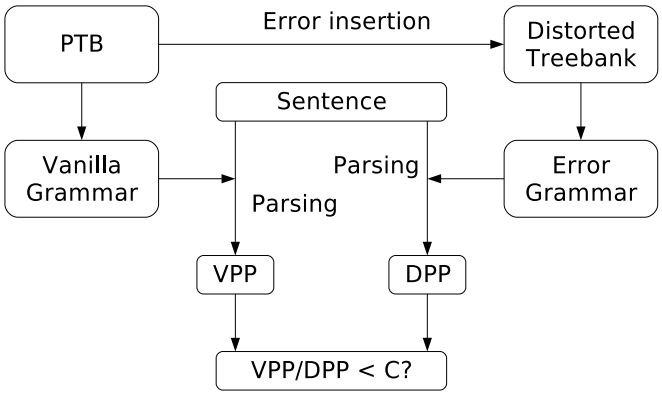


Figure 5.8: Rendering the distorted treebank method as an instance of the APP/EPP method; VPP = vanilla parse probability, DPP = distorted parse probability; compare with Figure 4.7 in Chapter 4 (p. 90)

parse probability (APP) of the APP/EPP method and the distorted parse probability (DPP) takes the role of the estimated parse probability (EPP). A threshold C is added to the method in the same way as for the APP/EPP method. In our work on accurate robust parsing (Foster et al., 2008), we implicitly set C = 1, or log(C) = 0, as we test for VPP < DPP.

5.4.2 Related Work

Artificial ungrammatical data has been used before to automatically induce linguistic models: in Chapter 4, we point to the discriminative language model of Okanohara and Tsujii (2007) and the anti-language model of Stolcke et al. (2000). However, to our knowledge, the induction of a probabilistic grammar from a treebank of ungrammatical sentences has only been attempted by Foster (2007b) and in our work (Foster et al., 2008). Similarly, there is previous work that automatically inserts errors into text to generate ungrammatical data (see Chapters 2 and 3) but the automatic generation of an ungrammatical treebank is also new to the aformentioned work.

Since the distorted treebank method is an instance of the APP/EPP method, most references given in Chapter 4 also apply here. In particular, we would like to bring to attention that Lee and Seneff (2006) also use parse probabilities to classify sentences as either grammatical or ungrammatical. However, they employ one grammar to compare multiple candidate corrections while we parse one sentence with multiple grammars.

Wong and Dras (2010) replicate the parsing and classification experiments of Foster et al. (2008) with the Stanford parser instead of the first-stage parser of Charniak and Johnson (2005)'s reranking parser. Overall, the accuracy of the Stanford parser is lower in the four configurations replicated and for all three test sets (vanilla treebank, distorted treebank, double distorted treebank).[23] However, the order of f-score results is different, e. g. training on the union of vanilla and distorted treebank degrades results on all three test sets. Wong and Dras (2010) combine the parse probabilities (VPP and DPP in our terminology) with a support vector machine and get accuracy figures similar to the simpler threshold-based classifier of Foster et al. (2008) — see also Section 6.2.4 and 6.4.1 of Chapter 6 for more details on this work.

In the broader context of our work on accurate robust parsing (Foster et al., 2008) where we aim to obtain robust parses that more accurately represent the intended meaning than parses of "regular" grammars induced from treebanks of grammatical language, the idea of training a parser on parse trees of ungrammatical sentences is related to self-training for domain adaptation (Bacchiani et al., 2006; Foster et al., 2007; van der Plas et al., 2009). Self-training can improve parse results (McClosky et al., 2006). However, we do not train on parser output but on distorted trees derived from gold trees which we expect to be better training material. See also Section 5.5.2.

The idea of processing input with multiple systems and then picking the (presumably) best output is also known from parsing, machine translation and other areas of NLP.[24] Henderson and Brill (1999) call the method parser switching. They evaluate two variants using a scoring function based on the similarity of the set of constituents and a probability model derived from the union of the candidate parses. The probability output of the individual parsers is not used. Henderson and Brill (1999) also test two methods for combining substructures of the candidate parses to form a new, better parse and get higher precision and f-score at the price of lower recall. Consequently, it is not surprising that later work focuses on combining parts of parse trees. Sagae and Lavie (2006) fill a chart with (label, start, end, weight) tuples obtained from multiple parsers and then parse bottom-up maximising weight. This is similar, but not identical, to inducing a

[23] The treebank is distorted twice by applying the distortion procedure to the distorted treebank again.
[24] See also Section 6.5.2 on system combination with voting in Chapter 6.

mini-grammar from the output of the initial parsers and parsing the input again with this specialised grammar. Fossum and Knight (2009) go in this direction as they combine productions found in the candidate parses instead of constituents. In addition, they select parses using Minimum Bayes Risk and extend the work to n-best parsing.

Way (2010) reviews the concept of multi-engine machine translation saying that "the best output from a number of MT hypotheses" is selected. However, the various architectures he summarises do not simply choose the output based on probability models or confidence scores, or, if they do, they combine parts of the outputs to form new output. In contrast, our accurate robust parsing work simply selects one of the parsers' outputs as a whole.

5.4.3 Experimental Setup

We parse our artificial BNC test data with the first-stage parser of Charniak and Johnson (2005)'s reranking parser as in Chapter 4 and three grammars:

- the vanilla treebank grammar induced from the PTB sections 2-21 as shipped with Charniak and Johnson (2005)'s reranking parser,
- a distorted grammar induced from the error treebank derived from PTB WSJ Sections 2-21, applying the error creation procedure twice to four sections[25] (Foster et al., 2008), and
- a distorted grammar induced from the union of the training data of the above two grammars (Foster et al., 2008).

The two possible combinations of the vanilla grammar and a distorted grammar are considered for the method. We calculate grammaticality scores log(VPP) - log(DPP) and apply thresholds log(C) to classify a test sentence as ungrammatical if $\log(\text{VPP}) - \log(\text{DPP}) < \log(\text{C})$.[26]

[25] (Foster et al., 2008) break down parsing results by clean, noisy and noisiest test data, the latter having two errors per sentence, and decided to also include more noisy data for treebank induction. For our purposes, we do not expect any disadvantages from the more noisy data as the errors often will be in different subtrees.

[26] If a sentence is not parsed (either because of the limit of 100 tokens that we imposed in order to parse the corpus more quickly or because of a parse failure), we assign a logarithmic parse probability between -2,000.001 and -2,000 which is less than the lowest observed value (-1,914.72). Random noise ensures that

The resulting classifiers are evaluated on 10 disjoint subsets of the test data for better comparison with methods requiring a cross-validation setup. However, since no training is involved and accuracy combines linearly, the average of accuracy corresponds to accuracy on the full test set. In addition to measuring accuracy of the classification, we also measure statistical properties of the distributions of probability values as in Chapter 4: mean square error (or here better called mean square difference) between log(VPP) and log(DPP) for grammatical test data, mean and variance on grammatical and ungrammatical data and σ-gap.

5.4.4 Results

We report properties of the distribution of log(VPP/DPP) grammaticality scores followed by accuracy curves of the error detection method.

Grammaticality Score Distribution

Tables 5.5a to 5.5c show statistical properties of grammaticality scores for grammatical and ungrammatical test data with the measures used in Chapter 4, as well as the accuracy of the classifiers that sets C such that the same accuracy is reached for grammatical and ungrammatical test data. Compared with Table 4.6 in Chapter 4 (p. 123), the variance of grammaticality scores of the distorted treebank method is higher and the sigma gap is smaller (while a large gap is desirable). Accuracy, however, is within the range of results obtained for variants of the APP/EPP method in Chapter 4.

The distribution shown in Figure 5.9 suggests that the variance of grammaticality scores of the distorted treebank method should be lower than the variance found in Chapter 4. A breakdown by parse failures and sentence length reveals that the 233 grammatical sentences that can only be parsed with one of the two grammars are extreme outliers and have a mean square error of over 1.8 million which is enough to affect the overall mean square error of the 2 million test sentences substantially. Table 5.5b shows the results without these test sentences and also excludes 3,703 sentences that cannot be parsed at all. Here, the mean square error is much lower than in Chapter 4. However, accuracy is almost

the grammaticality scores of affected sentences are unlikely to be exactly 0 avoiding a big jump of results as the threshold C passes this point.

Measure	Min	Avg	Max	Std. Dev.
MSE	188.8	242.6	304.3	38.3
Sigma gap	-31.53	-27.75	-24.29	2.43
Accuracy	60.61%	60.69%	60.77%	0.05%
μ_{G}	2.60	2.63	2.67	0.02
μ_{U}	-0.51	-0.45	-0.40	0.03
σ^2_{G}	182.03	235.67	297.30	38.25
σ^2_{U}	192.03	242.96	302.52	45.25

(a) with mixed grammar and all test data

Measure	Min	Avg	Max	Std. Dev.
MSE	29.9	30.2	30.4	0.2
Sigma gap	-8.51	-8.46	-8.42	0.02
Accuracy	60.63%	60.71%	60.79%	0.05
μ_{G}	2.45	2.48	2.50	0.02
μ_{U}	-0.62	-0.59	-0.55	0.02
σ^2_{G}	23.85	24.09	24.33	0.14
σ^2_{U}	43.50	43.89	44.07	0.19

(b) mixed grammar and excluding test sentences with parse failure(s)

Measure	Min	Avg	Max	Std. Dev.
MSE	238.9	280.2	332.4	29.8
Sigma gap	-32.75	-28.72	-26.53	1.93
Accuracy	60.70%	60.77%	60.79%	0.03%
μ_{G}	4.76	4.78	4.83	0.02
μ_{U}	0.92	0.95	1.03	0.04
σ^2_{G}	216.29	257.29	309.12	29.69
σ^2_{U}	239.48	274.68	360.04	40.18

(c) with error grammar trained on ungrammatical sentences only

Table 5.5: Evaluation results on test data in 10 cross-validation runs with measures as in Chapter 4; compare with Table 4.6 in Chapter 4 (p. 123). Note that MSE $= \sigma^2_{\mathrm{G}} + \mu^2_{\mathrm{G}}$.

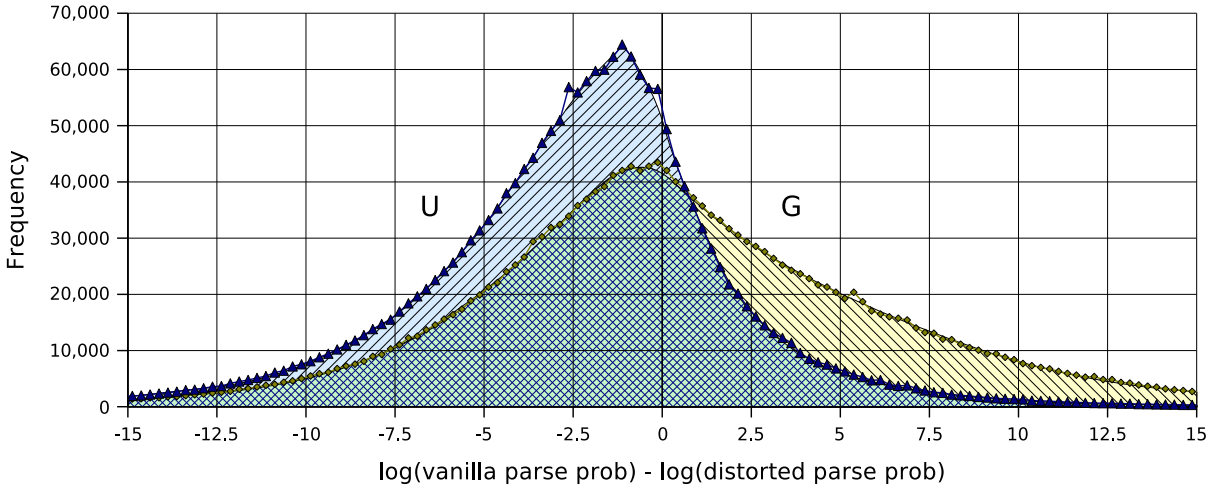


Figure 5.9: Distribution of grammaticality scores (difference of logarithmic parse probabilities of the vanilla treebank grammar and the distorted treebank grammar) for ungrammatical (U) and grammatical (G) test data; compare with Figure 4.19 in Chapter 4 (p. 125). Note that the range of scores is only half as wide as in Chapter 4.

the same. Table 5.5c shows higher MSE and variances for the other grammar combination that uses only ungrammatical data for the induction of the distorted grammar.

Figure 5.9 also shows an interesting shape of the distribution of grammaticality scores for ungrammatical test sentences (U) that is distinct from a Gaussian distribution typically caused by noise. This may indicate that some errors types have a constant effect on the grammaticality score log(VPP/DPP).

Accuracy Curve

The accuracy figures above are for the value of C that results in a classifier that has identical accuracy on grammatical and ungrammatical test data. Figure 5.10 shows the accuracy curve parameterised by C. For example, if the accuracy on grammatical data has to be 90% we would achieve approximately 33.77% accuracy on ungrammatical data. At this accuracy trade-off, this is by far the best result of Chapters 4 and 5. We will compare this method to other methods in more detail in Chapter 7. The accuracy reported in Tables 5.5a and 5.5c can be seen at the intersection of the curve with the bisecting line (not shown). Note that the two curves are almost identical.

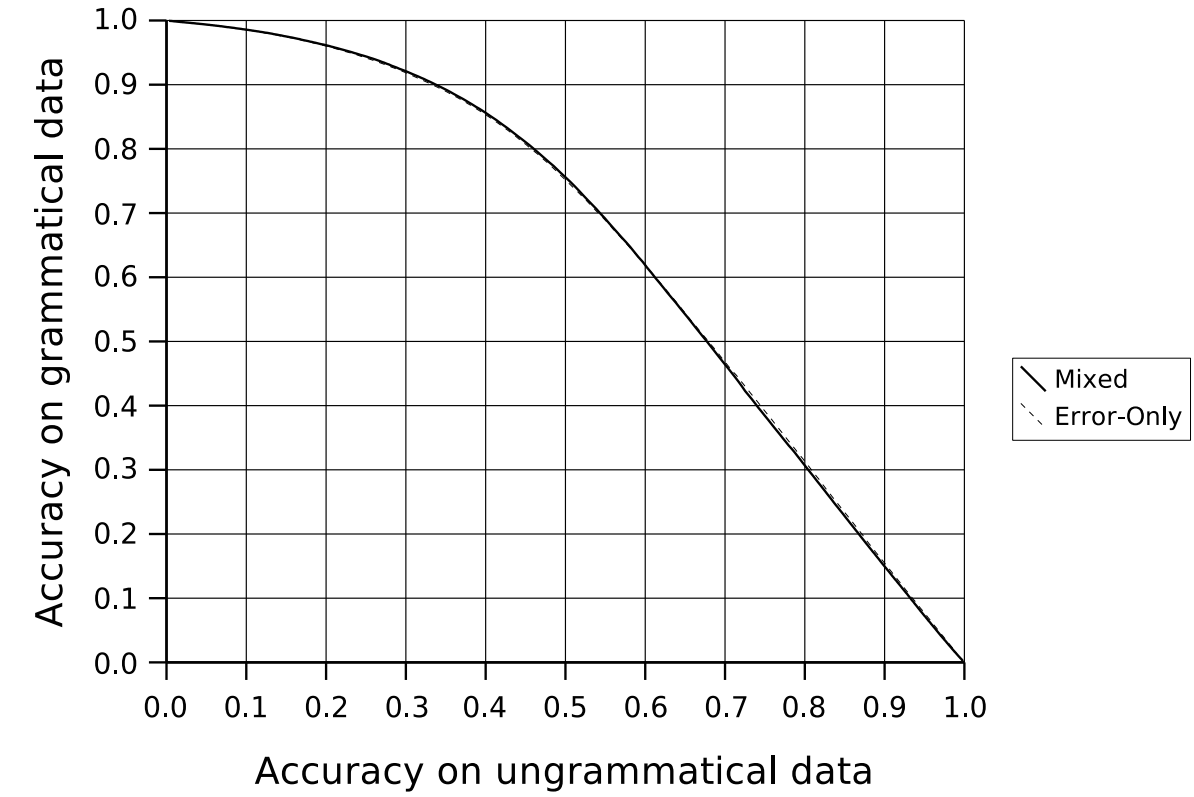


Figure 5.10: Accuracy curves of the distorted treebank method for two different distorted treebank grammars: the grammar derived from the union of the vanilla treebank and the error treebank (mixed) and the grammar derived only from ungrammatical sentences (error only). The probability offset C runs from e^{-25} to e^{25}. Note that the two curves are almost identical.

5.5 Summary and Future Work

We have evaluated four basic methods to judge a sentence as either grammatical or ungrammatical that do not rely on machine learning to set their parameters: *(a)* parsing with a hand-crafted precision grammar, *(b)* flagging unattested or rare part-of-speech n-grams, *(c)* pruning rare rules from PCFGs in an attempt to make them more discriminative, and *(d)* the distorted treebank method that compares parse results with vanilla and error-distorted treebank grammars. In the following, we point to future work expanding this evaluation.

5.5.1 Skipgrams

Sun et al. (2007) introduce the frequency ratio of non-continuous sequential patterns, i. e. skipgrams, between positive and negative reference data as a measure of the discriminativeness of a pattern. While they employ machine learning, we would like to see how a simple method based on this ratio (and possibly a confidence threshold) would perform in the task of judging the grammaticality of a sentence. See also Section 6.6.4 of Chapter 6.

5.5.2 Self-Training of Parser and the Distorted Treebank Grammar

All methods involving probabilistic treebank grammars could be implemented with self-trained parsers, i. e. parsers that have been trained on their own output, preferably on a larger amount of target domain data, here the BNC.

Parsing Grammatical Text and Adapting the Trees

The canonical way to obtain a distorted treebank for self-training would be to first parse grammatical sentences with the initial parser and then to apply the error insertion procedure with tree adaptation. In case of a parallel error corpus with authentic errors, one could also first parse the corrections and then apply the observed authentic errors to the trees, basically guiding the error insertion procedure where to make which edit operation.

Parsing Ungrammatical Text Directly

However, corrections of authentic errors that can guide the creation of gold trees are not always available. A question to explore would be if the distorted treebank method also works well if we parse (authentic) ungrammatical sentences with the initial parser to obtain the distorted treebank for self-training. We suspect that important mal-rules that the error insertion procedure will manifest in its output trees will not be produced by the initial parser given the corresponding ungrammatical strings. Nevertheless, the output trees might sufficiently differ from trees of grammatical input so that the distorted treebank method may work.

5.5.3 Distorted Grammar Probability and the APP/EPP Method

Another interesting question is whether the distorted treebank method can be improved by retrieving a correction factor from a reference corpus in the same way Section 4.5.8 proposes to improve LM predictions.

Chapter 6

Improving and Combining Classifiers with Machine Learning

The basic methods presented in Chapter 5 use individual features of the input sentence like its parsability or the presense of a rare event to classify it as either grammatical or ungrammatical. In this chapter, we build more complex classifiers that draw from additional features of the input sentence and that use compound decision rules instead of thresholds on a single feature. To handle the larger parameter spaces, we adopt a machine learning method, decision tree learning, to choose parameters. Section 6.1 explains how this learning method works. In Section 6.3, we apply decision tree learning to the individual methods of Chapter 5, moderately extending each feature set as described in Section 6.2. Then, in Section 6.4, we combine the feature sets of the individual methods step by step up to the full feature set to build decision trees that use various sources of information. Finally, we tune the accuracy trade-off with voting over multiple classifiers in Section 6.5.

6.1 Decision Trees

There are many different machine learning methods that can be used to automatically induce classifiers. Choosing one is not easy because performance is difficult to predict for new tasks. In order to successfully run experiments, it is also important that the method can be easily applied without time-consuming tuning of meta-parameters. In Chapter 4,

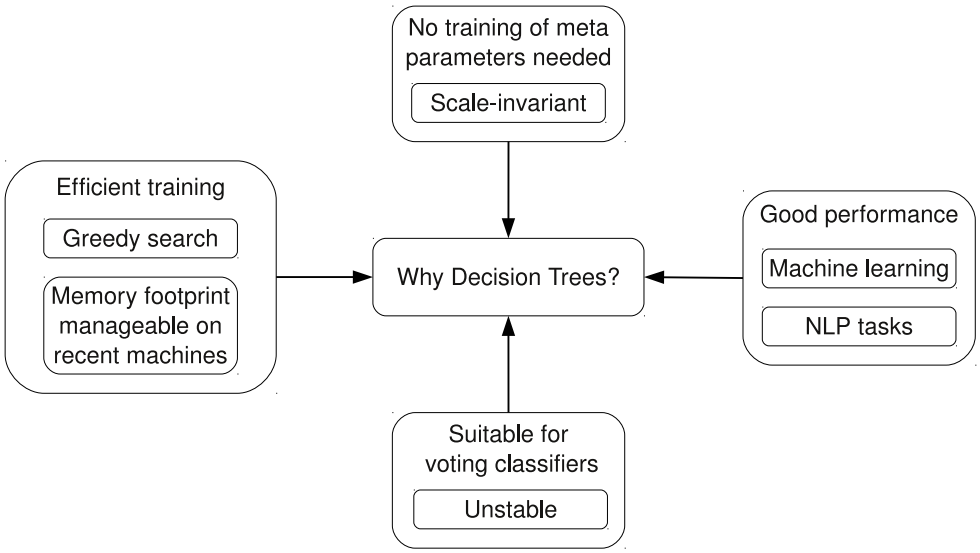


Figure 6.1: Reasons for choosing decision trees for the experiments with machine learning

we used the k-NN machine learning method to predict probabilities (real numbers) and we needed a substantial amount of computing resources to tune the model's parameters. The k-NN method could also be used in a binary classification task but would require the same optimisation of parameters. To avoid time-consuming optimisation, we choose decision trees which are scale-invariant and can be efficiently trained with a greedy search.[1] Decision trees perform well in many tasks (Breiman et al., 1984; Quinlan, 1993; Mitchell, 1997; Rokach and Maimon, 2007) and have been applied to a wide range of NLP tasks (Schmid, 2010). Finally, decision trees are particularly suitable for classifier combination via voting, a method we use in Section 6.5 in order to tune the accuracy trade-off of our classifiers. Figure 6.1 summarises the advantages of decision trees pointed out in this paragraph.

6.1.1 The Model and its Parameters

A decision tree is essentially a tree data structure storing a complex decision rule. Each leaf node of a decision tree is labelled with one of the target classes of the classification task. In our case there are two classes: grammatical and ungrammatical. Each non-terminal

[1]The decision tree software we use (see Section 6.3.1) further requires that the training data can be stored in main memory. At the time the experiments were carried out, we only had access to machines that can store half of our training data for our biggest feature set. We therefore limited our experiments accordingly, see Section 6.3.1 below.

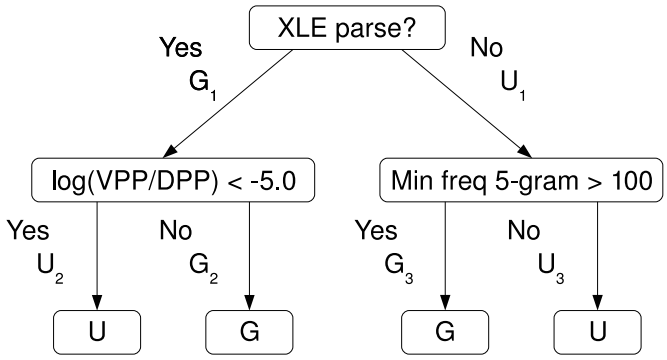


Figure 6.2: A manually written decision tree that refines XLE's grammaticality judgements with information from the distorted treebank and n-gram methods

node maps feature values to its daughter nodes. For example, real-valued features are usually compared to a threshold, the first daughter covers smaller values and the second daughter covers greater or equal values. Classification proceeds top-down from the root node walking towards a leaf node: at each node, the mapping from feature values to daughters directs the walk. If a leaf node is reached, its class label is the output of the classifier.

Figure 6.2 shows an example tree with three non-terminal nodes and four leaf nodes. It is a binary decision tree as each non-terminal nodes has two daughters. The root node tests whether the input sentence can be parsed with the core ParGram English LFG like the precision grammar method of Chapter 5. This splits our test data into two sets G_1 and U_1. The left daughter splits G_1 further into G_2 and U_2 using the distorted grammar method (Chapter 5) tuned for high accuracy on grammatical data ($C = e^{-5}$), i. e. the decision of the root node to classify an item as grammatical is only revised if the distorted grammar method is highly confident that the input is ungrammatical. Similarly, the root node's right daughter consults the part-of-speech 5-gram method of Chapter 5. The overall decision rule can be written as a disjunction of conjunctions where each conjunct corresponds to a path from the root node to a leaf node labelled G: a sentence is classified as grammatical by the example tree in Figure 6.2 if and only if the sentence can be parsed with the core LFG and has a distorted treebank grammaticality score[2] of -5 or higher, or if the sentence cannot be parsed with the core LFG but has a part-of-speech 5-gram

[2]This is log(VPP/DPP) where VPP is the vanilla parse probability and DPP the distorted treebank parse probability — see Section 5.4.1 of Chapter 5.

grammaticality score over 100.[3]

Decision trees can have a large number of nodes and an arbitrarily branching structure. Since each node can freely choose a feature, the number of combinations is exponential in the number of nodes which in turn is exponential in the height of the tree.[4] Real-valued features can be repeatedly compared to different thresholds, partitioning the feature space into regions with axis-parallel boundaries. Consequently, the parameter space of decision trees is huge and testing every possible tree is not feasible.

6.1.2 Top-Down Induction of Decision Trees

In order to find a good decision tree, the common approach is to run a recursive greedy search starting at the root node: the feature, the number of daughters and the mapping from feature values to daughters are optimised locally assuming that either the current node is a pre-terminal node or that any potential subtree will have no effect on the optimality of the current choice. Once a node is configured, the training data is split according to its mapping and for each daughter a subtree is built by applying the same procedure to the respective subset of the training data, unless a stopping criterion is reached which triggers the creation of a leaf node that is labelled according to the majority class of the training data subset assigned to it.

The commonly used decision tree induction algorithms differ with respect to the local optimisation criterion that splits the data at each node. Murthy (1996) surveys splitting criteria used in the literature and distinguishes two families of criteria: those based on information-theoretic entropy and those that compare the frequency distribution of classes.[5] In addition, Murthy (1996) investigates the effect of broadening the search by looking ahead one level and the effect of allowing linear combinations of features to be tested at non-terminal nodes.

It is not obvious how the splitting criteria impact the accuracy of the classifier. How-

[3]Note that while the example tree has been chosen intuitively for illustrative purposes without trying alternative configurations or thresholds, the resulting classifier achieves the accuracy point (61.12%, 64.65%) which is already above the accuracy curves of all methods of Chapters 4 and 5.

[4]The number of features appears in the base of the exponentiation. We assume that each feature can be re-used with a a new threshold even if it has been used in a node on the path from the current node to the root node and we do not count two decision trees as different if they only differ in threshold. (Otherwise, the number of possible decision trees is infinite even with a restriction to single node trees.)

[5]A newer overview of splitting criteria is provided by Rokach and Maimon (2007).

ever, the fact that the leaf nodes are labelled according to class majority suggests that there will be a bias towards high accuracy on test data with a uniform class distribution. The leaf nodes of a decision tree can also record the class distribution instead of just the majority class, providing probability estimates for each class conditioned on the path from the root (Breiman et al., 1984).[6]

Stopping criteria for the recursion can be the number of remaining training items, the size of the largest class within the remaining items, or the depth of the node. Finding the right size for a decision tree is important in order to exploit as much of the training data information as possible without over-fitting it. It is difficult to say when to stop while growing a tree top-down. Therefore, Quinlan (1993) firstly fits the training data as well as possible and then prunes the tree as long as it reduces the misclassification rate measured on held-out data. Section 6.3.1 describes the type of decision tree learning algorithm we use.

6.2 Feature Sets

Each classifier of Chapter 5 uses exactly one feature, e. g. the ratio of vanilla and distorted parse probability in the case of the distorted treebank method or the parsability with respect to the ParGram English LFG in the case of the precision grammar method. These classifiers correspond to decision trees with just one non-terminal node. In this section, we present the feature sets that we will use in order to build bigger decision trees. The feature sets are grouped according to the particular methods of Chapter 5, as we first augment each method individually and then combine methods in the following two sections.

Additional features range from side products of the process of producing a primary feature that is already present in Chapter 5, e. g. parsing statistics for parsability, over new but related features, e. g. measurements of the structural differences between parses obtained with vanilla and distorted grammars, to features derived from basic features, e. g. logarithmic parse probability normalised by sentence length. The latter is an example of how we try to expose information on the grammaticality of a sentence more clearly with

[6]Such probability estimates can be used as grammaticality scores. Applying other thresholds than 50% majority, the accuracy trade-off can be tuned — see Section 6.6.3.

derived features. Breiman et al. (1984), pp. 138–140, discuss the benefit of adding the latter type of features to decision trees.

We restrict ourselves to features related to the basic methods of Chapter 5 because the aim of this chapter is to shed light on how well the methods complement each other when combined in a machine learning framework. The primary reason why we extend the feature sets is to provide a basis for comparison and to see how much decision tree learning improves the individual methods.[7]

6.2.1 XLE Features

As in Chapter 5, we use the XLE command "parse-testfile" with the ParGram English LFG (Maxwell and Kaplan, 1996; Butt et al., 2002). In this mode, the parser outputs for each sentence the number of "optimal" and "unoptimal" parses,[8] the time it took to parse the sentence, the number of "subtrees" processed during parsing and the number of words. We add a feature for each of these numbers.

The XLE parser reports parser exceptions such as time-outs and out-of-memory using a negative number of solutions. In order to separate nominal from ordinal features, we move such error reports to a separate feature ("parse result" in the list below) and set the number of optimal and unoptimal parses to 0. If parsing succeeds, we use the nominal feature "parse result" to record whether XLE had to resort to its robustness techniques such as parse-fitting and error anticipation in order to parse the sentence. XLE marks such sentences with an asterisk (*).[9] In total, we extract six features:

- number of optimal parses
- number of unoptimal parses

[7] Secondarily, we add more features in order to obtain better classifiers with minimal effort. We limit our efforts in this regard as the absolute performance of classifiers does not contribute to answering our research questions. A final system for real-world applications may well have to incorporate more features, e. g. skipgrams (Sun et al., 2007) and parse tree fragments (Post, 2011), and use a suitable machine learning method that can deal with large feature sets.

[8] We keep XLE's terminology here to make clear that this is not a distinction between one or more highest scoring parses and all remaining suboptimal parses. Optimal parses are all parses that only use rules with an optimality ranking above ungrammatical, not just the parse(s) with highest optimality. This includes parses that use a rule with a dispreference mark (Frank et al., 1998). Correspondingly, unoptimal parses are parses that contain a rule that is marked as ungrammatical or "nogood".

[9] Note that the presense of an asterisk (and optionally also the presense of a parse failure) is the feature that we use in the basic classifier in Chapter 5.

- parsing time
- number of subtrees
- sentence length in words
- parse result (negative: parser exception, 0: parse with core grammar, 1: parse with robustness techniques)

6.2.2 Part-of-Speech n-gram Features

We use the part-of-speech n-gram features of Chapter 5, i. e. the frequencies (in grammatical reference data) of the least frequent n-grams in the sentence, with n ranging from 2 to 7. The difference between this and the n-gram method of Chapter 5 is that the decision trees can consult multiple features instead of having to choose one. The frequencies are obtained from a reference corpus of well-formed sentences, a subset of the British National Corpus (Burnard, 2000).[10]

6.2.3 Distorted Treebank Features

As in Chapter 5, the input sentence is parsed with three probabilistic grammars: the vanilla grammar induced from the original treebank, the distorted grammar induced from the distorted treebank, and the mixed grammar induced from the joined treebank. The vanilla grammar produces better parse results for grammatical sentences while the other two grammars are more suitable for ungrammatical input (Foster, 2007b), and these differences are reflected in the parse probabilities assigned by the probabilistic grammars (Foster et al., 2008). Therefore, we extract probabilistic and structural features of the parse results obtained with the three grammars (number of features in brackets):

- logarithmic parse probability of the best parse for each grammar (3)
- sentence length in tokens (1)
- number of non-terminal nodes in the parse tree (3)

[10] Note that padded n-gram frequencies that gave somewhat better results in Chapter 5 are missing from the n-gram feature set.

- height of the parse tree (3)
- difference in logarithmic parse probability between the first and 50th-best parse tree (3)
- number of parse results in 50-best output (usually 50, except for very short sentences) (3)

In addition, we use parser evaluation metrics to compare the best parse produced with each pair of grammars. One parse result is used as the gold tree and the other parse tree as the test tree. In the case of the leaf-ancestor metric (Sampson and Babarczy, 2003), we see an opportunity to detect relevant local differences instead of a global average difference: the vanilla leaf-ancestor metric calculates a score for each token and then outputs the average score. We replace the arithmetric average with (a) the geometric average and (b) the minimum. Using the minimum is similar to measuring the frequency of the rarest n-gram in our n-gram method instead of using the overall score assigned by an n-gram language model. The geometric average lies between the minimum and the arithmetric average. The features are (number of features in brackets):

- parseval precision, recall and f-score (Black et al., 1991) (9 features)[11]
- leaf-ancestor score (Sampson and Babarczy, 2003) (3)
- leaf-ancestor score using geometric average of lineage scores instead of arithmetic average (3)
- score of lowest-scoring lineage (3)
- leaf-ancestor score using longest common subsequence score for lineages, i. e. the number of shared nodes on the lineage divided by the length of the longer lineage, instead of edit distance (3)

Due to the fact that a decision tree can only represent axis-parallel decision boundaries accurately and has to approximate other boundaries with stepwise approximations, we

[11] Note that precision and recall swap roles when changing the direction of measurement and that f-score is symmetric.

derive additional features that exhibit important feature relations more clearly (number of features in brackets):

- difference in logarithmic parse probability between grammar pairs (3)
- difference in number of non-terminal nodes (3)
- difference in height of parse tree (3)
- difference in difference in logarithmic parse probability between first and 50th-best parse tree (3)

Since logarithmic parse probability and sentence length are strongly correlated, we provide features to exploit deviations from this correlation:

- logarithmic parse probability normalised by sentence length (3)
- the coordinates of the point (logarithmic parse probability, sentence length) after scaling the axes to fit the training data to the bisecting line and then rotating by 45 degrees (3)

Wagner et al. (2009) report experimental results using the above distorted treebank features.[12]

6.2.4 Discriminative Rule Features

While Section 5.3.3 of Chapter 5 argues that the frequency of a rule[13] in grammatical reference data is a weak indicator of grammaticality, we might obtain a useful indicator if we contrast the frequency in grammatical data with the frequency in ungrammatical data and derive the discriminativeness of each rule similarly to Sun et al. (2007) who measure the "discriminating ability" of sequential patterns. In the latter work, the presense of individual sequential patterns is used as features to train a support vector machine (SVM) and the measure of discriminativeness is only used to reduce the feature set from very large numbers to a few thousands (depending on the type of learner data used for training).

[12] Preliminary results with a subset of these features have been presented earlier (Wagner et al., 2008).

[13] We use the term *rule* to refer to a subtree of height 2, i. e. a node with all its daughters, that has been observed in parser output. Such rules are not necessarily part of the grammar that was used in parsing nor do they have to be in the format of the grammar.

Rule Set	Ratio	TG	TU
Top 20	above 10	95	1,744
Top 100	2.67–10	1,675	8,353
Top 250	1.58–2.67	9,012	23,132
Top 500	1.215–1.58	58,204	88,042
Bottom 50	below 0.723	3,799	2,168
PA Top 35	above 11.4	60	1,816
PA Top 160	2.76–11.4	1,919	9,927
PA Top 350	1.68–2.76	9,088	24,662
PA Top 800	1.211–1.68	62,725	96,036
PA Bot. 80	below 0.66	3,798	1,798

Table 6.1: Discriminative rule sets: PA=with parent annotation, ratio=range of frequency ratios of rules in this set, TG=number of instances in 100,000 grammatical sentences, TU=number of instances in 100,000 ungrammatical sentences

Recently, Wong and Dras (2010) train an SVM on a combination of parse probability features (see Section 5.4.2 of Chapter 5) and individual rule features. Like Sun et al. (2007), they have to reduce the number of features to make training of the SVM feasible. Therefore, Wong and Dras (2010) explore a number of feature selection strategies. In addition, their work differs from ours in that they use an accurate robust parser that combines parsers trained on grammatical and ungrammatical treebanks with a classifier, similarly to Foster et al. (2008) — see also Section 5.4 of Chapter 5. Wong and Dras (2010) use between 100 and 10,000 rule features depending on the feature selection strategy.

We have to reduce the number of features even further for our decision trees. Therefore, we aggregate them by counting the number of (non-terminal) rules that are in a set of the most discriminating rules. We consider ten such sets as shown in Table 6.1: eight sets that contain rules that are most likely to occur in ungrammatical sentences (first four rows of each section of Table 6.1) and two sets of rules that are indicative of grammatical sentences (bottom row of each section of Table 6.1).[14] The sets vary in size and in whether we annotate each (non-root) node with its parent category before rules are extracted from the treebank or a parse result. For example, in 99% of parent-annotated versions, the rule `NP -> DT NN` turns into one of

- `NP^VP -> DT^NP NN^NP` (48%),

[14]The frequencies are counted in held-out data, see also Section 6.3.1.

- `NP^S -> DT^NP NN^NP` (25%),
- `NP^PP -> DT^NP NN^NP` (14%) or
- `NP^NP -> DT^NP NN^NP` (12%),

depending on its context. It is important to note that the grammar is not changed. We do not re-train the parser on a parent-annotated treebank. The parent annotation only affects the feature extraction. To determine these sets, we extract rule frequencies from parsed grammatical and ungrammatical sentences taken from our BNC-derived artificial error corpus (100,000 sentences each; see Chapter 3), and score each rule with the frequency ratio. Rules that appear less than 20 times in total are discarded because they would be unreliable indicators and unnecessarily increase the size of the rule sets.[15, 16]

In addition to empirically chosen sets of discriminative rules, we also include two Markovisation rule sets (again with and without parent annotation). These are the infinite sets of all rules that do not appear in the treebank the grammar is induced from. Such rules can still appear in the parser's output because Charniak's history-based parser creates rules with a Markovisation process that can produce unattested rules[17] (see Section 5.3.4). In total, we consider 12 sets of rules (10 as of Table 6.1 and two Markovisation rule sets). Based on these 12 rule sets, we extract the following 88 features (breakdown of number of features in brackets):

- raw number of Markovisation rules (2)
- number of Markovisation rules normalised by sentence length, total number of rules, number of non-terminal nodes ($2 \times 3 = 6$)
- number of Markovisation rules normalised by the square roots of the above three values ($2 \times 3 = 6$)
- frequency of least-frequent rule (according to the treebank) that appears in the parse of the input sentence; always zero if there is a Markovisation rule (2)

[15] Note that the 90% interval for the null hypothesis of no difference between grammatical and ungrammatical data covers observations ranging from 7:13 to 13:7 for 20 events.

[16] Wong and Dras (2010) allow rule features with low overall frequency and remark that this could explain poor classification results with some of their feature selection strategies.

[17] Note, however, that the Markovisation process is used almost as often when parsing ungrammatical sentences as when parsing grammatical sentences.

- number of discriminative rules for each of the 10 rule sets; same normalisation as for Markovisation rules ($10 \times (1 + 3 + 3) = 70$)
- sentence length and total number of rules; This will allow the decision tree learner to make its own normalisations. (2)

6.3 Improving Individual Methods

In this section, we build four decision tree classifiers that judge whether a sentence is grammatical or not based on the four individual feature sets defined in the previous section. The aim of this experiment is to provide a basis for comparison for the combined methods of the next section so that we can tell how much of potential improvements over the basic methods of Chapter 5 is due to feature combination as opposed to the application of decision trees. Therefore, we will compare the decision tree methods to the respective basic methods.

Machine learning has been applied before to the task of classifying a sentence as either grammatical or ungrammatical and also to detect individual error types, e. g. preposition and determiner errors. Chapter 2 gives an overview. Previous work relevant due to its feature sets is mentioned in Section 6.2 above.

6.3.1 Experimental Setup

We train decision trees with each of the feature sets of Section 6.2 to classify sentences as either grammatical or ungrammatical. Training and test data is the BNC with artificial errors as described in Chapter 3. As in Chapters 4 and 5, we use 10-fold cross-validation. Differently from these chapters, we use only the first 200,000 sentences (100,000 of each class) instead of 400,000 parsed sentences of each cross-validation set for training (total: $9 \times 200{,}000 = 1{,}800{,}000$), and only the first 360,000 sentences of each cross-validation set for testing. The remaining parsed data is used as held-out data for counting rules in order to determine the discriminative rule sets of Section 6.2.[18] The frequencies of POS n-grams are still counted in the 2,409,265 BNC sentences we did not parse as in Section 5.2.2.

[18] The last 40,000 sentences (20,000 per class) of the first five cross-validation sets, i. e. 200,000 sentences in total, are used.

In a side-experiment, we investigate the effect of training decision trees on the same data that is used for counting rule frequencies. For this purpose, we train additional decision trees on the second 200,000 sentences that have been held out in the first 10 decision trees. If the effects are significant, we expect to see lower accuracy of these decision trees on truly unseen test data (the 10 test sets used in the first experiments).

We use the C4.5 decision tree induction algorithm of Quinlan (1993) as implemented by Witten and Frank (2000) in the Weka[19] suite of machine learning algorithms. With the exception of a minimum leaf size of 125 that we set in order to speed up tree induction, we use the default settings of the "J48" Weka module which include automatic pruning.

6.3.2 Results

Similarly to the precision grammar method of Chapter 5, we only have a single accuracy point to report for each method as the decision tree classifiers do not offer a parameter to set the accuracy trade-off.[20] Table 6.2a shows the range of accuracy on grammatical and ungrammatical test data over the ten cross-validation runs. Compared with Table 5.1 of Chapter 5, the results vary more over cross-validation runs. This can be explained by the fact that not just the test data is varied but also the classifiers as they are trained on different training sets. (The precision grammar method of Chapter 5 does not involve training.) However, there are also differences in the variance between the decision tree methods: the standard deviation shown in Table 6.2a, i. e. the square root of the variance, is roughly twice as large for the decision trees trained with discriminative rule features than for the other decision trees. The choice of features not only affects the accuracy trade-off but also how well the accuracy trade-off is reproduced when training on new data.

Table 6.2b shows the corresponding numbers for the decision trees trained on the held-out data that is partly used to define the rule features. For the XLE, n-gram and distorted

[19] Waikato Environment for Knowledge Analysis, `http://www.cs.waikato.ac.nz/ml/weka/`

[20] In exploratory experiments, we tried to change the accuracy trade-off of decision trees by varying the class distribution, i. e. the error density, of the training data. We find that the accuracy trade-off is difficult to control this way as small changes can have strong effects and the effects depend on the feature set used. The learning algorithm quickly jumps to producing trivial classifiers assigning only one class to all input as the class distribution of the training data becomes unbalanced. Therefore, we employ a different idea for tuning the accuracy trade-off in Section 6.5 below.

Feature Set	Grammatical				Ungrammatical			
	Min	**Avg**	**Max**	**SD**	**Min**	**Avg**	**Max**	**SD**
XLE	54.49%	55.01%	55.82%	0.47%	68.27%	69.19%	69.59%	0.42%
n-gram	63.90%	64.23%	65.05%	0.35%	59.77%	60.65%	61.12%	0.38%
Distorted	70.36%	70.96%	71.70%	0.43%	61.07%	61.68%	62.25%	0.41%
Rules	65.74%	66.93%	68.52%	0.80%	51.70%	53.21%	54.43%	0.78%

(a) decision trees trained on the first half of the training data

Feature Set	Grammatical				Ungrammatical			
	Min	**Avg**	**Max**	**SD**	**Min**	**Avg**	**Max**	**SD**
XLE	55.13%	55.81%	56.72%	0.45%	67.50%	68.43%	68.98%	0.46%
n-gram	62.27%	64.35%	65.70%	1.04%	59.05%	60.24%	62.30%	1.02%
Distorted	70.13%	70.87%	71.33%	0.42%	61.32%	61.83%	62.53%	0.40%
Rules	66.13%	67.22%	68.69%	0.93%	51.20%	52.89%	54.16%	0.99%

(b) decision trees trained on the second half of the training data which includes the data used to choose the rule sets for the rule features

Feature Set	Grammatical				Ungrammatical			
	Min	**Avg**	**Max**	**SD**	**Min**	**Avg**	**Max**	**SD**
XLE	54.49%	55.41%	56.72%	0.61%	67.50%	68.81%	69.59%	0.58%
n-gram	62.27%	64.29%	65.70%	0.78%	59.05%	60.44%	62.30%	0.80%
Distorted	70.13%	70.92%	71.70%	0.43%	61.07%	61.76%	62.53%	0.41%
Rules	65.74%	67.07%	68.69%	0.88%	51.20%	53.05%	54.43%	0.91%

(c) all 20 decision trees

Table 6.2: Accuracy range and standard deviation (SD) over 10 cross-validation runs for the decision trees trained on the four feature sets of Section 6.2

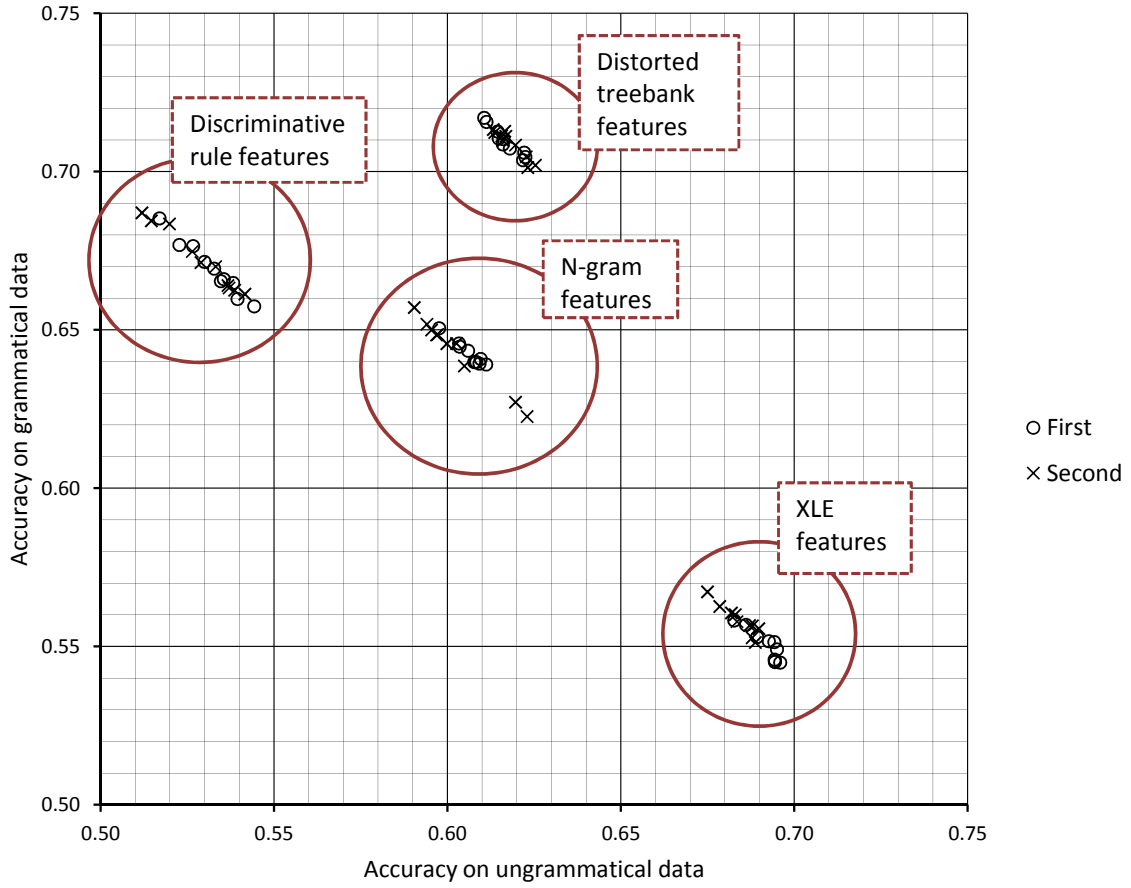


Figure 6.3: Scatter plot of the accuracy of the decision trees broken down by training set (first or second half) and feature set

treebank method, these results have equal status to the first results as the choice of rule features does not affect them. The range of values overlap well. However, the standard deviation of the n-gram method is much higher in the second experiment. This must be caused by an outlier. Indeed, a scatter plot for the individual decision trees shows two outliers (lower-right of circle labelled "N-gram features" in Figure 6.3).[21] As to the decision trees trained on discriminative rule features, Table 6.2b shows no negative effect of training on data that has partly been used to choose the rule sets: a 0.32 percentage point degradation of accuracy on ungrammatical data is compensated by a 0.29 percentage point improvement on grammatical data. Also, the scatter plots for the two experiments

[21] The same is true for the differences observed for the XLE method: accuracy differs by approx. 0.8 percentage points and the scatter plots are linearly separable with just three errors, e. g. with $y = 1.353x - 0.379$, as shown in the enlarged scatter plot in Figure 6.4. The experimental setup provides no other explanation than random noise. To reduce the chances of seeing such statistically significant but meaningless differences, one would have to increase the number of cross-validation runs, e. g. to 20-fold cross-validation.

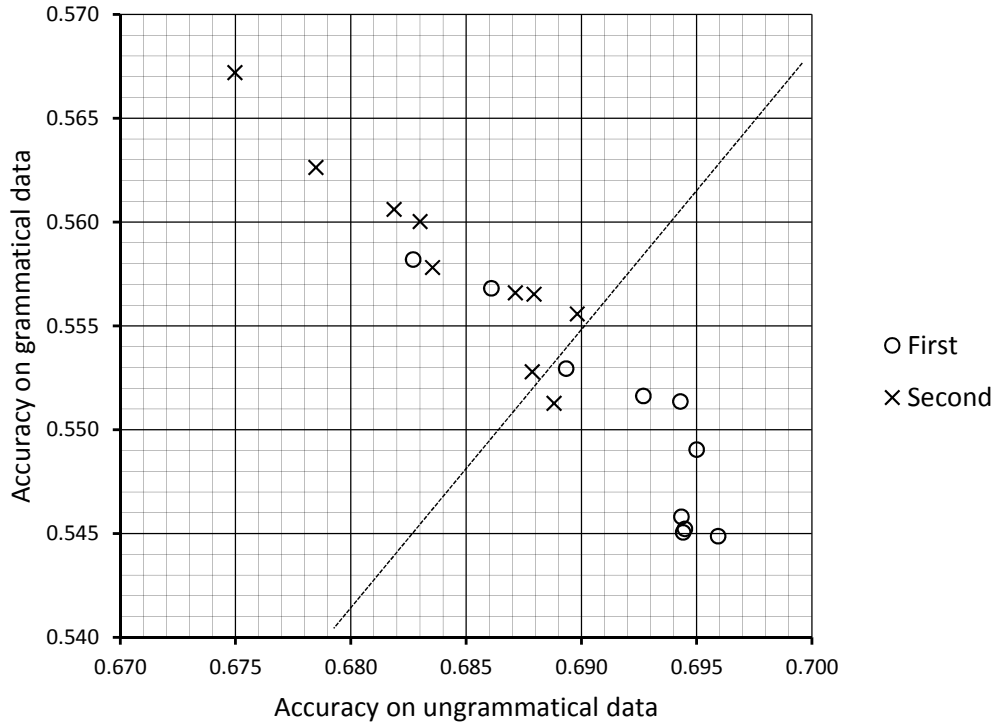


Figure 6.4: Enlarged scatter plot for the XLE feature set (see also Figure 6.3) and a line that separates the two sets with just three errors

agree well in Figure 6.3 (circle labelled "Discriminative rule features"). Therefore, it is permissable to join the results of the two experiments and to report average accuracy over all 20 decision trees for each feature set (Table 6.2c) in order to reduce noise.

Figure 6.3 also shows a strong negative correlation between accuracy on grammatical and ungrammatical data. The accuracy points do not uniformly distribute over the rectangular area defined by the intervals of Table 6.2c. Instead, they fall on curves similar to the accuracy curves of parameterised methods as we have seen in Chapters 4 and 5. This suggests that the decision trees are equally well optimised for some objective function and just represent different accuracy trade-offs.

Improvements over Basic Methods

Does decision tree learning with expanded feature sets improve over the basic methods of Chapter 5 which only use a simple decision rule corresponding to the root node of a decision tree?

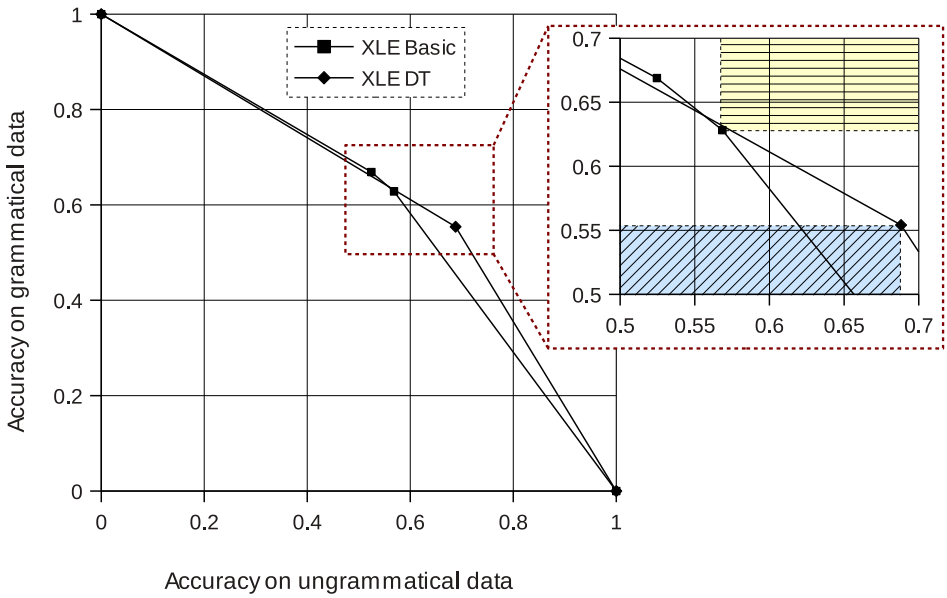


Figure 6.5: Accuracy of decision trees trained on XLE features compared to the two basic XLE classifiers of Chapter 5: out-of-memory errors and time-outs are either classified as grammatical (upper left accuracy point) or as ungrammatical (middle accuracy point); also shown is the interpolation to the trivial classifiers.

XLE features The decision trees trained on XLE features have higher accuracy on ungrammatical test data than the basic XLE-based classifiers of Chapter 5 but suffer a degradation of accuracy on grammatical data, i. e. the classifiers represent different accuracy trade-offs and we cannot say that one is generally better than the other. Figure 6.5 shows the classifiers in the accuracy plane with lines indicating interpolated accuracy with the trivial classifiers. There are two basic XLE classifiers as memory and time-out exceptions can be handled differently. All three classifiers are in the undecided region (as defined in Chapter 3) to each other. However, the interpolation line of the decision tree passes through the area of improvement of one of the two basic XLE classifiers. This means that we can obtain a classifier outperforming a basic XLE-based classifier with a decision tree if we randomly let some sentences pass as grammatical in order to change its accuracy trade-off. The improvement is marginal and hardly relevant though. On the other side, interpolation with the classifier flagging all input as ungrammatical (towards the lower right corner) does not bring the basic XLE-based classifiers near the accuracy of the decision tree.

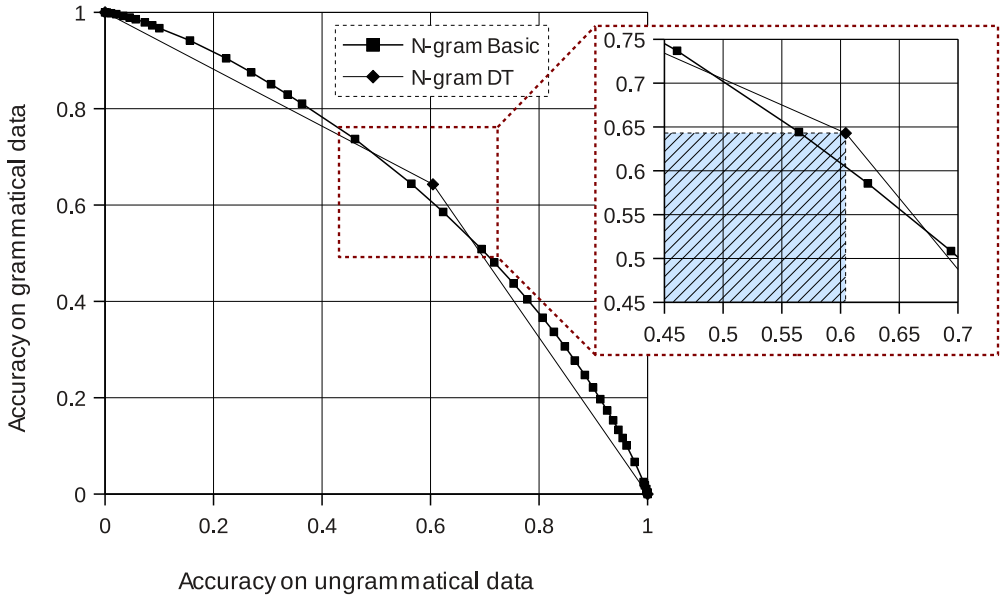


Figure 6.6: Accuracy of decision trees trained on n-gram features compared to the accuracy curve of the basic n-gram method of Chapter 5; also shown is the interpolation of the decision tree with the trivial classifiers.

***N*-gram features** The (average) accuracy point of the decision trees trained on n-gram features is shown together with the accuracy curve of the basic n-gram method of Chapter 5 in Figure 6.6. The decision tree outperforms the basic method for the segment of the accuracy curve that passes through the area of degradation (as defined in Chapter 3) of the decision tree. This area is highlighted in Figure 6.6. Furthermore, a wider segment is outperformed by linear interpolations of the decision tree and trivial classifiers (flagging all or no input). The Euclidean distance between the accuracy point of the decision tree and the accuracy curve of the basic method is approximately 2.70 percentage points.[22]

Distorted treebank features Figure 6.7 shows that the decision trees trained on distorted treebank features outperform a segment of the accuracy curve of the basic distorted treebank method of Chapter 5: the accuracy curve passes through the area of degradation (highlighted) of the decision tree. A wider segment can be outperformed if we interpolate the decision tree with trivial classifiers (the two lines meeting at the accuracy point). The lower line actually does not cross the curve at all (also not outside the range shown in

[22]Measured with the distance measure tool of the GNU Image Manipulator Program (GIMP) in a high-resolution screen shot of the graph.

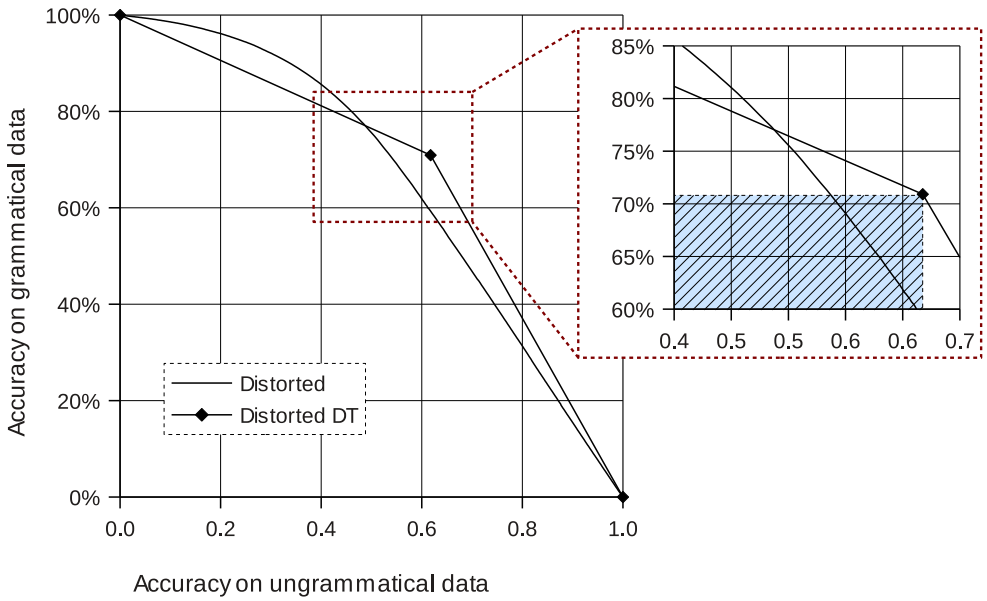


Figure 6.7: Accuracy of decision trees trained on distorted treebank features compared to the accuracy curve of the basic distorted treebank method of Chapter 5; also shown is the interpolation of the decision tree with the trivial classifiers.

Figure 6.7). The distance of the accuracy point of the decision tree to the curve is approx. 6.54 percentage points.

Discriminative rule features In Chapter 5, we did not consider a basic discriminative rule method, i. e. a method that uses parsed grammatical and ungrammatical reference data to identify discriminative rules but that works without machine learning (see Section 5.3.4 of Chapter 5 for a brief discussion of a "rare rule" method using only grammatical data). Consequently, there is no need to show the single accuracy point (0.5305, 0.6707) in a graph.[23]

6.4 Combining Methods with Decision Trees

In this section, we bring the different feature sets of Section 6.2 together to build classifiers that draw from different types of information, e. g. precision grammar (XLE) and n-gram features — see the method overview in Section 6.4.2 below.

[23]This accuracy point (and the interpolating lines to the accuracy points of the two trivial classifiers) will be shown in a different comparison in Figure 6.16 in Section 6.5 below.

6.4.1 Related Work

Machine learning naturally lends itself to combining heterogenous feature sets. Therefore, supposedly supportive features are often added to a core feature set. For example, Sun et al. (2007) add four types of features to an error detection method using sequential patterns with discriminating ability. Lee et al. (2007) train support vector machines (SVM) on learner data (and also on machine translation output) with diverse features including trigram language model perplexity, parse score of a probabilistic parser and pairs of words that are in certain dependency relations. They successively add feature sets to the method, starting with two feature sets and finally using five feature sets.

Andersen (2006) tests various combinations of n-gram and RASP parser features with a Naive Bayes classifier and finds that feature combination gives only small improvements over the best methods with a single feature set. A combination of lemma and POS unigram and bigram features seems to be the best choice as results are only marginally improved by adding dependency relation features extracted from RASP parse results.

Wong and Dras (2010) combine vanilla and distorted parse probability features (see Section 5.4 of Chapter 5) and disciminative rule features in a support vector machine. Their experiments show that rule features are weaker predictors of grammaticality than parse probability features and that results only modestly improve when all features are combined.

6.4.2 Method Overview

Given four feature sets, there are already six possible combinations of two sets and four combinations of three sets. We focus on a progression successively adding n-gram features, distorted treebank features and discriminative rule features to XLE features. We also test the combination of distorted treebank and discriminative rule features as a method that only relies on treebank-induced probabilistic grammars. Of course, we expect the combination of all four feature sets to give the best results. Figure 6.8 shows the feature sets in a hierarchy and the names we use for them. The feature set combinations we will test are marked with a star.

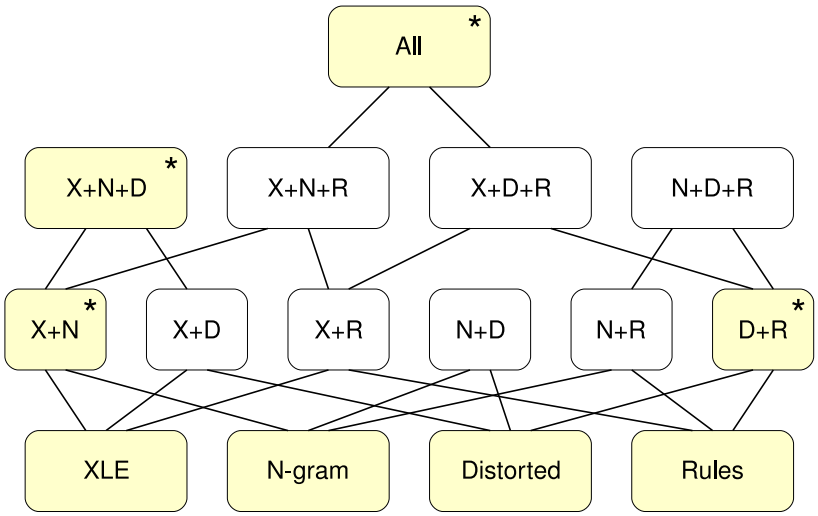


Figure 6.8: Method overview: all 11 possible combinations of the four feature sets; the combinations marked with a star will be studied in Section 6.4.

Feature	**Grammatical**				**Ungrammatical**			
Sets	**Min**	**Avg**	**Max**	**SD**	**Min**	**Avg**	**Max**	**SD**
X+N	61.65%	62.63%	63.20%	0.40%	65.39%	66.14%	67.14%	0.43%
D+R	70.56%	70.95%	71.32%	0.19%	63.17%	63.49%	64.04%	0.21%
X+N+D	69.32%	69.92%	70.64%	0.33%	65.41%	66.01%	66.55%	0.33%
All 4	70.38%	70.73%	71.35%	0.26%	65.82%	66.38%	66.77%	0.26%

Table 6.3: Accuracy range and standard deviation (SD) over 10 cross-validation runs and 2 decision trees per run trained on combinations of feature sets of Section 6.2

6.4.3 Experimental Setup

We train decision trees to classify sentences as either grammatical or ungrammatical as in Section 6.3 but with the combined feature sets of Section 6.2. Training and test data is the artificial BNC-based parallel error corpus of Chapter 3. All experiments are run with 10-fold cross-validation. As in Section 6.3, we train two decision trees per cross-validation run: one on the first half of the training data, one on the second half. We report average accuracy over all 20 results and compare the new classifiers to decision trees trained on the individual feature sets described in Section 6.3.

6.4.4 Results

Table 6.3 gives the accuracy range, average and standard deviation for the decision trees trained on the four combined feature sets we consider. With the exception of the combination of distorted treebank and discriminative rule features (D+R), the combined methods do not clearly outperform their component methods on both grammatical and ungrammatical test data. Therefore, we again use interpolation with trivial classifiers to compare methods with different accuracy trade-offs (Chapter 3, in particular Figure 3.5). In the following Figures 6.9 to 6.12, we show basic methods of Chapter 5 with dotted lines, decision tree results of Section 6.3 with triangles, the new combined methods with diamonds and interpolation of the latter classifiers and trivial classifiers with solid lines. Intentionally, the methods are not annotated with their feature sets in order to focus on the effect of feature combination and to leave the comparison of individual methods to Chapter 7. As in Section 6.3 above, the shaded rectangle is the area of degradation for the new classifier. In the following discussion of results, we also use the notions of indirect degradation and improvement as introduced in Chapter 3.

Accuracy Graphs for Combinations of two Feature Sets

Combining XLE and N-gram Features Figure 6.9 shows the average accuracy of our 20 decision trees trained on the combination of XLE and n-gram features (X+N) together with the individual XLE and n-gram results of Section 6.3 and Chapter 5. While there is no direct improvement over the previous decision trees (the triangles are outside the area of degradation), the decision trees trained on the individual XLE and n-gram feature sets are outperformed by interpolations of the combined decision tree and trivial classifiers. However, the distance is small. This is surprising since the two feature sets seem quite complementary.

Combining Distorted Treebank and Rule Features Figure 6.10 shows that combining distorted treebank and discriminative rule features (D+R) produces classifiers that perform similarly to the decision trees trained on the feature set that gives the better classifiers of the two sets, the distorted treebank classifier — see Chapter 7. There is a

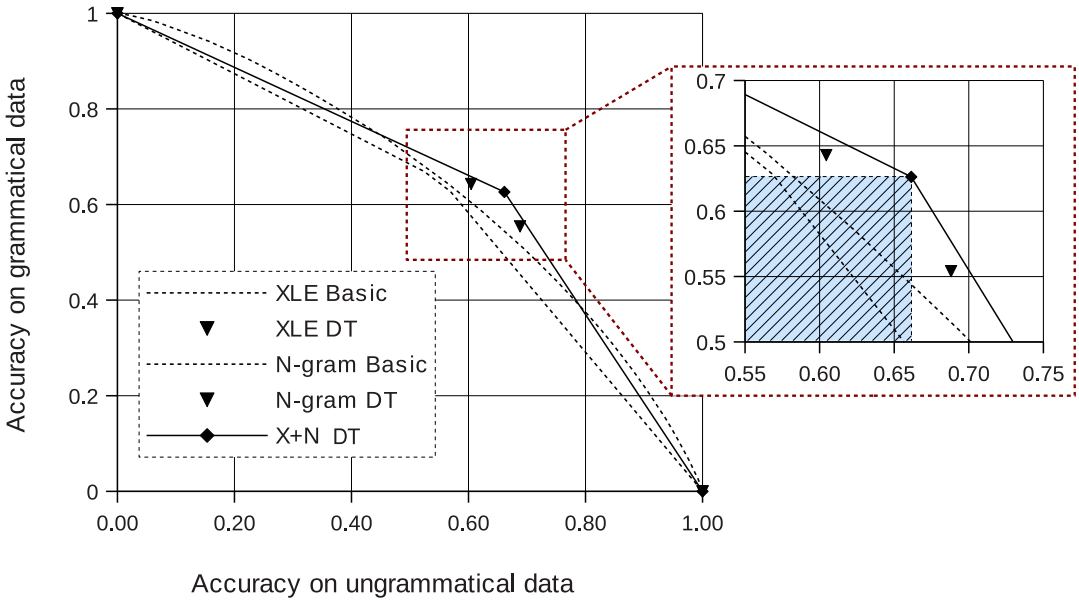


Figure 6.9: Combination of XLE and N-gram features and comparison with the individual XLE and n-gram methods: intentionally, we only differentiate between basic methods and decision trees.

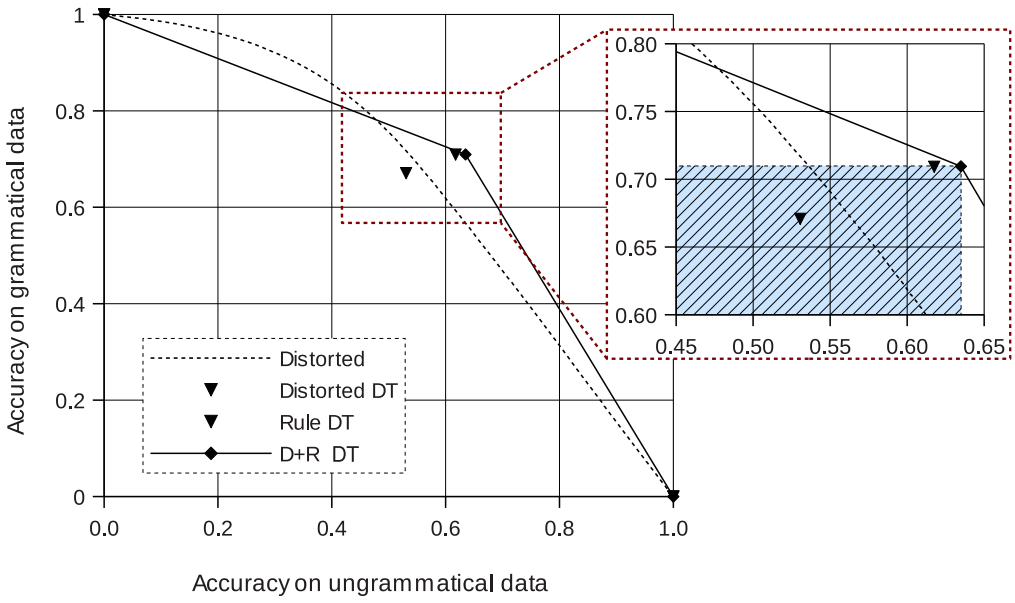


Figure 6.10: Combination of distorted treebank and discriminative rule freatures and comparison with the individual distorted treebank and discriminative rule methods: intentionally, we only differentiate between basic methods and decision trees.

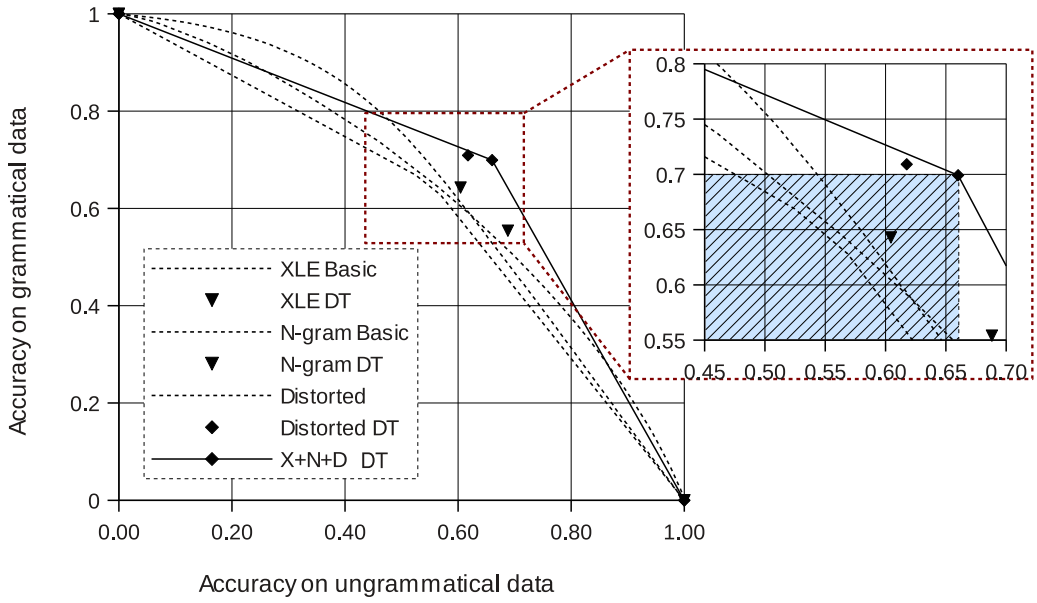


Figure 6.11: Combination of XLE, n-gram and distorted treebank feature sets and comparison with the corresponding three individual methods; intentionally, we only differentiate between the basic methods and those that use decision trees.

small improvement in accuracy on ungrammatical data.

Combining XLE, N-gram and Distorted Treebank Features Figure 6.11 shows the accuracy of the decision trees trained on the union of XLE, n-gram and distorted treebank feature sets (X+N+D) and the respective individual decision trees. While the new decision trees only outperform one of the three component methods directly, interpolation with the trivial classifiers can produce classifiers that have higher accuracy on both grammatical and ungrammatical test data than the decision trees trained on individual feature sets.

Combining all Four Features Sets Finally, Figure 6.12 compares the decision trees trained on the full set of features (All4) with the four individual methods. As in the case of the combination X+N+D discussed above, not all individual methods are directly outperformed but are outperformed indirectly via interpolation with the trivial classifiers.

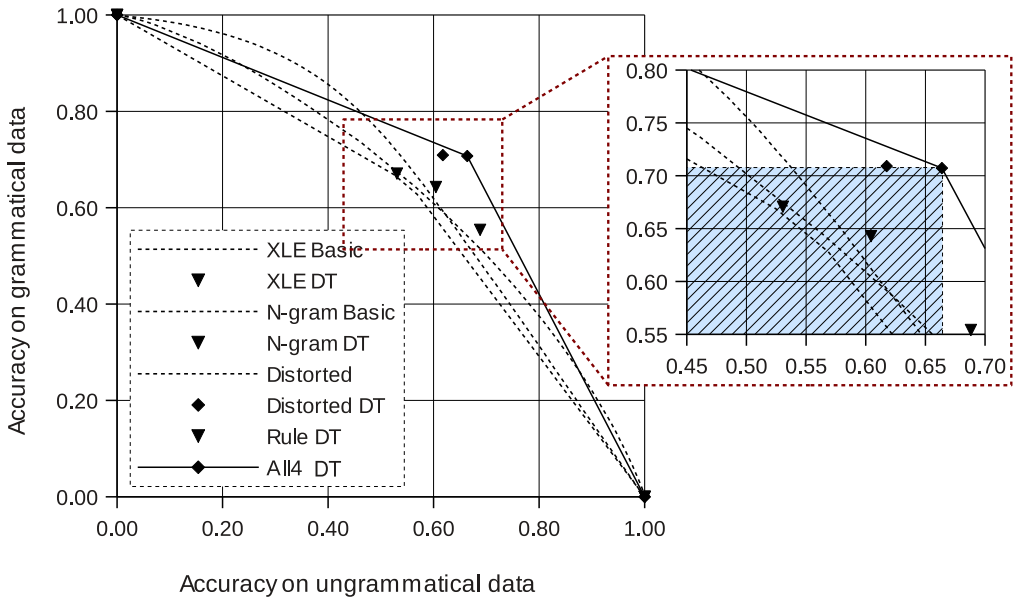


Figure 6.12: Combination of all four feature sets and comparison with the individual methods; intentionally, we only differentiate between the basic methods and those that use decision trees.

Summary

Due to different accuracy trade-offs, the decision trees we trained on the feature sets of Section 6.2 and combinations of these sets are difficult to compare. We can confirm that combining feature sets from a variety of linguistic sources improves results. However, we cannot quantify the improvements and therefore cannot, for example, say whether the XLE and n-gram features complement each other better than the distorted treebank and rule features. We will revisit these questions in the following section which introduces a method for tuning the accuracy trade-off and therefore should make it easier to compare methods.

6.5 Tuning the Accuracy Trade-Off with Voting Classifiers

The decision trees of Sections 6.3 and 6.4 offer only a limited range of accuracy trade-offs between high accuracy on grammatical data (minimal overflagging) and high accuracy on ungrammatical data (few errors missed): the highest accuracy on grammatical data is 70.95% for the combination of distorted treebank and rule features (D+R, Table 6.3)

and on ungrammatical data we reach 68.81% accuracy with decision trees trained on XLE features alone (Table 6.2c). Applications may need accuracy trade-offs outside this range, e. g. as a component of a grammar checker for native speakers or advanced L2 learners, a classifier with a much higher accuracy on grammatical sentences may be desired. Without a means to tune the accuracy trade-off of the decision tree methods, one would have to revert to one of the basic methods of Chapter 5 that offer a parameter that sets the accuracy trade-off, e. g. the n-gram frequency threshold (Section 5.2) or the probability offset of the basic distorted treebank method (Section 5.4).

The evaluation of our decision tree methods would also benefit from tunable accuracy trade-offs. In the previous sections, we employed interpolation with trivial classifiers in order to compare methods that do not directly outperform each other. As we have seen in Section 6.3, such interpolated classifiers can be inferior to basic classifiers. In the following, we extend our decision tree methods in such a way that the accuracy trade-off can be set with a parameter. A more appropriate method for tuning the accuracy trade-off than classifier interpolation can be expected to expand the superior performance of the decision tree methods over the corresponding basic methods from a small range of accuracy trade-offs to a wider range.

6.5.1 Proposed Method

We propose to train multiple classifiers (decision trees in our case) on subsets of the training data and have the classifiers vote for the final decision (Wagner et al., 2009). The accuracy trade-off can then be tuned by setting the number of votes that are required to flag a sentence as ungrammatical. For example, overflagging will be minimized if sentences are flagged only when all classifiers concordantly judge them as ungrammatical. However, for this method to work, the classifiers must not be identical. They have to disagree on some sentences for the voting to make a difference. Decision trees are particularly suitable because they are unstable, i. e. small changes to the training data can result in large changes to the tree (Breiman, 1996b; Bauer and Kohavi, 1999). As Bauer and Kohavi (1999) summarise various studies which show the positive effects of voting, we can expect to also see direct improvements over the decision trees of Sections 6.3 and

6.4 in addition to be able to set the accuracy trade-off. Improvements are plausible if we assume independence of the classifiers and that each classifier classifies > 0.5 of test items correctly: the probability of possible voting outcomes can be described with a binomial distribution and the probability of the majority being correct approaches one as the number of classifiers is increased, e. g. if each classifier has an accuracy of 0.6, the overall probability is 0.978 with 99 classifiers (Heath et al., 1993; Marsland, 2009).

6.5.2 Related Work

Breiman (1996a) introduces the idea of combining classifiers or predictors trained on different bootstrap samples[24] of training data, calls it bagging, an acronym for "bootstrap aggregating", and shows that it can improve classification and regression results. Bauer and Kohavi (1999) study these and other techniques for voting classification in more detail. However, Murthy (1996) points to earlier work combining multiple decision trees, e. g. Heath et al. (1993) discuss a majority voting scheme that uses decision trees that have been randomised using the split criterion.

To our knowledge, voting has previously only been used for system combination which we discuss in Section 5.4.2, not for tuning the accuracy trade-off. The following highlights some application of voting to NLP tasks.

Classifier voting has been applied to part-of-speech (POS) tagging. Màrquez et al. (1998) integrate POS tagger combination into a blend of self-training and co-training: In normal self-training, there is only one initial POS tagger and all additional (raw) training data would be annotated by it and then used to train a new tagger. Co-training would use the output of one tagger to train a different tagger, e. g. using a different learning algorithm. Màrquez et al. (1998), however, tag the data with two taggers and add only the intersection, i. e. the annotations the two taggers agree on, to the initial training data.[25]

[24] A bootstrap sample samples from the original data with replacement and has the same size as the original data. Another way of describing this is that the weight of each training item is set according to the Poisson distribution as each item is picked a small, discrete number of times in the sample (including 0 times). For large training sets, a bootstrap sample will contain approximately 63.2% unique items of the training set (Bauer and Kohavi, 1999; Rokach, 2009).

[25] In the context of combining decision trees for POS tagging, the work of Màrquez et al. (1999) is interesting: they average the POS tag probability distribution of multiple decision trees. Different randomisation methods for generating a set of varying decision trees are compared. Their experiment uses an unusual POS tagger, a "reductionist" tagger: for each ambiguity class, e. g. NN-ADJ, a separate predictor of the POS tag probability distribution is trained. Low probability POS tags are discarded and the process

Brill and Wu (1998) train two ensemble classifiers on the POS annotations of multiple POS taggers for the previous, current and next token: one classifier predicts the POS tag, the other classifier decides for each token which POS tagger to trust. The latter type of classifier interestingly achieves a slightly better (lower) annotation error rate. These and other ensemble learning methods for POS tagging are evaluated by van Halteren et al. (2001).

Xu and Jelinek (2007) build random forests of decision tree language models and evaluate the joined model in the task of automatic speech recognition. The decision trees are randomised with three techniques: *(a)* sampling the training data with replacement (bootstrapping; see also Footnote 24), *(b)* sampling the feature set to consider at each node of the decision tree, and *(c)* randomisation of the greedy search for the best split at each node. A decision tree language model uses decision trees to model the probability of a word given the history of previous words. Instead of predicting the most likely word, each leaf node stores a probability distribution over all possible words. Random forests are shown to improve results over previous language models (including basic decision tree language models) both in word error rate of the speech recognition task and in language model perplexity. Deoras et al. (2010) adapt decision tree language models to new domains by first splitting decision tree nodes according to the target data and then pruning the decision tree according to the source domain data. Filimonov and Harper (2011a,b) extend the work on decision tree language models.

In a broader sense, any combination method that aggregates the scores of candidate outputs of individual methods and picks the output with the best overall score can be viewed as a voting method. For example, Petrov (2010) combines 16 latent variable grammars in an unweighted product model, i. e. the probabilities assigned to structures by each grammar are multiplied. The 16 grammars have been automatically induced and "vary widely" due to randomness introduced by the EM training algorithm. Petrov finds that these grammars are highly suitable for combination: the combination outperforms parsing with discriminative reranking as of Charniak and Johnson (2005).

is iterated until all tags are disambiguated.

Feature Set	Additional Trees		Total Trees
	10 Trees, 3.6M	27 Trees, 3.24M	
XLE	✓	—	12
n-gram	✓	—	12
Distorted	✓	✓	12 and 29
Rules	—	—	2
X+N	✓	✓	12 and 29
D+R	—	✓	29
X+N+D	✓	—	12
All 4	—	✓	29

Table 6.4: Number of additional decision trees trained for voting and total number of trees (per cross-validation run) used in voting experiments including the two trees trained in Sections 6.3 and 6.4

6.5.3 Experimental Setup

The experimental setup follows the setup of our decision tree experiments in Sections 6.3 and 6.4 with the exception of the number of decision trees and the subsets of training data used for each tree. We run voting experiments with 12 and 29 trees per cross-validation run.[26] Since we re-use the decision trees trained in Sections 6.3 and 6.4 above, we train 10 or 27 additional trees. In order to increase the variance between trees, we half the amount of training data to $2 \times 9 \times 50{,}000 = 900{,}000$ sentences.[27] We use a shifting window to select the 900,000 training sentences from 3.6 million sentences available in each cross-validation run. In case of 10 trees, we move the window by (3,600,000 - 900,000) / (10 - 1) = 300,000 sentences for each tree. For training 27 additional trees, we exclude the last 10% of the training data which has partly been used to define the rule features (see Sections 6.2, 6.3.1 and 6.3.2) in favour of a cleaner setup. Correspondingly, the training data extraction window is moved by (3,240,000 - 900,000) / (27 - 1) = 90,000 sentences. Table 6.4 shows which setup(s) we employ for each method. We run experiments with both setups for the distorted treebank method and for the combination of XLE and n-gram features, allowing us to also investigate the effect of the number of trees in voting. We do not train additional decision trees for the discriminative rule method as there is no

[26] These numbers were chosen such that the training data can be divided easily using sets of 10,000 sentences and to keep computational costs low. Initial experiments used 12 trees (Wagner et al., 2009). We increased the number of trees to 29 in order to widen the covered accuracy range.

[27] There are two classes (grammatical, ungrammatical) and nine cross-validation sets used in training.

corresponding basic method in Chapter 5 to compare with.

6.5.4 Results

In the following, we present the results of our decision tree voting experiments in the order of methods of Table 6.4. We address the following questions:

- Can voting tune the accuracy trade-off in a better way than interpolation with trivial classifiers? In other words, to what extent can the voting method translate the superior performance of decision trees over their basic methods at their own accuracy trade-off (Section 6.3) to other accuracy trade-offs? It is not sufficient for voting over decision trees to provide some accuracy trade-off. In order to be useful, the accuracy curve should be above the two line segments that are produced by interpolating an individual decision tree with trivial classifiers as well as above the accuracy curve of the corresponding basic method that does not use machine learning, which leads us to the next question:

- Can voting with decision trees improve the basic methods of Chapter 5? While the main motivation for training decision trees on individual feature sets in Section 6.3 is to provide a baseline for comparision with the decision trees trained on combined feature sets in Section 6.4, we also tried to answer the question of whether and to what extent machine learning improves results over the basic methods of Chapter 5. With the notions of direct and indirect improvement of Chapter 3 which are based on interpolation with the trivial classifiers, we can only claim improvements over a small accuracy trade-off range as the interpolation lines quickly cross the accuracy curve of the respective basic method. It is unsatisfactory to be only able to improve over the basic method for a small range of accuracy trade-offs. Here, we repeat the comparison of Section 6.3.2, this time expanding the accuracy curve of the decision tree methods with voting.

- Finally, we also reinvestigate the improvements for combinations of feature sets as in Sections 6.4 using voting.

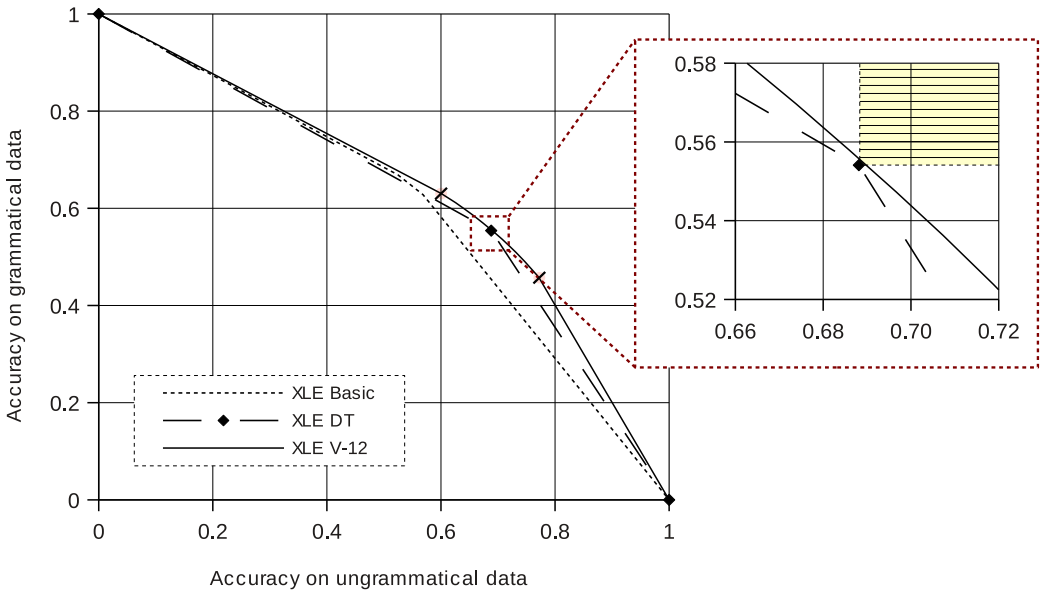


Figure 6.13: Voting applied to 12 decision trees (per cross-validation run) trained on the XLE features of Section 6.2; also shown are the basic XLE method of Chapter 5 and the XLE decision tree method of Section 6.3.

XLE Features

Figure 6.13 shows the accuracy curve of voting with 12 decision trees (solid curve) together with the XLE decision tree method of Section 6.3 (dashed line) and the basic XLE method of Chapter 5 (dotted curve). The accuracy range covered is fairly short, going from (60.05%, 63.05%) to (77.12%, 45.85%). Nevertheless, the accuracy curve of the voting method is fully above the curves of the basic method and the decision tree method. However, the curves get very close to the two accuracy points of the basic XLE method which are at 52.46% (classifier X_2) and 56.85% (classifier X_1) accuracy on ungrammatical data. The distance between the methods is considerably higher for accuracy on ungrammatical data between 65% and 85%. These improvements may mean that the ParGram English LFG "knows" more about grammaticality than it shows in its judgements based on parsability with its core grammar.

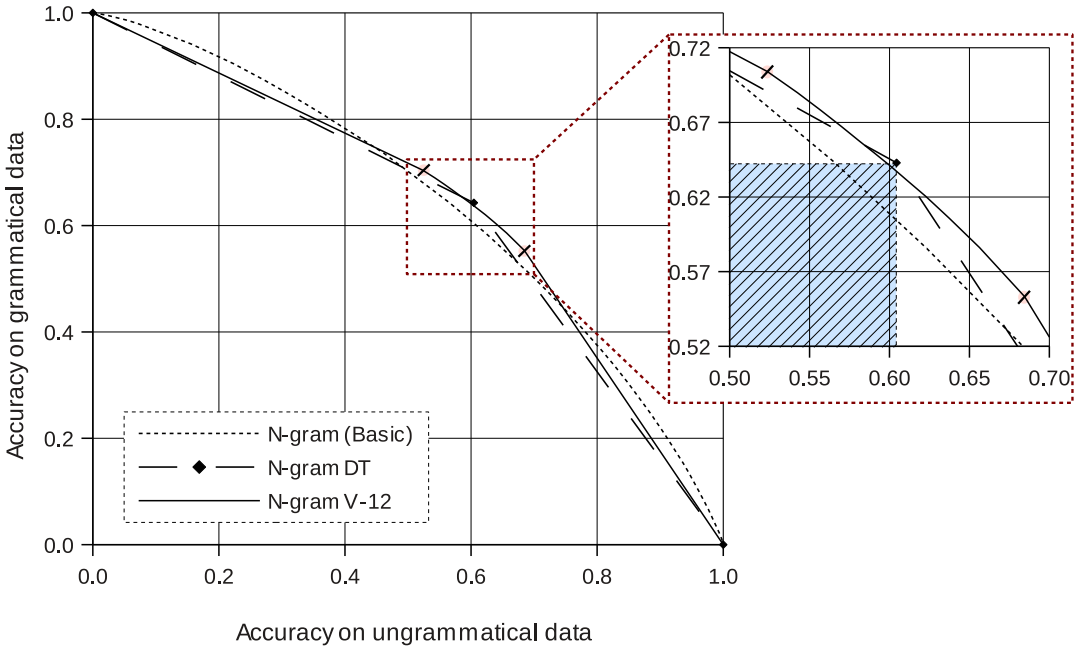


Figure 6.14: Voting applied to the n-gram method with 12 decision trees; also shown are the basic method of Chapter 5 and the decision tree method of Section 6.3.

N-gram Features

Figure 6.14 shows the basic, decision tree annd voting n-gram methods. Voting only moderately widens the accuracy range for which the basic method is outperformed compared to the decision tree method of Section 6.3. The short range of accuracy trade-offs means that the decision trees trained on n-gram features often agree. In addition, the accuracy curve of the voting method runs through the area of degradation of the decision tree method. Possibly, the decision trees trained for voting suffer from the reduced amount of training data in the experimental setup. As a consequence of these two shortcomings, the accuracy curve of the voting method with n-gram decision trees falls below the accuracy curve of the basic n-gram method almost as quickly as the interpolation lines of the decision tree method without voting. Therefore, the conclusion stays the same: the improvements are small and restricted to a small range of accuracy trade-offs. The basic n-gram method is difficult to improve on.

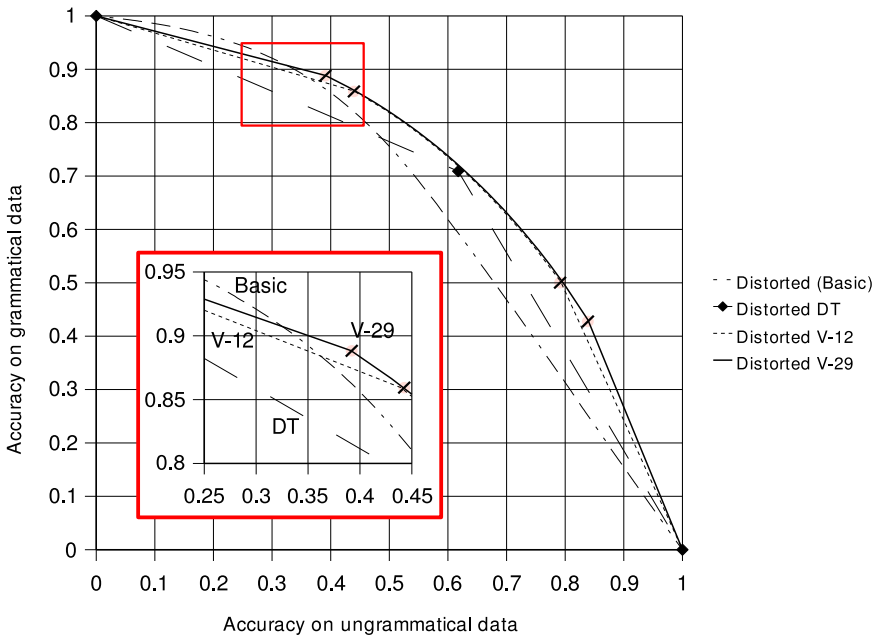


Figure 6.15: Voting applied to the distorted treebank method with 12 and 29 decision trees; also shown are the basic method of Chapter 5 and the decision tree method of Section 6.3.

Distorted Treebank Features

We expect the distorted treebank method to be most difficult to improve on. The basic method performs well, in particular for high accuracy on grammatical data: it intersects the 90% grammatical accuracy line at 33.77% accuracy on ungrammatical data (Section 5.4.4). As can be seen in the top-left hand corner of Figure 6.7 of Section 6.3.2, interpolated classifiers quickly fall below the accuracy curve of the basic distorted treebank method.

Figure 6.15 shows the accuracy curves obtained with voting with 12 (dotted curve) and 29 decision trees (solid curve) together with the distorted treebank methods of Chapter 5 (two dots and three dashes) and Section 6.3 (dashed line). The extreme voting classifiers requesting either only one vote or all votes to be for "ungrammatical" in order to classify a sentence as ungrammatical are highlighted to show the wide range of classifiers almost reaching 90% accuracy on grammatical data on one end and 85% accuracy on ungrammatical data on the other end. Within this range, the voting methods with 12 and 29

decision trees are almost identical. The method with 12 classifiers is slightly inferior and its accuracy range is shorter. In addition to tuning the accuracy trade-off, voting also gives a small direct improvement over the decision tree method of Section 6.3 (accuracy point marked with a diamond). All voting classifiers also stay above the accuracy curve of the basic method. With interpolation to the trivial classifiers, the range for which the curve stays above is extended somewhat to the top-left and fully reaches the bottom-right corner. Only for very high accuracy on grammatical data, the basic method is still superior. The right side of the graph is where we find the biggest improvements over the basic method: at 80% accuracy on ungrammatical data, accuracy on grammatical data is approximately 20 percentage points higher. The box zooming in on the top-left end of the accuracy range where multiple curves meet shows that the voting with 29 decision trees still outperforms the basic method at 90% accuracy on grammatical data. The accuracy curve of voting with 12 decision trees falls short earlier.

Discriminative Rule Features

The discriminative rule method of Section 6.3 has no basic counterpart in Chapter 5 as we concluded from the negative results of the PCFG pruning method (Section 5.3.4 of Chapter 5) that trying a rare rule approach would not be worthwhile. Figure 6.16 compares the decision tree method using discriminative rule features with the corresponding voting method. It shows a small effect of voting. However, note that only two decision trees vote as we do not train any additional decision trees on discriminative rule features in this section. Analysing the classification results of the individual decision trees, we find that they agree for 94.58% of test items. For comparison, the two decision trees trained in the same way on distorted treebank features (Section 6.3) agree on 86.78% of test items and voting with these two trees (using distorted treebank features) produces the accuracy points (55.17%, 77.52%) and (68.38%, 64.29%).

Reinvestigating Combination of Feature Sets

Our main motivation for combining feature sets in Section 6.4 is to confirm the expectation that performance of the classifiers should improve as more information becomes available

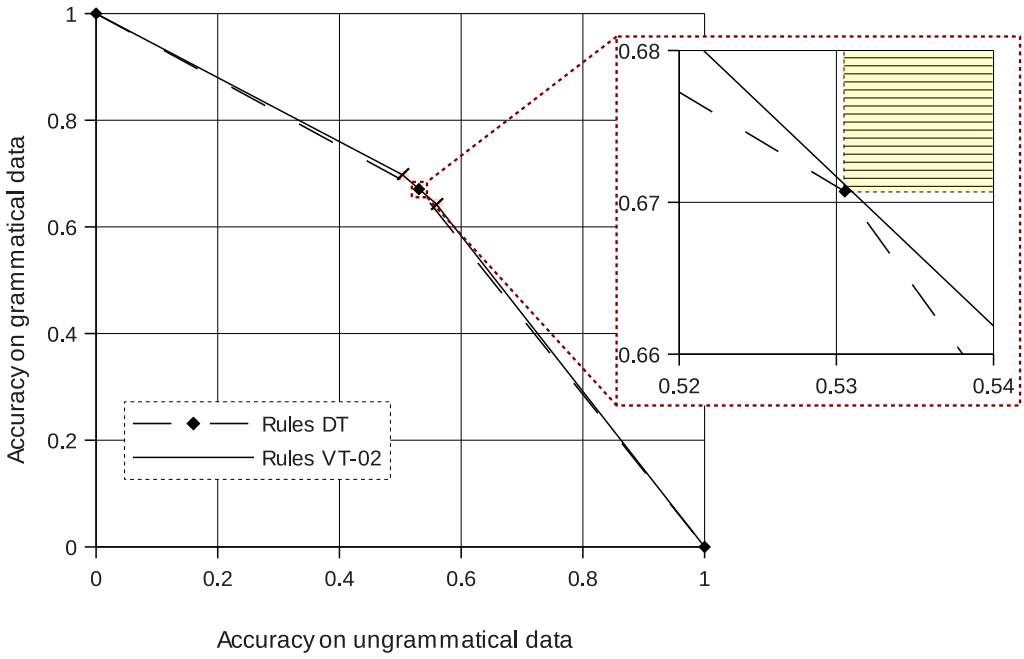


Figure 6.16: Voting applied to the discriminative rules method with only two decision trees; also shown is the decision tree method of Section 6.3.

to them. While no direct comparison is possible due to different accuracy trade-offs of the decision trees, indirect improvements via interpolation with the trivial classifiers are shown in Section 6.4. The improvements are small though as the interpolation method does not translate the performance of the decision trees well to other accuracy trade-offs. In the following, we compare the respective voting methods and briefly discuss the improvements due to feature set combination. For the method combining all four feature sets, we also compare the voting method to the plain decision tree method of Section 6.4.

XLE and N-gram Features The combination of XLE and N-gram features (X+N) is shown in Figure 6.17. There is a clear improvement for all accuracy trade-offs.

Distorted Treebank and Discriminative Rule Features Figure 6.18 shows a small improvement over the distorted treebank method when combined with discriminative rule features. The accuracy graph cannot tell us, however, whether this is due to the inferior performance of the discriminative rule method or whether the feature sets are partly redundant, i. e. represent similar linguistic information.

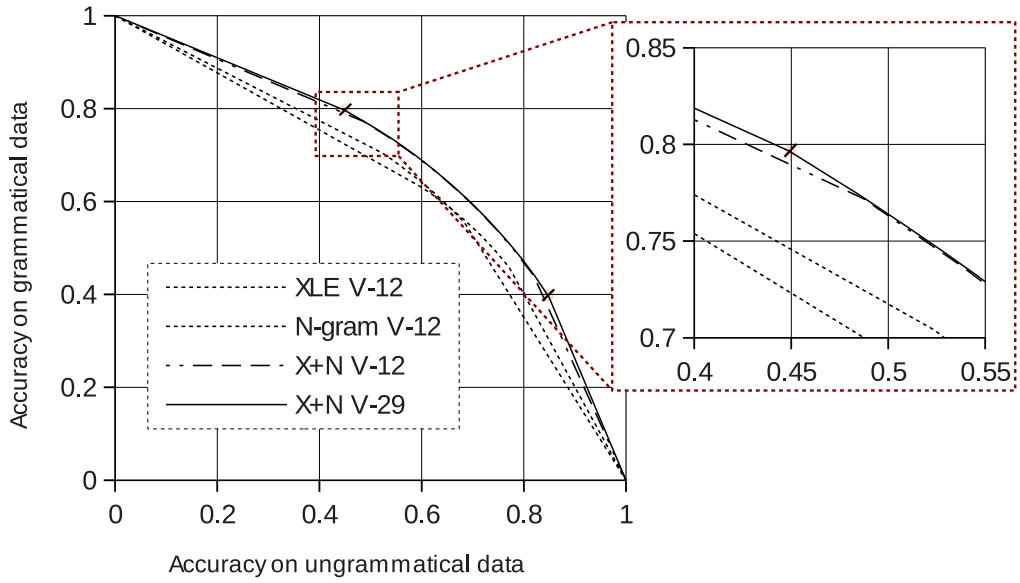


Figure 6.17: Voting applied to the combination of XLE and n-gram features (X+N) with 12 and 29 decision trees; also shown are voting with 12 decision trees trained on the individual XLE and n-gram feature sets

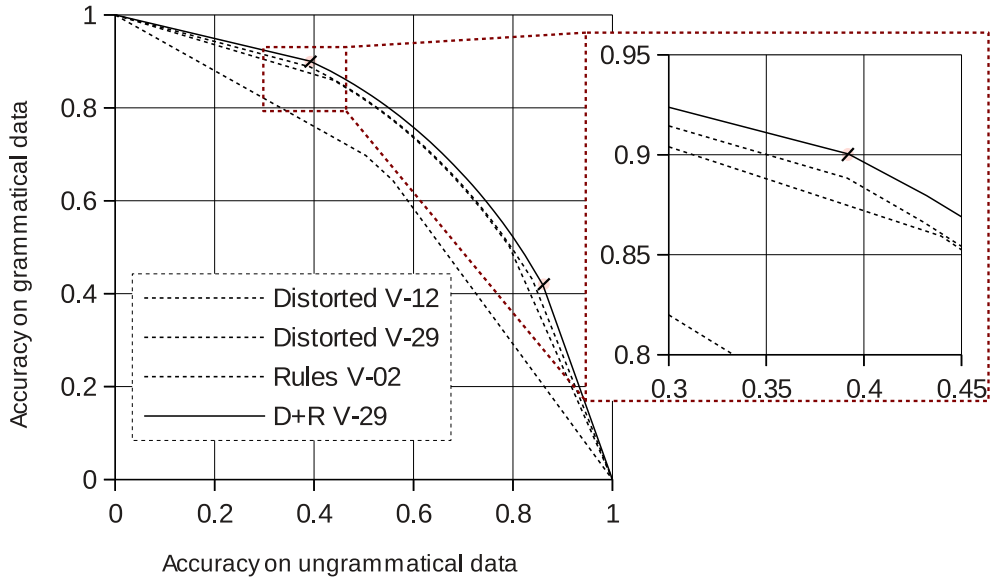


Figure 6.18: Voting applied to the combination of distorted treebank (D) and discriminative rule (R) features with 12 decision trees; also shown are voting with 2, 12 and/or 29 decision trees trained on the individual feature sets

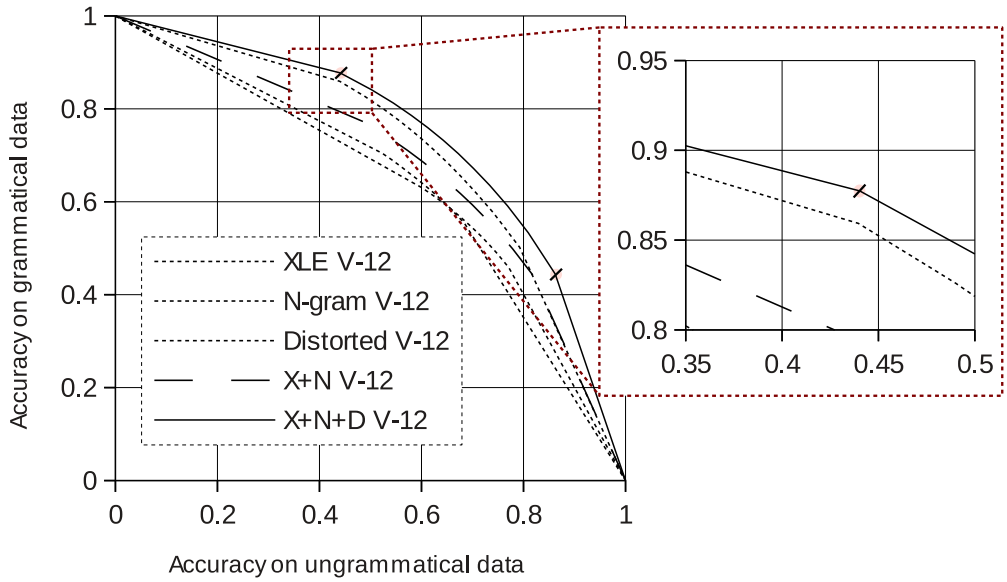


Figure 6.19: Voting applied to the combination of XLE, n-gram and distorted treebank features with 12 decision trees; also shown are voting with 12 decision trees trained on the individual feature sets

XLE, N-gram and Distorted Treebank Features Figure 6.19 shows the combination of XLE, n-gram and distorted treebank feature sets. The improvements over the distorted treebank method are somewhat bigger than for adding discriminative rule features in Figure 6.18, especially for high accuracy on ungrammatical data (right side of the graph).

All Four Feature Sets Finally, discriminative rule features are added to the method of the previous paragraph. Figure 6.20 shows the resulting method using all four feature sets together with the method with the first three features sets (as discussed in the previous paragraph) and the discriminative rule method. The improvement is small but consistent.

Figure 6.21 shows the same accuracy curve of the voting method with all four feature sets of Section 6.2 together with the basic n-gram and distorted treebank method and the combined decision tree method without voting. The voting method outperforms all other methods on a wide range of accuracy trade-offs. The box zooming in on the top-left end of the accuracy curve shows that an interpolated classifier reaches 41.72%

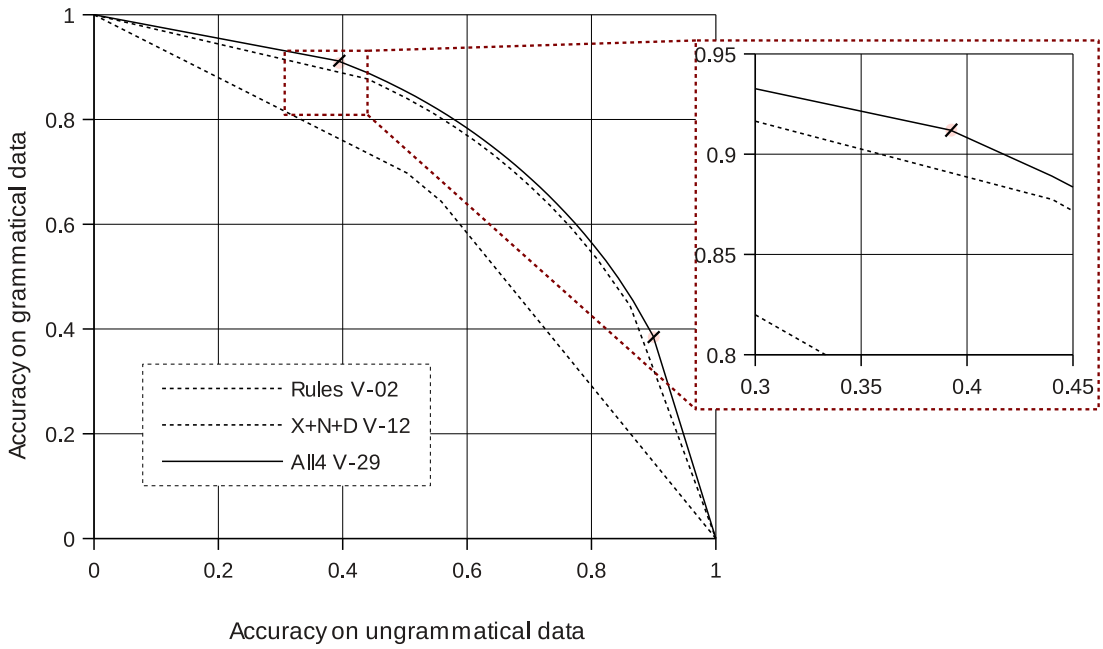


Figure 6.20: Voting applied to the combination of all four feature sets with 29 decision trees; also shown are the discriminative rule method with only two trees for voting and the combination of XLE, n-gram and distorted treebank features with 12 trees for voting

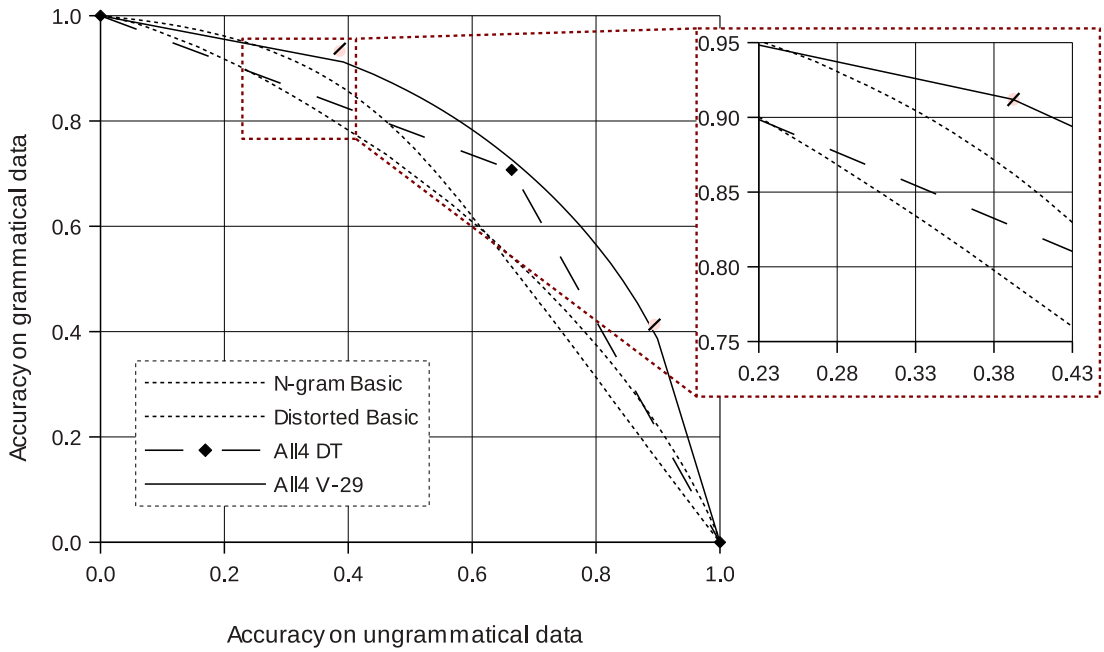


Figure 6.21: Voting applied to 29 decision trees (per cross-validation run) trained on the union of the four feature sets of Section 6.2; also shown are the two best-performing basic methods of Chapter 5 and the combined decision tree method of Section 6.4.

accuracy on ungrammatical data for 90% accuracy on grammatical data. The accuracy curve of the voting method crosses the curve of the basic distorted treebank method at the accuracy point (24.99%, 94.39%), the bisecting line at 69.50% and the 90% accuracy on ungrammatical data line at 38.35%. In addition, the decision tree method not using voting is directly outperformed as the accuracy curve passes above the accuracy point of the decision tree method. Only for accuracy on grammatical data above 94.39%, does the basic distorted treebank method perform better than voting.

Summary

The voting method extends the accuracy range, but the width of the range depends on the feature set. If the range is short as with the n-gram method, the voting method will not achieve the aim of outperforming basic methods on a wide range of accuracy trade-offs. In the case of the distorted treebank method, the voting method works well, though it also does not fully cover the performance of the basic method. Direct improvements over the plain decision tree methods are possible but not guaranteed.

6.6 Conclusions and Future Work

Machine learning in the form of decision tree induction produces classifiers with specific accuracy trade-offs. Direct or at least indirect improvements over the basic methods of Chapter 5 are observed but can be limited to a narrow range of accuracy trade-offs, e. g. for the n-gram method. Therefore, the basic methods may still be the best choice in some applications. Voting with multiple decision trees expands the accuracy range for which the basic methods are outperformed but the basic n-gram and distorted treebank method are not rendered obsolete. Quantifying the improvements remains difficult.

Combining feature sets improves accuracy of voting classifier in all combinations tested. The best method combining all four feature sets of Section 6.2 achieves 69.50% accuracy at the neutral accuracy trade-off and 41.72% accuracy on ungrammatical data when 90% accuracy on grammatical data is chosen. Still, the basic distorted treebank method is not outperformed for accuracy trade-offs which require accuracy on grammatical data to be above 94.39%. In the following, we point to ideas for future work related to the topics of

this chapter.

6.6.1 Weighted Voting with all Decision Trees

In Sections 6.3 to Sections 6.5, we trained over 160 decision trees per cross-validation run, see Table 6.4. From these decision trees, a voting classifier with a wide range of accuracy trade-offs could be built. However, decision trees with low accuracy may drag down overall accuracy. Therefore, superior decision trees should receive a higher weight. Also, the number of trees available per method should be considered. Unfortunately, no held-out data is left that we could use to optimise the voting weights. In future experiments, additional data should be held out for choosing these parameters.

6.6.2 Expand Investigation of Feature Set Combinations

In Section 6.4, we tested only two of the six possible pairs of the four feature sets of Section 6.2. Future work should also investigate the remaining four combinations X+D, X+R, N+D and N+R and develop a measure of improvements between accuracy curves, e. g. the improvement on the bisecting line.

6.6.3 Using Probability Estimates for Accuracy Trade-Off

Decision trees can store the class probabilities in leaf nodes instead of only storing the majority class. If the number of training items used for these probability estimates is sufficiently high, the accuracy trade-off of the classifier can be tuned varying the probability threshold (50% for majority voting and binary classification). Since the leaf nodes of the decision trees produced by the Weka toolkit are annotated with their size and the number of incorrectly classified items, the (unsmoothed) probabilities can easily be read from the trees. No new training is required.

6.6.4 Discriminative POS n-grams, Skipgrams and Parse Fragments

Similarly to the discriminative rule features, we could measure the discriminating ability of n-grams and add the discriminativeness of the most discriminative n-gram of the input sentence to the n-gram feature set. This would also be a step towards implementing Sun

et al. (2007)'s skipgram method which uses the discriminativeness of skipgrams to decide which skipgrams to include as features for training a support vector machine. (See also Section 5.5.1 of Chapter 5.) Furthermore, the work on discriminative rule features should be extended to larger parse fragments as the work of Post (2011) suggests that there is room for improvements using such features.

6.6.5 Trying other Machine Learning Methods

Andersen (2006) reports "markedly inferior performance" for decision trees compared to Naive Bayes, Maximum Entropy and Balanced Winnow. Even though this can be explained with the very different feature set, especially its size (a large number of binary features indicating the presense of n-grams) which is known to pose difficulties to decision trees that can only consult one feature at each node (and the number of nodes is limited by the training set) and while our review of machine learning literature suggests that decision trees are a good choice for our feature sets, other machine learning methods could be tried. Initial experiments with support vector machines (SVM) showed that we have to drastically reduce the amount of training data due to high computational costs. Still, results with other learning methods will be interesting.

6.6.6 Combining the Methods of Chapters 4 and 5

The distorted treebank probability could be added to the features of the EPP model of Chapter 4 in the same way language model probabilities are added, i. e. both as a k-NN feature and for "factoring out". Since the distorted treebank probability is a good predictor of parse probabilities of the vanilla grammar, this feature has the potential to improve the EPP model considerably. Of course, all features of Section 6.2 could be added as well.

In the other direction, we could train decision trees on the APP/EPP grammaticality score of Chapter 4 in combination with the feature sets of this chapter. The features of the k-NN model of Chapter 4 could also be added.

Chapter 7

Comparative Review of Methods

Chapters 4 to 6 presented 21 methods for classifying sentences as either grammatical or ungrammatical starting from the APP/EPP method over four basic methods to methods using decision tree learning, combinations of feature sets and classifier voting. In this chapter, we examine in more detail a selection of these methods. Section 7.1 compares methods that use only one source of information, i. e. *a)* the basic methods of Chapter 5 and *b)* the corresponding voting methods. We also include here the APP/EPP method of Chapter 4 which is difficult to categorise as either basic or machine learning-enhanced. Section 7.2 breaks down results by error type and sentence length. Section 7.3 then moves the evaluation to authentic error data. Finally, Section 7.4 draws conclusions and points to future work for analysing the results further.

7.1 Comparison of Methods

Chapters 4 and 5 presented individual methods and while Chapter 6 does contain some comparison, its focus is on the effect of machine learning and of combining feature sets. This section compares the accuracy curves across methods using individual sources of information.

7.1.1 Comparison of Basic Methods

A comparison of the basic methods is interesting as it shows differences unaffected by the suitability of each method's feature set for the machine learning method we chose, i. e.

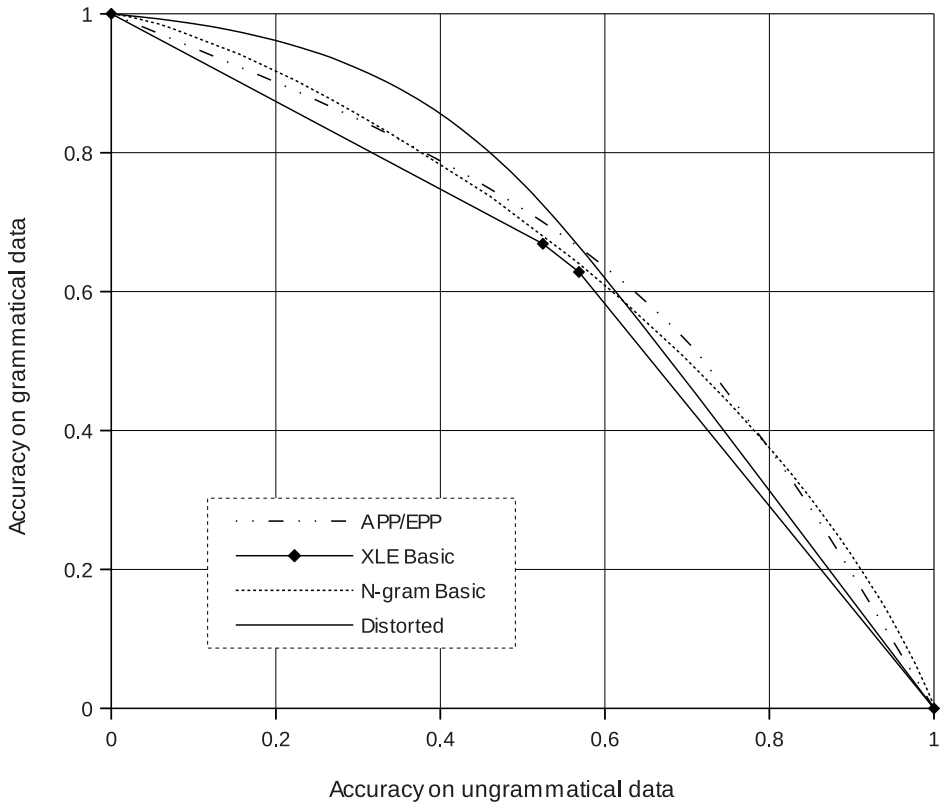


Figure 7.1: Accuracy graph for basic methods of Chapters 4 and 5 (excluding the PCFG pruning method of Section 5.3 which only marginally exceeds coin-flipping)

decision tree learning. In addition, Chapter 6 has shown that some of the basic methods of Chapter 5 outperform corresponding machine learning-enhanced methods for relevant parts of the accuracy curve.

Figure 7.1 shows the accuracy curves of the basic methods of Chapter 5[1] and the APP/EPP method of Chapter 4. On the left side of the graph, the distorted treebank method of Section 5.4 (solid line) spans an arc over the other methods, clearly outperforming them. The APP/EPP method (two dots, one dash) and the n-gram method (fine dots) of Section 5.2 are close together. For a balanced accuracy trade-off (middle of the graph), the APP/EPP is ahead of the n-gram method and at between 57.0% and 79.5% accuracy on ungrammatical data, the APP/EPP method is the best method in this com-

[1] with the exception of the PCFG pruning method of Section 5.3 which performed poorly

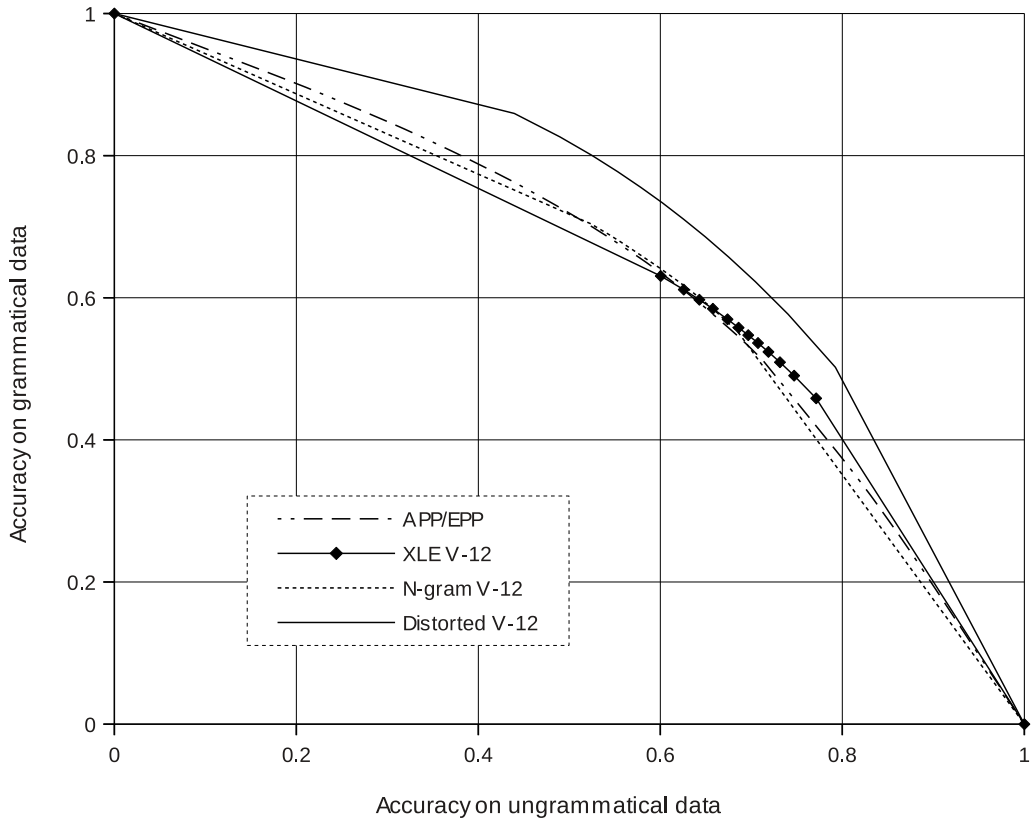


Figure 7.2: Accuracy graph for voting methods of Chapter 6 together with the APP/EPP method of Chapter 4 (excluding the rule feature method for which only two decision trees are available for voting, see Section 6.5.3)

parison. On the right fifth of the graph (accuracy on ungrammatical data greater than 79.5%), the n-gram method is best. Only the XLE method consistently stays below all other methods, though it gets close to the n-gram method when it is not interpolated with the trivial classifiers.

7.1.2 Comparison of Voting Methods with One Feature Set

In Chapter 6, we found that four out of the five voting methods that have been compared to their corresponding decision tree methods outperform the decision tree method, the only exception being the n-gram method (Section 6.5.4) — see the summary in Section 6.6. Figure 7.2 shows the accuracy curves of the voting methods that each use only one of the

four feature sets of Section 6.2 excluding the discriminative rule method, i. e. the voting methods of Figure 7.2 correspond to the basic methods discussed in the previous section. To aid a fair comparison, we show only the voting methods with 12 decision trees even though more decision trees are available for some of the methods. The APP/EPP method is included in Figure 7.2 as well. The most striking observation is that the distorted treebank method clearly outperforms the other three voting methods on the full range of accuracy trade-offs. The APP/EPP, XLE and n-gram methods are close together and their ordering changes compared to Figure 7.1: the XLE method is on top for 65.8% or higher accuracy on ungrammatical data (right side of the graph) and the n-gram method falls below the APP/EPP method for high accuracy on grammatical data (top-left quarter of the graph).

7.2 Influence of Error Type and Sentence Length

So far, we only looked at *overall* accuracy curves on artificially created ungrammatical sentences and grammatical BNC sentences. How do our classifiers perform for each of the error types of our artificial test data of Chapter 3 and for different sentence lengths?

7.2.1 Breaking down Results by Main Error Type

Naturally, for a breakdown by error type, the classifiers are evaluated on the subsets of the ungrammatical test data that contain an error of the respective type. It is difficult, though, to say whether grammatical test data should comprise the full set of grammatical sentences as in Chapters 4 to 6 and Section 7.1 above, or should be restricted to the grammatical sentences that are aligned to the ungrammatical test sentences in the parallel error corpus. We opt for the latter as we think that it is interesting to also investigate if there are differences between grammatical sentences that have been chosen by the error creation procedure for different error types.

In the following, we first give two examples of accuracy curves. Since we have to consider five error types and would like to cover most voting methods and the APP/EPP method, we then switch to a one-dimensional evaluation measure to present results more compactly.

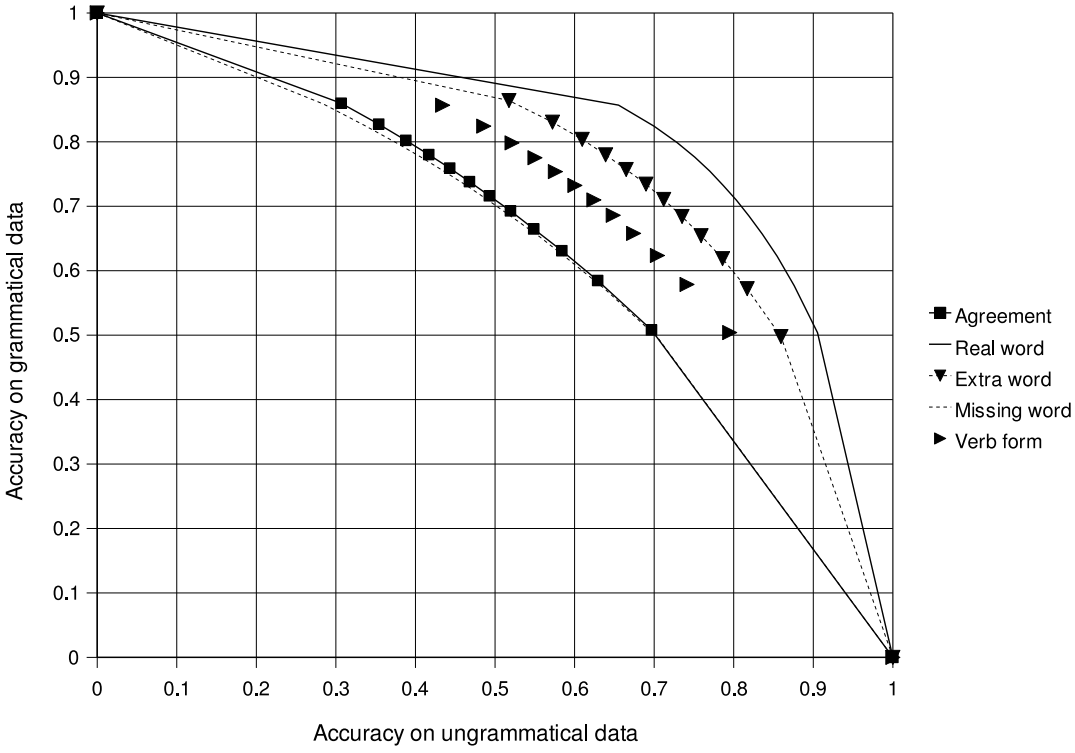


Figure 7.3: Accuracy graph for distorted treebank method with 12-classifier-voting broken down by main error type: to make it easier to distinguish the curves, three curves are shown with the accuracy points of the voting classifiers, one without the interpolating line segments and two with dashed lines.

Accuracy Curves for Main Error Types

Figure 7.3 shows the distorted treebank method with voting over 12 decision trees which is the method with the clearest separation of curves and accuracy points of the classifiers (shown in the graph for three error types only) among the basic voting methods of Chapter 6. The most challenging error types for this method are missing word errors (dashed line) and agreement errors (solid line with squares). Most reliably detected are real-word spelling errors (solid line). Lying somewhere in between are verb form errors (triangles pointing right, no line) and extra word errors (dashed line with triangles pointing down).

Since the accuracy points of individual voting classifiers are shown for three error types in Figure 7.3, we can see that accuracy on grammatical data is not affected much by restricting the test set to sentences aligned to particular error types. The n-th accuracy points of each curve line up horizontally with small deviations around one percentage point.

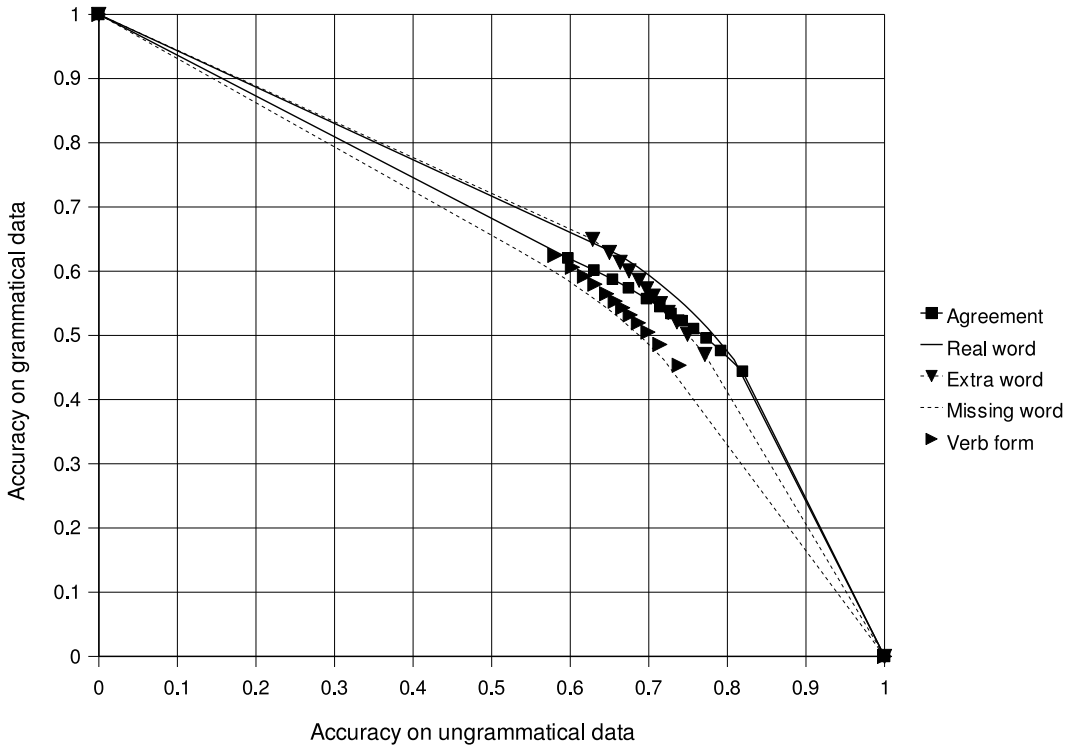


Figure 7.4: Accuracy graph for the XLE method with 12-classifier-voting broken down by main error type

In the upper part of the graph, accuracy on sentences that have been used to create verb form errors is up to 0.72 percentage points lower than for extra word errors. In the lower part of the graph, accuracy on grammatical sentences aligned to agreement errors are up to 1.27 percentage points higher than for sentences aligned to ungrammatical sentences with extra word errors. The wider accuracy range for extra word errors is plausible considering that a wide range means that the decision trees used in the voting method often disagree: for agreement and verb form errors, many sentences are excluded by the error creation procedure while extra word errors can always be inserted into a sentence — see Section 3.3.4 of Chapter 3.

Figure 7.4 shows the same type of graph as in Figure 7.3 for the XLE voting method of Chapter 6. The accuracy curves are difficult to distinguish as they fall closer together and have different slopes. Nevertheless, we can see in Figure 7.4 that missing word errors are also most challenging for the XLE method. However, the position of agreement errors and

	AG	RW	EW	MW	VF
APP/EPP	59.76% (4)	**63.77%** (1)	63.56% (2)	59.62% (5)	62.39% (3)
XLE V12	61.20% (3)	63.85% (2)	**63.93%** (1)	59.07% (5)	60.45% (4)
Ngram V12	55.96% (5)	64.12% (3)	**66.86%** (1)	59.44% (4)	65.20% (2)
Rule V02	58.40% (4)	62.19% (2)	**62.97%** (1)	56.84% (3)	56.60% (5)
Dist V12	60.72% (5)	**76.28%** (1)	71.13% (2)	60.48% (4)	66.65% (3)
Dist V29	60.86% (4)	**76.43%** (1)	71.27% (2)	60.60% (5)	66.77% (3)
All4 V29	65.13% (4)	**77.61%** (1)	73.97% (2)	62.99% (5)	70.14% (3)

Table 7.1: Accuracy parity points broken down by method and main error type

verb form errors swap in the graph compared to the distorted treebank method discussed above. Due to crossing curves, there is no single error type that is most reliably detected by the XLE voting method **XLE V12** for all accuracy trade-offs: below 64.2% accuracy on ungrammatical data, extra word errors are the best performing, followed by real-word spelling errors and, above 81.3%, agreement errors.

The differences in accuracy on grammatical data are more pronounced in the XLE method than in the distorted treebank method and are clearly visible in Figure 7.4: grammatical sentences permitting an agreement or verb form error to be inserted are misclassified as ungrammatical more often than unrestricted grammatical sentences, i. e. sentences permitting an extra word error to be inserted.

Accuracy at the Parity Point

The intersection of an accuracy curve and the bisecting line is the point where the method achieves identical accuracy on both grammatical and ungrammatical test data. In the following, we call this point the parity point. Accuracy at the parity point is a one-dimensional evaluation measure that we used in Chapter 4 as an objective function.

Table 7.1 shows the accuracy at the parity point for the APP/EPP method, the four basic voting methods and combined method using all four feature sets (Chapter 6) broken down by error type. The distorted treebank method is available both with 12 and 29 decision trees. The same data is shown as a bar chart in Figure 7.5. The poor detection of missing word errors is confirmed for all methods. Only the n-gram and discriminative rule methods struggle more with other error types: agreement errors in case of the n-

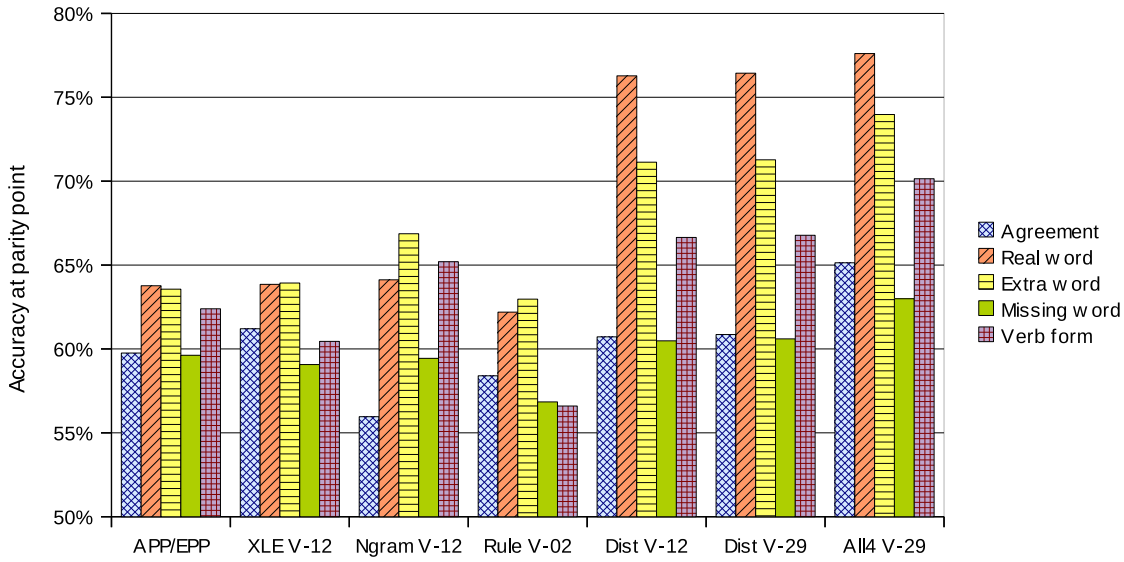


Figure 7.5: Accuracy parity points broken down by method and main error type

gram method and verb form errors in case of the discriminative rule methods. Real-word spelling and extra word errors are detected well across the board. Verb form errors are ranked second or third for all methods, with the exception of the XLE and discriminative rule methods.

7.2.2 Breakdown by Sentence Length

In many areas of NLP, performance depends on sentence length. For example, in parsing, long sentences tend to cause problems as ambiguity and processing time explode with sentence length and the risk increases that the sentence contains an uncovered construction misleading the analysis. Figure 7.6 shows the accuracy of selected classifiers from Chapters 4 and 6 over sentence length. The overall performance differences have been discussed in Section 7.1 above. Here, we focus on the influence of sentence length. A clear (negative) correlation with sentence length can be seen. The discriminative rule method (triangles without line) depends the least on sentence length: it maintains an accuracy around 56% for long sentences. (It only falls to 54% for sentence length 100, the limit we use in parsing.) The combined method (triangles on dashed line) and the distorted treebank method (squares on solid line) start from a high accuracy and decay with a slope similar to the average of the other slopes. The n-gram and XLE method are the most dependent on

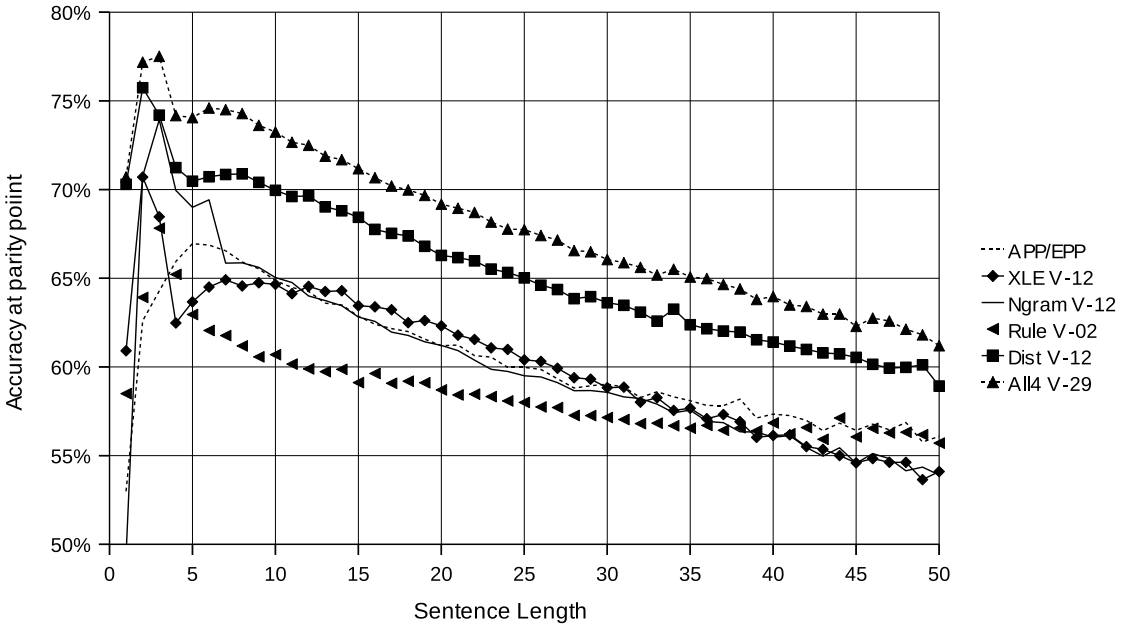


Figure 7.6: Influence of sentence length on accuracy of our classifiers

sentence length: they keep a distance of 4 percentage points from the distorted treebank method and fall more quickly than the APP/EPP method (dashed line).

For very short sentences, the ranking of methods and the absolute accuracies change considerably: the n-gram method performs well for sentences with between two and six tokens and peaks at 73.94% accuracy for three tokens. The discriminative rule method also performs best for sentences with three tokens (accuracy 67.82%). The peak shifts to two tokens for the XLE method (accuracy 70.70%) but leaves a surprising gap for sentences with four and five tokens for which the classifier deteriorates. The APP/EPP method shows a different behaviour: it peaks for sentences with five tokens (accuracy 66.95%) and does not perform well for sentences with three or fewer tokens. All basic voting methods apart from the distorted treebank method underperform for sentences with just one token.

Table 7.2 breaks down results further by main error type for the combined method **All4 V-29**. We stratify sentence length into eight groups for a more compact presentation. The length boundaries are chosen with recursive binary splits minimising the size difference of the two halves. Each stratum contains between 397,426 and 487,855 test sentences (on average: 450,000). The accuracy figures confirm both the correlation with sentence length

Error Type	Length range	Accuracy	|U|	|G|
Agreement	1–6	72.4%	17,614	17,602
Agreement	7–10	70.8%	35,130	35,132
Agreement	11–14	69.4%	41,140	41,144
Agreement	15–18	67.7%	44,533	44,538
Agreement	19–23	66.0%	54,515	54,510
Agreement	24–28	63.8%	47,474	47,474
Agreement	29–37	61.9%	59,069	59,073
Agreement	38–	58.5%	60,525	60,527
Real word	1–6	81.4%	38,653	38,653
Real word	7–10	80.2%	54,059	54,059
Real word	11–14	79.7%	49,396	49,396
Real word	15–18	79.0%	44,721	44,721
Real word	19–23	78.1%	49,322	49,322
Real word	24–28	76.7%	38,766	38,766
Real word	29–37	74.8%	44,619	44,619
Real word	38–	69.7%	40,464	40,464
Extra word	1–6	79.9%	64,267	77,466
Extra word	7–10	77.3%	52,101	50,793
Extra word	11–14	76.1%	44,857	43,272
Extra word	15–18	74.2%	39,641	38,618
Extra word	19–23	72.4%	43,884	42,465
Extra word	24–28	71.3%	35,384	33,421
Extra word	29–37	69.2%	41,380	38,535
Extra word	38–	65.5%	38,486	35,430
Missing word	1–6	68.5%	70,625	56,668
Missing word	7–10	66.3%	52,512	54,049
Missing word	11–14	64.1%	45,670	47,047
Missing word	15–18	62.9%	40,657	41,882
Missing word	19–23	61.4%	43,508	45,223
Missing word	24–28	60.1%	33,954	36,065
Missing word	29–37	59.0%	38,590	41,458
Missing word	38–	56.2%	34,484	37,608
Verb form	1–6	77.2%	28,847	28,847
Verb form	7–10	76.6%	45,693	45,693
Verb form	11–14	73.8%	46,240	46,240
Verb form	15–18	71.4%	45,883	45,883
Verb form	19–23	69.4%	52,553	52,553
Verb form	24–28	67.7%	43,061	43,061
Verb form	29–37	65.9%	50,633	50,633
Verb form	38–	62.4%	47,090	47,090

Table 7.2: Breakdown by error type and sentence length: accuracy at parity point of the combined method **All4 V-29**, number of ungrammatical test sentences (|U|) and number of grammatical sentences that fall in the sentence length range and are aligned to an ungrammatical sentence with an error of the respective type (|G|).

(see above) and, comparing numbers for fixed length ranges, the dependency on the error type observed in Section 7.2.1.

Table 7.2 also shows the number of ungrammatical and grammatical test sentences in each sentence length stratum and for each error type. The numbers vary considerably and cannot be explained by the sizes of the strata (see previous paragraph) or by random selection of sentences for the five error types. The artificial error data does not mirror the length distribution of the BNC (column $|G|$ for extra word errors in Table 7.2). Short sentences are under-represented in agreement, verb form and real-word spelling errors. The degree of under-representation correlates well with the number of sentences the error creation procedure produces for each error type (3.1, 3.5 and 5.4 million sentences, see Section 3.3.4 of Chapter 3), i. e. for error types for which the procedure has difficulties in inserting errors, the difficulties are the most pronounced in short sentences. It is plausible that the error creation procedure is more likely to match POS and token patterns it uses to find possible error insertion points in long sentences than short sentences.[2, 3]

7.2.3 Normalisation of the Sentence Length Distribution

The dependency of classifier performance on sentence length and the differences in the sentence length distributions between error types observed in Section 7.2.2 above can partly account for the differences in accuracy between error types shown in Section 7.2.1: the good results for extra word errors may just be caused by the higher fraction of short sentences in the test set while agreement suffer from a small number of short sentences. To counter the influence of the sentence length distribution on our evaluation, we normalise

[2]The higher than expected number of short sentences for extra word and missing word errors can be explained by our stratification balancing the overall length distribution including ungrammatical sentences. The shortage of short sentences for agreement, real-word spelling and verb form errors is compensated by additional short sentences for extra word and missing word errors.

[3]Table 7.2 shows small differences between the number of grammatical and ungrammatical sentences for agreement errors. Agreement errors should not change sentence length and, by the aligned construction of the test set, the numbers should be identical. An analysis of length differences shows the presence of 709 missing word errors. In a random sample of 8 cases, these errors are caused by wrong POS tags NNS or VBZ provided to the error creation procedure for single character tokens such as P (middle name), s (is) or V (in T V). These tokens are unknown tokens to the POS tagger, NNS and VBZ are reasonable POSs for unknown tokens and the context may suggest them, e. g. in the sentence *Birthday/NN card/NN* ***+/VBZ*** *Pressie/NNP for/IN Chris/NNP* where "+" is interpreted as a verb instead of a coordinating conjunction (CC). The error creation procedure assumes that nouns and verbs tagged NNS or VBZ have a suffix "s" which can be removed to cause a number agreement error. (Some exceptions, e. g. "children", "women" and "men", are considered.)

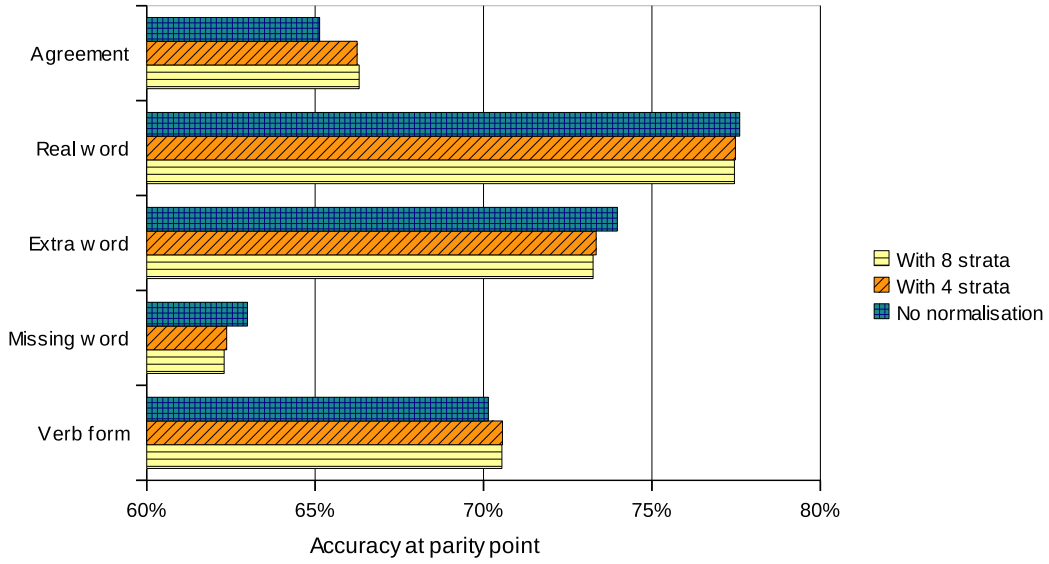


Figure 7.7: Stability of normalisation of sentence length distribution with increasing number of sentence length strata for each of the five main error types; method: All4 V-29

the accuracy figures using the sentence length stratification of Table 7.2.

Normalisation Method In the results of Section 7.2.1, certain sentence lengths are under-represented or over-represented for different error types. We address this by calculating a weighted average of the accuracy scores so that the weights normalise the impact of each accuracy score on the overall accuracy. To avoid unreliable statistics for long sentences, we use length strata as in Table 7.2. instead of individual sentence lengths. Each stratum represents roughly the same amount of test data. Therefore, the weights are one, i. e. we simply calculate the average accuracy of stratified results. How many strata are necessary? Figure 7.7 shows in the example of method **All4 V-29** that there is only a small difference in normalised accuracy between using four and eight strata compared to the difference between normalisation and no normalisation. We conclude that eight strata are sufficient.

Normalisation Results Figure 7.8 shows the revised, normalised bar diagram corresponding to Figure 7.5. Now, results for extra word errors are not as high as without normalisation and results for agreement errors improve for all methods (though only a little in case of the discriminative rule method). Normalisation changes the ranking of error

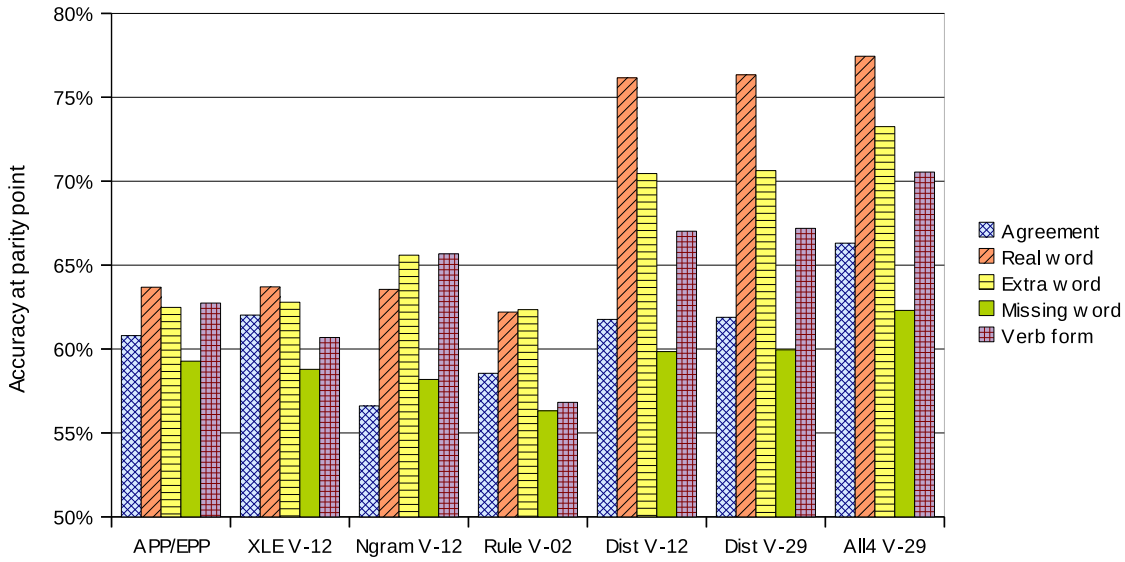


Figure 7.8: Normalised accuracy parity points broken down by method and main error type

types by performance for most methods: extra word and verb form errors swap positions for the APP/EPP method and the n-gram method. The XLE method now scores best for real-word spelling errors. Fourth and fifth place (missing word and verb form errors) change position for the discriminative rule method. The normalisation does not affect the ranking of error types for the distorted treebank method and the combined method.

7.3 Evaluation on Authentic Error Data

We now turn our attention to the performance of a selection of our various classifiers on authentic test data in order to see whether and how well results carry over to data different from the artificial training data used to build the classifiers. Section 3.2.2 of Chapter 3 lists three types of learner data we use for testing and as a fourth test set we have Foster's parallel error corpus, i. e. in total we have four test sets:

1. Advanced Learner Essays (608 sentences): this is the most realistic test set as both grammatical and ungrammatical sentences are authentic. It contains small subsets of the ICLE, JPU and Pelcra learner corpora — see also Chapter 3.

2. Spoken language (4602 sentences + 500 corrections): this corpus is annotated with

learner level and L1.

3. Mass noun errors (2 × 123 sentences): this corpus tests an error type not explicitly covered by our artificial training data. However, mass noun errors may be present as missing word, extra word or agreement errors:

   (7.1) *Drink **plenty tea** or other boiled or canned liquids.*

   (7.2) *I especially like drinking **the** tea.*

   (7.3) *I am ready to pay for those **paper**.*

4. Foster's error corpus (98 sentences held-out data): this test data is close to the training data as it has been collected in the same way as the data that informed the error creation procedure.

Note that not all sentences of the test data are annotated as either grammatical or ungrammatical, e. g. 113 of the 608 essay sentences (18.6%) are classified as "questionable" by our annotator. We exclude them from the evaluation as our evaluation measure (Chapter 3) requires a binary annotation as either grammatical or not.

The methods we consider are the three basic methods of Chapter 5 that perform well, namely the XLE, *n*-gram and distorted treebank method, the three corresponding voting methods of Chapter 6 and the combined method using the XLE, *n*-gram and distorted treebank feature sets.[4] Since the test data is separate from the training data, we can combine all 120 decision trees of the ten cross-validation runs with 12 trees each in voting classifiers. We call this method **X+N+D V-120**.

In the following, we first compare the accuracy curves for the test sets summarised above (Section 7.3.1), closely following Wagner et al. (2009). Then, we check whether machine learning with artificial training data can account for the poor performance of our classifiers.

7.3.1 Accuracy Curve for Combined Method X+N+D

Figure 7.9 shows that we lose some accuracy when we switch from artificial test data to real data. Method **X+N+D V-120** performs best on the held-out section of the corpus of

[4]This method is very close to the method using all four feature sets when measured on artificial test data (Figure 6.20 in Chapter 6) and results are available from Wagner et al. (2009).

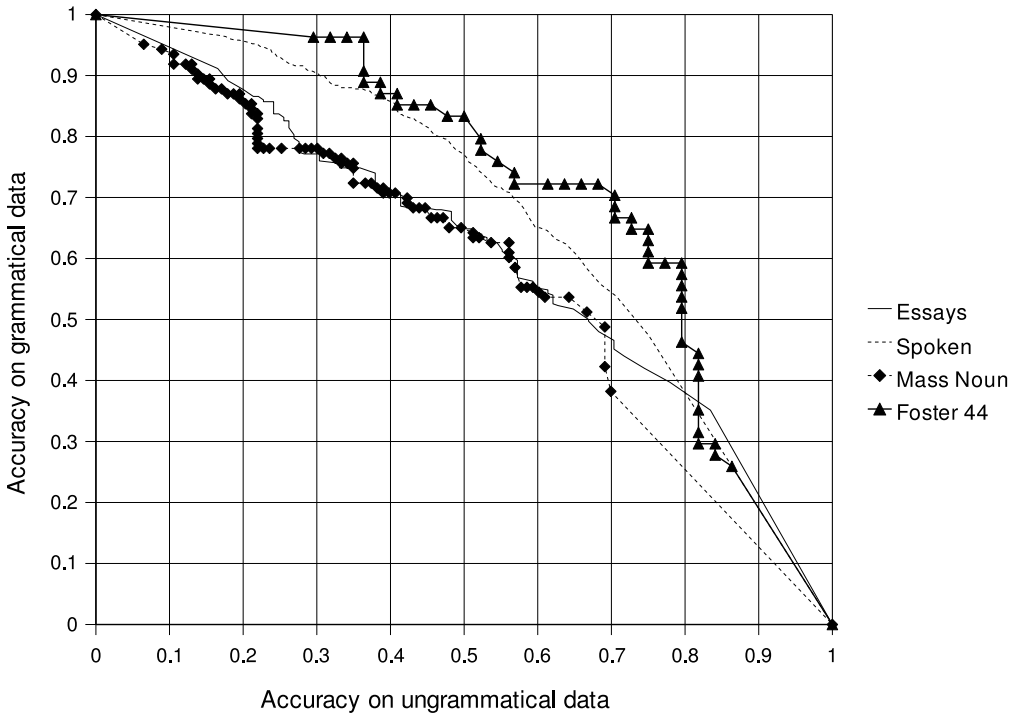


Figure 7.9: Accuracy graph for the combined method **X+N+D V-120** for the four authentic test corpora

naturally occurring errors that informed our automatic error insertion procedure ("Foster 44"). In contrast, the results for Essays and Mass Noun data are poor. At 70% accuracy on the grammatical side of the corpora, the baseline of randomly flagging 30% of all sentences is surpassed by only 10 percentage points to 40% accuracy on ungrammatical data. The results for spoken learner data are much better. Here, 57% accuracy is reached under the same conditions. At 95% accuracy on grammatical data, over 20% of ungrammatical spoken sentences are identified, more than 4 times over the 5% baseline.

The drop in accuracy observed when moving from synthetic test data (Figure 6.19) to real test data (Figure 7.9) confirms the well-known machine-learning maxim that training and test data should be as similar as possible. The best results for the real test data come from the Foster 44 corpus which has a similar distribution of error types as the synthetic training data. The curve oscillates around the curve for artificial data (not shown here but in Figure 6.20 in Chapter 6). The artificial data seems to mirror Foster's error corpus well. The low results for the Mass Noun data can be explained by the absence of this type of error from our training data. Sun et al. (2007) also report a large drop in accuracy (from approximately 82% to 58%) when they apply a classifier trained on Chinese English data to Japanese English test data.

The difference between the Essays and Spoken test sets might be due to the source of the grammatical sentences that are used to plot the accuracy curve. The grammatical essay sentences are produced by learners themselves along with the ungrammatical sentences, while the transcribed spoken sentences are corrected by a native speaker. It is possible that the level of the learner is also playing a role here. The sentences in the Essays test set have been produced by advanced learners, whereas the sentences in the Spoken test set have been produced by learners of various levels.

7.3.2 Basic Methods and Effect of Machine Learning

While Chapter 6 shows a positive effect of using machine learning with artificial training data when test data is also artificial, it is not guaranteed that the effect is positive also on authentic test data. The learned models might be too specific to BNC data. In particular, the error anticipation-free APP/EPP, basic XLE and basic n-gram methods (Chapters 4

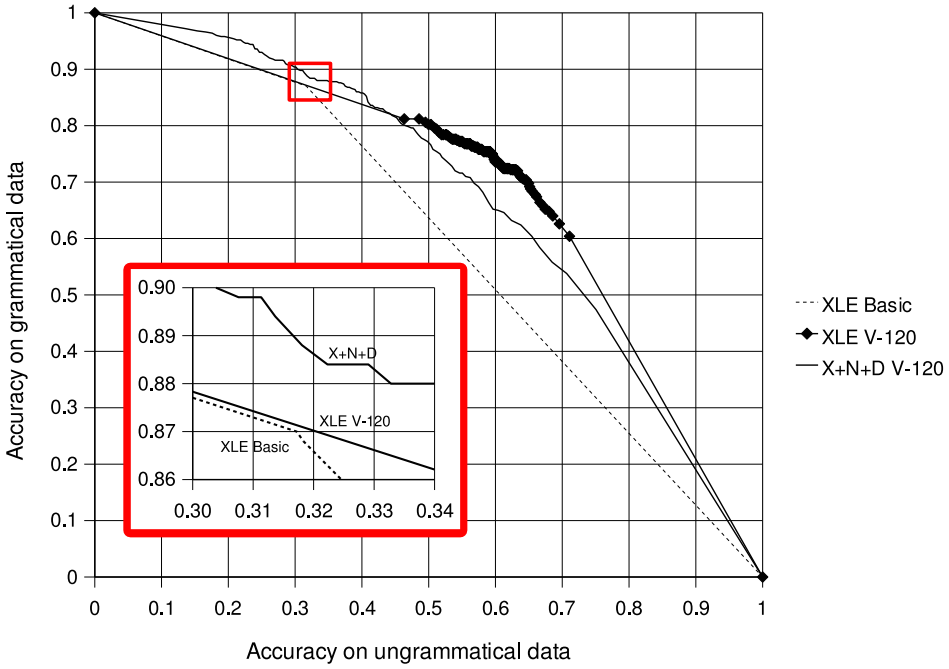


Figure 7.10: Accuracy graph for the XLE methods of Chapters 5 and 6 evaluated on spoken language learner data

and 5) may work well on authentic test data.

XLE Method

The basic XLE method of Chapter 5 has just two classifiers. These classifiers differ in the treatment of parser exceptions (time-outs and out-of-memory). For the spoken language learner test data, parser exceptions are rare and, consequently, the accuracy points fall close together: (31.7%, 87.0%) and (31.8%, 86.8%). Figure 7.10 shows the basic XLE method (dashed line, **XLE Basic**), the machine learning-enhanced XLE method (line with diamonds, **XLE V-120**) and the combined method with XLE, n-gram and distorted treebank features (solid line, **X+N+D V-120**). Also shown is a box that zooms in on the two accuracy points of the basic XLE method which are very close to the top-left interpolation line of Method **XLE V-120**. As these points, which are the turning points of the dashed line, fall below the interpolation line, the basic XLE method is technically outperformed by Method **XLE V-120** but the difference of approximately 0.15 percentage

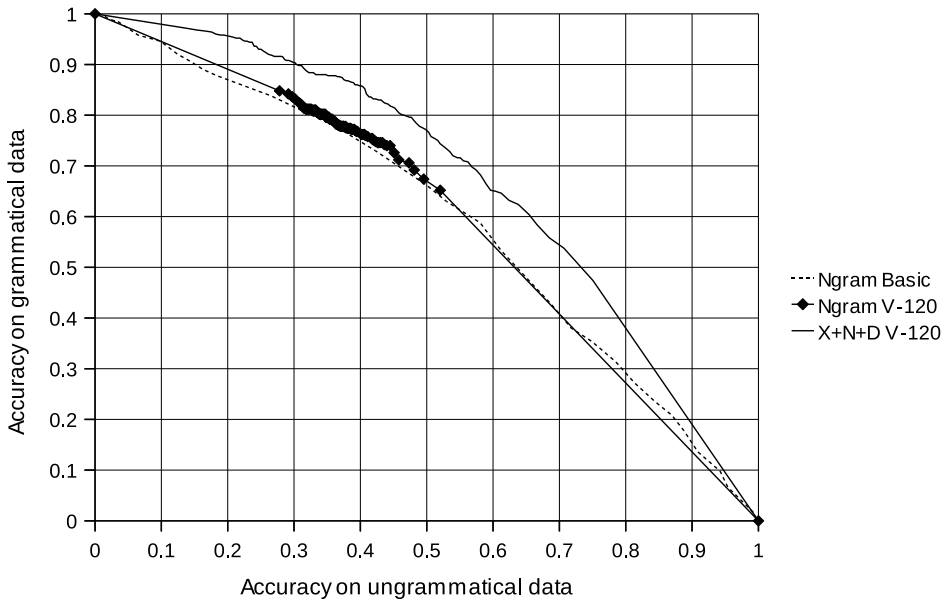


Figure 7.11: Accuracy graph for the n-gram methods of Chapters 5 and 6 evaluated on spoken language learner data

points is not statistically significant given that there are just 500 grammatical test items.[5] We can conclude from this alone that machine learning with artificial training data has no negative effect on the XLE method in our classification task. In addition, the methods are already close when evaluated on artificial test data (Figure 6.13 in Chapter 6). For both test sets, the machine learning-enhanced method has its strength at a different accuracy trade-off than the basic method.

It is interesting that Method **XLE V-120** outperforms the combined method for accuracy on ungrammatical data over 45% (right half of Figure 7.10). Apparently, good decision rules that are learned using just the XLE feature set are lost when feature sets are combined. The decision trees of the combined methods possibly prefer distorted treebank features as these give better results than XLE features on artificial test data (Figure 7.2).

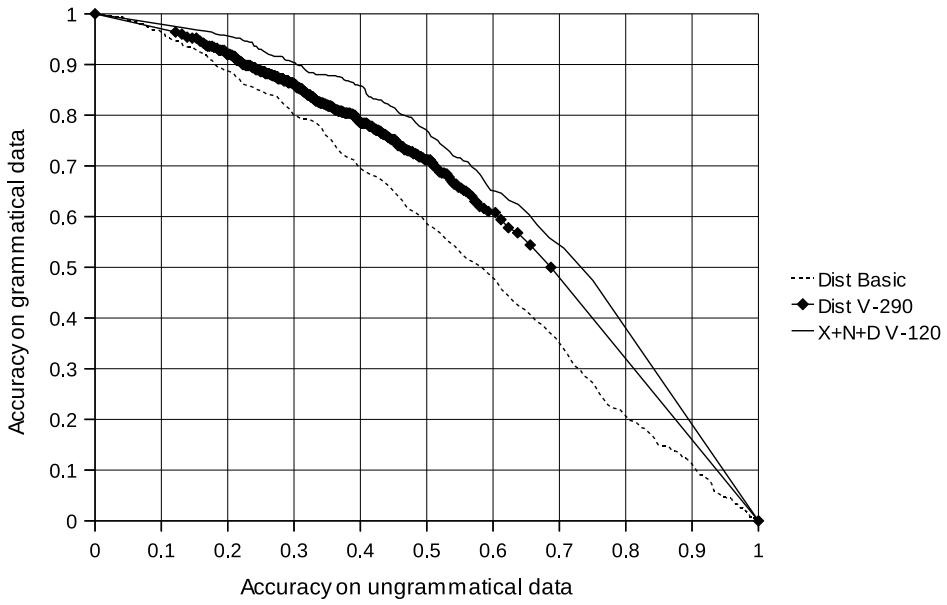


Figure 7.12: Accuracy graph for the distorted treebank methods of Chapters 5 and 6 evaluated on spoken language learner data

N-gram Method

Figure 7.11 shows the basic n-gram method (dashed line, **Ngram Basic**), the machine learning-enhanced n-gram method (line with diamonds, **Ngram V-120**) and the combined method with XLE, n-gram and distorted treebank features (solid line, **X+N+D V-120**). The accuracy curves of the two n-gram methods are very close. Again, as for the XLE method, machine learning with artificial training data does not harm the classifiers. However, again as for the XLE method, improvements were not expected as improvements on artificial test data are already small and limited to a short range of accuracy trade-offs (Figure 6.14 in Chapter 6).

Distorted Treebank Method

Figure 7.12 shows a consistent improvement over the distorted treebank method (dashed line, **Dist Basic**) for all accuracy trade-offs when machine learning is applied (line with diamonds, **Dist V-290**). The improvements are largest for accuracy on ungrammatical

[5]With 500 test items, accuracy must be a multiple of 0.2%. Smaller differences are only possible as we interpolate between classifiers calculating expectation values for randomly choosing between them — see Chapter 3.

data between 50% and 75%. This is similar to the improvements for high accuracy on ungrammatical data observed with artificial test data (Figure 6.15 in Chapter 6). In contrast to the XLE method, the accuracy curve stays below the curve of the combined method **X+N+D V-120** (solid line).

7.4 Summary and Future Work

We compared results of Chapters 4 to 6, expanded the evaluation to a breakdown by error type and tested selected methods on authentic learner data. Performance is varied and the ranking of methods depends not only on the desired accuarcy trade-off but also on error type, sentence length and the type of (test) data. In the following, we draw conclusions for future work.

7.4.1 Sentence Length Distribution

To exclude sentence length as a factor affecting the performance of methods for different error types, future work could make sure that the length distribution is the same for all error types in artificial test data, e. g. we could restrict the selection of sentences to sentences for which errors of all error types can be inserted, i. e. we could build a six times parallel error corpus: each alignment bead contains five ungrammatical sentences, one for each error type, and the grammatical source sentence. Of course, after the insertion of extra word or missing word errors, the length distribution would still be shifted by one token. On the other hand, there is no need to precisely rank error types. Rough performance figures are sufficient to identify a method's strengths and weaknesses which can then lead to ideas for improvements. Comparable figures are more important when the best method for an error type has to be selected and, in this scenario, we do have comparable figures in Section 7.2.

A related task for future work is to examine the sentence length distributions for different error types in authentic error corpora. For example, different error densities (per token) will cause different length distributions as we can see in Figure 7.13 which shows a sampling experiment with BNC data.

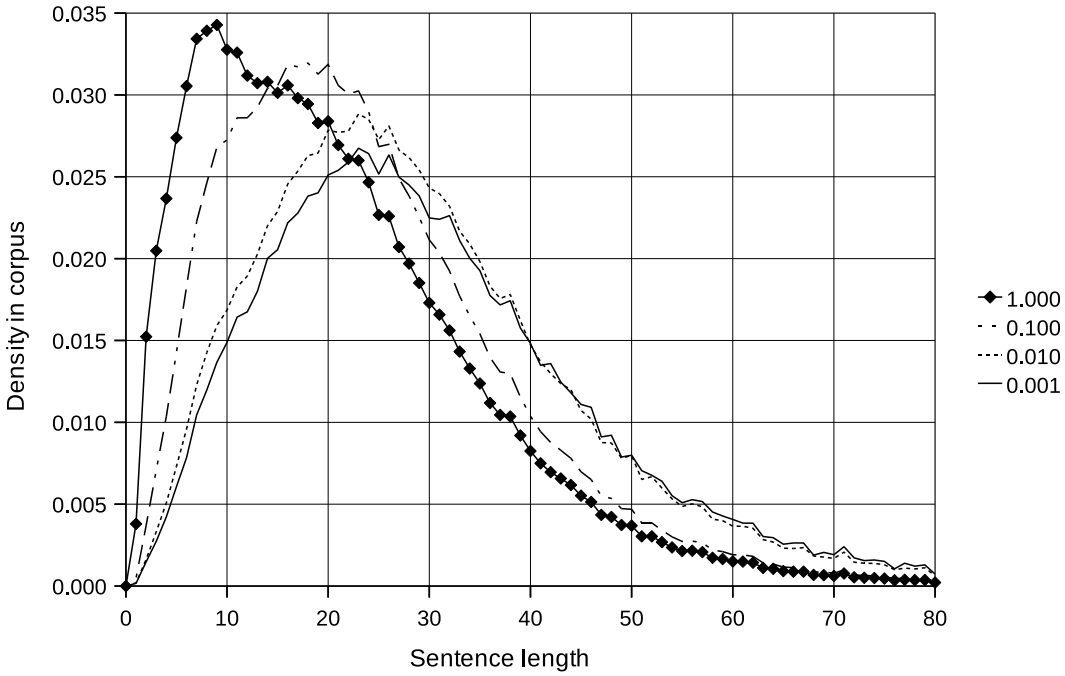


Figure 7.13: Effect of error density on the sentence length distribution: each token of the BNC is flagged as erroneous with the probability shown in the legend. The curves show the sentence length distributions for each subset of sentences with one or more errors. The curve for 1.0 shows the length distribution in the BNC as all sentences are included.

7.4.2 Expand Evaluation on Authentic Data

In Section 7.3, we tested nine methods on test data with authentic grammatical errors. There are many possibilities how the evaluation on authentic data can be expanded:

- The spoken learner data could be used to investigate the effect of learner level and L1 as it is annotated with this information.
- It would be interesting to look at the effect of machine learning for the essay data sets that underperform in Figure 7.9.
- We would like to compare the range of accuracy trade-offs covered with the same voting setup, i. e. 12 or 29 trees.
- A comparison of basic methods (Chapters 4 and 5) on authentic data would be interesting.

Chapter 8

Conclusions

This chapter summarises and draws conclusions from the work presented in Chapters 3 to 7. We start with our contributions in Section 8.1, summarise the experimental results (Section 8.2), and highlight what we learned from our research (Section 8.3). Finally, Section 8.4 discusses the implications for future research on error detection.

8.1 Contributions

In our opinion, the three main methodological contributions of this work to error detection research are (a) three methods for error detection using probabilistic parsing, (b) a method for selecting classifiers in the absence of a single measure of optimality, and (c) a method for evaluating error detection methods with measures that are independent of error density and for evaluating methods for the full range of trade-offs between finding all errors and only flagging errors. We summarise these contributions below.

Apart from methods, data is central to data-driven methods like parsing with treebank-induced grammars and training machine learning methods. Such data needs to be pre-processed and annotated (Chapter 3). We summarise the data sets produced in the course of our research in Section 8.1.4. The contribution section ends with a list of miscellaneous smaller contributions.

8.1.1 Error Detection with Probabilistic Parsing

The motivation of our choice of probabilistic parsing with treebank-derived grammars for error detection is its success in other areas of NLP. We presented and tested three new approaches of which two performed well.

Distorted Treebank

Our best-performing method for using probabilistic parsing for error detection uses an error treebank or distorted treebank that we automatically derive from a vanilla treebank. Two grammars are induced, one from each treebank. Input is parsed with both grammars and a grammaticality judgements is based on a comparison of the parse results obtained with the two different grammars.

The distorted treebank method comes in two flavours: *(a)* the basic method of Section 5.4 of Chapter 5 uses the likelihood ratio between the best parse trees according to two grammars: a vanilla treebank grammar and a grammar induced from a distorted treebank. *(b)* The machine learning-based method furthermore exploits differences between the tree structures and looks at parse results of three instead of two grammars (Section 6.2.3 of Chapter 6). The distorted treebank methods performs well — see also Section 8.2 below — and the computational overhead of parsing input with two or three parsers is not prohibitive.

APP/EPP Method

Chapter 4 introduced the error anticipation-free APP/EPP method that can be trained with positive data alone. Different from previous approaches that do not use negative training data, the APP/EPP method does not rely on the generation of candidate correction which would require an error model.[1] The method's performance compares well to other basic methods and outperforms parsing with the ParGram English LFG (Chapter 7). Also, it is computationally inexpensive for a parsing-based method as only one parse per input sentence is computed at test time. The list of ideas for future work in Chapter 4 indicates that there is room for improvements.

[1] Nevertheless, it should be possible to combine the candidate correction approach with the APP/EPP method. Future work has to show whether such a combination is beneficial — see Section 8.4.2.

(P)CFG Pruning

Given the observation in the literature that treebank-induced PCFGs parse almost any input and that coverage of grammatical language stays high if rules with a low frequency in the treebank are removed from the grammar, one may think that these rare rules are responsible for the robustness to ungrammatical input and that this can be exploited to detect grammatical errors. However, Section 5.3 of Chapter 5 showed that rare rules of a treebank-induced (P)CFG are as important for coverage of grammatical input as for robustness to ungrammatical input. Removing rare rules increases the fraction of unparsable sentences almost regardless of grammaticality.[2] Trying a wide range of frequency thresholds for pruning rare rules, we did not succeed in constructing a grammar that is substantially more discriminative than the vanilla grammar.

8.1.2 Convex Hull of Classifiers

Extending the comparison of classifiers in the accuracy plane from two to three and more classifiers, we concluded that a sequence of optimal classifiers for the full range of accuracy trade-offs can be derived from the convex hull of the accuracy points of all candidate classifiers (including the trivial classifiers flagging all or no sentences as ungrammatical). Effectively, we propose a meta-training method: various candidate classifiers are trained as usual, additional classifiers are derived, e. g. by varying a threshold or with classifier voting, and finally all classifiers that are inferior to a linear combination of other classifiers are discarded (Section 3.5.2 of Chapter 3). While this method is equivalent to the ROC convex hull method which has been developed in other fields (Section 3.5.3 of Chapter 3), its application to grammatical error detection is new.

8.1.3 Evaluation

Evaluation drives research in NLP forward with measures of how accurately methods solve the problem at hand, e. g. parsing sentences into phrase structure or dependency trees or, in our case, classifying sentences as either grammatical or ungrammatical. Successful methods are studied and refined further. Methods that do not reach the performance of

[2] A very small bias towards rendering ungrammatical input unparsable can be observed.

the best methods so far receive far less attention. Empirical evaluation of this kind often assumes that performance can be expressed in a single quantity and that methods can be ordered accordingly. However, error detection is a task with unclear costs of the two types of classification errors (failure to detect an error vs. falsely reporting an error) and with a varying a priori class probability (error density). These parameters depend on the application, the individual user or the text to be checked.

While one can argue that all basic NLP technologies face similar challenges, the accuracy curves of Chapters 6 and 7 show that there are marked differences in the ranking of methods depending on accuracy trade-off and error type. Therefore, we think that basic research on error detection methods should track these differences and advance a repertoire of methods addressing different settings rather than focusing on one setting. Applied research can then select and fine-tune the most suitable methods for each application. We propose to measure accuracy independently on grammatical and ungrammatical data (rather than precision and recall commonly used) as these two values *(a)* are independent of the error density of the test data and of misclassification costs, *(b)* allow anybody to reconstruct the confusion matrix of the classifications (if the amount of test data is reported as well), and *(c)* each combine linearly in weighted coin-flipping between classifiers.

The latter property draws attention to a method[3] for deciding between two classifiers in more cases than the trivial case that both accuracy measures agree on the ranking of classifiers (Section 3.4.4 of Chapter 3). This proved to be an important tool in Chapter 6 as classifiers often rank differently according to the two accuracy measures.

8.1.4 Data Sets

Based on an analysis of an authentic error corpus, we created a parallel artificial error corpus from the BNC which is first used outside DCU by Rethmeier (2011). Furthermore, the research for this thesis stimulated joint research on error detection that led to the annotation of learner and error corpora with sentence level grammaticality judgements and, for some corpora, with error type classifications and corrections (Chapters 3 and

[3]In retrospect, we think that a linear behaviour is not required and that the method can be adapted to non-linear combinations. However, we doubt that we would have found the method without the simplicity of the accuracy measure.

7). This annotation work which has been carried out by Jennifer Foster would have been challenging for the author as he is not a native speaker of English. We would like to highlight the following resources:

- the artifial error corpus derived from the BNC with 4.4 million grammatical sentences aligned to an ungrammatical sentence each (Section 3.3 of Chapter 3),
- the 608 sentences from advanced learner essays taken from three sources (ICLE, JPU and PELCRA) that we manually annotated with grammaticality judgements (Section 3.2 of Chapter 3),
- the corpus of 4,602 ungrammatical transcribed spoken learner sentences of which 500 have been hand-corrected (Section 3.2 of Chapter 3), and
- the distorted treebank derived from the WSJ part of the PTB (Section 5.4 of Chapter 5).

8.1.5 Other Contributions

In addition to the above contributions, we

- studied the effect of grammatical errors on parse probabilities (Wagner and Foster, 2009),
- provided Jennifer Foster with feedback and opinions on her error creation procedure which she continously developed further (Foster and Andersen, 2009),
- engaged in joint research on accurate robust parsing (Foster et al., 2008) and domain adaptation (Foster et al., 2007; Hogan et al., 2008; Foster et al., 2011a,b,c) — see also Section 5.4 of Chapter 5.

8.2 Summary of Experimental Results

In Chapters 4 to 7, we evaluated 21 methods for detecting grammatical errors. A subset of seven methods has been tested on authentic error data in Section 7.3 of Chapter 7. As mentioned in Section 8.1.3 above, the methods have different strengths: the ranking

of methods changes depending on (a) error type and (b) the accuracy trade-off between finding all ungrammatical sentences and not flagging any correct sentences. In the following, we highlight interesting results. Further details can be found in the summaries at the end of each chapter and in Chapter 7 which compares and tests a subset of methods on authentic test data.

- Section 4.2 of Chapter 4 shows that parse probabilities reflect grammaticality to some extent. Corrections often have a higher parse probability than the corresponding ungrammatical input.

- Three of the five main methods (Chapters 4 and 5) perform at very similar levels despite their different procedures, namely the basic XLE and n-gram methods and the APP/EPP method. Only the (P)CFG pruning method and the distorted treebank method show a markedly different performance.

- Although it is close, the basic XLE method does not reach the accuracy curve of the basic n-gram method (Figure 7.1 of Chapter 7). We expected a much better performance from a hand-crafted wide coverage precision grammar.

- The basic distorted treebank method performs very well for accuracy trade-offs with high accuracy on grammatical data.

- Machine learning produces classifiers that perform well at their inherent accuracy trade-off but fall below the corresponding basic method if extended to other accuracy trade-offs with interpolation to the trivial classifiers (flag all and flag none).

- The classifier voting method (Section 6.5 of Chapter 6) extends the range of accuracy trade-offs for which the basic methods are outperformed. Still, the basic n-gram method and the basic distorted treebank method are not outperformed for all accuracy trade-offs (Figures 6.13 and 6.14 of Chapter 6).

- Results vary for different error types, methods and sentence length. Generally, real word and extra word errors are detected most reliably while agreement errors and missing word errors pose difficulties for the classifiers — see Section 7.2.1 of Chapter 7.

- We got mixed results for different test sets with authentic error data. Naturally, results are best for the error data that informed the error model used to create the artificial training data. However, there are also marked differences between the accuracy curve of the transcribed spoken error corpus and the other two test sets (mass noun errors and L2 essays).

- The voting method combining XLE, n-gram and distorted treebank features is outperformed by the voting methods which just uses XLE features when tested on the transcribed spoken learner data. This indicates that the XLE features are more domain independent than the n-gram features (derived from BNC data) and the distorted treebank features (PTB data). This makes sense as the XLE grammar was not written with one domain in mind.

8.3 Lessons Learned

There are a number of interesting lessons to be learned from the work presented.

8.3.1 Grammaticality

We have seen that grammatical errors can be undetectable if the context does not suffice to indicate that a different sentence was or should have been intended (Sections 3.2.1 and 3.3.3 of Chapter 3). At the same time, superficially ungrammatical sentences can be grammatical in context, e. g. unfamiliar or new uses of a word. For example, names that share the same spelling with a function word can cause confusion when they appear at the start of a sentence, e. g. ***The*** *is a letter of the Cyrillic alphabet.* Also, interesting sentences arise when we refer to words. Lackowski (1963) gives examples such as *There's a big* ***if*** *in your plans*, *How do you spell* ***x****?* and *The Zambesi word for* ***hand*** *is* ***ugup***.

Therefore, if one judges sentences in isolation, grammaticality seems to reflect the effort one has to make to find a context in which the sentence makes sense on the syntactic level. This means that grammaticality is subjective and not necessarily consistent even for a single annotator. Furthermore, the availability of a plausible correction seems to stop the search for a possible context or interpretation in which the sentence is grammatical early.

This may be an explanation for the low inter-annotator agreement of the task of classifying a sentence as either grammatical or ungrammatical (Section 3.2.2 of Chapter 3). Future work should review psycholinguistics and second language aquisition research more deeply than we did in Chapter 2. As previous work linking reading effort and fluency of text, we came across Doherty and O'Brien (2009), further expanded by Doherty et al. (2010), who investigate whether eyetracking can be used to assess machine translation output and find that gaze time and fixation count correlate well with human-targeted translation error rate (HTER), an edit-distance-based measure of translation quality (Snover et al., 2006).

8.3.2 Basic Research or Application Focus?

Rozovskaya and Roth (2010c) draw attention to the shortcomings of casting error detection as the prediction of correct forms, e. g. articles, from context — see Section 2.2.4 of Chapter 2. They criticise previous work for generating unrealistic artificial training data, e. g. inserting exactly one error into each sentence. Machine learning works best if the training data closely resembles the test data. Therefore, it is not surprising that Rozovskaya and Roth (2010c) find that more precise modelling of the distribution of errors improves error detection and correction results. However, our results show that methods have different strengths (see Section 8.2). Since learner data varies considerably in factors such as error patterns, error densities, L1, genre, style and topic, there is no one-size-fits-all method for error detection. In applications, we are likely to have information about these factors or can estimate such statistics from the user's error history and from the new input text at hand.

Basic research on error detection should provide methods suitable for various applications, not just one method optimised for one particular test set. Both development and evaluation should take into account that error density and costs of misclassifications are not known in basic research on error detection.

8.3.3 Precision, Recall and F-Score

In our early research described in Appendix B, we used precision, recall and f-score for evaluation. Plotting curves for varying thresholds of the APP/EPP method, we found

that f-score can have flat maxima, that methods that are clearly better than coin-flipping can still have lower f-scores than the trivial "flag all" method, and that the optimal threshold for f-score can be very different from the optimal threshold for overall accuracy. Furthermore, we noticed here that precision (and hence also f-score) depend on the error density of the test data. In case of the "flag all" method, the dependency becomes simple: precision is then identical to the error density. See also Section 3.4 of Chapter 3.

8.3.4 Surprising Behaviour of the ParGram Grammar

It is well known that hand-writing grammars with high coverage is difficult and time consuming. The coverage of 62.82% of the core ParGram English LFG on the BNC which includes transcribed speech is expected. However, it was a surprise to see that coverage only drops to 43.15% for BNC sentences with artificially inserted errors (Section 5.1 of Chapter 5). We expected that grammatical errors would render sentences unparsable in most cases so that ungrammatical sentences can be flagged quite reliably and that limitation for error detection would only arise from the incomplete coverage of grammatical sentences. This is not the case. Two possible explanations are covert errors (Section 3.3.3 of Chapter 3) and implausible analyses.

8.3.5 Importance of Tuning Machine Learning Methods

With the k-nearest neighbour method, we chose a machine learning method in Chapter 4 that is simple, makes few assumptions and is reported to get close to the performance of the best methods. Nevertheless, we found improvements when its parameters were tuned. Also, the experiments of Chapter 6 show that combining various feature sets is important. Therefore, our impression from carrying out these experiments is that results depend more on the time and computational resources spent on parameter tuning and feature design than on the choice of machine learning method.

8.4 Impact on Future Research: What do to Next

Chapters 3 to 7 list ideas for future work at the end of each chapter. This section gives more general ideas for future work that either do not directly build on the experiments

presented or cannot be assigned to a particular chapter.

8.4.1 Expand Comparison of Methods

It would be interesting to include a candidate correction approach (Section 2.2.4 of Chapter 2) in the comparison of methods. Technically, a candidate generation module should reverse all possible error insertions of the artificial error creation procedure (Section 3.3 of Chaper 3). Given the missing word error type, the space of candidate corrections is unlimited. A possible solution could be to limit the proposed missing words to the most likely words according to an n-gram model. Still, the number of candidate corrections will be rather large if all error types are to be covered.

Sections 5.5.1 and 6.6.4 of Chapters 5 and 6 point to the skipgram method of Sun et al. (2007) who report high accuracy in the task of judging grammaticality (79.81 and 81.75% for two test sets). It would be interesting to see how these methods compare on our test data, in particular, for which accuracy trade-offs they are strong.

8.4.2 Beyond the Noisy Channel Model

In the candidate correction approach with a probabilistic target (grammatical) language model, the noisy channel model can be used to integrate an error model instead of applying ad-hoc thresholds as done in recent work — see also Section 2.2.4 of Chapter 2. However, the noisy channel model assumes that the target language model is a generative model while our research produces models that score the grammaticality of input strings. Future work should investigate how such grammaticality scores can be integrated into a candidate correction approach. A baseline method will rank candidate corrections by their grammaticality score. How can an error model be integrated in this setting? Is it redundant? How can models of ungrammatical language (source models in the terminology of the noisy channel model) be exploited?

8.4.3 Locating Errors

Each error detection method presented in Chapters 4 to 6 differs in how it could be extended to not only flag a sentence as ungrammatical but also to predict the error site

or, at least, a small number of candidate error sites. For example,

- the basic n-gram method of Chapter 5 can directly identify an n-gram as the candidate error and it should also be possible to apply the decision tree method of Chapter 6 to sub-sequences of a sentence.

- The distorted treebank method exploits differences between the parse trees obtained with vanilla grammar and distorted treebank grammar. Structural differences of the parse trees may point to candidate error sites. The lineage scores of the leaf ancestor metric assign a score to each token and these scores could be used to flag individual tokens.

In some applications, error site hypotheses may be useful on their own. Alternatively, candidate corrections could be generated for the identified errors sites. If the number of potential error sites is small, computational costs of candidate correction generation and ranking discussed as an issue in Section 8.4.1 above may be sufficiently low for the approach to be feasible.

8.4.4 Error Modelling: Error Types and Sentence Length

Section 2.3 of Chapter 2 and Section 3.3.1 of Chapter 3 list additional error types that others deemed relevant or built systems for. In addition to these error types, error detection work could be extended to optical character recognition, machine translation and speech recogniser errors.

Even though we think that using artificial data with one error per sentence is a valuable approach to designing and driving forward error detection methods, future work should investigate implications for experimental setup and possible improvements of modelling the distribution of errors more accurately in artificial error data. As a first step, one would have to analyse an authentic error corpus of sufficient size and close to the target text the system is developed for, e. g. learner corpora with a particular learner level and L1.

8.4.5 Dealing with Imperfections of Artificial Error Data

Using artificial error data is a means to get closer to authentic training data than it is possible with positive data only. Gamon (2010) proposes an alternative method to adjust an error detection system to the target data using a meta-classifier. His meta-classifier uses as features raw class probabilities from first-stage classifiers trained on a large corpus of positive training data and a small number of additional features. The meta-classifier learns how to map these features to final error detection and correction decisions using a small authentic error corpus — see also Section 2.3.2 of Chapter 2. Future work, therefore, should apply the meta-classifier idea to primary detection systems trained on artificial error data in order to adapt them to authentic error data before they are evaluated on such data.

8.4.6 L1 adaptation

Unless an error detection method is intentionally kept free of assumptions about the error types to anticipate, the L1 of the learner should be considered (a) as a feature for machine learning with error corpora (annotating training, development and test data with the L1) or (b) by training separate models for each L1 (splitting or adapting the training and/or development data). In both cases, at least some development data with errors made by various L1 speakers is needed. While there is previous work addressing L1 influence (Wang and Garigliano, 1992; Rozovskaya and Roth, 2010b), this aspect of error detection should receive more attention in future work. For example, the artificial error creation procedure proposed by Foster and Andersen (2009) could be adjusted to different L1s.

8.4.7 Convex Hull Method and Classifier Optimisation

The convex hull method introduced in Section 3.5.2 of Chapter 3 finds a subset of classifiers that span all accuracy trade-offs and are not outperformed by any of the classifiers not included in the subset. This method so far takes the individual classifiers as input and has no influence on how these classifiers are built. In Chapter 5, we varied parameters and thresholds to generate candidate classifiers. Future work should explore ideas for integrating the convex hull method and basic classifier interpolation in the accuracy plane

into machine learning. For example, one could

- change the objective function so that a machine learning algorithm optimises to prefer a pre-configurable accuracy trade-off and then train a sequence of classifiers for varying accuracy trade-offs.
- One could maintain a set of candidate classifiers and iteratively alternate between *(a)* expanding the set with variations or refinements of classifiers and *(b)* reducing the set to its convex hull.
- One could devise more specific learning algorithms, e. g. modify the split criterion of decision tree learning. Previous work on adapting classifiers to estimate class probabilities instead of only predicting the most likely class can give pointers to issues and solutions — see for example Schmid (2010) for decision trees.

8.5 Summary

We presented three new methods that address the task of classifying a sentence as either grammatical or ungrammatical using probabilistic parsing with treebank-induced grammars. We developed and evaluated these three methods, additional basic methods and combined methods on a newly created artificial error corpus that is grounded in an analysis of authentic error data. Seven methods are further tested on a corpus of transcribed spoken learner errors and one method is tested on three additional authentic error corpora.

The distorted treebank method is our best-performing method when high accuracy on grammatical data is demanded. The method combines well with n-gram and deep grammar-based approaches, as well as combinations thereof, in a machine learning-based framework. However, machine learning moved the strengths of the method to a different accuracy trade-off. It depends on various variables (error types, error density, sentence length, domain and text type) which method is most suitable in an application. We proposed to measure these individual strengths of each method with two accuracy values for grammatical and ungrammatical test data.

We provided a method for deciding between two classifiers in more cases than the trivial case that both accuracy measures agree on the ranking of classifiers. This method

has been extended to the case of multiple classifiers so that a small number of classifiers can be identified in a set of classifiers that cover the performance (as measured in accuracy on grammatical and ungrammatical data) of all remaining classifiers. We applied these methods in training and evaluation of classifiers.

Bibliography

Albert, C., Garnier, M., Rykner, A., and Saint-Dizier, P. (2009). Analyzing a corpus of documents produced by French writers in English: annotating lexical, grammatical and stylistic errors and their distribution. In Mahlberg, M., González-Díaz, V., and Smith, C., editors, *Proceedings of the Fifth Corpus Linguistics Conference (CL2009)*, University of Liverpool, UK. Article number 122.

Amaral, L., Meurers, D., and Ziai, R. (2011). Analyzing learner language: Towards a flexible NLP architecture for intelligent language tutors. *Computer Assisted Language Learning*, 24(1):1–16.

Amaral, L. A. and Meurers, D. (2009). Little things with big effects: On the identification and interpretation of tokens for error diagnosis in ICALL. *CALICO Journal (Special Issue of the 2008 CALICO Workshop on Automatic Analysis of Learner Language)*, 26(3):580–591.

Andersen, O. E. (2006). Grammatical error detection. Master's thesis, Girton College, University of Cambridge, Cambridge, UK.

Andersen, O. E. (2007). Grammatical error detection using corpora and supervised learning. In Nurmi, V. and Sustretov, D., editors, *Proceedings of the 12th ESSLLI Student Session (ESSLI-07)*, Dublin, Ireland. European Summer School for Logic, Language and Information.

Atwell, E. (1987). How to detect grammatical errors in a text without parsing it. In *Proceedings of the 3rd Conference of the European Chapter of the ACL (EACL'87)*, pages 38–45, Morristown, NJ. Association for Computational Linguistics.

Bacchiani, M., Riley, M., Roark, B., and Sproat, R. (2006). MAP adaptation of stochastic grammars. *Computer Speech and Language*, 20(1):41–68.

Baldwin, T., Kordoni, V., and Villavicencio, A. (2009). Prepositions in applications: A survey and introduction to the special issue. *Computational Linguistics - Special Issue on Prepositions*, 35(2):119–149.

Baroni, M. and Bernardini, S. (2004). BootCaT: Bootstrapping corpora and terms from the web. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-04)*, pages 1313–1316.

Barreno, M., Cárdenas, A. A., and Tygar, J. D. (2008). Optimal ROC curve for a combination of classifiers. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems (NIPS) 20*, pages 57–64. MIT Press, Cambridge, MA.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139.

Becker, M., Bredenkamp, A., Crysmann, B., and Klein, J. (1999). Annotation of error types for German news corpus. In *Proceedings of the ATALA Workshop on Treebanks*, Paris, France.

Bender, E. M., Flickinger, D., Oepen, S., Walsh, A., and Baldwin, T. (2004). Arboretum: Using a precision grammar for grammar checking in CALL. In Delmonte, R., Delcloque, P., and Tonelli, S., editors, *NLP and Speech Technologies in Advanced Language Learning Systems, Proceedings of InSTIL/ICALL2004 Symposium on Computer Assisted Language Learning*, pages 83–86, Padova, Italy. Unipress.

Berger, A., Della Pietra, S., and Della Pietra, V. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–72.

Bergsma, S., Lin, D., and Goebel, R. (2009). Web-scale n-gram models for lexical disambiguation. In *Proceedings for the 21st International Joint Conference on Artificial Intelligence*, pages 1507–1512.

Bestgen, Y., Lories, G., and Thewissen, J. (2010). Using latent semantic analysis to measure coherence in essays by foreign language learners? In Bolasco, S., Chiari, I., and Giuliano, L., editors, *Statistical Analysis of Textual Data: Proceedings of 10th International Conference Journées d'Analyse statistique des Données Textuelles (Jadt2010)*, Sapienza University of Rome.

Bigert, J. (2004). Probabilistic detection of context-sensitive spelling errors. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-04)*, volume 5, pages 1633–1636, Lisbon, Portugal.

Bigert, J. and Knutsson, O. (2002). Robust error detection: A hybrid approach combining unsupervised error detection and linguistic knowledge. In *Proceedings of the 2nd Workshop on Robust Methods in Analysis of Natural language Data (Romand'02)*, Frascati, Italy.

Bigert, J., Sjöbergh, J., Knutsson, O., and Sahlgren, M. (2005). Unsupervised evaluation of parser robustness. In *Proceedings of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICling-05)*, pages 142–154, Mexico City, Mexico.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press Inc., New York, USA.

Black, E., Abney, S., Flickinger, D., Gdaniec, C., Grishman, R., Harrison, P., Hindle, D., Ingria, R., Jelinek, F., Klavans, J., Liberman, M., Marcus, M., Roukos, S., Santorini, B., and Strzalkowski, T. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In Black, E., editor, *Proceedings of the HLT Workshop on Speech and Natural Language*, pages 306–311, Morristown, NJ, USA. Association for Computational Linguistics.

Bod, R. (1992a). A computational model of language performance: Data oriented parsing. In Boitet, C., editor, *Proceedings of the fifteenth International Conference on Computational Linguistics (COLING-92), Volume 3*, pages 855–859. GETA (IMAG) & Association Champollion / Association for Computational Linguistics.

Bod, R. (1992b). A computational model of language performance: Data oriented parsing (DOP). In *Computational Linguistics in the Netherlands - Papers from the second CLIN-meeting (1991)*, pages 26–39. Rijksuniversiteit Utrecht (RUU, now Universiteit Utrecht), Utrecht, The Netherlands.

Borin, L. (2002). What have you done for me lately? The fickle alignment of NLP and CALL. Technical Report 02-2002, Uppsala Learning Lab, Uppsala Universitet, Uppsala, Sweden. Presented at the EuroCALL 2002 Pre-conference Workshop on NLP and CALL in Jyväskylä, Finland.

Boyd, A. (2010). Eagle: an error-annotated corpus of beginning learner German. In Chair), N. C. C., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24:123–140.

Breiman, L. (1996b). Heuristics of instability and stabilization in model selection. *Annals of Statistics*, 24(6):2350–2383.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, USA.

Bresnan, J. (2001). *Lexical-Functional Syntax*. Blackwell, Oxford.

Brill, E. and Wu, J. (1998). Classifier combination for improved lexical disambiguation. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics (COLING-ACL'98), Volume 1*, pages 191–195, Montreal, Quebec, Canada. Association for Computational Linguistics.

Briscoe, T. and Carroll, J. (2002). Robust accurate statistical annotation of general text. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-02)*, pages 1499–1504, Las Palmas, Gran Canaria, Spain.

Briscoe, T., Carroll, J., and Watson, R. (2006). The second release of the RASP system. In *Proceedings of the Interactive Demo Session of the Joint Conference of the International Committee on Computational Linguistics and the Association for Computational Linguistics (COLING ACL 06)*, pages 77–80, Sydney, Australia. Association for Computational Linguistics.

Brockett, C., Dolan, W. B., and Gamon, M. (2006). Correcting ESL errors using phrasal SMT techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06): Vol. 1*, pages 249–256, Sydney, Australia. Association for Computational Linguistics.

Buchholz, S. and Green, D. (2006). Quality control of treebanks: Documenting, converting and patching. In *Proceedings of Workshop 12 on Quality assurance and quality measurement for language and speech resources (in conjunction with the 5th International Conference on Language Resources and Evaluation, LREC-2006 W12)*, pages 26–31, Genoa, Italy.

Burke, M. (2006). *Automatic Treebank Annotation for the Acquisition of LFG Resources*. PhD thesis, School of Computing, Dublin City University, Ireland.

Burnard, L. (2000). User reference guide for the British National Corpus. Technical report, Oxford University Computing Services.

Butt, M., Dyvik, H., King, T. H., Masuichi, H., and Rohrer, C. (2002). The parallel grammar project. In *Proceedings of COLING-2002 Workshop on Grammar Engineering and Evaluation*, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics.

Butt, M., King, T. H., Niño, M.-E., and Segond, F. (1999). *A grammar writers cookbook*. CSLI lecture notes 95, Stanford, CA.

Cahill, A. (2004). *Parsing with Automatically Acquired, Wide-Coverage, Robust, Probabilistic LFG Approximations*. PhD thesis, Dublin City University, Dublin, Ireland.

Cahill, A., Burke, M., O'Donovan, R., van Genabith, J., and Way, A. (2004). Long-distance dependency resolution in automatically acquired wide-coverage PCFG-based LFG approximations. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, pages 320–327, Barcelona, Spain.

Cahill, A., McCarthy, M., van Genabith, J., and Way, A. (2002). Parsing with PCFGs and automatic f-structure annotation. In Butt, M. and King, T. H., editors, *Proceedings of the Seventh International Conference on LFG*, pages 76–95, Stanford, CA. CSLI Publications.

Cahill, A. and van Genabith, J. (2006). Robust PCFG-based generation using automatically acquired LFG approximations. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06)*, pages 1033–1040, Sydney, Australia. Association for Computational Linguistics.

Carroll, G. and Rooth, M. (1998). Valence induction with a head-lexicalised PCFG. In *Proceedings of the 3rd conference on empirical methods in natural language processing (EMNLP)*, Granada, Spain.

Carter, S. and Monz, C. (2009). Parsing statistical machine translation output. In *Proceedings of the 4th Language & Technology Conference (LTC): Human Language Technologies as a Challenge for Computer Science and Linguistics*, Poznań, Poland.

Carter, S. and Monz, C. (2010). Discriminative syntactic reranking for statistical machine translation. In *Proceedings of the Ninth Conference of the Association for Machine Translation in the Americas (AMTA 2010)*, Denver, Colorado.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58.

Charniak, E. (1996). Tree-bank grammars. Technical Report CS-96-02, Department of Computer Science, Brown University. `ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-02.ps.Z`.

Charniak, E. (2000). A maximum entropy inspired parser. In *Proceedings of the First Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-00)*, pages 132–139, Seattle, WA.

Charniak, E. and Johnson, M. (2005). Course-to-fine n-best-parsing and maxent discriminative reranking. In *Proceedings of the 43rd Annual Meeting of the ACL (ACL-05)*, pages 173–180, Ann Arbor, Michigan. Association for Computational Linguistics.

Chodorow, M. and Leacock, C. (2000). An unsupervised method for detecting grammatical errors. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics (NAACL-00)*, volume 4, pages 140–147, San Francisco, CA, USA. Association for Computing Machinery (ACM) / Morgan Kaufmann Publishers Inc.

Chodorow, M., Tetreault, J. R., and Han, N.-R. (2007). Detection of grammatical errors involving prepositions. In *Proceedings of the 4th ACL-SIGSEM Workshop on Prepositions*, pages 25–30, Prague, Czech Republic.

Chomsky, N. (1957). *Syntactic Structures*. Mouton, The Hague.

Chrupała, G. and van Genabith, J. (2007). Using very large corpora to detect raising and control verbs. In Butt, M. and King, T. H., editors, *Proceedings of the Twelfth International Lexical Functional Grammar Conference (LFG07)*, Stanford, CA 94305-4115, USA. CSLI Publications, Stanford University.

Clarkson, P. R. and Rosenfeld, R. (1997). Statistical language modeling using the CMU-Cambridge toolkit. In Kokkinakis, G., Fakotakis, N., and Dermatas, E., editors, *Proceedings of the Fifth ESCA Conference on Speech Communication and Technology (EuroSpeech'97)*, pages 2707–2710, Rhodes, Greece. European Speech Communication Association (ESCA, now ISCA).

Collins, M. (1999). *Head-driven statistical models for natural language parsing*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA.

Collins, M. (2000). Discriminative reranking for natural language processing. In *Proceedings of the 17th International Conference on Machine Learning*, pages 175–182. Morgan Kaufmann, San Francisco.

Collins, M. and Koo, T. (2005). Discriminative reranking for natural language parsing. *Computational Linguistics*, 31(1):25–69.

Copestake, A. and Flickinger, D. (2000). An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the 3rd International Conference on Language Resources and Evaluation (LREC-02)*, Athens, Greece. Downloaded from http://www.cl.cam.ac.uk/∼aac10/papers/lrec2000.pdf, 11th May 2005.

Corston-Oliver, S., Gamon, M., and Brockett, C. (2001). A machine learning approach to the automatic evaluation of machine translation. In *Proceedings of 39th Annual Meeting of the Association for Computational Linguistics*, pages 148–155, Toulouse, France. Association for Computational Linguistics.

Crocker, M. W. and Keller, F. (2006). Probabilistic grammars as models of gradience in language processing. In Gisbert Fanselow, Caroline Féry, R. V. and Schlesewsky, M., editors, *Gradience in Grammar: Generative Perspectives*, pages 227–245. Oxford University Press Inc., New York, USA.

Daelemans, W., Van, A., Bosch, D., and Zavrel, J. (1999). Forgetting exceptions is harmful in language learning. *Machine Learning*, 34:11–41.

Daelemans, W., Van Den Bosch, A., and Weijters, T. (1997). IGTree: Using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, 11:407–423. 10.1023/A:1006506017891.

Dahlmeier, D. and Ng, H. T. (2011a). Correcting semantic collocation errors with L1-induced paraphrases. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP-2011)*. Association for Computational Linguistics.

Dahlmeier, D. and Ng, H. T. (2011b). Grammatical error correction with alternating structure optimization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 915–923, Portland, Oregon, USA. Association for Computational Linguistics.

Dale, R. (2004). Industry watch (column). *Natural Language Engineering*, 10(1):91–94.

Dalrymple, M. (2001). *Lexical-Functional Grammar*. San Diego, CA; London. Academic Press.

Davis, J. and Goadrich, M. (2006). The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning (ICML'06)*, pages 233–240, New York, NY, USA. ACM.

De Felice, R. and Pulman, S. G. (2007). Automatically acquiring models of preposition use. In *Proceedings of the 4th ACL-SIGSEM Workshop on Prepositions*, pages 45–50, Prague, Czech Republic.

De Felice, R. and Pulman, S. G. (2008). A classifier-based approach to preposition and determiner error correction in L2 English. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING-08)*, pages 169–176, Manchester, United Kingdom.

Deoras, A., Jelinek, F., and Su, Y. (2010). Language model adaptation using random forests. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Texas, USA.

Díaz-Negrillo, A., Meurers, D., Valera, S., and Wunsch, H. (2010). Towards interlanguage POS annotation for effective learner corpora in SLA and FLT. *Language Forum*, 36(1–2). Special Issue on New Trends in Language Teaching.

Dickinson, M. (2010). Generating learner-like morphological errors in russian. In *Proceedings of the 23rd International Conference on Computational Linguistics (COLING 2010)*.

Dickinson, M., Israel, R., and Lee, S.-H. (2010). Building a Korean web corpus for analyzing learner language. In *Proceedings of the 6th Workshop on the Web as Corpus (WAC-6)*, Los Angeles.

Doherty, S. and O'Brien, S. (2009). Can MT output be evaluated through eye tracking? In *MT Summit XII: proceedings of the twelfth Machine Translation Summit*, pages 214–221, Ottawa, Ontario, Canada.

Doherty, S., O'Brien, S., and Carl, M. (2010). Eye tracking as an MT evaluation technique. *Machine Translation*, 24(1):1–13. 10.1007/s10590-010-9070-9.

Douglas, S. and Dale, R. (1992). Towards robust PATR. In Boitet, C., editor, *Proceedings of the fifteenth International Conference on Computational Linguistics (COLING-92), Volume 2*, pages 468–474. GETA (IMAG) & Association Champollion / Association for Computational Linguistics.

Duwairi, R. M. (2006). A framework for the computerized assessment of university student essays. *Computers in Human Behavior*, 22:381–388.

Elghafari, A., Meurers, D., and Wunsch, H. (2010). Exploring the data-driven prediction of prepositions in English. In Huang, C.-R. and Jurafsky, D., editors, *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010) - Posters Volume*, pages 267–275, Beijing, China. Chinese Information Processing Society of China.

Elmi, M. A. and Evens, M. W. (1998). Spelling correction using context. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th international conference on Computational linguistics (COLING-ACL'98), Volume 1*, pages 360–364. Association for Computational Linguistics.

Erpenbeck, A., Koch, B., Kummer, N., Reuter, P., Tschorn, P., and Wagner, J. (2002). KOKS - Korpusbasierte Kollokationssuche. Technical report, Universität Osnabrück, Germany.

Evert, S. (2005). *The Statistics of Word Cooccurrences: Word Pairs and Collocations*. PhD thesis, Universität Stuttgart.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874. Special issue on ROC Analysis in Pattern Recognition.

Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, USA.

Filimonov, D. and Harper, M. (2011a). Generalized interpolation in decision tree LM. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 620–624, Portland, Oregon, USA. Association for Computational Linguistics.

Filimonov, D. and Harper, M. (2011b). Syntactic decision tree LMs: Random selection or intelligent design? In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP-2011)*. Association for Computational Linguistics.

Flach, P. A. (2010). ROC analysis. In Sammut, C. and Webb, G. I., editors, *Encyclopedia of Machine Learning*, pages 869–875. Springer US. 10.1007/978-0-387-30164-8_733.

Fossum, V. and Knight, K. (2009). Combining constituent parsers. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'09), Companion Volume: Short Papers*, pages 253–256, Morristown, NJ, USA. Association for Computational Linguistics.

Foster, J. (2004). Parsing ungrammatical input: An evaluation procedure. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-04)*, pages 2039–2042, Lisbon, Portugal.

Foster, J. (2005). *Good Reasons for Noting Bad Grammar: Empirical Investigations into the Parsing of Ungrammatical Written English*. PhD thesis, University of Dublin, Trinity College, Dublin, Ireland.

Foster, J. (2007a). Treebanks gone bad: Generating a treebank of ungrammatical English. In *Proceedings of the Workshop on Analytics for Noisy Unstructured Data (AND-07)*

at the 20th International Joint Conference on Artificial Intelligence (IJCAI-07), pages 39–47, Hyderabad, India.

Foster, J. (2007b). Treebanks gone bad: Parser evaluation and retraining using a treebank of ungrammatical sentences. *International Journal on Document Analysis and Recognition*, 10(3-4):129–145.

Foster, J. and Andersen, O. E. (2009). GenERRate: generating errors for use in grammatical error detection. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications (EdAppsNLP'09)*, pages 82–90, Morristown, NJ, USA. Association for Computational Linguistics.

Foster, J., Çetinoğlu, Ö., Wagner, J., Le Roux, J., Nivre, J., Hogan, D., and Hogan, S. (2011a). #hardtoparse: POS tagging and parsing the Twitterverse. In *Analyzing Microtext: Papers from the 2011 AAAI Workshop (WS-11-05)*, pages 20–25, Menlo Park, California, USA. The AAAI Press.

Foster, J., Çetinoğlu, Ö., Wagner, J., Le Roux, J., Nivre, J., Hogan, D., and van Genabith, J. (2011b). From news to comment: Resources and benchmarks for parsing the language of web 2.0. In *Proceedings of 5th International Joint Conference on Natural Language Processing (IJCNLP2011)*, pages 893–901, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Foster, J., Çetinoğlu, Ö., Wagner, J., and van Genabith, J. (2011c). Comparing the use of edited and unedited text in parser self-training. In *Proceedings of the 12th International Conference on Parsing Technologies (IWPT 2011, Dublin, Ireland)*, pages 215–219. Association for Computational Linguistics.

Foster, J. and Vogel, C. (2004a). Good reasons for noting bad grammar: Constructing a corpus of ungrammatical language. In Kepser, S. and Reis, M., editors, *Pre-Proceedings of the International Conference on Linguistic Evidence: Empirical, Theoretical and Computational Perspectives*, pages 151–152, Tübingen, Germany.

Foster, J. and Vogel, C. (2004b). Parsing ill-formed text using an error grammar. *Artificial Intelligence Review: Special AICS2003 Issue*, 21(3-4):269–291.

Foster, J., Wagner, J., Seddah, D., and van Genabith, J. (2007). Adapting WSJ-trained parsers to the British National Corpus using in-domain self-training. In *Proceedings of the Tenth International Conference on Parsing Technologies (IWPT'07)*, pages 33–35, Prague, Czech Republic. Association for Computational Linguistics.

Foster, J., Wagner, J., and van Genabith, J. (2008). Adapting a WSJ-trained parser to grammatically noisy text. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, Stroudsburg, PA, USA. Association for Computational Linguistics.

Frank, A., King, T. H., Kuhn, J., and Maxwell, J. (1998). Optimality theory style constraint ranking in large-scale LFG grammars. In *Proceedings of the 3rd Lexical Functional Grammar (LFG) Conference (LFG-98)*, Brisbane, Australia.

Gaizauskas, R. (1995). Investigations into the grammar underlying the Penn Treebank II. Technical Report CS-95-25, Department of Computer Science, Univeristy of Sheffield, UK.

Gamon, M. (2010). Using mostly native data to correct errors in learners' writing: A meta-classifier approach. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), Proceedings of the Main Conference*, pages 163–171, Stroudsburg, PA, USA. Association for Computational Linguistics.

Gamon, M. (2011). High-order sequence modeling for language learner error detection. In *Proceedings of the Sixth Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, pages 180–189, Portland, Oregon. Association for Computational Linguistics.

Gamon, M., Gao, J., Brockett, C., Klementiev, A., Dolan, W. B., Belenko, D., and Vanderwende, L. (2008). Using contextual speller techniques and language modelling for ESL error correction. In *Proceedings of the International Joint Conference on Natural Language Processing*, Hyderabad, India.

Gamon, M. and Leacock, C. (2010). Search right and thou shalt find... using web queries for learner error detection. In *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, pages 37–44, Stroudsburg, PA, USA. Association for Computational Linguistics.

Garside, R., Leech, G., and Sampson, G., editors (1987). *The Computational Analysis of English: a Corpus-Based Approach*. Longman, London.

Gisbert Fanselow, Caroline Féry, R. V. and Schlesewsky, M. (2006). *Gradience in Grammar: Generative Perspectives*. Oxford University Press Inc., New York, USA.

Gojenola, K. and Oronoz, M. (2000). Corpus-based syntactic error detection using syntactic patterns. In *Proceedings of the ANLP-NAACL 2000 Student Research Workshop*, pages 24–29, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Golding, A. R. (1995). A Bayesian hybrid method for context-sensitive spelling correction. In *Proceedings of the Third Workshop on Very Large Corpora*, pages 39–53, Boston, MA.

Golding, A. R. and Schabes, Y. (1996). Combining trigram-based and feature-based methods for context-sensitive spelling correction. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, Santa Cruz, CA. Morgan Kaufmann Publishers.

Goodman, J. (1996). Parsing algorithms and metrics. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, pages 177–183. Morgan Kaufmann Publishers.

Goodrich, M. T. and Tamassia, R. (1998). *Data Structures and Algorithms in Java*. John Wiley and Sons, Inc., New York, NY, USA, 2nd edition.

Granger, S. (1993). International corpus of learner English. In Aarts, J., de Haan, P., and Oostdijk, N., editors, *English Language Corpora: Design, Analysis and Exploitation*, pages 57–71. Rodopi, Amsterdam.

Hale, J. (2003). The information conveyed by words in sentences. *Journal of Psycholinguistic Research*, 32(2):101–123.

Hall, K. B. (2005). *Best-first Word-lattice Parsing: Techniques for integrated syntactic language modeling*. PhD thesis, Brown University, Providence, Rhode Island, USA.

Han, N., Tetreault, J., Lee, S., and Ha, J. (2010). Using an error-annotated learner corpus to develop and ESL/EFL error correction system. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Han, N.-R., Chodorow, M., and Leacock, C. (2004). Detecting errors in English article usage with a maximum entropy classifier trained on a large, diverse corpus. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-04)*, Lisbon, Portugal.

Han, N.-R., Chodorow, M., and Leacock, C. (2006). Detecting errors in English article usage by non-native speakers. *Natural Language Engineering*, 12(2):115–129.

Hashemi, S. S. (2003). *Automatic Detection of Grammar Errors in Primary School Children's Texts — A Finite State Approach*. PhD thesis, Göteborg University, Göteborg, Sweden.

Hashemi, S. S. (2007). Ambiguity resolution by reordering rules in text containing errors. In *Proceedings of the 10th International Conference on Parsing Technologies (IWPT 2007)*, pages 69–79, Stroudsburg, PA, USA. Association for Computational Linguistics.

Heath, D., Kasif, S., and Salzberg, S. (1993). k-DT: A multi-tree learning method. In Michalski, R. S. and Tecuci, G., editors, *Proceedings of the Second International Workshop on Multistrategy Learning (MSL-93)*, pages 138–149, Fairfax, VA, USA. George Mason University.

Heilman, M., Zhao, L., Pino, J., and Eskenazi, M. (2008). Retrieval of reading materials for vocabulary and reading practice. In *Proceedings of the Third ACL Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2008)*, pages 80–88, Columbus, Ohio, USA. Association for Computational Linguistics.

Henderson, J. C. and Brill, E. (1999). Exploiting diversity in natural language processing: Combining parsers. In Fung, P. and Zhou, J., editors, *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 187–194, New Brunswick, NJ, USA. Association for Computational Linguistics.

Hermet, M. and Désilets, A. (2009). Using first and second language models to correct preposition errors in second language authoring. In *Proceedings of the Fourth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 64–72. Association for Computational Linguistics.

Hermet, M., Désilets, A., and Szpakowicz, S. (2008). Using the web as a linguistic resource to automatically correct lexico-syntactic errors. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odjik, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA). http://www.lrec-conf.org/proceedings/lrec2008/.

Hogan, D., Foster, J., Wagner, J., and van Genabith, J. (2008). Parser-based retraining for domain adaptation of probabilistic generators. In *Proceedings of the Fifth International Natural Language Generation Conference (INLG 2008)*, pages 165–168, Salt Fork, Ohio, USA. Association for Computational Linguistics.

Holst, A., Ekman, J., and Gillblad, D. (2004). Deviation detection of industrial processes. *ERCIM News (European Research Consortium for Informatics and Mathematics): Special on Analysis, Diagnosis, Planning and Simulation of Industrial Systems*, 56:13–14.

Horváth, J. (1999). *Advanced Writing in English as a Foreign Language: A Corpus-based Study of Processes and Products*. PhD thesis, Janus Pannonius University, Pécs, Hungary.

Izumi, E., Uchimoto, K., and Isahara, H. (2004). The overview of the SST speech corpus of Japanese learner English and evaluation through the experiment on automatic detection

of learners' errors. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC-04)*, volume 4, pages 1435–1439, Lisbon, Portugal.

Jaakkola, T., Meila, M., and Jebara, T. (2000). Maximum entropy discrimination. In *Advances in Neural Information Processing Systems 12 (Proceedings of NIPS 1999)*, pages 470–476, Cambridge, MA, USA. The MIT Press.

James, C. (1998). *Errors in Language Learning and Use: Exploring Error Analysis*. Addison Wesley Longman.

Jebara, T. (2004). *Machine Learning: Discriminative and Generative*. Kluwer Academic Publishers, Norwell, MA, USA and Dordrecht, The Netherlands.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'02)*, pages 133–142, New York, NY, USA. Association for Computing Machinery (ACM).

Johnson, M. (1998). PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632.

Johnson, M., Geman, S., Canon, S., Chi, Z., and Riezler, S. (1999). Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL-99)*, pages 535–541, San Francisco, California.

Jones, G. J., Burke, M., Judge, J., Khasin, A., Lam-Adesina, A., and Wagner, J. (2005). Dublin City University at CLEF 2004: Experiments in monolingual, bilingual and multilingual retrieval. In Peters, C., Clough, P., Gonzalo, J., Jones, G., Kluck, M., and Magnini, B., editors, *Multilingual Information Access for Text, Speech and Images: 5th Workshop of the Cross-Language Evaluation Forum (CLEF 2004)*, volume 3491 of *Lecture Notes in Computer Science*, pages 207–220. Springer, Heidelberg.

Kaplan, R. and Bresnan, J. (1982). Lexical functional grammar, a formal system for grammatical representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, MA.

Kaplan, R., Riezler, S., King, T. H., Maxwell III, J. T., Vasserman, A., and Crouch, R. (2004). Speed and accuracy in shallow and deep stochastic parsing. In Dumais, S., Marcu, D., and Roukos, S., editors, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2004)*, pages 97–104, Boston, Massachusetts, USA. Association for Computational Linguistics.

Kato, T. (1994). Yet another chart-based technique for parsing ill-formed input. In *Proceedings of the Fourth Conference on Applied Natural Language Processing*, pages 107–112, Morristown, NJ, USA / San Fransisco, CA, USA. Association for Computational Linguistics / Morgan Kaufmann.

Khader, R. A., King, T. H., and Butt, M. (2004). Deep CALL grammars: The LFG-OT experiment. http://ling.uni-konstanz.de/pages/home/butt/dgfs04call.pdf. Presentation slides presented at the DGfS Workshop Computerlinguistik und elektronisches Lernen.

Kilgarriff, A. (2007). Googleology is bad science. *Computational Linguistics*, 33(1):147–151.

Kiss, T., Keßelmeier, K., Müller, A., Roch, C., Stadtfeld, T., and Strunk, J. (2010). A logistic regression model of determiner omission in PPs. In *Coling 2010, 23rd International Conference on Computational Linguistics, Posters Volume*, pages 561–569, Haidian District, Beijing, China. Chinese Information Processing Society of China.

Kiss, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 423–430, Sapporo, Japan. Association for Computational Linguistics.

Klenner, M. and Visser, H. (2003). What exactly is wrong and why? tutorial dialogue for intelligent CALL systems. *Linguistik online*, 17:81–98.

Knight, K. and Chander, I. (1994). Automated postediting of documents. In *Proceedings*

of the Twelfth National Conference on Artificial Intelligence (AAAI'94), pages 779–784, Menlo Park, California, USA. The AAAI Press.

Koontz-Garboden, A. and Jaeger, T. F. (2003). An empirical investigation of the frequency-grammaticality correlation hypothesis. Essay prepared for LIN 200 Foundations of Linguistics, received or downloaded on 2006-03-13, source unknown.

Krotov, A., Hepple, M., Gaizauskas, R., and Wilks, Y. (1999). Evaluating two methods for treebank grammar compaction. *Natural Language Engineering*, 5(4):377–394.

Kukich, K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439.

Lackowski, P. (1963). Words as grammatical primes. *Language*, 39(2):211–215.

Lavergne, T., Urvoy, T., and Yvon, F. (2011). Filtering artificial texts with statistical machine learning techniques. *Language Resources and Evaluation*, 45:25–43.

Leacock, C., Chodorow, M., Gamon, M., and Tetreaul, J. (2010). *Automated Grammatical Error Detection for Language Learners*. Morgan & Claypool Publishers, San Rafael, CA, USA.

Lee, J. and Seneff, S. (2006). Automatic grammar correction for second-language learners. In *Interspeech 2006 - ICSLP, Ninth International Conference on Spoken Language Processing*, pages 1978–1981, Pittsburgh, PA, USA. International Speech Communication Association (ISCA).

Lee, J. and Seneff, S. (2008). Correcting misuse of verb forms. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-08: HLT)*, pages 174–182, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lee, J., Zhou, M., and Liu, X. (2007). Detection of non-native sentences using machine-translated training data. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics*

(NAACL'07); Companion Volume, Short Papers, pages 93–96, Morristown, NJ, USA. Association for Computational Linguistics.

Lee, J. S. Y. (2009). *Automatic Correction of Grammatical Errors in Non-native English Text*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA.

Lin, C.-Y. and Hovy, E. (2003). Automatic evaluation of summaries using n-gram co-occurrence statistics. In Hearst, M. and Ostendorf, M., editors, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL 2003)*, pages 71–78. Association for Computational Linguistics.

Lonsdale, D. and Strong-Krause, D. (2003). Automated rating of ESL essays. In *Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing (BEA 2003) - Volume 2*, pages 61–67, Morristown, NJ, USA. Association for Computational Linguistics.

Malmsten, M. and Klasen, S. (2005). Grammar checker. In Nugues, P. and Johansson, R., editors, *Språkbehandling och datalingvistik: Projektarbeten 2004*, pages 55–60. Lunds Universitet, Institutionen för Datavetenskap, Lund, Sweden. Project report of the course "Language processing and computational linguistics" of 2004/2005.

Manning, C. D. and Schütze, H. (1999). *Foundations of statistical natural language processing*. MIT Press, Cambridge, MA, London.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The Penn Treebank: Annotating predicate argument structure. In Weinstein, C. J., editor, *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey (the 1994 ARPA HLT Workshop)*, pages 114–119, Princeton, New Jersey. Morgan Kaufmann.

Màrquez, L., Padró, L., and Rodríguez, H. (1998). Improving tagging accuracy by using voting taggers. In *Proceedings of International Conference on Natural Language Processing and Industrial Applications (NLP+IA 98) / Conference internationale sur*

le traitement automatique des langues et ses applications industrielles (TAL+AI 98), Moncton, New Brunswick, Canada.

Màrquez, L., Rodríguez, H., Carmona, J., and Montolio, J. (1999). Improving POS tagging using machine-learning techniques. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 53–62, New Brunswick, NJ, USA. Association for Computationl Linguistics.

Marsland, S. (2009). *Machine learning: an algorithmic perspective*. Chapman & Hall/CRC, Taylor & Francis Group, LLC, Boca Raton, FL, USA.

Maxwell, J. and Kaplan, R. (1996). Unification-based parsers that automatically take adventage of context freeness. In Butt, M. and King, T. H., editors, *Proceedings of the First International Conference on Lexical Functional Grammar (LFG-96)*, Grenoble, France. Presentation title *An Efficient Parser for LFG*.

Mays, E., Damerau, F. J., and Mercer, R. L. (1991). Context based spelling correction. *Information Processing and Management*, 23(5):517–522.

McCarthy, D. (2009). Word sense disambiguation: An overview. *Language and Linguistics Compass*, 3(2):537–558.

McClosky, D., Charniak, E., and Johnson, M. (2006). Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06)*, pages 337–344, Sydney, Australia. Association for Computational Linguistics.

Mellish, C. S. (1989). Some chart-based techniques for parsing ill-formed input. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL-89)*, pages 102–109, Morristown, NJ, USA. Association for Computational Linguistics.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1990). Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, pages 235–244. Special Issue on WordNet.

Minnen, G., Bond, F., and Copestake, A. (2000). Memory-based learning for article generation. In Cardie, C., Daelemans, W., Nédellec, C., and Tjong Kim Sang, E., editors, *Proceedings of the 4th conference on Computational natural language learning (CoNLL'00) and the 2nd workshop on Learning language in logic (LLL'00) - Volume 7*, pages 43–48, New Brunswick, NJ, USA and Stroudsburg, PA, USA. Association for Computational Linguistics.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Singapore, Boston or New York.

Mitton, R., Hardcastle, D., and Pedler, J. (2007). BNC! handle with care! — spelling and tagging errors in the BNC. In *Proceedings of the Fourth Corpus Linguistics Conference (CorpLing'07)*, Birmingham, UK.

Mount, D. M. (2005). ANN programming manual. Technical report, University of Maryland.

Murthy, K. V. S. (1996). *On growing better decision trees from data*. PhD thesis, The Johns Hopkins University, Baltimore, MD, USA.

Mutton, A., Dras, M., Wan, S., and Dale, R. (2007). GLEU: Automatic evaluation of sentence-level fluency. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 344–351, Prague, Czech Republic. Association for Computational Linguistics.

Nagata, R., Kakegawa, J., and Kutsuwa, T. (2010). Detecting missing sentence boundaries in learner English. In Bolasco, S., Chiari, I., and Giuliano, L., editors, *Statistical Analysis of Textual Data: Proceedings of 10th International Conference (Journées d'Analyse statistique des Données Textuelles, JADT 2010)*, Sapienza University of Rome.

Nagata, R., Morihiro, K., Kawai, A., and Isu, N. (2006). Reinforcing English countability prediction with one countability per discourse property. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06): Main Conference Poster Sessions*, pages 595–602, Sydney, Australia. Association for Computational Linguistics.

Nerbonne, J. (2002). Computer-assisted language learning and natural language processing. In Mitkov, R., editor, *Handbook of Computational Linguistics*, pages 670–698. Oxford University Press.

Nesselhauf, N. (2005). *Collocations in a Learner Corpus*. John Benjamins, Amsterdam, The Netherlands.

Nicholls, D. (1999). The Cambridge learner corpus - error coding and analysis. In *Summer Workshop on Learner Corpora*, Tokyo, Japan.

Och, F. J., Gildea, D., Khudanpur, S., Sarkar, A., Yamada, K., Fraser, A., Kumar, S., Shen, L., Smith, D., Eng, K., Jain, V., Jin, Z., and Radev, D. (2004). A smorgasbord of features for statistical machine translation. In Susan Dumais, D. M. and Roukos, S., editors, *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 161–168, Boston, Massachusetts, USA. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL02)*, pages 295–302, Morristown, NJ, USA. Association for Computational Linguistics.

O'Donovan, R., Burke, M., Cahill, A., van Genabith, J., and Way, A. (2004). Large-scale induction and evaluation of lexical resources from the Penn-II Treebank. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, pages 368–375, Barcelona, Spain.

Okanohara, D. and Tsujii, J. (2007). A discriminative language model with pseudo-negative samples. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 73–80, Prague, Czech Republic. Association for Computational Linguistics.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on*

Association for Computational Linguistics (ACL02), pages 311–318, Morristown, NJ, USA. Association for Computational Linguistics.

Pecina, P. and Schlesinger, P. (2006). Combining association measures for collocation extraction. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06): Main Conference Poster Sessions*, pages 651–658, Sydney, Australia. Association for Computational Linguistics.

Pedler, J. (2007). *Computer Correction of Real-word Spelling Errors in Dyslexic Text*. PhD thesis, Birkbeck College, University of London, London, UK.

PELCRA (2004). Pelcra: Polish and English language corpora for research and applications. `http://pelcra.ia.uni.lodz.pl/`. Corpus sample accessed 9th November 2004.

Petrov, S. (2010). Products of random latent variable grammars. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), Proceedings of the Main Conference*, pages 19–27. Association for Computational Linguistics, Stroudsburg, PA, USA.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING ACL 06)*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Pinchak, C., Lin, D., and Rafiei, D. (2009). Flexible answer typing with discriminative preference ranking. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 666–674. Association for Computational Linguistics.

Pollard, C. and Sag, I. A. (1994). *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications.

Post, M. (2011). Judging grammaticality with tree substitution grammar derivations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 217–222, Portland, Oregon, USA. Association for Computational Linguistics.

Post, M. and Gildea, D. (2008). Parsers as language models for statistical machine translation. In *The Eighth Conference of the Association for Machine Translation in the Americas (AMTA 2008)*.

Prince, A. and Smolensky, P. (1993). Optimality theory: Constraint interaction in generative grammar. Technical report, Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts, USA. Rutgers Optimality Archive, `http://roa.rutgers.edu/files/537-0802/537-0802-PRINCE-0-0.PDF`, received 2009-09-26; also available as a book through Wiley-Blackwell (2004).

Provost, F. and Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, USA.

Ratnaparkhi, A. (1997). A simple introduction to maximum entropy models for natural language processing. Technical Report IRCS-97-08, University of Pennsylvania, Institute for Research in Cognitive Science, Philadelphia, PA, USA.

Rethmeier, N. (2011). Using language models to detect errors in second-language learner writing. Bachelor thesis, Bauhaus-Universiät Weimar, Fakultät Medien, Mediensysteme, Weimar, Germany.

Reuer, V. (2003). *PromisD - Ein Analyseverfahren zur antizipationsfreien Erkennung und Erklärung von grammatischen Fehlern in Sprachlehrsystemen*. PhD thesis, Humboldt-Universität zu Berlin, Berlin, Germany.

Rieger, W. (1995). *SGML für die Praxis. Ansatz und Einsatz von ISO 8879*. Springer, Berlin, Germany.

Riezler, S., King, T. H., Kaplan, R. M., Crouch, R., Maxwell III, J. T., and Johnson, M. (2002). Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL'02)*, pages 271–278, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Ringlstetter, C., Mihov, S., and Schulz, K. U. (2006). Orthographic errors in web pages: Toward cleaner web corpora. *Computational Linguistics*, 32(3):295–340.

Roark, B., Saraclar, M., and Collins, M. (2007). Discriminative n-gram language modeling. *Computer Speech and Language*, 21(2):373–392.

Rokach, L. (2009). Taxonomy for characterizing ensemble methods in classification tasks: a review and annotated bibliography. *Computational Statistics and Data Analysis*, 53(12):4046–4072.

Rokach, L. and Maimon, O. (2007). *Data Mining with Decision Trees: Theory and Applications*, volume 61 of *Series in Machine Perception and Artificial Intelligence*. World Scientific Publishing.

Rosén, V. and de Smedt, K. (2007). Theoretically motivated treebank coverage. In Nivre, J., Kaalep, H.-J., Muischnek, K., and Koit, M., editors, *Proceedings of the 16th Nordic Conference of Computational Linguistics (NODALIDA-2007)*, pages 152–159.

Rosenfeld, R. (1996). A maximum entropy approach to adaptive statistical language modeling. *Computer Speech and Language*, 10(3):187–228.

Rozovskaya, A. and Roth, D. (2010a). Annotating ESL errors: Challenges and rewards. In *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications (BEA)*, pages 28–36, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rozovskaya, A. and Roth, D. (2010b). Generating confusion sets for context-sensitive error correction. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 961–970, MIT, Massachusetts, USA. Association for Computational Linguistics.

Rozovskaya, A. and Roth, D. (2010c). Training paradigms for correcting errors in grammar and usage. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL HLT 2010), Proceedings of the Main Conference*, pages 154–162, Stroudsburg, PA, USA. Association for Computational Linguistics.

Sagae, K. and Lavie, A. (2006). Parser combination by reparsing. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of [sic!] Computational Linguistics, Companion Volume: Short Papers*, pages 129–132, New York City, USA. Association for Computational Linguistics.

Sampson, G. and Babarczy, A. (2003). A test of the leaf-ancestor metric for parse accuracy. *Natural Language Engineering*, 9(4):365–380.

Sampson, G. R. (2007). Grammar without grammaticality. *Corpus Linguistics and Linguistic Theory*, 3(1):1–32.

Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *Proceedings of International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK.

Schmid, H. (2000). LoPar — design and implementation. Technical report, IMS, Universität Stuttgart, Germany.

Schmid, H. (2004). Efficient parsing of highly ambiguous context-free grammars with bit vectors. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING 2004)*, pages 162–168, Geneva, Switzerland. COLING.

Schmid, H. (2010). Decision trees. In Clark, A., Fox, C., and Lappin, S., editors, *The Handbook of Computational Linguistics and Natural Language Processing*, pages 180–196. Wiley Blackwell, Chichester, West Sussex, UK.

Schneider, D. and McCoy, K. F. (1998). Recognizing syntactic errors in the writing of second language learners. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and the 17th international conference on Computational*

linguistics (COLING-ACL'98), Volume 2, pages 1198–1204, Morristown, NJ, USA. Association for Computational Linguistics.

Selinker, L. (1972). Interlanguage. *International Review of Applied Linguistics*, 10(3):209–231.

Sjöbergh, J. (2006). Chunking: an unsupervised method to find errors in text. In Werner, S., editor, *Proceedings of the 15th NODALIDA conference (Nodalida-05)*, pages 180–185, Joensuu, Finland. University of Joensuu electronic publications in linguistics and language technology.

Smith, N. A. and Eisner, J. (2005a). Contrastive Estimation: Training log-linear models on unlabeled data. In *Proceedings of the 43rd Annual Meeting of the Association of Computational Linguistics*, pages 354–362, Ann Arbor.

Smith, N. A. and Eisner, J. (2005b). Guiding unsupervised grammar induction using contrastive estimation. In *Proceedings of the IJCAI Workshop on Grammatical Inference Applications*, pages 73–82, Edinburgh.

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., and Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas (AMTA)*, pages 223–231.

Snow, C. and Meijer, G. (1976). On the secondary nature of syntactic intuitions. In Greenbaum, S., editor, *Acceptability in Language*, pages 163–177. Mouton Publishers, The Hague.

Srinivasan, A. (1999). Note on the location of optimal classifiers in ROC space. Technical Report PRG-TR-2-99, Oxford University.

Stehouwer, H. and van den Bosch, A. (2009). Putting the t where it belongs: Solving a confusion problem in dutch. In Verberne, S., van Halteren, H., and Coppen, P.-A., editors, *Computational Linguistics in the Netherlands 2007: Selected Papers from the 18th CLIN Meeting*, pages 21–36, Groningen, The Netherlands.

Stemberger, J. P. (1982). Syntactic errors in speech. *Journal of Psycholinguistic Research*, 11(4):313–345.

Stolcke, A. (2002). SRILM — an extensible language modeling toolkit. In Hansen, J. H. L. and Pellom, B., editors, *Proceedings of the 7th International Conference on Spoken Language Processing (ICSLP2002)*, volume 2, pages 901–904, Baixas, France. International Speech Communication Association (ISCA).

Stolcke, A., Bratt, H., Butzberger, J., Franco, H., Rao Gadde, V. R., Plauché, M., Richey, C., Shriberg, E., Sönmez, K., Weng, F., and Zheng, J. (2000). The SRI March 2000 hub-5 conversational speech transcription system. In *Proceedings of the NIST Speech Transcription Workshop*, College Park, MD.

Stymne, S. and Ahrenberg, L. (2010). Using a grammar checker for evaluation and post-processing of statistical MT. In Chair), N. C. C., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, pages 2175–2181, Valletta, Malta. European Language Resources Association (ELRA).

Sun, G., Liu, X., Cong, G., Zhou, M., Xiong, Z., Lee, J., and Lin, C.-Y. (2007). Detecting erroneous sentences using automatically mined sequential patterns. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 81–88, Prague, Czech Republic.

Tanaka-Ishii, K., Tezuka, S., and Terada, H. (2010). Sorting texts by readability. *Computational Linguistics*, 36(2):203–227.

Tetreault, J. and Chodorow, M. (2008a). The ups and downs of preposition error detection in ESL writing. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING-08)*, pages 865–872, Manchester, United Kingdom. Association for Computational Linguistics.

Tetreault, J. and Chodorow, M. (2009). Examining the use of region web counts for ESL error detection. In *Web as Corpus Workshop (WAC-5)*, San Sebastian, Spain.

Tetreault, J., Foster, J., and Chodorow, M. (2010a). Using parse features for preposition selection and error detection. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010), Short Papers Volume*, pages 353–358, Stroudsburg, PA, USA. Association for Computational Linguistics.

Tetreault, J. R. and Chodorow, M. (2008b). Native judgments of non-native usage: Experiments in preposition error detection. In *Proceedings of the COLING Workshop on Human Judgements in Computational Linguistics*, pages 24–32, Manchester, UK.

Tetreault, J. R., Filatova, E., and Chodorow, M. (2010b). Rethinking grammatical error annotation and evaluation with the Amazon mechanical turk. In *Proceedings of the NAACL HLT 2010 Fifth Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2010)*, pages 45–48, Los Angeles, California. Association for Computational Linguistics.

Titov, I. and Henderson, J. (2006). Loss minimization in parse reranking. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing (EMNLP-06)*, Sydney, Australia.

van der Plas, L., Henderson, J., and Merlo, P. (2009). Domain adaptation with artificial data for semantic parsing of speech. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL'09), Companion Volume: Short Papers*, pages 125–128, Morristown, NJ, USA. Association for Computational Linguistics.

van Genabith, J. (2006). Rapid treebank-based acquisition of multilingual LFG resources. In Butt, M., Dalrymple, M., and King, T. H., editors, *Intelligent Linguistic Architectures: Variations on themes by Ronald M. Kaplan*, CSLI Lecture Notes, pages 111–136. CSLI Publications, Stanford, CA, USA. Festschrift.

van Halteren, H., Zavrel, J., and Daelemans, W. (2001). Improving accuracy in wordclass tagging through combination of machine learning systems. *Computational Linguistics*, 27(2):199–229.

van Zaanen, M. (1999). Error correction using DOP. In de Roeck, A., editor, *Proceedings of the Second UK Special Interest Group for Computational Linguistics (CLUK2)*, pages 1–12. University of Essex. Second Issue.

Verberne, S. (2002). Context-sensitive spell checking based on word trigram probabilities. Master's thesis, University of Nijmegen.

vor der Brück, T., Hartrumpf, S., and Helbig, H. (2008). A readability checker with supervised learning using deep syntactic and semantic indicators. In Erjavec, T. and Gros, J. Ž., editors, *Proceedings of the 11th International Multiconference: Information Society - IS 2008 - Language Technologies*, pages 92–97, Ljubljana, Slovenia.

Wagner, J. (2004). A false friends exercise with authentic material retrieved from a corpus. In Delmonte, R., Delcloque, P., and Tonelli, S., editors, *NLP and Speech Technologies in Advanced Language Learning Systems, Proc. of InSTIL/ICALL2004 Symposium on Computer Assisted Language Learning*, pages 115–118, Padova, Italy. Unipress.

Wagner, J. and Foster, J. (2009). The effect of correcting grammatical errors on parse probabilities. In *Proceedings of the 11th International Conference on Parsing Technologies (IWPT'09)*, pages 176–179, Paris, France. Association for Computational Linguistics.

Wagner, J., Foster, J., and van Genabith, J. (2007a). A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 112–121, Prague, Czech Republic. Association for Computational Linguistics.

Wagner, J., Foster, J., and van Genabith, J. (2008). Using decision trees to detect and classify grammatical errors. Presentation at the Calico '08 Workshop on Automatic Analysis of Learner Language: Bridging Foreign Language Teaching Needs and NLP Possibilities, 18th of March 2008.

Wagner, J., Foster, J., and van Genabith, J. (2009). Judging grammaticality: Experiments in sentence classification. *CALICO Journal (Special Issue of the 2008 CALICO Workshop on Automatic Analysis of Learner Language)*, 26(3).

Wagner, J., Seddah, D., Foster, J., and van Genabith, J. (2007b). C-structures and f-structures for the British National Corpus. In Butt, M. and King, T. H., editors, *Proceedings of the Twelfth International Lexical Functional Grammar Conference (LFG07)*, pages 418–438, Stanford, CA 94305-4115, USA. CSLI Publications, Stanford University.

Wang, Y. and Garigliano, R. (1992). An intelligent language tutoring system for handling errors caused by transfer. In Frasson, C., Gauthier, G., and McCalla, G. I., editors, *Intelligent Tutoring Systems: Second International Conference, ITS'92, Montreal, Canada, June 1992, Proceedings*, volume 608 of *Lecture Notes in Computer Science*, pages 395–404. Springer Berlin / Heidelberg.

Way, A. (2010). Machine translation. In Clark, A., Fox, C., and Lappin, S., editors, *The Handbook of Computational Linguistics and Natural Language Processing*, pages 531–573. Wiley Blackwell, Chichester, UK.

Weber, H., Spilker, J., and Görz, G. (1997). Parsing n best trees from a word lattice. In Brewka, G., Habel, C., and Nebel, B., editors, *KI-97: Advances in Artificial Intelligence*, volume 1303 of *Lecture Notes in Computer Science*, pages 279–288. Springer Berlin / Heidelberg. 10.1007/3540634932_22.

Wilcox-O'Hearn, A., Hirst, G., and Budanitsky, A. (2008). Real-word spelling correction with trigrams: A reconsideration of the Mays, Damerau, and Mercer model. In Gelbukh, A., editor, *Computational Linguistics and Intelligent Text Processing - 9th International Conference, CICLing 2008, Haifa, Isarael, February 17–23, 2008 - Proceedings*, volume 4919/2008, pages 605–616. Springer Berlin/Heidelberg, Germany. 2006 draft version available on http://ftp.cs.toronto.edu/pub/gh/WilcoxOHearn-etal-2006.pdf.

Witten, I. H. and Frank, E. (2000). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers.

Wong, S.-M. J. and Dras, M. (2010). Parser features for sentence grammaticality classification. In *Proceedings of the Australasian Language Technology Workshop (ALTA 2010)*, pages 67–75, Melbourne, Australia. Australasian Language Technology Association and Association for Computational Linguistics.

Xu, P. and Jelinek, F. (2007). Random forests and the data sparseness problem in language modeling. *Computer Speech and Language*, 21(1):105–152.

Yarowsky, D. (1994). Decision lists for lexical ambiguity resolution: Application to accent restoration in Spanish and French. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL'94)*, pages 88–95, Las Cruces, New Mexico, USA. Association for Computational Linguistics.

Yi, X., Gao, J., and Dolan, W. B. (2008). A web-based English proofing system for English as a second language users. In *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume II*, pages 619–624, Hyderabad, India.

Zwarts, S. and Dras, M. (2008). Choosing the right translation: A syntactically informed classification approach. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, volume 1, pages 1153–1160, Manchester, UK. Association for Computational Linguistics.

Appendix A

Preprocessing Details

A.1 BNC Sentence Extraction

In the original BNC, the start but not the end of each sentence is marked. Usually, the end of a sentence is indicated by the start of the next sentence or the end of the document. Very occasionally (18 cases), we found text after SGML tags such as paragraph markers without a sentence marker or at the very start of documents (BNC files A8E and A8L). Table A.1 shows the positions of these 18 cases in the original BNC material and the extracted strings. Most cases are caused by attribute values containing the character > without escaping them. This is legal in SGML (Rieger, 1995, p. 126). However, the SGML parser module we use (Python's sgmllib) does not handle this situation correctly. We extracted a total of 6,228,111 sentences including the above 18 sentences. One BNC ID (HJ5.3786) was detected as a duplicate (lines 22566 and 22569 in file HJ5) and we added the letter "b" to the ID of the second occurrence.

A.2 Soft Hyphen Disambiguation

Table A.2 shows the 14 most frequent tokens containing soft hyphens in the BNC together with the substitution candidates of the replacement straftegy of Section 3.1.1.

Filename	Line	String	Note
A0D	8196	actual text' ed=OUCS	3
A8E	155	23-NOV-89 edition, page 29 " org=SEQ>	1
A8E	471	23-NOV-89 edition, page 29 " org=SEQ>	1
A8F	1152	23-NOV-89 edition, page 22 " org=SEQ>	1
A8F	2027	23-NOV-89 edition, page 37 " org=SEQ>	1
A8F	2169	23-NOV-89 edition, page " org=SEQ>	1
A8F	2255	23-NOV-89 edition, page " org=SEQ>	1
A8F	2343	23-NOV-89 edition, page 26 " org=SEQ>	1
A8F	2698	23-NOV-89 edition, page " org=SEQ>	1
A8F	2798	23-NOV-89 edition, page 26 " org=SEQ>	1
A8H	1197	23-NOV-89 edition, page 012 " org=SEQ>	1
A8J	1380	23-NOV-89 edition, page 008 " org=SEQ>	1
A8K	1963	23-NOV-89 edition, page 6 " org=SEQ>	1
A8K	4790	23-NOV-89 edition, page 4 " org=SEQ>	1
A8L	155	23-NOV-1989 edition, page 18 " org=SEQ>	1
GX5	1752	" ed=OUP>	2
GX5	1940	" ed=OUP>	2
KDP	2773	Two fifty, sixty, two seventy. "	3

Table A.1: BNC locations of out-of-sentence strings found by our end-of-sentence heuristics; Notes: 1 = greater than sign in attribute value, 2 = full tag in attribute value, probably unintended, 3 = invalid SGML

Form	Soft Hyphen	Hyphen/Minus	Bigram	One Word
- (stand-alone hyphen)	76	**2,437**	—	—
hard-working	6	**201**	53	88
full-time	5	**2,137**	586	37
non-existent	5	**395**	10	39
self-evident	5	**273**	28	0
time-consuming	5	**382**	209	1
well-established	5	331	**517**	0
long-standing	4	156	118	**193**
much-needed	4	197	**211**	0
near-certainty	4	**6**	**6**	0
self-destructive	4	**59**	0	0
well-being	4	**726**	100	149
well-intentioned	4	**83**	30	0
well-known	4	1,492	**1,762**	5

Table A.2: Most frequent tokens containing soft hyphens in the BNC and frequency of candidate substitutes; the highest frequency is shown in bold for each row and marks the substitution that will be chosen by our substitution heuristic.

Translation to American English

We modified the varcon tool to not change “For” to “Four” because this substitution would also apply to the preposition “for” at the start of a sentence or in headings.

Appendix B

Early APP/EPP Experiments

This section documents proof of concept work that might have influenced some design decisions. The headings follow the corpora that were used in the different stages of this research, with the exception of Section B.3 on k-NN techniques.

B.1 Pelcra Learner Data

The PELCRA project at the University of Lodz, Poland, aims to build various corpora including the Polish Learner English Corpus. The project website[1] suggests that the project was most active between 1996 and 2001 and that the work has not been finished. Nevertheless, a sample of the learner corpus was available for download in November 2004. It contains 2 essays with 30 and 36 sentences respectively.

I identified 6 sentences that contain exactly one grammatical error but no other errors or questionable constructions. Later error annotation added in November 2007 confirms this low yield of ungrammatical language, see Table B.1. An additional 6 grammatical sentences were selected from the essays. I doubled the set of sentences by correcting the 6 ungrammatical sentences and inserting artificial errors into the 6 grammatical sentences. The resulting 24 sentences were parsed with LoPar (Schmid, 2000) and the English head-lexicalised PCFG grammar available for download on the IMS website[2]. According to the information on the website, the grammar is either based on or identical to the grammar

[1] `http://pelcra.ia.uni.lodz.pl/`

[2] `http://www.ims.uni-stuttgart.de/tcl/SOFTWARE/English-HLPCFG-en.html`

Frequency	Error Type	Description
2	BLEND	two words amalgamated, i. e. missing space
7	COMP	composite or multiple errors
3	MISSING	a word is missing
3	SUBST	wrong word, but not a context-sensitive spelling error
16	questionable	needing correction but not clearly ungrammatical
35	gram	no error

Table B.1: Errors in the 66 sample sentences of the Pelcra Leaner corpus according to Foster's error annotation

Pair	$\mathbf{c}_2$	$\mathbf{c}_3$	$\mathbf{c}_4$	$\mathbf{c}_5$	$\mathbf{c}_6$	$\mathbf{e}_3$
Decrement (logarithmic)	0.2	2.3	1.0	0.6	1.1	5.0

Table B.2: Drop of parse probability for those sentences that did not change length

of Carroll and Rooth (1998) who estimate probabilistic parameters of a hand-written grammar on BNC data.

Figure B.1 shows the logarithmic parse probabilities over sentence length for the 24 sentences derived from the Pelcra sample corpus. Generally, a strong log-linear relationship between sentence length and parse probability can be seen.[3] However, the point of interest is whether ungrammatical sentences have a significantly lower parse probability than correct sentences. The small data set does not support statistical significance testing. One can easily see that + and − signs marking correct and ungrammatical sentence are highly intermixed. There are six + signs and seven − signs above a manually drawn best-fit straight line. The situation is improved if only pairs of sentences that remained the same length after correction or insertion of errors are considered. Table B.2 shows the amount of drop. Parse probabilities for c_2 did not change much, c_3, c_4, c_5 and c_6 drop in parse probability after error insertion and e_3 drops even though it has been corrected. The 4:1 ratio (or 5:1 if c_2 is included) suggests that parse probabilities are somewhat lower for ungrammatical sentences than for corresponding correct sentences and provides a motivation to continue working on this approach to error detection.

Early plans for the APP/EPP method included ordinary n-gram language models as EPP models. Therefore, the 24 sentences were also plotted with logarithmic parse

[3]It is approximately $p = 0.002^{x-2}$ for this data set, p being the probability and x sentence length.

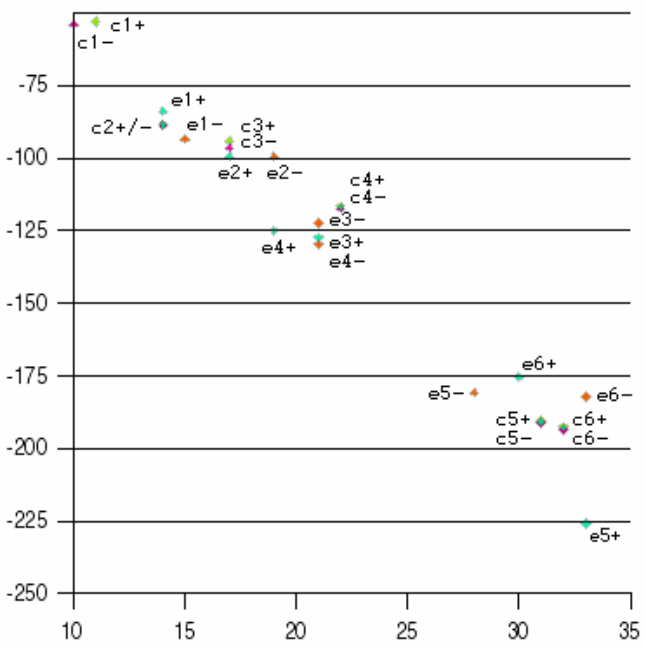


Figure B.1: Logarithmic parse probability over sentence length — c_i^+ authentic and correct, c_i^- error-inserted version of c_i^+, e_i^- authentic ungrammatical, e_i^+ corrected version of e_i^-

probability over language model perplexity measured with the CMU-Cambridge Statistical Language Modeling toolkit[4] by Clarkson and Rosenfeld (1997), but the results are similarly mixed.

B.2 Glasgow Herald

During my participation in CLEF[5] information retrieval experiments (Jones et al., 2005), I had access to a collection of large corpora. British English is represented by the Glasgow Herald 1995. Balancing the copyright holders' interests and the way the data is utilised in my experiments, I decided that it should be fair to use a random sample of 5,000 sentences to build my first EPP models. The corpus is parsed with LoPar and the same grammar as in the Pelcra experiments in the previous section.

[4] `http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html`
[5] `http://www.clef-campaign.org/`

B.2.1 Corpus Preprocessing

The CLEF corpus is encoded in SGML with tags for headings and captions, but otherwise the text is raw, for example consecutive numbers appear inside sentences presumably marking the location of page breaks in the original print document. I made an effort to clean up the data because the EPP model should be based on grammatical language. The cleaned corpus was then passed through the IMS TreeTagger of Schmid (1994) to detect sentence boundaries. Some post-filtering was applied to correct some types of errors, for example heuristics are used to detect abbreviations that are not part of the lexicon of the tagger.[6] This resulted in a segmentation of the corpus into approximately 1.2 million sentences. Finally, 5,000 sentences were randomly selected to be used in this work.

B.2.2 k-NN Implementation and Experimental Setup

An efficient exact k-NN search structure for 2 dimensions has been implemented. The experiments are conducted with $k = 50$ and an exponential weighting function that reduces the influence of training items with a mismatch of sentence length and/or tree height. The data is split into 4 sections of 1,250 sentences each. 2 sections are used for training, 2 for testing, resulting in 6 possible runs for each experiment. However, in order to collect even more data on the variance of results caused by the test data, we repeatedly test on random subsets of 1,250 sentences of the 2,500 sentences of each of the 6 runs. I exhaustively search for optimal parameters for scaling sentence length and tree height.

B.2.3 Terminal Rule Probabilities

LoPar's parse tree output contains annotation of PCFG rule probabilities. I read the terminal rule probabilities off the parse trees and then use them as described in Section 4.5.8.

B.2.4 Evaluation Measures and Results

In addition to the mean square error of the prediction of logarithmic parse probability of grammatical sentences 2 more evaluation metrics are introduced: a) the fraction of grammatical sentences for which the prediction error is below 3, i. e. $|\ln(\mathrm{EPP}/\mathrm{APP})| < 3$, and

[6] The sentence boundary detection module of student project KoKS Erpenbeck et al. (2002) was re-used.

Method	M. S. Error	$\lvert\ln(\mathbf{EPP}/\mathbf{APP})\rvert < 3$	$\mathbf{EPP}/\mathbf{APP} > e^3$
Sentence length only	21.1 – 31.8	10%–15%	36%–43%
Adding tree height	21.2 – 31.3	11%–15%	34%–42%
Factoring out t. rules	7.6 – 18.8	34%–40%	23%–29%

Table B.3: First results with Glasgow Herald corpus

b) the fraction of grammatical sentences which would be misclassified as ungrammatical if the threshold C was set to 3, i. e. EPP/APP $> e^3$.

Table B.3 shows the results. The tree height feature does not help. I only found noise while searching for optimal parameters. However, the exclusion of terminal rule probabilities reduces the prediction error and misclassifications of grammatical sentences considerably.

B.3 k-Nearest Neighbour Experiments

I experimented with various weighting functions and values for k in simple k-NN model predicting parse probabilities with just 1 or 2 features (sentence length and tree height) and a small number of training items. The small impact of the tree height feature made it easy to interpret plots of slices of the resulting functions (keeping tree height fixed, see Figure B.2) or colour-coded plots with 2 independent variables. Local linear regression, i. e. fitting a linear function to the k-nearest items, is also tested as it can be expected to remove a bias of constant functions in uneven distributions.

The observations suggest that weighting functions have a similar effect to reducing k in areas with a low density of training items. Since our data is highly noisy this is counter-productive. We have to average over a large number of items. Linear regression, however, only improved results marginally and therefore was abandoned (and also because it would be more difficult to implement for a higher number of dimensions).

B.4 Europarl

From November 2005 to March 2006 experiments were conducted on the EuroParl corpus which contains proceedings of the European Parliament from 1996 to 2003 in various

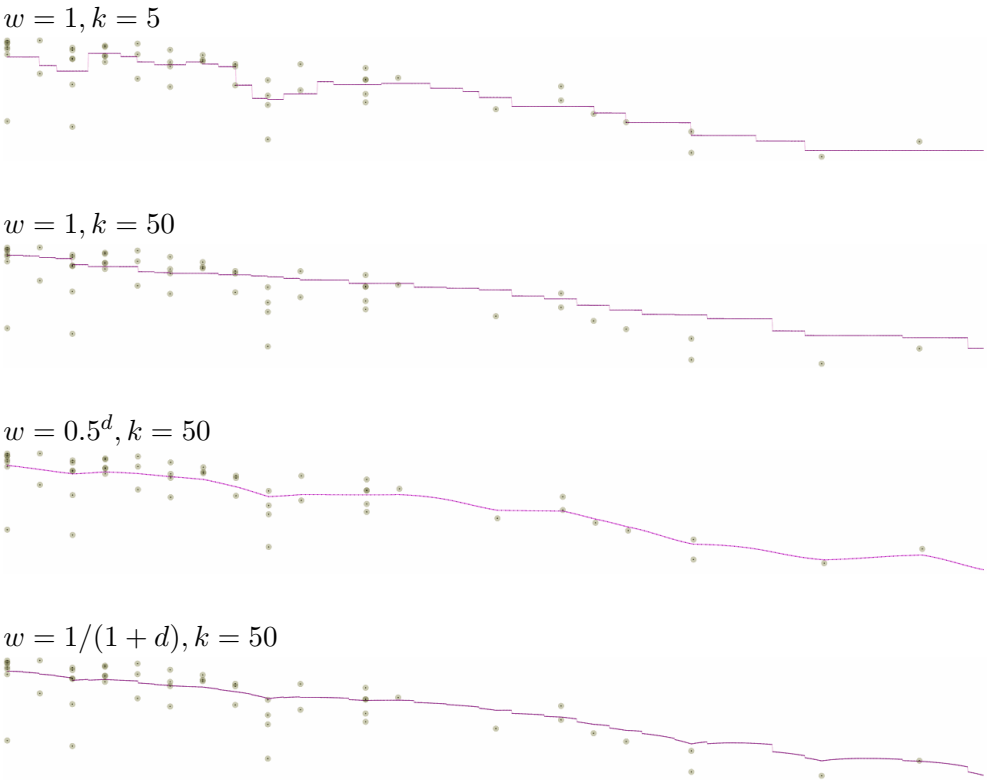


Figure B.2: Effect of weighting functions on k-NN

languages. The move was made because it opens the door to train and compare the APP/EPP method for different languages. At the same time, we switched to Charniak and Johnson (2005)'s parser, and to Mount (2005)'s approximate k-NN implementation. As often, corpus pre-processing was laborious. Quotes, apostrophes and erroneous SGML markup needed special attention.

The experiments add various features (number of nodes, character trigrams, LM probabilities, terminal rule probabilities) and scale the axes of the k-NN vector space to optimise the mean square error. Scaling the axes is equivalent to tuning the feature weights of the similarity function which we cannot manipulate directly in the ANN implementation.[7]

We visualise results with a box plot. Since we only have 10 results per experiment (10-fold cross-validation), we define the interquartile range which will be represented by a box to go from the 4th to the 7th item and we do not mark outliers. Instead of choosing a definition for the median of an even number of items, we simply plot the 5th and 6th value as dots inside the box. If the box for one method is fully below the box of another method, we regard this as a significant improvement.[8] Figure B.3 shows 2 typical charts out of a series of 28 charts evaluating over 200 EPP models.[9]

B.4.1 Observations and Results

The bin method performs better than k-NN if only sentence length is used. However, as soon as we add a second feature (tree height and number of nodes have been tested), the bin method becomes inferior to k-NN for $20 \leq k \leq 50$. The improvements are larger for adding the number of nodes feature than for the tree height feature. Combining all 3 features does not improve results any further.

Reducing the size of the training data to 20% and 4% shows a moderate increase of the mean square error from 585 to 618 and 673 (middle of the interquartile box) and the relative positions of the interquartile boxes suggest that k should be lowered, though

[7] The ANN library only allows to change the function by re-compiling the library. The manual shows macro definitions for the parameterised Minkowski L_p norm which is the Euclidean norm for $p = 2$ and the Manhatten norm for $p = 1$.

[8] Assuming that the results are drawn from a single normal distribution, the probability of 2 boxes not overlapping is 17.8%, i. e. the p-value of this test is 17.8%. Including the 3rd and 8th item into the interquartile box would tighten the p-value to 2.3%, but this was not considered at the time.

[9] Not only feature weights were optimised but also normalisation parameters for frequencies and probabilistic features and the parameter k.

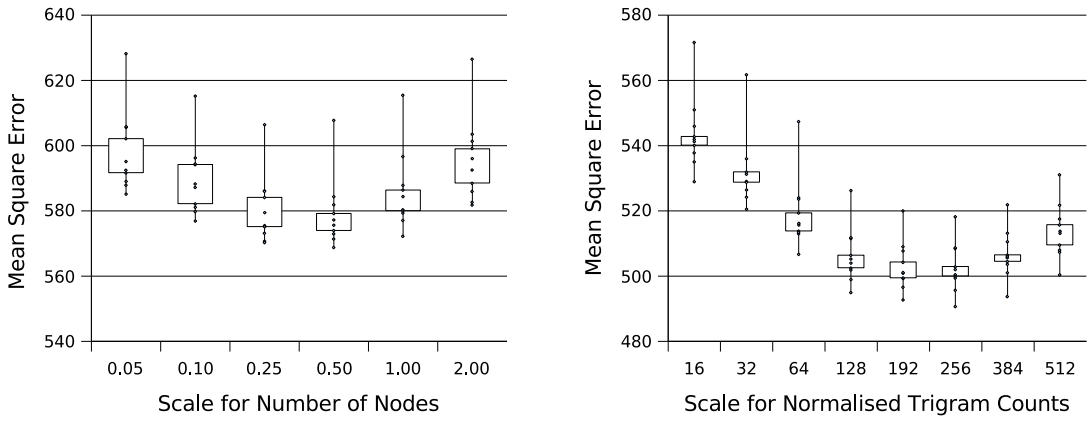


Figure B.3: Box plot of cross-validation results over a range of feature weights. Left: k-NN model with 3 features (sentence length, tree height and number of nodes). Right: 6 experiments later weights for the first 3 features scaled with the optimal factors previously found and the 4th to 9th feature (character trigram frequencies) are added and scaled by a single factor as a group.

$k = 20$ remains optimal within the set of tested values (7, 20, 50, 120 and 360).

Tuning the feature weights yields small but significant improvements (MSE 567). Repeating the optimisation for a feature after other features have been added and tuned shows that the independence assumptions are not correct, but no significant improvement can be achieved by tuning two features simultaneously. The best parameters for the first 3 feature are $k = 30$, tree height weight 0.22 and number of nodes weight 0.30. (The weight of the sentence length feature always stays 1.0.)

The 6 character trigram features reduce the mean square error considerably to 502 if the frequency values are normalised by sentence length in characters and scaled by a factor between 192 and 384.[10] We also considered normalisations by the square root of the length and no normalisation. The optimal weights are 12 and 0.5 respectively, i. e. they highly depend on the normalisation. No improvement could be gained.

How lexical information is best integrated into the k-NN model depends on whether we train the LM on the Penn Treebank, i. e. the data the parser is also trained on, or EuroParl data.[11] Figure B.4 shows box plots for the various options. (Some of the boxes

[10] We did not optimise individual weights here. Within the reported weight range there is no significant difference.

[11] Note that we used out own LM implementation with a simple discounting method in the EuroParl

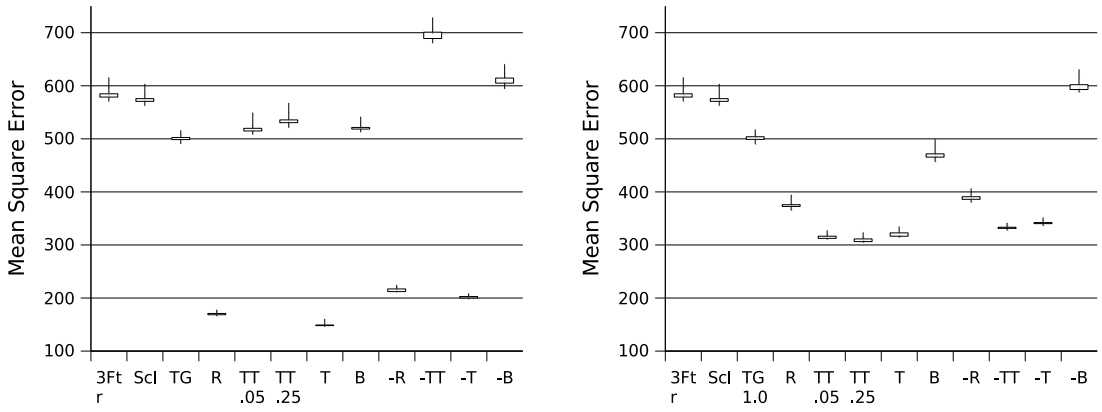


Figure B.4: Integrating LM and terminal rule probabilities into the EPP model. Left: trained on PTB gold parse trees. Right: Trained on parsed EuroParl. R=terminal rules, TT=tagged tokens, T=unigram token, B=bigram token, minus sign=factoring out method instead of k-NN feature, 3Ftr=baseline method with sentence length, tree height and number of nodes, Scl=with scaling, TG=with character trigram features

and sticks are so short that they appear as little crosses.) On the left (PTB-trained), we see that terminal rule probabilities (R) and token unigram LM probabilities (T) improve the mean square error to 149, much more than for the other features. The factoring out method (-) is somewhat inferior to using the probabilities as features in the k-NN model. If the probability estimates are obtained from EuroParl data, the LM on POS-tagged tokens performs best followed by plain token unigrams and terminal rule probabilities. However, the mean square error remains above 300.

Using both PTB-trained terminal rule and token unigram features in the k-NN model reduces the mean square error to 140 (not shown in Figure B.4; weights are 0.10 and 0.25 respectively). Combining this method with the factoring out method (we only tested the token unigram LM here) gives us the best EPP model with a mean square error of 138.6. However, according to our criterion of non-overlapping interquartile boxes, the difference to the pure k-NN method is not significant.

experiments, not SRILM as described in Section 4.5.6. SRILM will be used in Section 4.6.

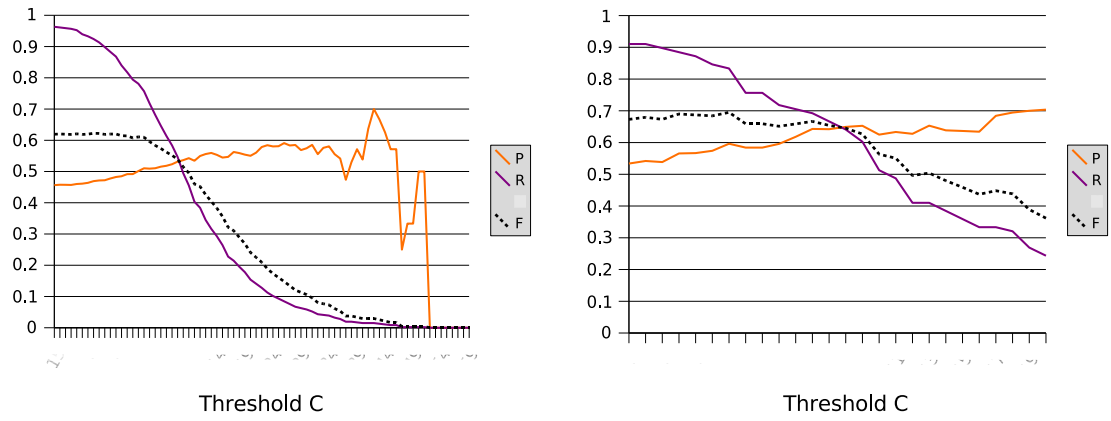


Figure B.5: Precision, recall and f-score graphs for the EPP model trained on EuroParl data. Left: evaluation on a development section of the Foster 2005 corpus (461 ungrammatical, 568 grammatical sentences). Right: evaluation on 78 sentences with real-word spelling errors and their corrections (error density 50%)

B.5 Evaluation on Foster's Error Corpus

With good progress in the development of EPP models, it became urgent to include authentic errors in the evaluation procedure. I selected Foster (2005)'s parallel error corpus with contains ungrammatical sentences taken from written text and one (or sometimes 2) corrections. Each test item is labelled with one of the 2 classes "grammatical" or "ungrammatical" and the automatic classifiers can be evaluated with precision and recall metrics. We plotted precision over recall for a range of values for threshold C. In order to break down results by error type, I re-annotated the parallel error corpus according to Foster's error taxonomy with a semi-automatic annotation tool which reduces the choice of possible annotations based on a word alignment of correction and original and which I implemented for this work. Figure B.5 exemplifies 2 of the various graphs that have been drawn. We can see that the EPP method works in principle: there is a precision-recall trade-off and precision almost reaches 60%. Unfortunately, we have to set the threshold C so high that recall drops below 10%. (If it is increased further, very few sentences are flagged as ungrammatical and the precision values become noisy.) However, precision is poor for values of C that result in high recall. The overall f-score does not exceed the baseline of flagging all input as ungrammatical (except for a tiny, hardly visible bump in

the f-score curve at C = −9).[12] The best results we can report are for context-sensitive spelling errors. For this error type the f-score peaks at C=1 with precision = 59.6%, recall = 83.1% and f-score = 69.5%. For the other error types, results are worse and also less reliable as only very few test items are in the corpus. A breakdown by sentence length has also been made, but remained mostly inconclusive. The recall and f-score curves are less steep for long sentences.

B.6 Early BNC Experiments

It was suggested to switch to a larger corpus, the British National Corpus, for which parser output was available in house. Unfortunately, the archived parse trees did not include parse probabilities. However, the POS annotation implicitly available in the parse trees could be used for applying the automatic procedure for creating an artificial error corpus described in Chapter 3. Therefore, raw text was extracted from the parse trees and parsed again with Charniak's parser, this time with 2-best parsing enabled because Charniak's parser outputs parse probabilities if n-best parses are requested with $n > 1$. In order to speed up parsing, we limited the sentence length to 20 tokens (exclusive).

B.6.1 Results

The size of the training data only doubled due to the length restrictions. In the first experiment, we reduced the size of the training data by 1 and 2 orders of magnitude to get a rough learning curve. The precision-recall curve[13] does not change. The curve is almost linear with the following support points: (10%, 65.5%), (50%, 60%), (90%, 52.5%) and (100%, 50%).[14] The breakdown by error type shows a strange result: context-sensitive spelling errors showed a precision below 50%, i. e. the system preferred to flag grammatical sentences as ungrammatical over flagging these errors.

Adding the POS features improves the mean square error from approximately 96 to 87.5 and also increases the precision of the classifier for extra word errors. (Other error

[12] With the error density p and recall = 1.0 (selecting all), we can calculate the f-score = $2p/(p+1)$, i. e. $461/745 \approx 61.9\%$ for the left graph and $2/3 \approx 66.7\%$ for the right graph.

[13] This time we plot precision over recall leaving the parameter C implicit.

[14] Below 10% recall the curve is too noisy despite 200,000 test items.

types were not tested.)

B.6.2 Built-In Language Model

An experiment with the trigram LM output of Charniak's parser has been made. The parser can only output this additional information if it has been requested at training time, i. e. the grammar had to be re-trained. Therefore, probability values are not comparable to the off-the-shelf parser. The results show that factoring out the LM probabilities reduces the mean square error from 85 to 58 when using the square root of the LM probability, but the precision of the corresponding classifier is lower than before. As a feature in the k-NN model, the parser's LM output again can lower the mean square error to 57 and the classifier is very similar to the previous one. Combining the 2 methods also does not make a difference. This motivated an experiment in which the LM features added in the EuroParl experiments are excluded to see the effect on the precision-recall graph. However, results confirm that the old LM features are beneficial to both mean square error and the classifier.

B.7 BLEU Score

We retrieve the k_1-nearest sentences ($k_1 = 20, 100, 300, 1000, 5000$) according to our previous k-NN model and then re-rank the k_1 sentences according to BLEU score (Papineni et al., 2002).[15] The top $k_2 = 20$ sentences are used to calculate the EPP. Due to the time needed to retrieve k_1 sentences from disk for each test sentence (the reference corpus does not fit into memory), we use only 500 test sentences.

The first result is that we cannot use 4-grams in the BLEU measure because too many input sentences do not find any matches, i. e. all scores are 0. Best improvement of mean square error is observed for BLEU with unigrams and bigrams with $k_1 = 1000$. However, the improvement is small, just about significant: the mean square error decreases from approximately 85 to 80.

BLEU score calculated on POS tags gives us more matches than with tokens and higher scores. However, the improvements of mean square error are very small and the

[15] Lin and Hovy (2003)'s description was also useful in the implementation.

same as for tokens (80). Reducing the tag set does not help either. BLEU score on token frequency classes performs worse: it does not even outperform the baseline of 85.

It is interesting though that results do not degrade. BLEU somehow avoids moving unsuitable reference sentences to the top of the list. The BLEU reranker outperforms a reranker that randomly shuffles items on the k_1-best list for all tested values of k_1. This suggests that we should try to combine BLEU score and k-NN distance to a single, improved score. We exhaustively searched linear models. However, we did not find any weights that show an improvements over using BLEU alone.

B.8 Using the Web as a Reference Corpus

The purpose of this experiment is to see if we can improve BLEU score-based EPP models by retrieving reference sentences with the help of a web search engine.

B.8.1 Building a Corpus from Seed Key Words

We used the BootCaT tool (Baroni and Bernardini, 2004) to retrieve a corpus from the web using seed tokens. The tool uses a Google API to query the Google search engine. The tool then downloads the top 20 documents, expands the seed set and downloads more documents. We normalise special characters and use a simple sentence boundary detector based on the idea of identifying abbreviations with statistics on their co-occurrence with token-final periods (Kiss and Strunk, 2006).

B.8.2 Experiment

For the sentence

- Stuart Quarrie is a yachting consultant specialising in race training and coaching. [BNC G37.1945]

we use the seeds "stuart quarri [sic] yacht consult special race train coach". For 25 of the 336 possible triplets we retrieve the top 20 documents, pre-process the text, in particular fixing multiple times UTF-8 encoded special characters, and get 6.4 MB of plain text. Besides character set normalisation, pre-processing includes sentence boundary detection,

abbreviation detection (removes sentence boundaries) and tokenisation. The final corpus has 55,985 sentences of which 17,814 fall in the length range 10–19 which was uses in the BNC experiments of that time (summer 2006).

Results: No BLEU matches (score > 0) with $N = 3$. 400 matches with $N = 2$, but best one only scores 0.073 while with BNC reference data ($k_1 = 20000$, see Section B.7), we find 268 matches, best 0.085.

B.8.3 Feasibility

For a single experiment, we have to query at least 5,000 sentences. For each sentence BootCaT sends 25 queries to the Google API. However, the Google API is limited to 1,000 calls per day, i. e. an single experiment would take 125 days. In addition, we estimate that we would have to download 5,000 $\times$6.4 MB = 1.2 TB of text (not including HTML markup which we cannot avoid to also download).

Appendix C

Additional Material

C.1 Character Trigram Candidates for the EPP Model

Triplets of rank, trigram and frquency (see Section 4.5.4):
(1, ‘ th’, 3,296,137), (2, ‘the’, 2,492,768), (3, ‘he ’, 2,043,239), (4, ‘ , ’, 1,358,622), (5, ‘on ’, 1,092,690), (6, ‘ion’, 1,069,184), (7, ‘ . ’, 1,000,582), (8, ‘ in’, 989,176), (9, ‘ to’, 981,259), (10, ‘ of’, 971,219), (11, ‘of ’, 944,517), (12, ‘ co’, 932,258), (13, ‘to ’, 923,266), (14, ‘ an’, 889,868), (15, ‘nd ’, 849,354), (16, ‘and’, 802,101), (17, ‘is ’, 781,033), (18, ‘ent’, 769,213), (19, ‘ed ’, 721,597), (20, ‘tio’, 711,706), (21, ‘in ’, 708,603), (22, ‘ing’, 694,781), (23, ‘ng ’, 676,141), (24, ‘at ’, 661,388), (25, ‘es ’, 655,723), (26, ‘e t’, 628,729), (27, ‘ re’, 607,771), (28, ‘re ’, 602,811), (29, ‘nt ’, 581,528), (30, ‘n t’, 561,422), (31, ‘ pr’, 555,322), (32, ‘ be’, 552,081), (33, ‘er ’, 538,251), (34, ‘hat’, 533,785), (35, ‘e a’, 522,299), (36, ‘tha’, 520,432), (37, ‘men’, 490,416), (38, ‘e c’, 489,001), (39, ‘ is’, 487,893), (40, ‘ati’, 486,635), (41, ‘t t’, 485,290), (42, ‘al ’, 477,408), (43, ‘for’, 473,975), (44, ‘s a’, 466,221), (45, ‘ a ’, 444,917), (46, ‘ly ’, 419,301), (47, ‘an ’, 414,740), (48, ‘s t’, 407,251), (49, ‘ fo’, 405,509), (50, ‘d t’, 403,927), (51, ‘ wh’, 402,710), (52, ‘as ’, 402,179), (53, ‘com’, 402,030), (54, ‘or ’, 401,841), (55, ‘ve ’, 400,632), (56, ‘f t’, 396,031), (57, ‘ ha’, 394,822), (58, ‘thi’, 389,538), (59, ‘ we’, 389,463), (60, ‘res’, 385,861), (61, ‘ on’, 379,286), (62, ‘ wi’, 375,756), (63, ‘pro’, 370,145), (64, ‘con’, 361,317), (65, ‘e i’, 349,208), (66, ‘e o’, 347,273), (67, ‘ts ’, 345,750), (68, ‘t i’, 340,169),

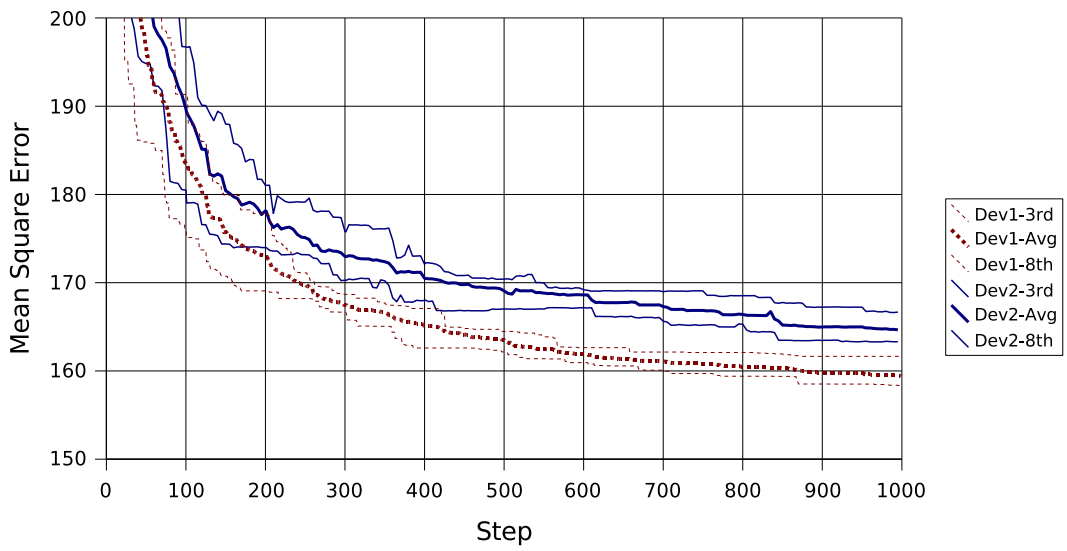


Figure C.1: Optimisation progress: mean square error of the best shot measured on development data (dev1) and validated on unseen data (dev2); the gap is statistically significant with a p-value of 2.3 from step 235 except for steps 415 and 420

(69, ‘ no’, 327,498), (70, ‘ it’, 326,885), (71, ‘ate’, 324,161), (72, ‘en ’, 321,123), (73, ‘her’, 320,586), (74, ‘ll ’, 315,720), (75, ‘se ’, 313,699), (76, ‘ons’, 313,437), (77, ‘his’, 310,574), (78, ‘e s’, 306,976), (79, ‘we ’, 305,870), (80, ‘st ’, 300,309), (81, ‘s o’, 296,648), (82, ‘e p’, 295,752), (83, ‘it ’, 290,236), (84, ‘ i ’, 289,698), (85, ‘ch ’, 289,625), (86, ‘le ’, 282,738), (87, ‘ de’, 281,218), (88, ‘are’, 278,916), (89, ‘e e’, 277,200), (90, ‘ns ’, 276,983), (91, ‘ere’, 273,800), (92, ‘ ma’, 271,756), (93, ‘n a’, 270,919), (94, ‘rop’, 269,337), (95, ‘ter’, 266,437), (96, ‘ ar’, 266,013), (97, ‘ope’, 265,213), (98, ‘omm’, 262,026), (99, ‘ort’, 261,059) and (100, ‘all’, 257,012).

C.2 EPP Model Optimisation

C.2.1 Additional EPP Over-fitting Graph

Figure C.1 shows the respective graph for optimising the mean square error. Here, we do not see any over-fitting effects. The slope of the curves is decreasing. In contrast to the accuracy optimisation, we do not expect to be able to improve the mean square error substantially. However, in both optimisations, we do not yet have a clear sign that the process should be stopped due to over-fitting.

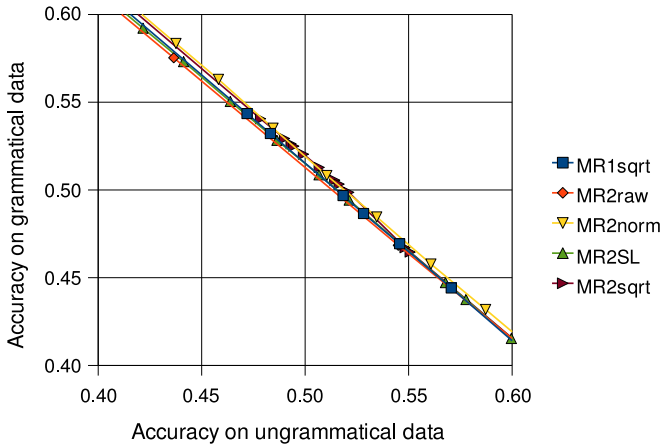


Figure C.2: Accuracy curves for methods based on the frequency of Markovisation rules in the parser output

C.3 Basic Methods

C.3.1 n-gram Parameters

Table C.1 compares accuracy results of the n-gram method of Section 5.2 for training data and test data.

C.3.2 PCFG Pruning Parameters

Table C.2 produces a corresponding comparison for the PCFG pruning method (Section 5.3).

C.3.3 Markovisation Rules

Figure C.2 shows low accuracy for classifiers based on the number of Markovisation rules (with and without parent annotation and normalised in different ways) in the parser output: for a balanced accuracy trade-off (equal accuracy on grammatical and ungrammatical data), accuracy is approximately 51% which is is a marginal improvement over coin-flipping.

Parameter	Avg. tr.	Avg. test	Min. test	Max. test
raw, 2, 6	0.31% 99.96%	0.31% 99.96%	0.30% 99.95%	0.32% 99.96%
raw, 2, 25	1.11% 99.81%	1.11% 99.81%	1.06% 99.79%	1.15% 99.83%
padded, 3, 1	2.06% 99.62%	2.06% 99.62%	2.02% 99.61%	2.12% 99.63%
padded, 3, 2	3.35% 99.25%	3.35% 99.25%	3.27% 99.22%	3.40% 99.28%
padded, 3, 3	4.54% 98.90%	4.54% 98.90%	4.49% 98.87%	4.63% 98.94%
padded, 3, 4	5.67% 98.57%	5.67% 98.57%	5.64% 98.54%	5.74% 98.61%
padded, 3, 6	7.34% 97.92%	7.34% 97.92%	7.29% 97.89%	7.38% 97.96%
padded, 3, 8	8.70% 97.30%	8.70% 97.30%	8.65% 97.26%	8.75% 97.36%
padded, 3, 10	10.02% 96.71%	10.02% 96.71%	9.94% 96.64%	10.09% 96.77%
padded, 4, 1	15.65% 94.14%	15.65% 94.14%	15.54% 94.09%	15.75% 94.21%
padded, 4, 2	22.33% 90.45%	22.33% 90.45%	22.17% 90.36%	22.45% 90.52%
padded, 4, 3	26.94% 87.54%	26.94% 87.54%	26.80% 87.43%	27.06% 87.62%
padded, 4, 4	30.60% 85.08%	30.60% 85.08%	30.49% 84.98%	30.75% 85.19%
padded, 4, 5	33.67% 82.94%	33.67% 82.94%	33.55% 82.84%	33.79% 83.06%
padded, 4, 6	36.31% 81.03%	36.31% 81.03%	36.15% 80.87%	36.46% 81.18%
padded, 5, 1	46.08% 73.69%	46.08% 73.69%	45.94% 73.53%	46.21% 73.85%
padded, 5, 2	56.46% 64.42%	56.46% 64.42%	56.29% 64.12%	56.65% 64.62%
padded, 5, 3	62.33% 58.56%	62.33% 58.56%	62.18% 58.26%	62.60% 58.77%
padded, 5, 5	69.40% 50.85%	69.40% 50.85%	69.31% 50.64%	69.59% 51.02%
padded, 5, 6	71.74% 48.08%	71.74% 48.08%	71.66% 47.90%	71.94% 48.26%
padded, 5, 8	75.31% 43.71%	75.31% 43.71%	75.19% 43.59%	75.52% 43.89%
padded, 5, 10	77.83% 40.42%	77.83% 40.42%	77.72% 40.25%	78.09% 40.56%
padded, 5, 13	80.67% 36.60%	80.67% 36.60%	80.55% 36.44%	80.87% 36.77%
padded, 5, 16	82.73% 33.67%	82.73% 33.67%	82.60% 33.50%	82.97% 33.82%
padded, 5, 20	84.75% 30.64%	84.75% 30.64%	84.62% 30.49%	84.94% 30.75%
padded, 5, 25	86.59% 27.72%	86.59% 27.72%	86.45% 27.55%	86.77% 27.82%
padded, 5, 32	88.44% 24.72%	88.44% 24.72%	88.28% 24.55%	88.62% 24.82%
padded, 5, 40	89.95% 22.12%	89.95% 22.12%	89.80% 21.95%	90.09% 22.21%
padded, 5, 50	91.29% 19.70%	91.29% 19.70%	91.16% 19.53%	91.43% 19.82%
padded, 5, 63	92.56% 17.35%	92.55% 17.34%	92.38% 17.12%	92.69% 17.45%
padded, 5, 79	93.65% 15.31%	93.65% 15.31%	93.48% 15.13%	93.79% 15.40%
padded, 5, 100	94.63% 13.32%	94.63% 13.32%	94.48% 13.17%	94.73% 13.45%
padded, 5, 126	95.41% 11.64%	95.41% 11.64%	95.26% 11.49%	95.52% 11.75%
padded, 5, 158	96.12% 10.07%	96.12% 10.07%	96.00% 9.99%	96.20% 10.18%
padded, 4, 1259	97.63% 6.67%	97.63% 6.67%	97.52% 6.60%	97.68% 6.75%
padded, 5, 1000	99.26% 2.49%	99.26% 2.49%	99.24% 2.45%	99.28% 2.56%
padded, 4, 3162	99.35% 2.24%	99.35% 2.24%	99.33% 2.20%	99.39% 2.29%
padded, 4, 3981	99.50% 1.79%	99.50% 1.79%	99.48% 1.75%	99.53% 1.85%
padded, 4, 5012	99.77% 1.03%	99.77% 1.03%	99.75% 1.00%	99.78% 1.08%
padded, 5, 5012	99.97% 0.32%	99.97% 0.32%	99.97% 0.29%	99.98% 0.33%
padded, 6, 5012	99.99% 0.27%	99.99% 0.27%	99.98% 0.25%	99.99% 0.28%
padded, 7, 5012	99.99% 0.26%	99.99% 0.26%	99.99% 0.23%	99.99% 0.27%

Table C.1: Accuracy results for the n-gram method on training data (on which the selection of optimal parameters is based) and test data: parameters are padding, n and threshold t; only a subset of possible thresholds t is considered — see text

Parameter	Avg. tr.	Avg. test	Min. test	Max. test
5	1.74% 98.57%	1.74% 98.57%	1.69% 98.52%	1.84% 98.64%
6	1.86% 98.47%	1.86% 98.47%	1.81% 98.42%	1.95% 98.54%
21	7.85% 93.28%	7.85% 93.28%	7.68% 93.16%	7.97% 93.41%
23	7.94% 93.21%	7.94% 93.21%	7.78% 93.08%	8.06% 93.33%
27	10.12% 91.19%	10.12% 91.19%	10.03% 91.05%	10.24% 91.28%
31	10.63% 90.70%	10.63% 90.70%	10.51% 90.53%	10.78% 90.82%
79	20.18% 81.59%	20.18% 81.59%	20.03% 81.27%	20.48% 81.77%
82	20.19% 81.59%	20.18% 81.59%	20.03% 81.26%	20.48% 81.77%
598	60.35% 42.65%	60.35% 42.65%	60.18% 42.44%	60.58% 42.83%
602	60.54% 42.46%	60.54% 42.46%	60.35% 42.25%	60.76% 42.64%
603	64.85% 38.20%	64.85% 38.20%	64.62% 38.05%	64.99% 38.35%
992	82.64% 19.48%	82.64% 19.48%	82.55% 19.32%	82.78% 19.61%

Table C.2: Accuracy results for training and test data (PCFG pruning method)