ERROR DETECTION AND CORRECTION IN ANNOTATED CORPORA

DISSERTATION

Presented in Partial Fulfillment of the Requirements for

the Degree Doctor of Philosophy in the

Graduate School of The Ohio State University

By

Markus Dickinson, B.A., M.A.

* * * * *

The Ohio State University

2005

Dissertation Committee:

Professor W. Detmar Meurers, Adviser

Professor Christopher H. Brew

Professor Robert Levine

Approved by

Adviser
Department of Linguistics

ABSTRACT

Building on work showing the harmfulness of annotation errors for both the training and evaluation of natural language processing technologies, this thesis develops a method for detecting and correcting errors in corpora with linguistic annotation. The so-called variation n-gram method relies on the recurrence of identical strings with varying annotation to find erroneous mark-up.

We show that the method is applicable for varying complexities of annotation. The method is most readily applied to positional annotation, such as part-of-speech annotation, but can be extended to structural annotation, both for tree structures—as with syntactic annotation—and for graph structures—as with syntactic annotation allowing discontinuous constituents, or crossing branches.

Furthermore, we demonstrate that the notion of variation for detecting errors is a powerful one, by searching for grammar rules in a treebank which have the same daughters but different mothers. We also show that such errors impact the effectiveness of a grammar induction algorithm and subsequent parsing.

After detecting errors in the different corpora, we turn to correcting such errors, through the use of more general classification techniques. Our results indicate that the particular classification algorithm is less important than understanding the nature of the errors and altering the classifiers to deal with these errors. With such alterations, we can automatically correct errors with 85% accuracy. By sorting the errors, we can

relegate over 20% of them into an automatically correctable class and speed up the re-annotation process by effectively categorizing the others.

to my wife

Stephanie Dickinson

whose love and encouragement has exceeded all my expectations

ACKNOWLEDGMENTS

I would like to thank my adviser Detmar Meurers for his insight into this thesis and for his support and encouragement during my entire time in graduate school. He has taught me how to find my way in computational linguistic research and how to successfully collaborate. I look forward to even more collaboration in the future.

I would also like to thank Chris Brew for many discussions on issues both great and small related to this thesis and related to almost any aspect of computational linguistics. He has always been able to provide insightful questions and encourage my research to go in new directions.

Bob Levine has also been of enormous help in ensuring that my time in the PhD program was well-spent. He has provided long discussions on syntactic issues and has provided detailed, constructive, and useful feedback on topics outside his specialty, and for that I am both impressed and extremely grateful.

All of my professors at OSU have throughout my graduate student career afforded more many opportunities for exploration and discussion and have never let me forget that linguistics is an essential part of computational linguistics.

The OSU computational linguistics discussion group at, CLippers, has been invaluable in providing feedback on many different aspects of this thesis. I also thank the audiences at EACL-03, TLT-03, MCLC-04, MCLC-05, the NODALIDA-05 special

session on treebanks, and ACL-05, where portions of this work have been presented before.

I also thank the OSU Department of Linguistics and the Edward J. Ray Travel Award committee for the financial support that allowed me to travel to these conferences; and the OSU Graduate School and the OSU Center for Slavic Studies for providing the fellowships that supported my studies in general.

Finally, this thesis would not have been possible without the support of all my family and friends. They are too many to name here, but I will specifically point out a few. My cohort of Wes Collins, Robin Dodsworth, and Anton Rytting have been a delightful group to work with and to get to know.

And, of course, my wife Stephanie has been a blessing in letting me know when I needed to keep working and when I needed to rest. She has provided wonderful emotional support and has additionally lent me her statistical expertise. From simple calculations of confidence intervals to broader discussions of how best to analyze data, her skills and knowledge have been an asset to this dissertation.

VITA

March 28, 1978	Born – Peoria, IL
2000	BA, Linguistics, University of Illinois at Urbana-Champaign
2002	MA, Linguistics, The Ohio State University
2003-2004	Graduate Research and Teaching Associate, The Ohio State University

FIELDS OF STUDY

Major Field: Linguistics

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND MOTIVATION

In this thesis, we will pursue a method for detecting errors in annotated corpora, and as a first step we set out here the desiderata for this line of research. Minimally, there are the following considerations: 1) The annotation of corpora actually contains errors. Although effective error detection can prove if there are errors, we at least need indications that the search is not futile. 2) The presence of annotation errors is detrimental to the intended uses of the corpora. If errors are harmless, or even helpful, then again there is little point in isolating them and correcting them. 3) It is actually possible to find the errors in a systematic way. With annotated corpora as large as they currently are—in the millions and hundreds of millions of words—methods for detecting errors must be automatic and generally applicable.

That corpora are erroneous and that the errors are problematic is demonstrated in sections 1.1.2 and 1.2, respectively, and previous attempts at systematically finding errors are presented in section 1.3. Building on these attempts, this dissertation will develop a particularly useful method for both detecting and correcting errors in annotated corpora, based on inconsistencies in the annotation, i.e., recurrences of the same material with different annotations. First, however, we must look at the different kinds of annotated corpora and the range of uses for such corpora.

1.1 Gold standard annotated corpora

1.1.1 The use of annotated corpora

Annotated corpora are large bodies of text with linguistically-informative mark-up. As such, they provide training material for research on natural language processing algorithms and serve as a gold standard for evaluating the performance of such tools.

Corpora have been essential for training and testing algorithms in a wide range of areas, such as in tagging and morphological analysis (e.g., Brill, 1995a; Daelemans et al., 1996), parsing (e.g., Atwell, 1993; Briscoe, 1994; Ramshaw and Marcus, 1995; Collins, 1996), term and name identification (e.g., Bikel et al., 1999), word sense disambiguation (e.g., Brown et al., 1991; Gale et al., 1992; Segond et al., 1997), and anaphora resolution (e.g., Rocha, 1997; Mitkov et al., 1997). Annotated corpora are also used to develop human language technology applications such as machine translation (e.g., Berger et al., 1994), document classification (e.g., Ragas and Koster, 1998; Zavrel et al., 2000), or information retrieval (e.g., Riloff, 1993; Soderland et al., 1995), as well as in linguistic research (e.g., Wichmann, 1993; Sampson, 1996; Meurers, 2005).

The supervised models mostly used in statistical natural language processing which underlie many of these applications require training on corpora that are annotated with the particular linguistic properties intended to be learned. The nature of the annotations and the annotation schemes for written corpora include a wide range of linguistic information, such as morphological (e.g., Leech, 1997; Santorini, 1990; Sampson, 1995; Schiller et al., 1995), syntactic (e.g., Marcus et al., 1993; Hajič, 1998; Sampson, 1995; Skut et al., 1997), semantic (e.g., Kingsbury et al., 2002; Hajičová,

1998; Erk et al., 2003), and discourse (e.g., Allen and Core, 1996) distinctions. The information encoded in these annotations represents significant abstraction and generalization from the data and at present require manual correction or annotation to obtain annotated corpora of high-quality.

Each annotation layer is important for a variety of uses, including serving as input for processing of other annotation layers. For example, part-of-speech (POS) annotated text is used as input for syntactic processing, for practical applications such as information extraction, and for linguistic research making use of POS-based corpus queries. Annotated corpora are also essential as gold standards for testing the performance of human language technology, regardless of whether the model is statistical or rule-based in nature. Additionally, the linguistic use of corpora (e.g., Wichmann, 1993; Sampson, 1996; Meurers, 2005) often corresponds to using the annotations to search for a particularly theoretically-interesting pattern. Thus, for both computational and theoretical linguistic purposes, it is crucial that annotations be of a sufficiently high quality.

1.1.2 The nature of a gold standard corpus

As mentioned, reference corpora annotated with part-of-speech tags or syntactic labels, such as the British National Corpus (BNC) (Leech et al., 1994a), the Penn Treebank (Marcus et al., 1993), or the German NEGRA Treebank (Skut et al., 1997) play an important role for current work in computational linguistics, so great care has gone into developing such corpora. For example, the gold-standard POS-annotation for such large reference corpora is generally obtained using an automatic tagger to produce a first annotation, followed by human post-editing. While Sinclair (1992)

provides some arguments for prioritizing a fully automated analysis, human post-editing has been shown to significantly reduce the number of POS-annotation errors. Brants (2000a) discusses that a single human post-editor reduces the 3.3% error rate in the STTS annotation of the German NEGRA corpus produced by the TnT tagger to 1.2%. Baker (1997) also reports an improvement of around 2% for a similar experiment carried out for an English sample originally tagged with 96.95% accuracy by the CLAWS tagger. And Leech (1997) reports that manual post-editing and correction done for the 2-million word core corpus portion of the BNC, the BNC-sampler, reduced the approximate error rate of 1.7% for the automatically-obtained annotation to less than 0.3%.

Despite these gains, the presence of human annotators or correctors introduces another source of potential error: humans are prone to mistakes, fatigue, and misinterpretation of guidelines. Unlike a purely automatic system, human annotators can introduce inconsistencies: if a post-editing correction has been made once, there is no guarantee it will be made every time it applies across the corpus. Ratnaparkhi (1996) demonstrates this vividly: he runs a consistency check on the Wall Street Journal (WSJ) corpus by examining how certain words were labeled by different annotators. He finds biases for different parts of speech based on who the annotator was: the tagging distribution for a word changes when the annotator does.[1]

Corpora which have been in existence for some time and have undergone careful revisions (e.g., the Lancaster-Oslo-Bergen (LOB) corpus (Johansson, 1986; Garside et al., 1987) and the SUSANNE corpus (Sampson, 1995)) likely have low error rates,

[1]Because of these facts, Ratnaparkhi (1996) suggests training and testing on texts annotated by the same person. This admittedly does not alleviate problems of intra-annotator inconsistency, and does nothing to handle the problem of annotation which is consistently wrong.

although no error estimates are provided for these particular corpora. Likewise, the error rates for carefully-constructed and documented corpora are likely to contain relatively few errors. However, with 200 million words, even a carefully-constructed corpus like the Bank of English (Järvinen, 2003) is likely to contain errors. And the Penn Treebank, one of the most commonly-used corpora, has an estimated error rate of 3% (Marcus et al., 1993). It is clear, then, that "gold standard" corpora contain errors in their annotation.

1.2 Errors and their effects

Even if there are errors, it might be the case that they are harmless, or possibly even helpful, for the intended uses. We will show, however, that the proliferation of errors in corpora has a negative impact on natural language processing technologies which train and test on them. To do so, let us first demonstrate why and how errors might be problematic. As a concrete example of the kind of system where the validity of a corpus is important for learning, induced grammars (Charniak, 1996) are automatically derived from treebanks. They rely on the corpus annotation to form grammar rules. If many of these rules are ill-formed due to errors in the input, this can result in extra rules, increasing the size of the set of grammar rules. Detecting erroneous annotation can potentially reduce the number of rules and thereby improve the efficiency of a parser using the grammar. Furthermore, as we will show in section 3.3, removing erroneous annotation can also improve the quality of the grammar.

To take another example of the potential effect of errors, consider the case where the original data is equally correct and incorrect for a particular construction, and an algorithm has to separate the signal from the noise. Focusing in on part-of-speech tagging, for example, Dickinson and Meurers (2003a) report that in the Wall Street Journal (WSJ) corpus, a part of the Penn Treebank 3 project (Marcus et al., 1993), in the bigram *Salomon Brothers*, *Brothers* is tagged 42 times as NNP and 30 times as NNPS,[2] and this is only one of many NNP/NNPS confusions. Both taggings have approximately the same weight, so it is not clear what the appropriate rule is to learn from this situation.

1.2.1 Effects on classifiers

Training data errors Errors in the training data can have a big impact on classifiers. Ali and Pazzani (1996) show that when they introduced "class noise" to various data sets, their ensemble classifiers performed less well than single model classifiers. Class noise is noise in the labeling of an instance in the data (as opposed to noise in the attributes, or features, pertinent to an instance); thus, errors in the part of speech in a corpus are instances of class noise. As Zhu and Wu (2004) point out, one relevant source of class noise is that of contradictory examples, i.e., "The same examples appear more than once and are labeled with different classifications." This case is of particular interest for the approach we will develop.

Zhu and Wu (2004) additionally note that in a series of experiments with different classifiers, "the classification accuracies decline almost linearly with the increase of the noise level" for almost all datasets they tested. Quinlan (1986) also showed that

[2]The tags have the following meanings: NNP = singular proper noun, NNPS = plural proper noun. See section 2.2.3 for more details on this distinction.

"[d]estroying class information [adding errors to the data] produces a linear increase in error so that, when all class information is noise, the resulting decision tree classifies objects entirely randomly."

In the realm of part-of-speech tagging, the presence of errors in the training data used has been well-noted. Brill and Pop (1999) note that extracting a lexicon from a tagged corpus for their tagger results in a noisy lexicon. For instance, in the WSJ (Marcus et al., 1993), the word *the* is tagged six different ways, but the distribution is highly skewed due to errors: DT occurs 57,973 times, NNP occurs 5 times, JJ occurs 3 times, and VBP, NN|DT, and NN occur once each.[3] By filtering out a certain percentage of infrequently-occurring tags, Brill and Pop (1999) improve the accuracy of one tagger from 81.3% to 95.9% and another from 82.6% to 90.6%. These dramatic increases in quality are likely why Schmid (1997) also removed tags from his lexicon, filtering out tags which occurred less than 1% of the time for a word.

We should mention that a problem with such an approach for handling errors is that it potentially removes true linguistic information along with erroneous data since rare examples do occur (e.g., see the discussion of Zipf's law in chapter one of Manning and Schütze (1999)). This is also the case for syntactic annotation, where approaches to grammar induction like Gaizauskas (1995) and Krotov et al. (1998) obtain reasonable results by thresholding and removing non-frequent rules. One might be able to achieve good results with such techniques, but at some point, the coverage of the grammar, or model, will be limited because rare patterns will have been eliminated. Indeed, Daelemans et al. (1999) performed experiments on

[3]The tags have the following meanings: DT = determiner, NNP = singular proper noun, JJ = adjective, VBP = non-3rd person singular present verb, NN = singular or mass noun. A bar (|) indicates uncertainty between two tags.

four different NLP tasks[4] and found that removing atypical instances from training was detrimental to learning. As they state, "for language learning tasks, it is very difficult to discriminate between noise on the one hand, and valid exceptions and sub-regularities that are important for reaching good accuracy on the other hand." With errors in the training data, we cannot sufficiently answer the question of what the methods are doing when filtering: are they removing noise? Or are they betting that correct rules will never appear?

Ratnaparkhi (1996) examines the effects of errors on his maximum entropy tagger in detail, specifically errors which are due to inconsistencies in the corpus. Since the maximum entropy tagger maintains a close fit with the training data, it is likely that any noise in the data will be modeled by the tagger. Indeed, many words with variation in tagging, such as *that* and *about*, are found to be problematic for the tagger. Ratnaparkhi (1996) lexicalizes the tagger, i.e., allows the context to consist of words and not just tags, to deal with these problematic words. Because this lexicalization provides more information about these words, it should result in an improvement in tagging accuracy. The tagging, however, gets no better and even becomes worse for some words, due to inconsistencies found in the training data, i.e., no single consistent pattern can be deduced which is correct. Although it depends on the underlying model, the experiments in Ratnaparkhi (1996) show that inconsistent training data can lead to worse tagger performance.

[4]The four tasks are: grapheme-phoneme conversion, part-of-speech tagging, prepositional phrase attachment, and base noun phrase chunking. See references in Daelemans et al. (1999) for similar work on other NLP tasks, such as pseudo-word sense disambiguation.

This fact is confirmed by both Kübler and Wagner (2000) and Květǒn and Oliva (2002). Kübler and Wagner (2000) distinguish four kinds of errors (described in section 1.2.2) which they introduced into a corpus. They show that for a training corpus with any one of those kinds of errors, tagger performance is significantly worse as compared to a corpus without such introduced errors. For example, a tagger trained on an unchanged 24,082 sentences obtained a 1.95% error rate on the test corpus. For a training set with potentially ambiguous words ("AMBI") being erroneously tagged, the error rate is 2.86% on the same test set; other training sets result in even higher error rates.

Květǒn and Oliva (2002) demonstrate that with their method of finding errors in a corpus (described in section 1.3.3), they are able to improve tagger results. One experiment consisted of training and testing on their cleaned data, resulting in a lowering of the error rate to 3.07%, as compared to an error rate of 3.14% when training and testing on the original data. More enlightening is the fact that training on the uncorrected version of the corpus, but testing on the clean version, resulted in an error rate of 3.41%, significantly higher than the 3.07% error rate obtained when the training data was also cleaned. By keeping all other factors equal, this disparity shows that the quality of the training data can significantly impact the performance of a tagger.

Testing data errors While training data errors affect the classification model, testing data errors are vitally important in classifier evaluation. This was just shown in Květǒn and Oliva (2002): keeping the training data constant as the original data, the cleaned testing data has an error rate of 3.41%, while the original testing data

has a significantly lower error rate, that of 3.14%. This again highlights the negative effect of corrupted training data, but it also shows that the nature of the testing data has a profound impact on what error rates will be reported by classifiers.

The conclusion that the quality of the testing data affects precision and error rates is backed up by other researchers. Working with the written half of the BNC-sampler (Leech, 1997), van Halteren (2000) examines cases where his WPDV[5] tagger (van Halteren et al., 2001) disagrees with the BNC annotation; he reports that in 13.6% of the cases, the cause is an error in the BNC annotation. The percentage of disagreement caused by BNC errors rises to 20.5% for a tagger trained on the entire corpus. van Halteren (2000) goes on to show that many of the errors fall into various inconsistency classes, such as inconsistent choices made between adjective (JJ) and verbal gerund (VVG) for noun-modifying words ending in *ing*. Blaheta (2002) likewise shows that for his function tag assignment algorithm, 18% of the disagreements with the WSJ benchmark are treebank errors, and 13% more are not covered by the guidelines.

Turning to cross-corpus comparison, van Halteren et al. (2001) compare the results of taggers trained and tested on the WSJ corpus (Marcus et al., 1993) with results from training and testing on the LOB corpus (Johansson, 1986) and find that the results for the WSJ are significantly worse. By examining a tenth of the test sets of both corpora, van Halteren et al. (2001) discover that these lower accuracy figures are due to inconsistencies in the WSJ corpus. They found that 44% of the errors for their best tagging system were caused by "inconsistently handled cases." Thus,

[5]WPDV = Weighted Probability Distribution Voting algorithm.

both van Halteren (2000) and van Halteren et al. (2001) show that benchmark errors, especially errors which are inconsistencies, have a profound impact on the accuracy figures reported for classifiers.

To emphasize this point, building on a mention of tagging errors caused by noise in the data in Marquez and Padro (1997), Padro and Marquez (1998) mathematically demonstrate that tagger evaluation is questionable when the testing data contains errors. Depending on the corpus error rate, the true accuracy of a classifier could be much better or worse than reported. This is, of course, a major problem when state-of-the-art taggers claim superiority based on slightly better precision rates.

Given a reported tagger accuracy rate K and a corpus error rate C, we want to find the true tagger accuracy rate, x. Padro and Marquez (1998) show that x is between x_{min} and x_{max}, as given in (1), where p is the probability that the tagger and the corpus are both wrong in the same way.

(1) a. $x_{min} = K - Cp$

b. $x_{max} = \left\{ \begin{array}{ll} K + C & \text{if } K \leq 1 - C \\ 1 - \frac{K+C-1}{p} & \text{if } K \geq 1 - C \end{array} \right\}$

Thus, x_{min} is calculated by subtracting from the reported accuracy rate the percentage of the corpus where both the corpus and the tagger are wrong in the same way. This accuracy (x_{min}) number accounts for only those situations where the corpus and the tagger are both right; situations where the tagger is right but the corpus is wrong are not considered, which is why x_{max} is needed. The x_{max} value is calculated by one of two methods. In the first case, we add the corpus error rate (C) to the tagger accuracy rate (K), i.e. assume every time the corpus is wrong, the tagger is right. In the second, we calculate similarly, but factor out the times when the corpus and the tagger are both wrong (p).

Coming up with reasonable estimates for p affects what the true tagger accuracy estimates are, but we know that $0 \leq p \leq 1$, since p is a probability. So, if a tagger has a 93% reported accuracy on a corpus with 3% errors, the real tagger accuracy x is between the ranges of $[0.93, 0.96]$ ($p = 0$) and $[0.90, 0.96]$ ($p = 1$).[6]

Padro and Marquez (1998) show how, given narrower, more reasonable estimates for p, the true tagger accuracy of two taggers can overlap. The example they use is of taggers with the following rates for ambiguous words: $K_1 = 91.35\%$ and $K_2 = 92.82\%$. Both were tested on the WSJ, with an estimated error rate of 3%. The range of true accuracy rates, x_i, is shown in (2), where the important thing to note is how they overlap (e.g. between 92.82 and 94.05 in the case of $p_i = 0.4$). Thus, because of errors in the testing data, it is not clear which tagger is actually better.

(2)

	$p_i = 0.4$	$p_i = 1$
$K_1 = 91.35$	$x_1 \in [91.35, 94.05]$	$x_1 \in [90.75, 93.99]$
$K_2 = 92.82$	$x_2 \in [92.82, 95.60]$	$x_2 \in [92.22, 95.55]$

1.2.2 Types of errors

As mentioned above, van Halteren (2000) and van Halteren et al. (2001) found classes of inconsistencies in their work on POS annotation. That is, there were recurring patterns of variation between certain labels for items that were the same (e.g., the same word). Ratnaparkhi (1996) also distinguishes inconsistent labeling as a particularly problematic kind of error. As described in chapters 2 and 3, Dickinson and Meurers (2003a,b) confirm the presence of inconsistencies in both POS-annotated and syntactically-annotated corpora. Inconsistencies are clearly a prevalent kind of error,

[6] As a quick shorthand, then, one can simply add or subtract the corpus error rate from the tagger accuracy rate.

but in distinguishing different kinds of errors we can make other classifications. Much work has gone into classifying the types of corpus errors, with the hope that this will aid in corpus error detection and error correction.

Blaheta (2002) provides a categorization of errors into three classes, based on whether an error is detectable and automatically correctable (type A),[7] fixable but needs human intervention to correct (type B), or is not covered by the annotation guidelines (type C).[8] Using this typology, human inspection and hand-written rules are used for error detection and correction.

The distinction between type A and type B errors is also made in Oliva (2001), who writes rules to detect errors in a corpus, noting that some rules need human assistance to correct, while others can be automatically corrected. Leech et al. (1994b) also report that post-editors for the BNC-sampler were instructed to focus on certain tag combinations, highlighting the fact that certain recurring tag patterns were problematic and needed human post-editors to determine the correct analysis.

As an example of an error which needs human correction, Marquez and Padro (1997) discuss the distinction between adjective (JJ) and verbal gerund (VBG) in the Penn Treebank which occurs in "sentences with no structural differences." Likewise, distinctions between adjectives (JJ) and nouns (NN) often come down to semantic

[7]A distinction not made in Blaheta (2002) is between automatically detectable errors which need manual correction and errors which require even manual detection. Since manual detection would be highly labor-intensive, we focus on automatic detection only.

[8]Blaheta (2002) refers to type C errors as *systematic inconsistencies*, but we will not use that term to avoid confusion with the use of *inconsistency* to simply mean that tagging practices were not consistently applied over the whole corpus, regardless of if the source of inconsistency was unclear/missing guidelines, misapplication of guidelines, or general human error. It should be noted that Ule and Simov (2004) also make a distinction between *violations of annotation guidelines* and *violations of language principles not covered by the annotation guidelines*; we will say more on annotation guidelines in section 1.3.2.

differences (Marquez and Padro, 1997). In this case, the reason for a non-automatic method of correction is due to the design of the tagset, which is discussed more in section 2.2.3.

Errors like this arise because of ambiguous words, and one way of defining error types is to look more specifically at problematic ambiguities. Volk and Schneider (1998), for example, view error types for classifier output as ⟨corpus label, tagger label⟩ pairs. A set of these pairs for a corpus can give a good indication of which tagging distinctions are difficult for a tagger to maintain over the entire corpus. Likewise, the development of the BNC included the use of *portmanteau tags*, a tag indicating a disjunction of possible tags (Eyes and Leech, 1992; Leech et al., 1994b). These were added after an initial tagger had already been run, based on decisions known to be problematic for their tagger.

As van Halteren (2000) and van Halteren et al. (2001) point out, frequent occurrences of difficult tagging decisions are likely to indicate an error in the benchmark corpus. In fact, van Halteren (2000), van Halteren et al. (2001), and Dickinson and Meurers (2003a) all show that we can define an error type as a ⟨corpus label 1, corpus label 2⟩ pair, or as a set of corpus labels. Certain label variations can be problematic regardless of context, or they can be problematic in a given context (Dickinson and Meurers, 2003a). As an example of the former, van Halteren et al. (2001) discuss the inconsistent use of the labels NNP (proper noun) and NNPS (plural proper noun) in the WSJ: e.g., *Securities* (146 NNP vs. 160 NNPS).[9] The latter case of two labels for the same word/string in the same context is discussed in chapter 2. Note that the

[9]This problematic tagging distinction was independently discovered in Dickinson and Meurers (2003a).

NNP/NNPS distinction cannot be resolved by the definitions given in the guidelines and so falls into what Blaheta (2002) defines as type C errors. In general, though, these errors based on ambiguities/variations can be any one of type A, B, or C.

What this research is pointing to is a definition of an ambiguity class. An ambiguity class (e.g., Cutting et al., 1992) is a set of words with the same ambiguities. Words could be grouped into classes based on the *amount* of ambiguity they manifest (akin to classes with the same polysemy levels in Kilgarriff (1998)), but given that it is the particular distinctions which have a strong impact on tagging, the notion of an ambiguity class is more useful for defining errors. Ambiguities can lead to variation, and variation between two or more labels for a given word or string seems to be a common problem.

Do we know, however, that these so-called variation errors specifically affect tagger performance? Kübler and Wagner (2000) define four error types (alternatively viewed as two binary-valued types): 1) the wrong tag is a possible tag for the word ("AMBI"), 2) the wrong tag is not a possible tag for the word ("NOT-AMBI"), 3) the major part-of-speech category is correct, but not the more fine-grained distinctions ("MCAT"), and 4) the major part-of-speech category is incorrect ("NOT-MCAT"). This distinction helps pinpoint the most problematic errors: NOT-AMBI and NOT-MCAT cause more degradation in the accuracy of the tagger than their counterparts, presumably because they provide data which can never be true. That is, errors where the wrong tag is a possible tag for that word (AMBI) are actually less problematic than errors where the tag is not even possible (NOT-AMBI). And, due to lexica, NOT-AMBI (and NOT-MCAT) errors are easier to detect (Kübler and Wagner, 2000).

However, this does not tell us how much of an impact variation errors have on tagger performance. Although ambiguities often lead to variations, AMBI errors and variation errors are not the same thing; variation between two labels can in principle be a NOT-AMBI error, as the example of *the* in the WSJ mentioned before demonstrates. In that case, *the* has variation between six different tags, five of which are not possible tags for it. There is a good deal of overlap between AMBI errors and variation errors, though, and so it is important to note that AMBI errors are still a problem: as mentioned before, training on them increased the error rate on one corpus from 1.95% to 2.86%.

Also noteworthy is that Kübler and Wagner (2000) gathered their training data of erroneous corpora by running a poorly-trained tagger on the corpus and keeping the appropriate kind of errors, as compared to the original corpus. Even though this does give a good indication of how different kinds of errors are problematic, it gives no indication of how many errors are actually present in the corpus. So, practically speaking, we cannot tell which errors are most problematic from this work, and other work (Ratnaparkhi, 1996; van Halteren, 2000; van Halteren et al., 2001; Dickinson and Meurers, 2003a,b) indicates that errors stemming from ambiguous words are indeed problematic.

1.3 The elimination of errors

We have established the existence of annotation errors in corpora and shown that they are a hindrance to both training and testing natural language technologies. We now turn to methods which have been proposed to prevent or remove errors from

a corpus. In addition to confirming the preponderance of errors in corpora, the following will show that there are systematic ways to find and fix certain errors, and our approach will build on this groundwork.

1.3.1 Manual and semi-automatic error detection and correction

As will be more fully discussed in the *problematic cases* part of section 2.2.3, the first step to preventing errors comes by having a well-defined set of guidelines. Indeed, as Wynne (1996) mentions, "Probably the best way to ensure that the tagging of a large body of text remains as consistent as possible is to build up a 'caselaw' of ... tagging decision[s] as they are made." This helps for constructing an initial corpus and for defining standards of annotation, but we are here interested in improving the state of an already-annotated corpus.

Even though human post-editing can significantly improve the corpus (Baker, 1997; Leech, 1997; Brants, 2000a), for very large corpora—often in the range of 100-200 million words—it is in practice impossible to detect and correct all errors manually. A small subset of the corpus, however, can be examined and fixed (Eyes and Leech, 1992; Järvinen, 2003). Problematic decisions can be identified and searched for in the rest of the corpus. Similarly, to build a sense-tagged corpus, Kilgarriff (1998) suggests sampling from the word types in the corpus and from those word types sampling their corpus instances. From these samples, one can find what kind of tagging issues arise.

Skut et al. (1997) mention using previously-annotated data to train their classifiers for further annotation on unseen data in constructing a corpus. This idea can be adapted to fixing the annotation of seen data by training on cleaned data in order

to compare the learned model to the parts of the corpus which have not yet been manually checked. In this way, decisions made in the carefully cleaned part of the corpus can be extended to the rest of the corpus. This line of thinking is similar to active learning (e.g., Cohn et al., 1994; Dagan and Engelson, 1995; Thompson et al., 1999). In the case of active learning, one uses judgments on labeled data to select the next data to label, and this is done by finding new data which is sufficiently different from previously-seen data. Here, we are using cleaned labeled data to select corpus positions which are sufficiently similar yet do not match in annotation.

In an effort to generalize a judgment made one time to the rest of the corpus, Wallis (2003) argues for moving from a sentence-by-sentence correction approach (*longitudinal* correction) to what he calls *transverse* correction: correction on a construction-by-construction basis across the whole corpus. In this way, problematic constructions can be identified and treated in a consistent fashion. Likewise, Oliva (2001) and Blaheta (2002) manually write rules to identify errors across the whole corpus. In all these cases, although the searching is done automatically, a human must specify what patterns to search for, and there is no way to ensure that all errors, or all types of errors, will be found.

Hinrichs et al. (2000) also mention running "automatic consistency checks" on the Tübingen treebanks. A manually-selected type of annotation is searched for, and the non-majority annotations are flagged as possible errors. Kordoni (2003) describes this for the Verbmobil project as follows:

> In brief, each time it was called the program was given the type of phrase to be checked, and it extracted all strings of words in the corpus which were annotated with the specific phrasal type. If the annotations differed, the program selected the annotation which was used in most cases and output the deviant cases for further inspection by the annotator.

The program described in Kaljurand (2004) can be seen as one way to implement this and similar consistency checks by searching for strings of words, POS-tags, or syntactic function labels which receive different annotations. Similar to these approaches, but driven by the data, the proposal we will make in chapter 2 presents an automatic way to specify problematic constructions.

Alternatively, one can specify which constructions are allowed in the corpus. A technique used by Bond et al. (2004) and Oepen et al. (2004) to ensure consistency in a corpus with syntactic annotation is to build a treebank in parallel with a grammar. The grammar feeds into the annotation, and the treebank provides feedback for the grammar. The result of this is that every treebank annotation must have a well-formed parse; otherwise, the grammar could not have suggested it. This approach, while useful and theoretically attractive, requires a significant amount of resources to construct the grammar, ensure its robustness, and to manually disambiguate the parser output.

1.3.2 Interannotator agreement

Another technique to ensure that corpora are closer to what humans view as the true analysis compares the analyses of multiple annotators for the same corpus positions. Kilgarriff (1998) outlines what he views as the requirements for producing a gold standard corpus: 1) require multiple people to label [the same corpus positions], 2) calculate the rate of interannotator agreement, and 3) determine if the agreement rate is high enough.

Running interannotator experiments produces varied results for different researchers. As a method of post-editing, Marcus et al. (1993) report agreement rates of 96.5%. This seems to support claims in Church (1992) of a 97% upper bound to tagging, i.e. there is a 3% residue of the corpus for which human annotators cannot agree on the right analysis.[10] However, Baker (1997) reports interannotator consistency rates of 98.8% on the BNC-sampler, and Brants (2000a) mentions agreement rates of 98.57% for POS-tagging in the NEGRA corpus.

The agreement rate does not tell the whole story, and so Kilgarriff (1998) states that discrepancies in labeling need to be examined to see if the disagreement between annotators is a real disagreement or an accident. If it is an accident, the correct label can be easily agreed upon. If the disagreement is real, the source of disagreement can probably be attributed to the annotation scheme; the variety of annotation schemes is what causes such a variety of results above.

As Kilgarriff (1998) points out, if the disagreement is real, the cause is either a true ambiguity in the data or a poor definition of the labels, i.e., poor guidelines. Both of these can be dealt with by specifying a clearer annotation scheme (Baker, 1997). In fact, Voutilainen and Järvinen (1995) show that 100% interannotator agreement is possible for both part-of-speech and syntactic annotation when difficult distinctions are eliminated. Interannotator agreement rates are thus highly determined by the annotation scheme and guidelines, and there is some research (e.g. Sampson and Babarczy, 2003) on defining annotation schemes which do not require human annotators to make distinctions that cannot be made reliably.

[10]Baker (1997) reports that Church has stated (in 1996) that there is a 5% residue.

Corpus development work like Brants et al. (2002) uses interannotator disagreements as indicators of where the annotation scheme needs improvement. And work such as Brants (2000a) shows that testing interannotator agreement can pinpoint problematic tagging decisions, highlighting where either the scheme or the guidelines need to be revised.

We have mostly discussed interannotator agreement for positional annotations; testing interannotator agreement for structural and other complex forms of annotation requires more work to determine how similar the analyses are. Defining similarity of complex annotations is non-trivial, as shown by the difficulties surrounding the comparison of parser output (Carroll et al., 2002). Brants and Skut (1998) discuss automating the comparison of two syntactic annotations of a sentence in interannotator agreement testing. Based on Calder (1997), they propose to select the nodes from one annotation and search for a node with the same terminal yield in the second annotation. The process is driven by the selection of non-terminals from one of the annotations and thus is asymmetric in nature. As a result, the full comparison involves running the process in both directions, selecting from the first annotation and comparing with the second as well as the other way around. We will return to the issue of comparing complex annotations in chapters 3 and 4.

Indeed, interannotator agreement rates for syntactic annotation tend to be lower than those reported for POS annotation, as given above. Brants (2000a) reports an F-score agreement of 92.43%. However, Brants and Skut (1998) show that, after discussion, annotators are able to obtain identical nodes and labels 97.4% of the time. And, as stated above, Voutilainen and Järvinen (1995) are able to obtain near 100%

agreement after negotiations.[11] Thus, we can see that checking interannotator agreement can prevent errors from appearing in corpora, as well as help define annotation standards. Given that annotation guidelines are the ultimate authority in a corpus project (i.e. the benchmark by which "gold standard" corpora can be said to have errors), it is vital that annotation standards are as complete and coherent as possible and include a rich documentation of case law for difficult cases. As we will see in sections 1.3.3 and 2.2.3, error detection research can also provide feedback to revising annotation guidelines.

1.3.3 Error detection research

Nelson et al. (2002) report for the International Corpus of English (ICE) that "[e]rror correction has continued on an *ad hoc* basis." This kind of passive approach to corpus correction, supplemented or not with other methods, is undoubtedly practiced for every corpus: correct errors as they are found, no matter how they are found. However, a body of recent research is trying to systematically find errors in corpora. In this way, not only can more errors be found, but systemic causes of the errors (e.g., inadequate guidelines) can be rooted out and prevented in future work. A common theme in this error detection work is that errors are items which do not fit a consistent pattern; they deviate from the "norm" in some way. How consistency and deviation are defined is what separates the methods from one another.

[11] As another example, Hinrichs et al. (2000) claim that they reduced incorrect/inconsistent error rates in the Verbmobil corpus by deciding to remove predicate-argument structure information from the annotation scheme for tree configurations; however, they provide no estimate of error rate or interannotator agreement rate.

For example, Eskin (2000) discusses how to use a sparse Markov transducer as a method for what he calls anomaly detection. The notion of an anomaly essentially refers to a rare local tag pattern, which is found by using a "mixture model," a statistical technique used to find outliers. The method flags 7055 anomalies for the Penn Treebank, about 44% of which hand inspection shows to be errors.[12] The low precision of this method for detecting errors means that the repair process has to deal with a high number of false positives from the detection stage. Furthermore, Eskin notes that "if there are inconsistencies between annotators, the method would not detect the errors because the errors would be manifested over a significant portion of the corpus." As we saw in section 1.2.2, though, inconsistencies are a prominent issue to deal with in cleaning corpora.

Similar to Eskin (2000), Nakagawa and Matsumoto (2002) also search for exceptional elements. Their method uses support vector machines (SVMs). The SVMs provide weights for each corpus position; the larger the weight, the more difficulties the SVM had assigning a label to it. This gives a first set of potential error candidates.[13] The second step is to find similar examples in the corpus, based on a window of two words and tags, as well as affix information in the focus word. More specifically, they search for examples with the smallest distance metric from the heavily-weighted word but having a different label. That is, they look for something which is the same but not labeled as such. Of 1740 positions above a certain threshold in the WSJ, they sampled the first 200 in the corpus and found 199 to be errors (precision =

[12]The probability for the 25% most likely errors increases to 69%.

[13]This, of course, ignores errors made by the SVM where it is highly confident in its label but is nonetheless wrong.

99.5%),[14] and on two Japanese corpora, they obtained precision rates of 93.8% and 100%. Artificially introducing random changes to the tags led them to estimate a recall of up to 18.5% (but no precision figures are given for the recall experiments).

Ule and Simov (2004) employ a similar idea to Eskin (2000) and Nakagawa and Matsumoto (2002) for treebank error detection. Their idea is that by using a method called Directed Treebank Refinement (DTR) (Ule, 2003), they can find unexpected tree productions. To find the most unexpected tree node—what they call a focus node (f)—for each iteration of the method, Ule and Simov (2004) use information about the context c (i.e., the type of parent node) and the production type p (i.e., the children). An event (c, f, p) which is unexpected, based on the χ^2 metric, is deemed likely to be an error. Each node is classified into one of four classes: error in f, error in c, error in p, or no error, and errors are sorted into a list, with the more likely errors appearing first. Ule and Simov are somewhat successful in finding errors in a treebank of 580 sentences: of the first 27 error candidates in a hand-checked test corpus, 11 were errors and five were the result of unclear guidelines. They also tested their approach on a corpus into which they had introduced errors by switching node labels, which allowed them to test not only precision but recall. Results varied by corpus and by the number of errors introduced, but the best results were of 72% precision and 72% recall. Admittedly, however, this does not test performance on errors due to "misinterpreted larger structures."

The methods we have discussed so far rely on the fact that errors correspond to unexpected events, i.e., rare events. However, being rare does not necessarily mean that something is incorrect, as shown in the well-known Zipf's law (Manning and

[14]Since they took the first 200 items and did not randomly sample, we do not know the precision of the next 1540 items.

Schütze, 1999, ch. 1), where rare linguistic events are predicted to occur. In practice, these rare but correct events can play a crucial role in NLP technology (Daelemans et al., 1999). In addition to the methods mentioned so far, therefore, other error detection methods need to be developed.

Taking the idea of identifying instances which probably should not happen and making it stronger, Květoň and Oliva (2002) employ the notion of an invalid bigram to locate corpus positions with annotation errors. An invalid bigram is a POS-tag sequence that cannot occur in a corpus, and the set of invalid bigrams is derived from the set of possible bigrams occurring in a hand-cleaned sub-corpus, as well as from linguistic intuition. Using this method, Květoň and Oliva (2002) report finding 2661 errors in the NEGRA corpus (containing 396,309 tokens).[15]

Since errors in annotation can affect the performance of NLP systems, some researchers have used the failure of systems to help locate errors. For example, Hirakawa et al. (2000) and Müller and Ule (2002) are two approaches which use the POS annotation as input for syntactic processing—a full syntactic analysis in the former and a shallow topological field parse in the latter case—and single out those sentences for which the syntactic processing does not provide the expected result. Thus, the syntactic analysis is able to revise the POS-annotation.[16] The disadvantage of such a design for error detection is that these approaches require a sophisticated, language-specific grammar and a robust syntactic processing regime so that the failure of an analysis can confidently be attributed to an error in the input and not an error in the grammar or the processor.

[15]Errors were found (semi-)automatically, but correction of the 2661 errors was assumedly done manually.

[16]These techniques are related to the general field of correcting errors while POS-tagging (e.g., Elworthy, 1994; Yao et al., 2002).

Similarly, Ma et al. (2001) detect errors as a part of their POS tagger. They use modular neural networks; assuming n part of speech tags, and thus an n-way decision in deciding a tag for a word, they break the n-way tagging problem (n = number of tags) into approximately $\binom{n}{2}$ two-class problems.[17] Each module only deals with the smaller problem of a choice between two tags; thus, the idea is that if a module does not converge with an answer, the problem is probably not with the module, but might instead be attributable to a problem in the data. The non-convergence of the modules is usually caused by inconsistent data. Specifically, for the same given context in a corpus, a word might have two different labels, where the context is defined in their experiment by a window of two tags and the current word. They report that out of 97 pairs of contradictory learning data found by their method, 94 had an error, making for a precision of 96.9%.

Independent of the underlying model, POS taggers can be used more generally to find errors. As mentioned in section 1.2.1, van Halteren (2000) determines that benchmark errors are a problem in the evaluation of taggers, by showing that variation in annotation can indicate an annotation error. Using the idea that automatic taggers are designed to detect "consistent behavior in order to replicate it," places where the automatic tagger and the original annotation disagree are deemed likely to be inconsistencies in the original annotation.[18] This idea is successful in locating a number of potential problem areas, but van Halteren concludes that checking 6326 areas of disagreement only unearths 1296 errors, making error detection precision

[17]There are approximately and not exactly $\binom{n}{2}$ two-class problems because some part-of-speech decisions are more complex and require further decomposition into smaller problems.

[18]Abney et al. (1999) suggest a related idea based on using the importance weights that a boosting algorithm employed for tagging assigns to training examples; but they do not explore and evaluate such a method.

rather low (20.49%). However, since one tagger's disagreement with the corpus can indicate a potential error, it could be even more informative to use the disagreements of multiple taggers. Places in the corpus which are erroneous and inconsistent will be difficult for most taggers to accurately assess and may result in disagreement between taggers. Therefore, van Halteren (2000) proposes a two-phase method for detecting errors: 1) use tagger disagreements to pinpoint contexts of inconsistency, such as an *-ed* word being confused between an adjective and a past participle verb, and 2) examine instances of those contexts in the full corpus. This method, however, is not completely tested in van Halteren (2000); van Halteren et al. (2001), though, show that a combination tagger can point to even more errors—44% of the "errors" for a combination tagger on the WSJ were in actuality benchmark errors.

Summing up, the body of error detection research indicates that there are several ways to rephrase the error detection question, revolving around finding the contexts where labeling decisions are difficult to make. One way of viewing the task is to detect a spot where something has gone wrong. Another view, as just exemplified in van Halteren (2000), is to detect spots which are the "same" yet have different annotations. That is, we can view error detection as a task of finding, not necessarily where something has gone wrong, but where some element does not fit with the evidence elsewhere in the corpus.

1.4 The current proposal

This dissertation expands on the previous error detection research, especially van Halteren (2000), by looking for labelings which are inconsistent with other parts of the corpus. The essential idea is that identical strings with identical context but found in different parts of the corpus should be annotated consistently.

In addition to this error detection, we also want to correct the errors we find. As pointed out in various works (cf. Oliva, 2001; Blaheta, 2002), the task of correcting annotation can be viewed as consisting of two steps: i) detecting which corpus positions are incorrectly annotated, and ii) finding the correct label for those positions.

The first step, *detection*, considers each corpus position and classifies the label of that position as correct or incorrect. Given that this task involves each corpus position, only a fully automatic detection method is feasible for a large corpus.

The second step, *repair*, considers those positions marked as errors and determines the correct tag. For POS annotation, we can take the performance of current automatic taggers as baseline for the quality of the "gold-standard" annotation we intend to correct. For English we can assume that repair needs to consider less than 3% of the number of corpus positions.[19] This makes automation of this second step less critical, as long as the error detection step has a high precision (which is relevant since the repair step also needs to deal with false positives from detection).

[19]Typical reported error rates for taggers are around 3%. The assumption is that hand-corrected corpora will not be worse than automatically-tagged corpora.

Chapters 2 through 4 address the first issue, detecting errors, deferring correction to chapters 5 and 6. We focus on detecting errors automatically and with high precision.[20] To do so, we propose a method relying on internal corpus variation. The starting point of our approach, that variation in annotation can indicate an annotation error, essentially is also the starting point of the approach to annotation error detection of van Halteren (2000), as described in section 1.3.3.

This notion is the basis of our error detection method, which we call variation n-gram method. We describe the variation n-gram error detection method in detail for various levels of annotation in the following chapters. In chapter 2, the variation n-gram method is outlined for detecting errors in positional annotation, as exemplified by part-of-speech annotation. An efficient algorithm and results for several corpora are given. The method can be extended to more complex forms of annotation, and so the necessary extensions for detecting errors in structural annotation are presented in chapter 3, using syntactic annotation as the example. The method has to be broken down into several runs in order to efficiently calculate recurring strings of all relevant lengths. Results which validate the usefulness of the method are also given, as well as the results of a related approach based on inconsistent local trees. The approach to structural annotation in chapter 3 assumes contiguous units of data and so in chapter 4, the method is extended to cover annotation containing discontinuous elements, focusing on treebanks with discontinuous constituents. The linguistic motivation behind the method for discontinuous constituents is virtually

[20]Recall is less relevant in our context since eliminating any substantial number of errors from a "gold-standard" is a worthwhile enterprise.

the same as with continuous constituents, but several techniques are needed to make an efficient extension, all of which are elucidated in chapter 4. Again, the practical benefit of using such a method is there displayed.

Although hand-correction of a corpus should be feasible once errors have been accurately detected by the variation n-gram method, automatic and semi-automatic methods can speed up the process and ensure a new level of consistency in the corpus. In chapters 5 and 6, we turn to applying various classification techniques to the task of correcting a corpus, based on the results of the variation n-gram detection method. For this work, we focus on part-of-speech annotation. Using various off-the-shelf tagging techniques—Hidden Markov Models, memory-based learning, and decision-tree techniques—chapter 5 attempts to take automatic correction (with classifiers) as far as possible. Although we discover new methods and insights into the general process of part-of-speech tagging, lingering problems remain for the task of automatic correction of corpus annotation errors. Thus, in chapter 6, we integrate automatic correction methods into a broader picture of error correction. We automatically classify errors into groups with differing levels of confidence: some errors can be automatically corrected with a high degree of confidence, while others need human intervention.

CHAPTER 2

POS ANNOTATION

2.1 Introduction

This chapter will introduce a general error detection method for corpus annotation, the so-called *variation n-gram method*, by showing how it can be applied to part-of-speech (POS) annotation. Although, following Dickinson and Meurers (2003a), it is here illustrated for POS annotation, it is generally applicable for any positional annotation—i.e., annotation where each position has a label—since it is simply based on inconsistencies in the annotation of recurring strings. The method presented in this chapter assumes a one-to-one relationship between a token and an annotation label, which is true for POS annotation. However, the method can be used as long as a one-to-one relationship between the corpus data and the annotation can be established. Chapters 3 and 4 will describe extensions to the method, accounting for annotation which does not have such an immediate one-to-one mapping.

The underlying idea of the method presented in this chapter is that corpus annotation should be consistent across a corpus. One way to determine if the annotation is consistent is to find two corpus instances of the "same element" (defined more precisely below) and to see if those same elements have the same annotation. If they do not have the same annotation, then it is likely an error. For example, if a word

appears in the middle of ten other words, it should probably have the same part-of-speech tag as when the same word again appears in the middle of the same ten words.

The usefulness of the variation n-gram error detection method can be seen in that it automatically detects errors in corpus annotation which remain despite human post-editing and are usually caused by it. Furthermore, the method is independent of the language and tagset of the corpus and requires no additional language resources such as lexica.

As we will see in section 2.2, the method detects variation in the POS annotation of a corpus by searching for n-grams[21] which occur more than once in the corpus and include at least one difference in their annotation. We discuss how all such *variation n-grams* of a corpus can be obtained and show that together with some heuristics they are highly accurate predictors of annotation errors. We illustrate the applicability and effectiveness of this method by reporting the results of applying it to the Wall Street Journal (WSJ) corpus as part of the Penn Treebank 3 release, which was tagged using the PARTS tagger and manually corrected afterwards (Marcus et al., 1993). In section 2.3, we discuss the applicability of the variation n-gram method on a larger range of corpora. Section 2.4 then presents two related ideas which attempt to detect different kinds of inconsistencies.

[21] An *n-gram* is a stretch of n tokens (or words) in the corpus.

2.2 A method for detecting errors

2.2.1 Using the variation in a corpus

For each word that occurs in a corpus, there is a lexically determined set of tags that can in principle be assigned to this word. The tagging process reduces this set of lexically possible tags to the correct tag for a specific corpus occurrence. A particular word occurring more than once in a corpus can thus be assigned different tags in a corpus. We will refer to this as *variation*. For example, in (3), the word *decided* varies between a past tense verb (VBD) and a past participle (VBN) in the WSJ.

(3) a. And the city decided/VBD to treat its guests more like royalty or rock stars than factory owners .
b. I 've decided/VBN for personal reasons to take early retirement .

Variation in corpus annotation is caused by one of two reasons: i) *ambiguity*: there is a word ("type") with multiple lexically possible tags and different corpus occurrences of that word ("tokens") happen to realize the different options,[22] as was seen in (3), or ii) *error*: the tagging of a word is inconsistent across comparable occurrences, as shown in (4). We can therefore locate annotation errors by zooming in on the variation exhibited by a corpus, provided we have a way to decide whether a particular variation is an ambiguity or an error—but how can this be done?

(4) a. Mr. Suominen may have decided/VBD to cut Finland 's losses once and for all .
b. Many farmers, too removed to glean psyllium 's new sparkle in the West , have decided/VBN to plant mustard

[22]For example, the word *can* is ambiguous between being an auxiliary, a main verb, or a noun and thus there is variation in the way *can* would be tagged in *I can play the piano*, *I can tuna for a living*, and *Pass me a can of beer, please*.

Variation n-grams

The key to answering the question lies in a classification of contexts: the more similar the context of a variation, the more likely it is for the variation to be an error. But we need to make concrete what kinds of properties the context consists of and what count as similar contexts. In this chapter, we focus on contexts composed of words,[23] and we require identity of the context, not just similarity. We will use the term *variation n-gram* for an n-gram (of words) in a corpus that contains a word that is annotated differently in another occurrence of the same n-gram in the corpus. The word exhibiting the variation is referred to as the *variation nucleus*.

For example, in the WSJ, the string in (5) is a variation 12-gram since *off* is a variation nucleus that in one corpus occurrence of this string is tagged as preposition (IN), while in another it is tagged as particle (RP).

(5) to ward off a hostile takeover attempt by two European shipping concerns

Note that the variation 12-gram in (5) contains two variation 11-grams, which one obtains by eliminating either the first or the last word, as shown in (6).

(6) a. to ward off a hostile takeover attempt by two European shipping
b. ward off a hostile takeover attempt by two European shipping concerns

Algorithm

To compute all variation n-grams of a corpus, we make use of the just mentioned fact that a variation n-gram must contain a variation $(n-1)$-gram to obtain an algorithm efficient enough to handle large corpora. The algorithm, which essentially is an instance of the a priori algorithm used in information extraction (Agrawal and

[23] Other options allowing for application to more corpus instances would be to use contexts composed of POS tags or some other syntactic or morphological properties. Such options are explored for syntactic and discontinuous syntactic annotation in chapters 3 and 4.

Srikant, 1994), takes a POS-annotated corpus and outputs a listing of the variation n-grams, from $n = 1$ to the longest n for which there is a variation n-gram in the corpus. It proceeds as follows:

1. Calculate the set of variation unigrams in the corpus and store the variation unigrams and their corpus positions.[24]

2. Based on the corpus positions of the variation n-grams last stored, extend the n-grams to either side (unless the corpus ends there).[25] For each resulting $(n+1)$-gram, check whether it has another instance in the corpus and if there is variation in the way the different occurrences of the $(n+1)$-gram are tagged. Store all variation $(n+1)$-grams and their corpus positions.

3. Repeat step 2 until we reach an n for which no variation n-grams are in the corpus.

Running the variation n-gram algorithm on the WSJ corpus produced variation n-grams up to length 224.[26] The table in figure 2.1 reports two results for each n: the first is the number of variation n-grams that were detected and the second is the number of variation nuclei that are contained in those n-grams. For example, the second entry reports that 17,384 variation bigrams were found, and they contained 18,499 variation nuclei, i.e., for some of the bigrams there was a tag variation for both

[24]In chapter 3, we will see how the definition of a unigram can be generalized for syntactic annotation.

[25]In chapter 4, we will see how the notion of extending an n-gram can be generalized for discontinuities to include any material which is interleaved with the n-gram.

[26]Irrespective of variation, this 224-gram is the longest string which appears multiple times at all in the WSJ corpus. This fact was calculated using the a priori algorithm without the condition on variation, but note that for extremely large corpora, the suffix array method presented in Yamamoto and Church (2001) would likely be more efficient for such a calculation.

1.	7033	7033	57.	946	3558	113.	343	1846	169.	90	395
2.	17384	18499	58.	932	3558	114.	338	1820	170.	87	380
3.	12199	13002	59.	918	3557	115.	333	1794	171.	84	365
4.	6576	7181	60.	904	3556	116.	328	1768	172.	81	350
5.	4097	4646	61.	889	3550	117.	323	1742	173.	78	335
6.	2934	3478	62.	873	3545	118.	318	1716	174.	75	320
7.	2333	2870	63.	857	3536	119.	313	1689	175.	72	305
8.	2027	2583	64.	841	3519	120.	308	1661	176.	69	290
9.	1825	2405	65.	825	3497	121.	303	1632	177.	66	274
10.	1678	2296	66.	809	3473	122.	298	1602	178.	63	258
11.	1579	2249	67.	793	3449	123.	293	1571	179.	60	242
12.	1516	2241	68.	777	3426	124.	288	1540	180.	57	226
13.	1475	2260	69.	762	3405	125.	283	1509	181.	54	210
14.	1456	2305	70.	747	3376	126.	278	1478	182.	51	194
15.	1429	2333	71.	733	3348	127.	273	1446	183.	48	178
16.	1413	2378	72.	720	3315	128.	268	1413	184.	45	162
17.	1395	2431	73.	708	3283	129.	263	1379	185.	42	146
18.	1381	2484	74.	696	3250	130.	258	1345	186.	40	137
19.	1376	2547	75.	683	3211	131.	253	1311	187.	38	128
20.	1376	2615	76.	670	3171	132.	248	1277	188.	37	126
21.	1367	2671	77.	656	3134	133.	243	1243	189.	36	124
22.	1355	2721	78.	642	3093	134.	237	1205	190.	35	122
23.	1343	2764	79.	629	3052	135.	231	1167	191.	34	120
24.	1330	2808	80.	616	3011	136.	225	1134	192.	33	118
25.	1318	2846	81.	603	2966	137.	219	1100	193.	32	116
26.	1304	2877	82.	594	2928	138.	213	1066	194.	31	114
27.	1291	2911	83.	585	2890	139.	207	1032	195.	30	112
28.	1283	2950	84.	577	2853	140.	202	1001	196.	29	110
29.	1273	2987	85.	568	2814	141.	197	970	197.	28	108
30.	1264	3028	86.	558	2765	142.	193	948	198.	27	106
31.	1255	3072	87.	547	2714	143.	189	926	199.	26	104
32.	1243	3116	88.	536	2661	144.	185	904	200.	25	102
33.	1234	3164	89.	526	2617	145.	181	882	201.	24	100
34.	1220	3203	90.	517	2573	146.	176	853	202.	23	98
35.	1211	3241	91.	505	2516	147.	171	828	203.	22	96
36.	1201	3275	92.	493	2457	148.	167	809	204.	21	94
37.	1188	3305	93.	481	2398	149.	163	790	205.	20	92
38.	1177	3337	94.	469	2339	150.	159	770	206.	19	90
39.	1169	3371	95.	459	2298	151.	155	750	207.	18	88
40.	1158	3397	96.	449	2259	152.	151	729	208.	17	86
41.	1147	3419	97.	439	2218	153.	147	708	209.	16	84
42.	1134	3432	98.	430	2185	154.	143	687	210.	15	82
43.	1124	3444	99.	421	2150	155.	139	666	211.	14	80
44.	1114	3454	100.	412	2114	156.	135	645	212.	13	78
45.	1106	3468	101.	405	2084	157.	131	623	213.	12	76
46.	1097	3481	102.	399	2066	158.	127	600	214.	11	74
47.	1087	3495	103.	393	2048	159.	123	575	215.	10	72
48.	1074	3503	104.	388	2032	160.	119	550	216.	9	68
49.	1059	3507	105.	383	2017	161.	115	525	217.	8	64
50.	1045	3510	106.	378	2002	162.	111	500	218.	7	59
51.	1030	3510	107.	373	1987	163.	108	485	219.	6	53
52.	1018	3521	108.	368	1969	164.	105	470	220.	5	46
53.	1004	3529	109.	363	1948	165.	102	455	221.	4	38
54.	989	3538	110.	358	1924	166.	99	440	222.	3	29
55.	975	3548	111.	353	1898	167.	96	425	223.	2	20
56.	961	3556	112.	348	1872	168.	93	410	224.	1	10

Figure 2.1: Variation n-grams and nuclei in the WSJ

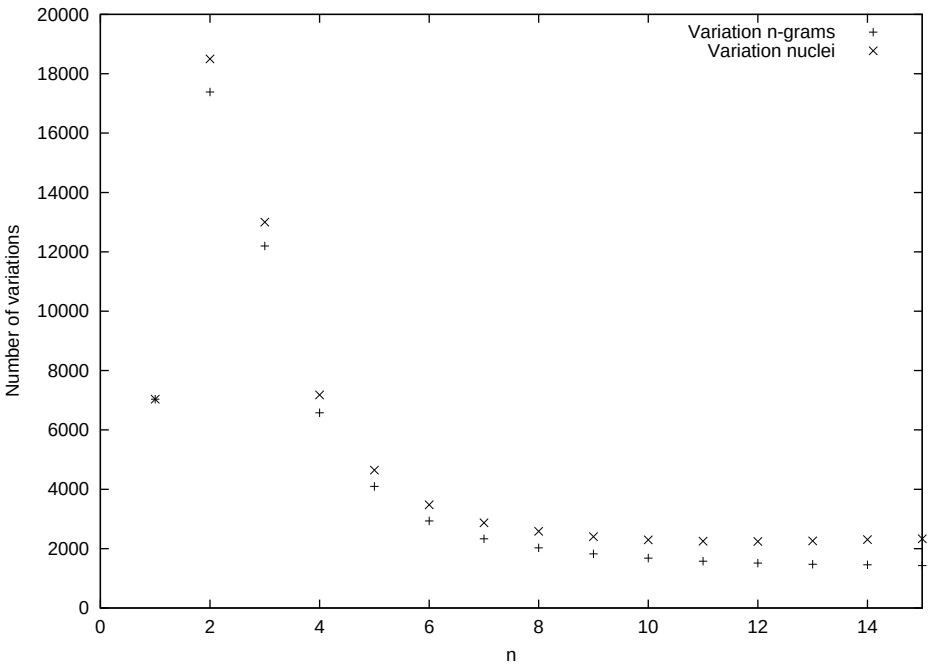

Figure 2.2: Variation n-grams and nuclei in the WSJ for n up to 15

of the words. At the end of the table is the single variation 224-gram, containing 10 different variation nuclei, i.e., spots where the annotation of the (two) occurrences of the 224-gram differ.[27]

The table reports the level of variation in the WSJ across identical contexts of different sizes, graphically seen in figures 2.2 and 2.3. In the next section we turn to the issue of detecting those occurrences of a variation n-gram for which the variation nucleus is an annotation error.

[27]The table does not report how often a variation n-gram occurs in a corpus since such a count is not meaningful in our context: The variation unigram *the*, for instance, appears 56,317 times in the WSJ, but 56,300 of these are correctly annotated as determiner (DT).

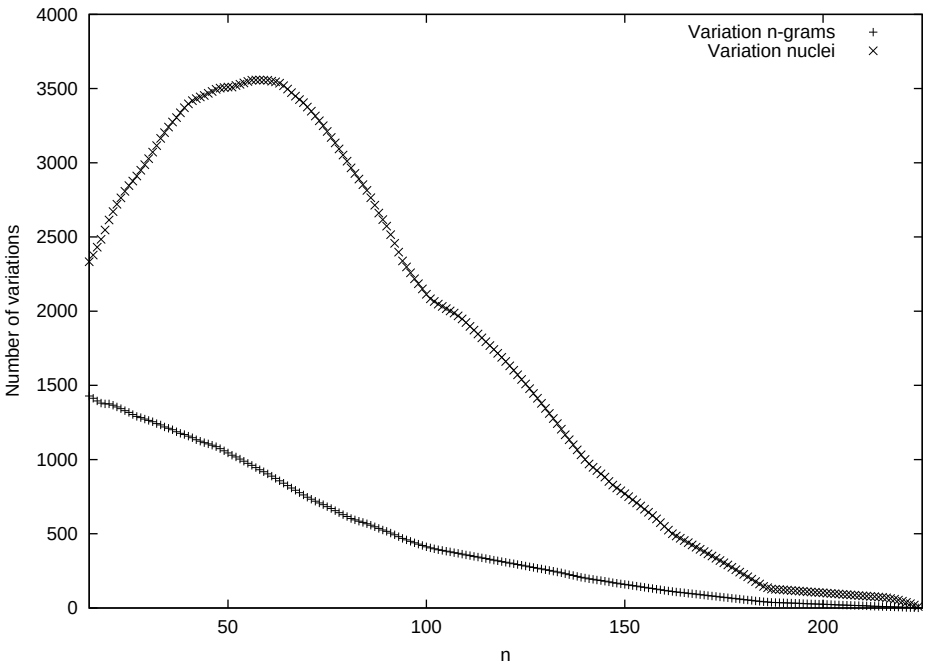

Figure 2.3: Variation n-grams and nuclei in the WSJ for n above 15

2.2.2 Heuristics for classifying variation

Once the variation n-grams for a corpus have been computed, heuristics can be employed to classify the variations into errors and ambiguities. These heuristics capture typical regularities of language, but they are only heuristics. They do not capture language facts in the way that a full grammatical analysis would, but we want to see how effective simple, linguistically-informed heuristics can be. The first heuristic encodes the basic fact that the tag assignment for a word is dependent on the context of that word. The second takes into account that natural languages favor the use of local dependencies over non-local ones. Both of these heuristics are independent of a specific corpus, tagset, or language.

Variation nuclei in long n-grams are errors The first heuristic is based on the insight that a variation is more likely to be an error than a true ambiguity if it occurs within a long stretch of otherwise identical material. In other words, the longer the variation n-gram, the more likely that the variation is an error.

For example, *lending* occurs tagged as adjective (JJ) and as common noun (NN) within occurrences of the same 184-gram in the corpus, as shown in figure 2.4. It is very unlikely that the context (109 identical words to the left, 74 to the right) supports an ambiguity, and the adjective tag does indeed turn out to be an error. Similarly, the already mentioned 224-gram, shown in figure 2.5, includes 10 different variation nuclei, all of which turn out to be erroneous variation.

% four months ; 8 7/16 % to 8 5/16 % five months ; 8 7/16 % to 8 5/16 % six months . LONDON INTERBANK OFFERED RATES (LIBOR) : 8 11/16 % one month ; 8 5/8 % three months ; 8 7/16 % six months ; 8 3/8 % one year . The average of interbank offered rates for dollar deposits in the London market based on quotations at five major banks . FOREIGN PRIME RATES : Canada 13.50 % ; Germany 9 % ; Japan 4.875 % ; Switzerland 8.50 % ; Britain 15 % . These rate indications are n't directly comparable ; lending practices vary widely by location . TREASURY BILLS : Results of the Monday , October 23 , 1989 , auction of short-term U.S. government bills , sold at a discount from face value in units of $ 10,000 to $ 1 million : 7.52 % 13 weeks ; 7.50 % 26 weeks . FEDERAL HOME LOAN MORTGAGE CORP . (Freddie Mac) : Posted yields on 30-year mortgage commitments for delivery within 30

Figure 2.4: A 184-gram which appears twice in the WSJ with one variation

. The Wall Street Journal " American Way of Buying " Survey consists of two separate , door-to-door nationwide polls conducted for the Journal by Peter D. Hart Research Associates and the Roper Organization . The two surveys , which asked different questions , were conducted using national random probability samples . The poll conducted by Peter D. Hart Research Associates interviewed 2,064 adults age 18 and older from June 15 to June 30 , 1989 . The poll conducted by the Roper Organization interviewed 2,002 adults age 18 and older from July 7 to July 15 , 1989 . Responses were weighted on the basis of age and gender to conform with U.S. Census data . For each poll , the odds are 19 out of 20 that if pollsters had sought to survey every household in the U.S. using the same questionnaire , the findings would differ from these poll results by no more than 2 1/2 percentage points in either direction . The margin of error for subgroups – for example , married women with children at home – would be larger . In addition , in any survey , there is always the chance that other factors such as question wording could introduce errors into the findings . (See related story : " The American Way of Buying :

Figure 2.5: A 224-gram which appears twice in the WSJ with ten variations

While we have based this heuristic solely on the length of the identical context, another factor one could take into account for determining relevant contexts are structural boundaries. A variation nucleus that occurs within a complete, otherwise identical sentence is very likely to be an error.[28]

For example, the 25-gram in (7) is a complete sentence that appears 14 times, four times with *centennial* tagged as JJ and ten times with *centennial* marked as NN, with the latter being correct according to the description in the tagging guide (Santorini, 1990). The fact that variation exists for *centennial* for n-grams up to length 32 (for three instances) seems to be irrelevant.

(7) During its centennial year , The Wall Street Journal will report events of the past century that stand as milestones of American business history .

[28]Since sentence segmentation information is often available for POS-tagged corpora, we focus on those structural domains here. For treebanks, other constituent structure domains could also be used for the purpose of determining the size of the context of a variation that should be taken into account for distinguishing errors from ambiguities. In chapter 4, we will experiment with using sentence and dialog turn domains, partly for reasons of efficiency.

In fact, for the JJ/NN distinction, all that really seems to be relevant are the preceding and following words. We will exploit this later on in obtaining more robust results (see section 2.2.3).

Distrust the fringe Turning the spotlight from the n-gram and its properties to the variation nucleus contained in it, an important property determining the likelihood of a variation to be an error is whether the variation nucleus appears at the fringe of the variation n-gram, i.e., at the beginning or the end of the context which is identical over all occurrences. This is due to the fact that morphological and syntactic properties are generally governed locally.

For example, *joined* occurs as past tense verb (VBD) and as past participle (VBN) within a variation 37-gram, as shown in example (8). It is the first word in the variation 37-gram and in one of the occurrences it is preceded by *has* (8b) and in another it is not (8a). Despite the relatively long context of 36 words to the right, the variation thus is a genuine ambiguity, enabled by the location of the variation nucleus at the left fringe of the variation n-gram. By taking only non-fringe variations as indicative of an error, we eliminate examples like (8) from consideration.

(8) a. John P. Karalis ...
b. John P. Karalis has ...
joined the Phoenix , Ariz. , law firm of Brown & Bain . Mr. Karalis , 51 , will specialize in corporate law and international law at the 110-lawyer firm . Before joining Apple in 1986 ,

2.2.3 Results for the WSJ

There are 1,289,201 tokens in the WSJ, corresponding to 51,457 word types. Nearly half the word types—23,497 words—appear only once in the corpus and so cannot possibly show up in our method as potentially erroneous. But these account

for only 1.8% of the tokens. Even though this is a small percentage of the corpus, we would still like to examine them for correctness of annotation, in order to better approach the detection of all errors. The variation n-gram method offers two possibilities of extension to detect errors in the annotation of words appearing once in a corpus. 1) One can generalize the notion of a nucleus to include POS information and thus include single-occurrence words in a class. 2) Hand-inspection of the errors detected by the method can point to what tags and kinds of words (e.g. hyphenated words) need more attention. In the style of Blaheta (2002) and Oliva (2001), patterns can be specified to search for throughout the whole corpus.

Focusing on the words which occur multiple times (98.2% of the tokens), we want to know which ones are potentially problematic. From figure 2.1, we can see that there are 7033 variation unigrams; this number corresponds to 13.7% of the word types and sets an upper bound on the number of word types we can hope to correct. These 7033 unigram types correspond to 711,994 word tokens, or 55.2% of the corpus. Thus, we find that a first pass of the unigrams restricts our attention to about half of the corpus. This is not to say that there are no errors in the 45.8% of the corpus which does not contain variation unigrams, but that we are more likely to find systematic errors in the remaining portion.

With that in mind, we can turn to the main results for the WSJ. We first examined only n-grams of length six or higher in order to evaluate the efficiency of the trust long contexts heuristic. The variation n-gram algorithm found 2495 distinct variation nuclei[29] of n-grams with $6 \leq n \leq 224$, where by *distinct* we mean that each corpus

[29]These 2495 distinct nuclei correspond to 559 distinct words, or types.

position is only taken into account for the longest variation n-gram it occurs in.[30] The number of distinct variation n-grams and nuclei for each n can be seen in figures 2.6 and 2.7. Note that for the first 15 values (figure 2.6), we get a curve very similar to the variation n-grams in figure 2.1. However, distinct n-grams stop only at the longest length, and so the variation n-grams thin out as n gets larger, and what we see in figure 2.6 is something of an asymptotic curve, but with much more noise.

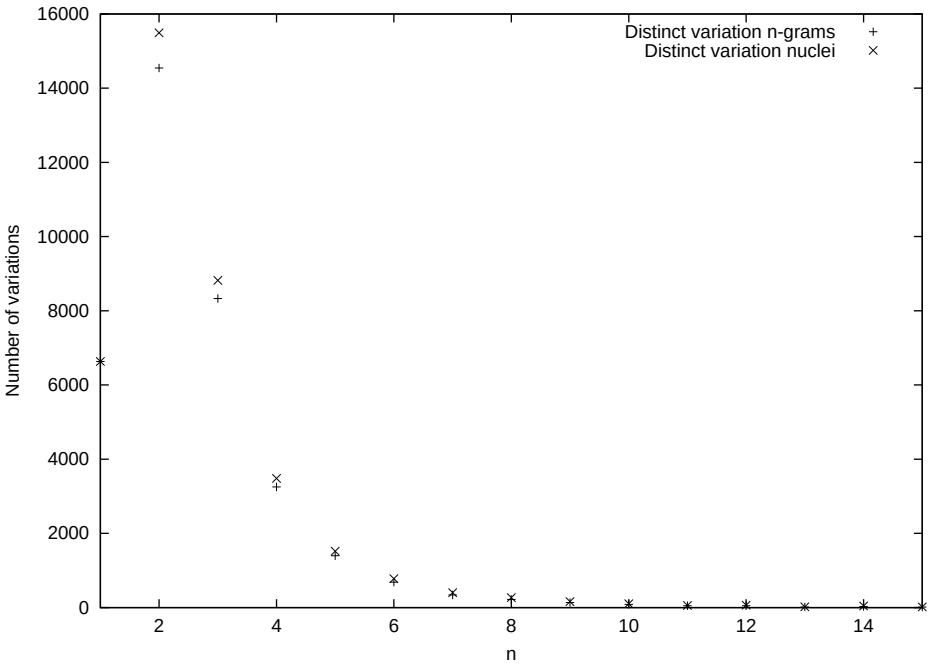

Figure 2.6: Distinct variation n-grams and nuclei in the WSJ for n up to 15

[30]This eliminates the effect that each variation n-gram instance also is an instance of a variation (n-1)-gram, a property exemplified by (5) and the discussion below it.

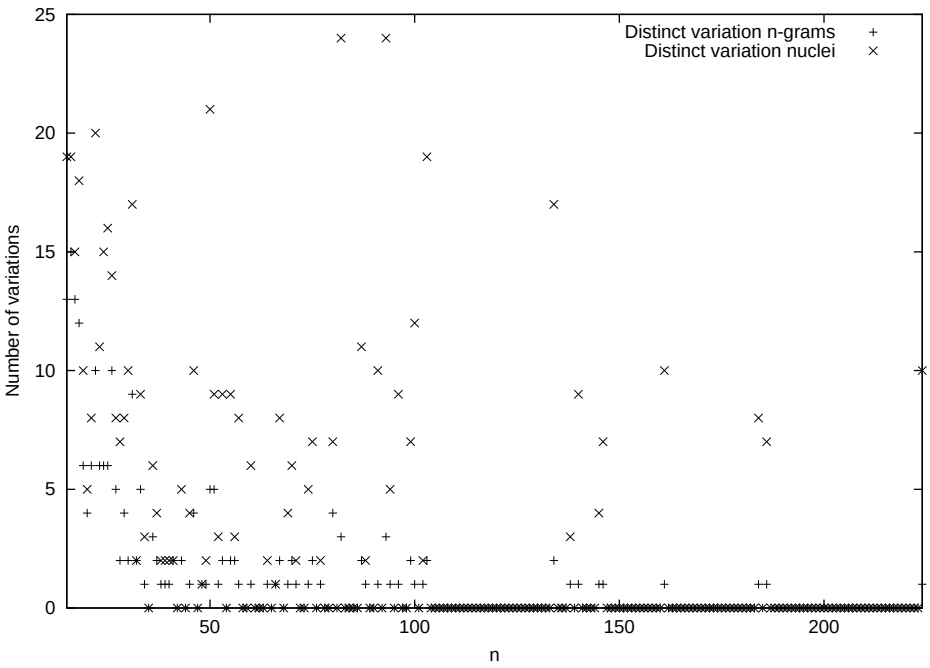

Figure 2.7: Distinct variation n-grams and nuclei in the WSJ for n above 15

To evaluate the precision of the variation n-gram algorithm and the heuristics for error detection, we need to know which of the variation nuclei detected actually include tag assignments that are real errors. We thus inspected the tags assigned to the 2495 variation nuclei that were detected by the algorithm and marked for each nucleus whether the variation was an error or an ambiguity.[31] We found that 2436, or 97.64%, of those variation nuclei (types) are errors, i.e., the variation in the tagging of those words as part of the particular n-gram was incorrect. To get an idea for

[31]Generally, the context provided by the variation n-gram was sufficient to determine which tag is the correct one for the variation nucleus. In some cases we also considered the wider context of a particular instance of a variation nucleus to verify which tag is correct for that instance. In theory, some of the tagging options for a variation nucleus could be ambiguities, whereas others would be errors; in practice this did not occur for this data set.

how many tokens in the corpus correspond to the variation nuclei that our method correctly flagged as being wrongly tagged, we hand-corrected the mistagged instances of those words. This resulted in a total of 4417 tag corrections.

For the heuristic of trusting long contexts, an n-gram length of six turns out to be a good cut-off point for the WSJ. This becomes apparent when one takes a look at where the 59 ambiguous variation nuclei arise: 32 of them are variation nuclei of 6-grams, 10 are part of 7-grams, 4 are part of 8-grams, and the remaining 13 occur in longer n-grams.

The second heuristic, distrust the fringe, however, turns out to be a stronger heuristic. 57 of the 59 ambiguous variation nuclei that were found are fringe elements, i.e., occur as the first or last element of the variation n-gram. The two exceptions are *and* ***use*** *some of the proceeds to* and *buy and* ***sell*** *big blocks of*, where the variation nuclei *use* and *sell* are ambiguous between base form verb (VB) and third-person singular present tense verb (VBP) but do not occur at the fringe. As an interesting aside, more than half of the true ambiguities (31 of 59) occurred between past tense verb (VBD) and past participle (VBN) and are the first word in their n-gram, as exemplified in (8). To completely evaluate the fringe in this experiment, one must also take into account the fact that there were 156 total fringe elements, of which 99 were actually errors. So, at least above the 5-grams, it may not be necessary to distrust the fringe, depending on if one is more concerned about precision or recall. Expecting the observed drop in precision to continue, however, the heuristic seems to be a a practical filter for generally determining if the n-gram contains an ambiguity or an error.

Indeed, we further tested the cut-off point for long n-grams against the heuristic of distrusting the fringe and in this way increased our recall. It is possible that in a language like English, simply ignoring fringe elements is enough to obtain reliable results. Concretely, we took all 7141 distinct non-fringe variation n-grams (i.e. types)—i.e. $3 \leq n \leq 224$—sampled 125 of them, and marked for each one whether it was erroneous variation (i.e. there was no need for variation), true ambiguity, or whether it was too difficult to decide for sure.[32] Of the 125 variations, 116 were erroneous variation, with 4 being true ambiguities, and 5 being cases of uncertainty, giving an estimated precision of at least 92.8%. The 95% confidence interval for the point estimate of .928 is (.8827, .9733), i.e., the number of real errors detected in the 7141 cases is estimated to be between 6303 and 6950.[33] Note that these are counts of distinct variation nuclei (i.e., recurring strings, or types), but we can reasonably assume that the token counts will be similar, if not larger.

If we take the estimated error rate of 3% in the WSJ as accurate, this means that there are about 38,767 errors to find. Our method locates approximately 6626 errors, which means that our recall is approximately 17%. While this leaves many errors yet to be found, to our knowledge it is significantly more than previously found systematically in the WSJ. Furthermore, correcting these errors would lead to a corpus error rate of 2.5%, a significant improvement. We can see, then, that solely

[32]This is essentially the same evaluation approach we will use in future chapters, as hand-checking all results is not generally feasible.

[33]The 95% confidence interval was calculated using the standard formula of $p \pm 1.96\sqrt{\frac{p(1-p)}{n}}$, where p is the point estimate and n the sample size.

relying on the heuristic "distrust the fringe" gives high accuracy while boosting the recall of the method. And with such high accuracy, hand-correction of the errors is eminently practical.

Kaljurand (2004) critiques the variation n-gram method by pointing out that the nature of the inconsistency is not taken into account, i.e., the frequency of distribution of tag variations. He proposes using a skew value to measure how skewed the frequency distribution is. High skew values are supposed to indicate a preference for one tag; low skew values have approximately equivalent frequencies for tags. Although using this kind of information could be useful, it is not clear whether we can reliably gain any information from this skew value, and Kaljurand (2004) does not test it. For example, although *executive* varies between JJ and NN, often incorrectly, with 466 NN instances and 285 JJ instances, *made* has about the same ratio (430 VBN to 290 VBD), and this is a desired ambiguity. A method such as one using this skew value or variance might be useful for ranking errors, but the variation n-gram method is already quite precise, and no such ranking seems necessary.[34]

Problematic cases When we speak of ambiguity, it is clear that we are speaking of cases of where there is ambiguity within an n-gram, but for each instance the part of speech can be determined. At other times, even within the context of the whole corpus, the word's part of speech is not clearly limited to one tag. A case where the word is truly ambiguous between two or more tags can be seen in the end of a 28-gram in (9), where *Rally* is shouted three times (in one instance, the next 2 *rally*'s are lowercase, while in the other they are uppercase, so they do not appear in the 28-gram).

[34]We will employ a variant of this skew value, however, for ranking error corrections; see section 6.4.

(9) ... Then, nine minutes later, Wall Street suddenly rebounded to a gain on the day . “ Rally

The shouting of this single word could either mean something akin to “This is a rally!” or “Let’s rally!”, corresponding to noun and verb uses, respectively. Oddly, such indeterminate cases were very rare, and it is noteworthy that, although the Penn Treebank project provides a vertical slash for such cases where two tags are possible (Santorini, 1990), what we find is one annotation of common noun (NN) and one of base form verb (VB).

Of the 2436 erroneous variation nuclei we discussed above, 140 of them deserve special attention here in that it was clear that the variation was incorrect, but it was not possible to decide based on the tagging guide (Santorini, 1990) which tag would be the right one to assign. Even without knowing the correct tag, it is clear that the context demands a uniform tag assignment. Most of those cases concern the distinction between singular proper noun (NNP) and plural proper noun (NNPS). For example, in the bigram *Salomon Brothers*, *Brothers* is tagged 42 times as NNP and 30 times as NNPS; similarly, *Motors* in *General Motors* is an NNP 35 times and an NNPS 51 times. A clearer case is the trigram *Salomon Brothers Inc*, where *Brothers* is no longer a word that could be affecting subject-verb agreement, yet is NNP 27 times and NNPS 22 times.

It is possible to design and document a tagset in a way that allows for 100% inter-judge agreement for POS-annotation (Voutilainen and Järvinen, 1995), but this requires clearly documenting difficult cases and eliminating structurally unjustified POS-ambiguity, e.g., certain types of noun-adjective homographs and the distinction

between proper and common nouns.[35] The results of the variation n-gram algorithm support the notion that such highly error-prone distinctions exist in the Penn Treebank tagset. Whether or not these distinctions should be kept is another issue, and further research will determine what internal and external criteria constitute a good tagset—see, e.g., Przepiórkowski and Woliński (2003).[36] In the section 2.3, we discuss in more detail the tagset's influence on the n-gram method.

2.3 Other corpora

To gauge the generality of the variation n-gram error detection method, we must examine other corpora with different properties. The method is not specifically designed for a particular language, tagset, or corpus, but certain issues arise when applying the method and the heuristics to other corpora, due to the kind of language and the tagset used in the corpus.

The utility of a method using variation n-grams relies on a particular word appearing several times in a corpus with different annotations. It thus works best for large corpora and hand-annotated or hand-corrected corpora, or corpora involving other sources of inconsistency. As discussed in section 1.2.1, interannotator bias creates inconsistencies which a completely automatically-tagged corpus does not have (Ratnaparkhi, 1996). And Baker (1997) makes the point that a human post-editor also decreases the internal consistency of the tagged data since he will spot a mistake made by an automatic tagger for some but not all of its occurrences. As a result, our

[35]Current research by Sampson and Babarczy (2003) is trying to determine if previously-defined tagsets such as the one for the SUSANNE corpus are equally capable.

[36]Note that our heuristics would work best on a tagset like the one proposed in Przepiórkowski and Woliński (2003), which is based purely on morphological and syntactic properties, since semantic and pragmatic properties are not directly visible in word forms and word order.

variation n-gram approach is well suited for the gold-standard annotations generally resulting from a combination of automatic annotation and manual post-editing, as the WSJ was. The other corpora we tested the method on, as discussed in sections 2.3.1 (BNC-sampler) 2.3.2 (SUSANNE), and 2.3.3 (MULTEXT-East), all contain some degree of manual tagging. Figure 2.8 gives a quick comparison of some of the properties relevant for the variation n-gram method, including the total size of the corpus (number of tokens), the largest varying n, and the number of variation nuclei.

Corpus	Size	Largest n	Nuclei
WSJ	1,289,201	224	7033
BNC-sampler	2,427,451	692	8710
SUSANNE	156,622	9	1501
MULTEXT-East	118,424	127	1048

Figure 2.8: A comparison of the different corpora examined

2.3.1 BNC-sampler

The British National Corpus sampler, or BNC-sampler (Leech, 1997), provides an ideal scenario for testing the n-gram method on a larger corpus, in this case one roughly twice as large as the WSJ (see Figure 2.8), and one which contains spoken data. With a large corpus, more and longer n-grams are expected, but the corpus is reputed to be much better-tagged than the Penn Treebank corpora, and spoken data changes the nature of the context. As previously cited (Leech, 1997), 0.3% of the BNC-sampler is estimated to be erroneous, while the estimates for the part-of-speech

annotation in the Penn treebank are closer to 3% (Marcus et al., 1993, p. 19). Thus, testing the method on the BNC-sampler will give a clearer indication of how general the method is.

Running the variation n-gram algorithm on the BNC-sampler found 8710 variation unigrams and variation n-grams up to length 692 (one 692-gram with 4 variation nuclei), but for the 6-grams and above, there were only 872 distinct variation nuclei (cf. 2495 in the WSJ), many of which (385) were on the fringe. With 34,312 distinct variation nuclei (8527 non-fringe nuclei) between the 3 and the 692-grams, however, there still is much potential for detecting a large number of errors.[37]

Of the 8527 non-fringe distinct variation nuclei (types), we sampled 125 in order to obtain an estimate of precision. As with the WSJ evaluation, we marked for each one whether it was erroneous variation (i.e. there was no need for variation), true ambiguity, or whether it was too difficult to decide for sure. With the BNC-Sampler, there were two reasons we could not decide: 1) the guidelines were not sufficiently clear, as was often the case with indecisions in the WSJ, and 2) the text—much of it spoken dialogue with restarts and unclear phrases—made it too difficult to determine what the speaker was trying to say and thus what tag is appropriate. The latter was much more prevalent, but we will conflate the two into a single category, designated *uncertain*.

[37]Compare the 16,319 distinct variation nuclei (7141 non-fringe) between the 3 and 224-grams in the WSJ corpus.

One further note is in order. When checking the results by hand, sometimes it was unclear if the variation was an error or an ambiguity, but one was deemed (much) more likely. In the tables below, we report the results of grouping these cases with the *uncertain* category (Uncertain) vs. grouping them with the appropriate *error* or *ambiguity* (Error/Ambiguity) category.

Of the 125 samples, then, we can see in figure 2.9 that at least 51 are errors (40.80%) and most likely 65 (52%) are errors. The 95% confidence interval for the point estimate of .408 is (.3218, .4942), meaning that we are 95% confident that there are between 2744 and 4213 errors; likewise, for the point estimate of .52, we obtain a 95% confidence interval of (.4324, .6076), corresponding to between 3687 and 5180 errors. While the number of errors is useful for correction and the precision is decent (compare the precision rates of 44% (Eskin, 2000) and 20% (van Halteren, 2000)), these precision figures are considerably lower than the WSJ results, and so we now turn to some of the causes of this.

	Error	Ambiguity	Uncertain	Total	Precision
Uncertain	51	51	23	125	40.80%
Error/Ambiguity	65	55	5	125	52.00%

Figure 2.9: Results of the method on the BNC-sampler corpus

Spoken language issues As mentioned, one difference between the BNC and the WSJ is that the BNC includes spoken language. The BNC-sampler, in fact, is comprised of about half spoken language. Aside from false starts and sudden endings

(which potentially cause odd sequences of words to appear next to each other), one noticeable feature of spoken language in the BNC is the presence of pauses. These pauses indicate that the words preceding and following them may or may not be syntactically related. In other words, they mask the context, in that we really do not know what the surrounding context is. We can see an example of this in (10), where *to* varies between II (general preposition) and TO (infinitive marker).[38] What appears to be variation in the same context really does not share a context at all.

(10) a. down from London <**pause**> **to** <**pause**> just within ten miles of Bury St Edmunds .
b. he just does n't seem to have anything <**pause**> **to** <**pause**> volunteer

Removing pauses which are on the fringe and next to a nucleus gives us the results in figure 2.10, which shows us that the rate of finding errors slightly goes up: 41.52% (49/118) on the low end and 52.54% (62/118) on the high end. Removing pauses from the overall set of 8527 distinct variation nuclei gives 8177 remaining nuclei—i.e., 350 nuclei have fringe pauses next to them.[39]

	Error	Ambiguity	Uncertain	Total	Precision
Uncertain	49	48	21	118	41.52%
Error/Ambiguity	62	51	5	118	52.54%

Figure 2.10: Results on the BNC-sampler after removing examples where only a <*pause*> makes them non-fringe

[38]We will here notate an n-gram by putting it in bold and its nucleus by underlining it and shading it gray.

[39]In order to better assess the impact of these pauses on the method in the future, one could divide the BNC-sampler into the written and spoken halves and run the variation n-gram code on the separate halves individually.

Tagset considerations A bigger factor in the accuracy of the method turns out to be the tagset. Most problematic for our method are the variations between two different verbal tags. We can see this illustrated in example (11), where (11a) gets the tag VV0 (base form of lexical verb) and (11b) gets the tag VVI (infinitive). These tags are clearly correct, as the VV0 instance is finite and the VVI one is not, but the context is not sufficiently large enough to determine the differences in use. In this case, one more word to the left would have clearly distinguished their uses (i.e. *did* requires an infinitive to follow). Similar problems exist between other verbal pairings, notably between VVD (past tense of lexical verb) and VVN (past participle of lexical verb).[40]

(11) a. Really ni , I mean **you get in** there and you feel ...
b. What time did **you get in** from the shopping ?

Removing from consideration examples with verbal variation significantly increases the accuracy of the method, as shown in figure 2.11. There is 53.49% precision on the low end and 65.12% precision on the high end. Removing these examples from the full set of 8527 n-grams results in 5340 remaining distinct variation n-grams. Thus, with a 95% confidence interval of (.4295, .6403) for the point estimate of .5349, we are 95% confident that there are between 2293 and 3419 errors. For the point estimate of .6512, we obtain a 95% confidence interval of (.5524, .7519), corresponding to between 2939 and 4015 errors.

Recall It has been reported (Leech, 1997) that the BNC-sampler has an estimated corpus error of 0.3%. If this is true, then of the 2,427,451 tokens, about 7282 are

[40]More technically, we should say: between the regular expressions V.D and V.N, as the verbs *be*, *do*, and *have* have their own tags, e.g., VBD and VBN for forms of *be*.

	Error	Ambiguity	Uncertain	Total	Precision
Uncertain	46	22	18	86	53.49%
Error/Ambiguity	56	25	5	86	65.12%

Figure 2.11: Results on the BNC-sampler after removing examples involving variation between verbal tags

estimated to be errors. All of the results we have looked at so far correspond to distinct variation types, but each erroneous type corresponds to one token error, which provides an estimate of the number of token errors we can expect to find. With the confidence intervals obtained from figure 2.9, we can estimate the recall of our method. On the low end (grouping questionable cases into the *uncertain* category), we have between 2744 and 4213 errors, or between 37.69% and 57.86% recall. On the high end (grouping questionable cases into their appropriate error or ambiguity groups), we have between 3687 and 5180 errors, or between 50.63% and 71.15% recall. Thus, if Leech (1997) is right, we could say that this method has detected approximately half of the errors in the BNC-sampler; as Leech only estimates the error rate, what we can say with more firmness is that we have detected a large number of errors, none of which had been caught during the BNC Tag Enhancement project (Baker, 1997).

2.3.2 SUSANNE

The SUSANNE corpus, version 5 (Sampson, 1995), is worth investigating for a number of reasons. Its annotation is composed of a much finer-grained tagset than the Penn Treebank tagset, with 357 total tags (vs. the 48 in the Penn Treebank and 147 in the BNC-sampler). Secondly, it is a much smaller corpus, with only

156,622 tokens,[41] making recurrence of strings much less likely. Indeed the longest recurring text is only 16 tokens long. Finally, since it has "undergone considerable proof-checking" (Sampson, 2003), it is likely to be very well-tagged.

Indeed, with so few words, there are much fewer n-grams detected, extending up to a much smaller length. There are only 1501 variation nuclei, and only 104 distinct variation n-grams between the 4-grams and the 9-grams, of which 40 are non-fringe nuclei. And of these 40, 12 are errors, a surprisingly low amount. This low percentage, however, can be explained in light of two factors: alteration of the original data and ditto tags.

Firstly, there were 11 varying words called ⟨*formul*⟩, with variation between either FO (indeterminate formula) and FOx (formula or acronym for chemical substance, molecule, or subatomic particle) or between FOc (algebraic expression with nominal as opposed to equative function) and FOx. These all denote different kinds of formulas, but the original formulas have been replaced with a simple ⟨*formul*⟩ token, meaning that the original token distinctions have been lost and are now (partially) encoded in their tags. What our method has discovered is not erroneous tagging, but tokens which were originally different. In addition, there are meta-tags like ⟨*bital*⟩ (begin italics) and ⟨*eital*⟩ (end italics) which give no indication of the surrounding contentful words, making the immediate context less informative.

[41] Sampson (1995) reports that there are about 130,000 words, but for our purposes we take into account all tokens in the original file, which includes punctuation, words that have been split into multiple tokens (e.g., the one-word *term-end* is split into the three-token sequence *term* ⟨*hyphen*⟩ *end*), and meta-word tokens (e.g., markers for beginning and ending italics, sentence-break markers). Performance of the algorithm without the inclusion of these so-called "meta-tokens" is expected to be the same, if not better, based on our discussion of the ⟨*pause*⟩ meta-tokens in the BNC-sampler.

Secondly, eight of the varying words which are non-errors have variation between a regular tag and a ditto tag, such as *by* and *the* in *by the way*.[42] Ditto tags encode the notion that a token is not an individual unit, but rather is a (somewhat non-compositional) part of a larger "idiom." The ditto tags, RR31 and RR32 in this case, indicate the part of speech (RR – general adverb), the total number of elements in the idiom (3 in this case), and the position of the current word within the idiom (1 and 2, respectively). For these multi-token words, the surrounding context is less informative for distinguishing between an error and an ambiguity than under normal conditions because neighboring tokens are a part of the same "word" and thus contain the same information.

Despite the small number of errors, it is significant that the method detects any errors at all since the corpus is small and has been carefully cleaned. We ran the algorithm on an earlier version of the corpus, version 1 from 1992, eight years before version 5 (2000), the version on which we obtained our results above. The current version is generally much better-tagged, but not by very much when it comes to the variation n-grams. In the earlier version, there were 117 distinct variation n-grams between the 4-grams and the 12-grams, of which 46 were non-fringe nuclei. And of these 46, 18 were errors (compare 12 out of 40). Thus, even though efforts have been made to clean the corpus (Sampson, 2003), the kind of errors the variation n-gram method detected were mostly not found in the eight years between versions.

[42]The word *way* also varies in the 4-gram, but only as a right fringe element and so is ignored here.

2.3.3 MULTEXT-East

The MULTEXT-East corpus, version 2.1, (Dimitrova et al., 1998; Erjavec et al., 2003) is also a small corpus (under 200,000 tokens), but it has very different properties than the SUSANNE corpus. It is a parallel corpus consisting of translations of George Orwell's novel *1984* in six different Eastern European languages, as well as in the original English version. We will only examine the tagging of the English version of the corpus, a text of 118,424 words,[43] but note that the method is being used for at least Slovenian in the preparation of a newer version of the corpus (Tomaž Erjavec, personal communication).

Even though it is of comparable size to the SUSANNE corpus, by the nature of the material, it contains a much longer repetitive stretch of text, namely a 127-word passage which appears twice. The 127-gram contains one variation nucleus (which is an error), and it is noteworthy that the next smallest distinct variation n-gram is only of length 11. Despite containing fewer variation unigrams than the SUSANNE corpus (1048, as compared to 1501), the MULTEXT-East corpus has a greater number of non-fringe elements between the 4 and 127-grams (147, as compared to 40). It also has a higher percentage of errors: 99 out of 147, or about 67.3%.

The tagset for the MULTEXT-East project, specified in Erjavec (2001), makes distinctions which cause unique problems for the variation n-gram method. The tagset is a compositional tagset, with a slot for part of speech and slots for the different features within each part of speech. For example, the tag **Ncns** refers to noun-common-neuter-singular. For this kind of tagset, a question emerges about the

[43]The version we worked with had no capitalized words; this did not seem to greatly affect the method.

nature of an error, for two tags can differ and yet still be compatible. In example (12), for instance, the word *other* varies between **Pg** (pronoun-general) and **Pg3** (pronoun-general-third person).

(12) , one or other of them

The question this example raises if whether consistency in tagging is made up of more than the right part of speech. As we interpret it, consistency requires each tag to contain the same amount of information in the same situation. If one tag subsumes another, the guidelines should outline what approach to take; in the BNC project, for example, ambiguities between a general tag and a more specific tag are avoided by "allowing the more general tag to subsume the more specific" (Wynne, 1996). Lönneker and Jakopin (2004) refer to situations such as this one in the MULTEXT-East corpus as cases of underspecification—i.e., not specifying all the relevant information that can be provided—and these often point to errors. Lönneker and Jakopin (2004) also discuss overspecification and how both kinds of non-compliance can point to erroneous tagging. Note that both overspecification and underspecification potentially fall into a larger class of errors, namely those between a real tag and a tag which should not exist in the corpus, although this can be a difficult task for a compositional tagset with thousands of possibilities.

Although instantiations vary from language to language, the tagset has been specified in such a way as to cover a wide range of languages. So, for example, distinction such as feminine and masculine have been retained in the English tagset for all pronouns. This can cause a problem when we have a situation as in example (13).

(13) . one of them was

In one situation, *one* refers to a female and in another it refers to a rat, garnering the tags **Pp3fs** (pronoun-personal-third-feminine-singular) and **Pp3ms** (pronoun-personal-third-masculine-singular), respectively.[44] The distinction here is non-local, whereas it is morphologically marked for other personal pronouns in English and for virtually all nominals in many Eastern European languages.

One final note is in order regarding the MULTEXT-East corpus. Being composed of the novel *1984*, the corpus contains passages of Newspeak, George Orwell's thought-enforcing version of English, evidenced in example (14).[45]

(14)

times	3.12.83	reporting	bb	dayorder	d[o]ubleplusungood	refs	unpersons
			X			X	
			Np			Ncnp	

rewrite	fullwise	upsub	antefiling .
Vmn	X	Af	
Afp	Afp	Ncns	

The task of accurate tagging here could be said to be impossible: in an afterword to *1984*, Orwell claimed that for these words, there "was an almost complete interchangeability between different parts of speech."[46] However, Newspeak can be handled with proper guidelines, and this example highlights the fact, mentioned in the problematic cases part of section 2.2.3, that, even without knowing the correct tag, or in this case without even fully understanding the language, it is clear that there is no need for variation in tagging. The mere fact that it is the same sentence is sufficient to know that tagging should be consistent.

[44] Assumedly, it was a male rat.

[45] Note that the two spellings of *d(o)ubleplusungood* caused the method to find two separate n-grams, a 5-gram before *d(o)ubleplusungood* and a 6-gram after it, instead of a single 12-gram. Modifying the method to map alternate spellings onto the same class would be useful in this context, but would require a significant amount of effort for what is likely a small payoff.

[46] http://www.newspeakdictionary.com

2.4 Two related ideas

Aside from the main proposal of this chapter—to use a variation n-gram analysis combined with heuristics for detecting corpus errors—there are two simple ideas for detecting errors which we want to mention here. These techniques are independent of the variation n-gram method, but they share with it an emphasis on enforcing consistency and can be combined with it in a pipeline model.

Closed class analysis

Lexical categories in linguistics are traditionally divided into open and closed classes. Closed classes are the ones for which the elements can be enumerated (e.g., classes like determiners, prepositions, modal verbs, or auxiliaries), whereas open classes are the large, productive categories such as verbs, nouns, or adjectives.

Making practical use of the concept of a closed class, one can see that almost half of the tags in the WSJ tagset correspond to closed lexical classes. This means that a straightforward way for checking the assignment of those tags is available. One can search for all occurrences of a closed class tag and verify whether each word found in this way is actually a member of that closed class. This can be done automatically, once a list of tags corresponding to closed classes and a list of the few elements contained in each closed class have been enumerated.

Conversely, one can also search for all occurrences of a particular word that is a member of a closed class and check that only the closed class tag is assigned. Some of these words are actually ambiguous, though, so that additional lexical information

would be needed to correctly allow for additional tag assignments for such ambiguous words. More broadly, if a list of unambiguous words can be derived, then checking that there is no ambiguity in their annotation in a corpus is easy.

The WSJ annotation uses 48 tags (including punctuation tags), of which 27 are closed class items. Searching for determiners (DT) we found 50 words that were incorrectly assigned this tag. Examples for the mistagged items include *half* in both adjectival (JJ) and noun (NN) uses, the predeterminer (PDT) *nary*, and the pronoun (PRP) *them*. We have not fully evaluated this method, but looking through four closed classes (CC, DT, IN, and RP), we detected 94 such tagging errors.

In sum, such a closed class analysis seems to be useful as an error detection/correction method, which can be automated and requires very little in terms of language specific resources.

Implementing tagging guide rules

As mentioned in section 1.3.1, many researchers have used hand-written rules to find errors. For example, Baker (1997) discusses that the BNC Tag Enhancement Project used context sensitive rules to fix annotation errors. The rules were written by hand, based on an inspection of errors that often resulted from the focus of the automatic tagger on a few properties in a small window.

Tagging guides such as the one for the WSJ (Santorini, 1990) often specify a number of specific patterns and state explicitly how they should be treated. One can therefore use the same technology as Baker (1997), Oliva (2001) and others and write rules which match the specific patterns given in the manual, check whether the correct tags were assigned, and correct them where necessary. This provides

valuable feedback as to how well the rules of the tagging guide were followed by the corpus annotators and allows for the automatic identification and correction of a large number of error pattern occurrences.

For example, the WSJ tagging manual states: "Hyphenated nominal modifiers ... should always be tagged as adjectives." (Santorini, 1990, p. 12). While this rule appears to be obeyed for 8605 occurrences in the WSJ, there are also 2466 cases of hyphenated words tagged as nouns preceding nouns, most of which are violations of the above tagging manual guideline, such as, for instance, *stock-index* in *stock-index futures*, which is tagged 41 times as JJ and 36 times as NN. We again did not fully explore this method, but we note that several times in the WSJ manual, such hard-and-fast rules are stated, explicitly mentioning certain tags which should or should not accompany certain words or collocations (e.g., *off* in *worse off*).

2.5 Summary for POS annotation

We have introduced a method for detecting annotation errors that remain in gold-standard corpora despite human post-editing and have illustrated it with POS annotation, in addition to presenting two smaller related methods. The proposal detects variation of annotation within comparable contexts and classifies such variation as error or ambiguity using heuristics based on the nature of the context. We showed that an instance of this method based on identity of words in the variation contexts—the so-called variation n-grams—successfully detects a significant number of errors in the WSJ corpus and extends reasonably well to a range of other corpora. The method can be used not only to point to specific errors in the corpus, but also to point to general tagging confusions and problems with the tagging guidelines.

The detection method can be automated, is independent of the particular language and tagset of the corpus, and requires no additional language resources such as lexica. The method can be applied to any corpus with annotation which is positional in nature, and the heuristics can be adapted as long as locality is still a general property of the language and the annotation scheme. As introduced in this chapter, the method works for positional annotations; in the next chapter, we show how the general variation n-gram method can be extended to structural annotation, in particular constituency-based syntactic annotation, where initially there is no one-to-one mapping between annotation and text.

The variation n-gram approach works well for part-of-speech annotation because part-of-speech labels are usually determined by the surrounding context of words. A word appearing multiple times with the same surrounding words is usually being used in the same manner. A question for more complex layers of annotation is whether the same assumption holds, i.e., whether the annotations are also disambiguated by the surrounding context, or whether other features are needed to disambiguate, or whether such disambiguation is not possible based only on the information explicitly present in the corpus. If the annotations can be disambiguated by the surrounding context, then we can continue to use a context of surrounding words to determine if two corpus instances are the same; if the annotation is dependent upon other factors, however, then surrounding words are less helpful and we will have to use those other factors to determine similarity of corpus instances. We will see in the next chapter that syntactic annotation also can be disambiguated by surrounding

words; this should not be surprising, as the results we have seen in this chapter are for part-of-speech, or morphosyntactic positional, labels, which strongly interact with the syntactic structure of a sentence.

CHAPTER 3

SYNTACTIC ANNOTATION

As outlined in the previous chapter, the variation n-gram method is effective for detecting errors in part-of-speech annotation, and the algorithm can be easily applied to any annotation which has a one-to-one mapping between each text token and its annotation label. The algorithm and heuristics were shown to work for a linguistic annotation which is determined by the surrounding context, being composed in chapter 2 of identical words.

Syntactic annotation presents a challenge both computationally and linguistically. First, being comprised of labels over strings of text, syntactic annotation and other structural annotations have no direct one-to-one mapping between a token and the annotation, and therefore many labels dominate a particular word. For example, in figure 3.1 below, the adjective *biggest* is dominated by two different NPs. Thus, detecting errors in syntactic annotation using the variation n-gram method requires a non-trivial extension to the method, which is described in section 3.1, building on Dickinson and Meurers (2003b). Secondly, even if the method can be adapted to handle syntactic units, instead of only individual words, it is still not clear if this will be useful in any practical sense. Syntactic annotation can be affected by items far removed from the immediately-surrounding local context. However, as we will

demonstrate with the results of the method as applied to the WSJ in section 3.2, the variation n-gram method is also effective for detecting errors in syntactic annotation because so many distinctions are indeed still local. We also show in that section the effectiveness of using a more general disambiguating context, that of identical part-of-speech tags. In section 3.3, we turn to a related method of finding errors in treebanks, based on the variation between local trees, and display its impact on the task of grammar induction.

3.1 Detecting variation in syntactic annotation

3.1.1 Defining variation nuclei for syntactic annotation

To adapt the variation n-gram method for the detection of errors in syntactic annotation, we must define what constitutes a nucleus as the unit of data for the comparison of annotations. As described in the previous chapter, for POS annotation single words (tokens) are the unit of data, and each word is paired with a part-of-speech tag as the annotation we are interested in comparing. For syntactic annotation it is not as straightforward to determine a unit of data with a one-to-one relation to the syntactic category annotation since the syntactic category labels annotate constituents, which are strings (token lists) of different length. In other words, the length of the string making up a constituent is determined by the annotation, but we are looking for a theory-independent, data-driven definition of a nucleus, which will allow nuclei to be compared across identical contexts.

As a solution to this problem, we decompose the variation n-gram detection for syntactic annotation into a series of runs with different nucleus sizes. Each run detects the variation in the annotation of strings of a specific length, i.e., the mapping is

between a string and a label. By performing such runs for strings from length 1 to the length of the longest constituent in the corpus, we ensure that all strings which are analyzed as a constituent somewhere in the corpus are compared to the annotation of other occurrences of that string.

Two points are in order about this method for comparing syntactic annotation. Firstly, when comparing the annotation of a string of a specific length, only the category assigned to that entire string is compared. The internal structure of a constituent—the syntactic annotation of its substrings—is inspected when the nuclei of the length of the respective substrings are compared, i.e., during a different run. Secondly, since we organize the variation detection as a data-driven search from strings of a particular length to the syntactic categories assigned to those complete strings, we need to decide how to handle the cases in which a string has an occurrence in which it is not analyzed as a constituent and therefore not assigned a syntactic category. To handle those cases, we assign all non-constituent occurrences of a string the special label NIL. Note that this has the effect that a string occurring multiple times in the corpus without ever being annotated as a constituent will not show up as variation. With such an approach, we are aiming to detect errors in bracketing in addition to category label errors.

Our method does not detect differences in constituents which have one but not both constituency borders within the nucleus, as e.g., the variation between $A[B\underline{\text{X Y}}]C$ and $D[E\underline{\text{X}]\ \text{Y}}F$ or $[DE\underline{\text{X Y}}F]$ (with the nucleus shown in the underlined gray box). One could explore using a range of special labels to encode more information about those differences, but we do not pursue this here since it violates the general idea of our approach to compare the annotation of identical recurring nuclei within a similar

context given that A, B, and C differ from D, E, and F (if they are identical, the variation *is* caught by our approach when the nuclei B X Y, E X, and D E X Y F are investigated).

Let us take a look at an example from the WSJ treebank, the variation 12-gram in (15), which includes a nucleus of size two.

(15) market received its biggest jolt last month from Campeau Corp. , which

The string *last month* is a variation nucleus in this 12-gram because in one instance in the corpus it is analyzed as a noun phrase (NP), as in Figure 3.1 while in another it does not form a complete constituent on its own, as shown in Figure 3.2.

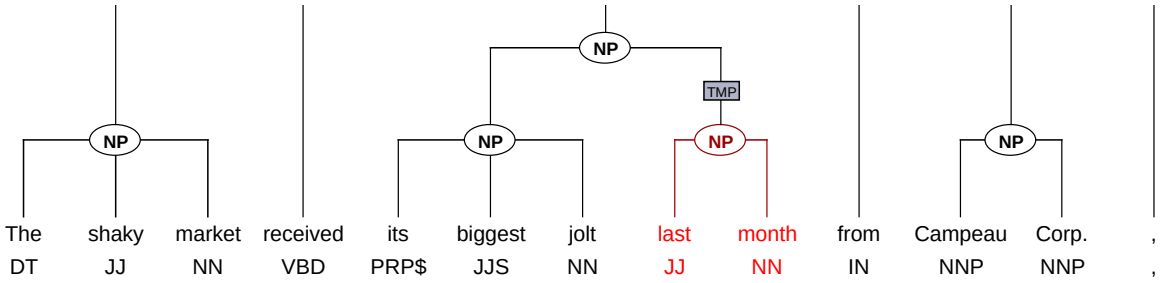

Figure 3.1: An occurrence of "last month" as a constituent

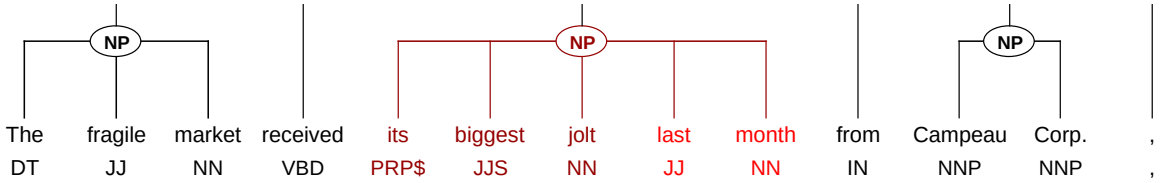

Figure 3.2: An occurrence of "last month" as a non-constituent

As another example, take the variation 4-gram *from a year earlier*, which appears 76 times in the WSJ. Out of those, the nucleus *a year* is labeled noun phrase (NP) 68 times, and 8 times it is not annotated as a constituent (because it is bracketed with *earlier* as an NP). An example with variation between two syntactic categories involves the nucleus *next Tuesday* as part of the variation 3-gram *maturity next Tuesday*, which appears three times in the WSJ. Twice it its labeled as a noun phrase (NP) and once as a prepositional phrase (PP).

Having clarified how we can define the notion of a variation nucleus to detect variation in syntactic annotation and shown that it can be used for detecting category label and bracketing errors, we now take a closer look at how the variation nuclei and the variation n-grams are computed for a given treebank.

3.1.2 Computing the variation nuclei of a treebank

A simple way of calculating all variation nuclei would be to perform the following procedure for all i between 1 and the length of the longest constituent in the corpus: First, step through the corpus and store all stretches of length i with their category label, or with the special label NIL if that list of corpus positions is not annotated as a constituent. Second, eliminate the non-varying stretches.

However, such a generate-and-test methodology which for every corpus position stores the list of words of length i for all i up to the length of the longest constituent in the corpus is clearly not feasible for dealing with larger corpora. Instead, we can make use of the observation from the last section that a variation necessarily

involves at least one constituent occurrence of a nucleus. We thus arrive at the following algorithm to calculate the set of nuclei for a window of length i ($1 \leq i \leq$ *length-of-longest-constituent-in-corpus*):

1. Compute the set of nuclei:
   a) Find all constituents of length i, store them with their category label.
   b) For each distinct type of string stored as a constituent of length i, add the label NIL for each non-constituent occurrence of that string.
2. Compute the set of variation nuclei by determining which of the nuclei were stored in step 1 with more than one label.

In addition to calculating the variation nuclei in this way, we generate the variation n-grams for these variation nuclei as defined in section 2.2.1, i.e., we search for instances of the variation nuclei which occur within a similar context. The motivation for the step from variation nuclei to variation n-grams is that a variation nucleus occurring within a similar context is more likely to be an error, and again we define a similar context as a context of identical words surrounding the variation nucleus.

3.1.3 Context generalization

To increase the recall of the error detection method, one can also experiment with using different types of context, such as requiring the context surrounding the nucleus to consist of identical POS tags instead of identical words. This will allow nuclei which appear next to low-frequency words to be grouped with other strings sharing the same POS labels. Given that treebanks generally also have part-of-speech information

available for every token, implementing this idea requires no extra technology.[47] We thus experimented with using this more general context on the original set of variation nuclei.

3.2 A case study: Applying the method to the WSJ treebank

To test the variation n-gram method as applied to syntactic annotation, we performed a case study with the WSJ corpus as part of the Penn Treebank 3 (Marcus et al., 1999). Before presenting the results of the case study, there are a few points to note about the nature of the corpus and the format we used.

3.2.1 Properties of the corpus

Syntactic categories and syntactic functions The syntactic annotation in the WSJ includes syntactic category and syntactic function information. The annotation of a constituent consists of both pieces of information joined by a hyphen, e.g., the label NP-TMP is used to annotate a constituent with a category of noun phrase (NP) functioning as a temporal modifier (TMP). The syntactic category of a constituent is generally determined by the lexical material in the covered string and the way this material is combined; the syntactic function of a constituent, on the other hand, is determined by the material outside of the constituent. For example, one can determine that the string *last month* is an NP based solely on the string, but we have to look at the surrounding material in order to determine what function *last month* has within the sentence. As a consequence, the consistency test of the mapping from strings to

[47] Grouping some words into other classes, such as numbers, dates, and company names could also be highly beneficial, in that we would expect greater recall with little drop in precision. As this requires additional technology, in the form of an intelligent tokenizer or chunker, we choose not to pursue this here.

their syntactic annotation we are proposing in this chapter is most appropriate for the syntactic category annotation, and we thus focus only on this annotation for our case study. Nevertheless, the variation n-gram approach is also applicable to syntactic function annotation because a variation n-gram is a nucleus within a context, i.e., within an environment which constrains the syntactic function of the nucleus.

For the case study, we used the TIGERRegistry developed at the University of Stuttgart to import the corpus into TIGER-XML format (König et al., 2003). The TIGERRegistry import filter we used removes the function labels from the categories and places them as edge labels onto the edge above the category; e.g., a temporal noun phrase (NP-TMP) becomes a noun phrase node (NP) under an edge labeled temporal (TMP). Thus, the TIGER-XML format allows easy access to the tokens and the syntactic category labels using XSLT (Clark, 1999), and we generate a file with all the (non-NIL) constituents this way. Once the NIL strings have also been calculated, our variation n-gram algorithm then runs on the output of this process.

Null elements In addition to providing the syntactic category and function annotation, the WSJ annotators also modified the corpus text by inserting so-called null elements, e.g., markers for arguments and adjuncts which are realized non-locally, or unstated units of measurement (cf. Bies et al., 1995, p. 59). For example, **T** is inserted to mark the trace of A' movement, marking "the interpretation location of certain constituents that are not in their usual argument position," as in *What*$_1$ *are you thinking about *T**$_1$*?* (p. 61). The syntactic annotation of these empty elements is largely determined through theoretical considerations and the non-local occurrence of linguistic material, not the empty element itself or its local context. In other words,

the variation in the annotation of a null element as the nucleus is largely independent of the local environment so that such nuclei should be ignored when testing for the consistency of the mapping from strings to their syntactic annotation. We will see an illustration of this in the discussion of the case study results in the next section.[48]

Unary branching Finally, we need to discuss a special type of syntactic constituency which is directly relevant when talking about the possible mappings from strings to constituency: categories dominating only a single daughter. The syntactic annotation in the WSJ makes use of such unary branches, which are motivated by theoretical considerations. For our discussion, the important aspect is that a unary branch causes the same string to be annotated by two distinct categories, which would be detected as a variation in the annotation of this string without further modification to the algorithm. To instead obtain the interpretation that the two syntactic categories conjunctively characterize the string, we replaced all unary syntactic structures with a category label consisting of the mother and the daughter category. For example, as discussed in Bies et al. (1995), a quantifier phrase (QP) which is missing a head noun, such as *10 million*, in the WSJ is dominated by a noun phrase (NP) node

[48]Since null elements are inserted by the makers of the treebank, one could consider whether or not the null elements have been consistently inserted throughout the corpus. The variation n-gram method could be adapted for this purpose, but we do not go into details here.

in a unary structure. We replace this structure with the new category label NP/QP. This conversion added 70 syntactic category labels to the original 27 labels[49] used in the WSJ.[50]

3.2.2 Results from the WSJ

Using the syntactically-annotated WSJ corpus—a corpus of 1,253,013 tokens in 49,208 sentences—in the format described above, we ran the variation n-gram method for every possible nucleus size. As shown in Figure 3.3, the WSJ contains constituents from size 1 to size 271, making these the only possible nucleus sizes.

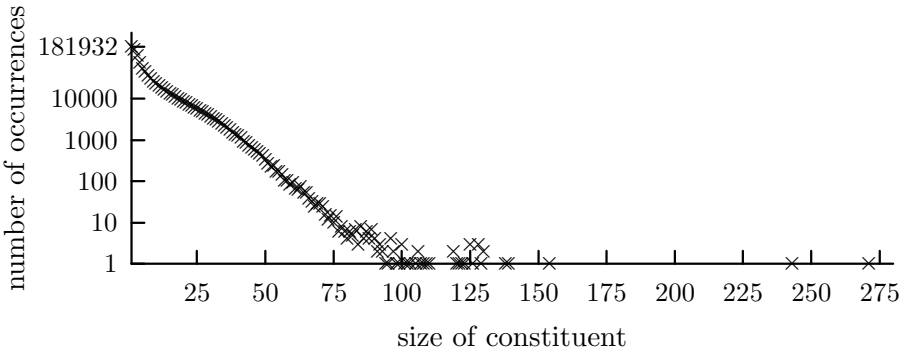

Figure 3.3: Constituent length in the Wall Street Journal Corpus

[49]The manual (Bies et al., 1995) defines 26 category labels; additionally the part-of-speech label CD occurs as a category label in the corpus, which is likely an error. For more on using the manual in this way to detect errors, see section 2.4.

[50]Of the 70 added labels, two are noteworthy in that they display multiple levels of unary branching, namely NP/NP/QP for *3 1/2* and PRN/FRAG/WHADJP for *how incompetent at risk assessment and evaluation*. The former appears in variation and is an error, while the latter appears to be correct.

The largest repeating string with variation in its annotation is of size 46. Figure 3.4 shows the number of variation nuclei for sizes 1 through 46.[51] In total, there are 34,564 variation nuclei, more variation than with the part-of-speech variation analysis where there are 7033 variation nuclei, as shown in figure 2.1 in the last chapter.[52]

i	nuclei	i	nuclei	i	nuclei	i	nuclei	i	nuclei
1	9150	8	122	15	8	22	3	35	2
2	14654	9	87	16	10	23	3	36	1
3	6156	10	69	17	10	24	3	37	1
4	2520	11	45	18	9	26	1	45	2
5	1022	12	37	19	9	27	2	46	2
6	393	13	18	20	6	28	4		
7	196	14	15	21	2	31	2	total	34,564

Figure 3.4: Nucleus size and number of different nuclei

To evaluate the precision of the variation n-gram algorithm, we need to know which of the detected variation nuclei actually include category assignments that are real errors. To do so, we examine the distinct variation nuclei, as we did with POS annotation, where by *distinct* we mean that each corpus position is only taken into account for the longest variation n-gram it occurs in. Furthermore, as shown in section 2.2.3, variation nuclei which appear on the fringe of a variation n-gram (i.e., nuclei which border words that are not part of the variation n-gram) are unreliable for

[51] Nucleus sizes with zero counts are omitted.

[52] This number is what was reported in Dickinson and Meurers (2003b). However, if one changes the way in which traces are handled, there are 36,859 variation nuclei. Originally, tokens such as **T*-1* and **T*-2* were treated uniquely, even though *1* and *2* are simply coindices which point to other parts of the sentence. Grouping all instances of null elements with coindexation markers into the same type, e.g., in this case **T**, gives more variation nuclei. We group all instances of **T** together for *A second test* below and show that the results obtained are parallel.

determining whether there is an error or not. Even though one might think a larger nucleus is as good as a longer context, we make no such assumption because the contextual information still affects arbitrarily long material (discussed more below). Thus, for syntactic error detection, we examined only non-fringe nuclei, giving us a total of 6277 distinct variation nuclei, as shown in Figure 3.5.

i	nuclei	i	nuclei	i	nuclei	i	nuclei	i	nuclei
1	3165	8	92	15	2	22	1	35	3
2	1235	9	75	16	13	23	2	36	2
3	705	10	64	17	10	24	4	37	2
4	338	11	58	18	18	26	5	45	3
5	152	12	37	19	22	27	4	46	4
6	131	13	29	20	6	28	4		
7	82	14	6	21	1	31	2	total	6277

Figure 3.5: Non-fringe distinct nuclei counts

From these 6277, we randomly sampled 100 examples and marked for each nucleus whether the variation in the annotation of the instances of this nucleus was an annotation error or an ambiguity. We found that 71 out of 100 examples were errors. The 95% confidence interval for the point estimate of .71 is (.6211, .7989), i.e., the number of real errors detected in the 6277 cases is estimated to be between 3898 and 5014.

It should also be noted that most of the variations involved were between a regular category and a NIL string. Of the 6277 distinct variation nuclei, 4813 (76.68%) of them were variation between NIL and a single other category, and 130 more nuclei involved variation between NIL and at least two other labels. Thus, methods like

those mentioned in Kaljurand (2004) and Ule and Simov (2004) which do not deal with problems in a "lack of structure" will miss three-quarters of the errors we are able to find. These kinds of errors in bracketing, as discussed in section 3.3, are especially detrimental to natural language processing technologies such as grammar induction algorithms because one cannot hope to learn the correct label for a string if it is not even bracketed correctly.

Ambiguities We examined the 29 ambiguous nuclei in the sample to see if they had certain identifiable properties which could be used to remove them from consideration and increase the precision of the method. Of the 29 ambiguous nuclei, ten are a null element (nucleus size of one) which varies between two different categories, and the ambiguity arises because the null element occurs in place of an element realized elsewhere. For instance, in example (16), the null element **EXP** (expletive) can be annotated as a sentence (S) or as a relative/subordinate clause (SBAR), depending on the properties of the clause it refers to.

(16) a. For cities losing business to suburban shopping centers , it **EXP** $_S$ may be a wise business investment [$_S$ * to help * keep those jobs and sales taxes within city limits] .

b. But if the market moves quickly enough , it **EXP** $_{SBAR}$ may be impossible [$_{SBAR}$ for the broker to carry out the order] because the investment has passed the specified price .

Removing null elements as variation nuclei of size one reduces our set of non-fringe distinct variation nuclei to 5584, and changes our proportion of errors to 71 out of 90 (78.9%). The 95% confidence interval becomes (.7046, .8732), meaning that out of the 5584 examples, we are 95% confident that there are between 3934 and 4875 errors. Thus, eliminating these null elements increases the precision without much of a reduction in the number of estimated errors, as summarized in figure 3.6.

Condition	Precision	Errors
Original	71%	3898–5014
Without nulls	79%	3934–4875

Figure 3.6: Accuracy rates for the WSJ

Another six ambiguities deal with coordinate structures. In the guidelines of the Penn Treebank (Bies et al., 1995), there is a distinction made for simple and complex coordinate elements. Even if an element is simple (i.e., non-modified), it is annotated like a complex element when it is conjoined with one. In figure 3.7, for example, *interest* is part of a flat structure because all the nouns are simple.

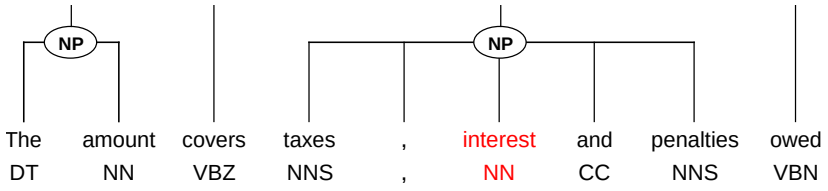

Figure 3.7: An occurrence of "interest" in a flat structure

In figure 3.8, on the other hand, *interest* is an NP because it conjoins with a modified noun, which has to be labeled NP. These coordination ambiguities are fairly systematic, but different from our decision to exclude null elements from being variation nuclei, it is not as clear how to completely eliminate this ambiguity because the conjunct is not always visible in the n-gram. In the next section, however, we will discuss eliminating fringe coordinates from consideration.

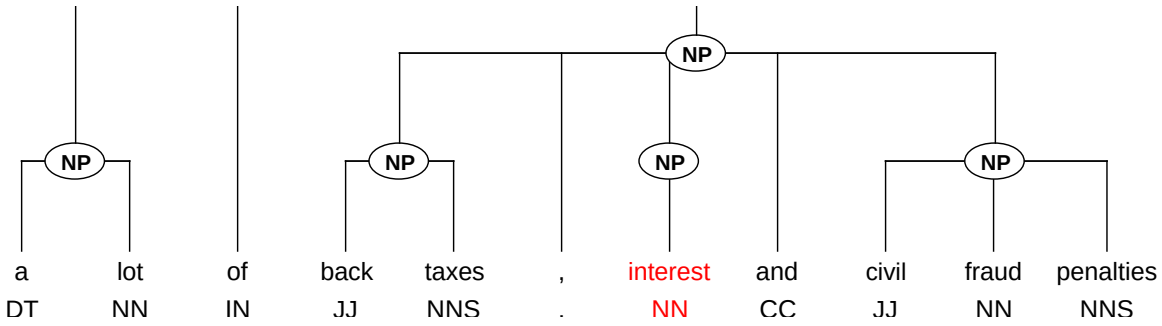

Figure 3.8: An occurrence of "interest" in a complex coordinate structure

One further note is in order regarding inconsistencies in the Penn Treebank. The way the guidelines are written (Bies et al., 1995), there is at times not only an acknowledgment of potential inconsistencies, but what appears to be an acceptance of it. The examples from (17) are taken directly from the bracketing guidelines. Even if one has the same sentence in different parts of the corpus, there is no restriction that they must be annotated in the same manner. Although the task is a difficult one and at times subjective, clear documentation about what to do in problematic cases is necessary; see section 2.2.3 (under *Problematic cases*) for a discussion of corpus annotation guidelines in the context of part-of-speech annotation.

(17) a. "Use of PRN is determined ultimately by annotator intuition" (p. 40).
b. "Certain constructions that are sometimes analyzed using *NOT* are more likely to be annotated more simply, usually using PRN or FRAG or just PP" (p. 90). [Followed by five different possible annotations]
c. (Talking about *wh*-phrases) "There are a number of tricky cases for which there is no policy. ... Possible bracketings are listed for each case" (p. 166).

A second test In addition to using distinct non-fringe variation n-grams to evaluate the precision of the method, we also experimented with examining the shortest possible n-grams which were non-fringe, in order to place a precedence on the "distrust

the fringe" heuristic. This gave us 3965 "shortest" non-fringe n-grams. Note that we have fewer of these than with the distinct variation n-grams evaluation because each type of non-fringe variation n-gram now corresponds to more token occurrences. Also noteworthy is that over half of these—2015 examples—contained nuclei of length one, due largely to the fact that individual words are most likely to be repeated. From all 3965 cases, we sampled 100 and found that 67 of them pointed to an error; four more examples were deemed too unclear to judge. This 67% is consistent with the 71% found using distinct variation n-grams, especially considering the four unclear cases.

The 95% confidence interval for this point estimate of .67 is (.5778, .7622), meaning that there are between 2291 and 3022 errors in this set with 95% confidence. Note that these are counts of the shortest variation nuclei (i.e., recurring strings), and the number of instances of erroneous annotation could be much higher. By their definition, each variation nucleus has at least two instances which differ in their annotation; each variation that is not an ambiguity thus corresponds to at least one instance (but possibly more instances) of erroneous annotation.

Many of the ambiguities were again due to empty elements and nuclei whose only context on either the right or the left was a conjunction. Removing the seven examples with coordinates from the sample results in a precision of 68.82% (64/93), and removing the thirteen examples with null items results in a precision of 75.86% (66/87), paralleling the jump in precision we saw with the distinct variation n-gram method of evaluation. By removing both kinds of examples, we obtain a precision of

78.75% (63/80) and narrow down the set of non-fringe n-grams to 3019 by removing those 946 examples (600 with coordinates as context, 346 with an empty element as the nucleus).[53] The results are summarized in the top half of figure 3.11.

One might suspect that longer nuclei would have a greater precision for finding errors since they have more information in them internally. To get at this question, we broke down our results by nucleus size, as presented in figure 3.9. We see the general trend that with longer nuclei the precision stays high. But often when there is a long nucleus, the context is also the same, so it is unclear if a longer nucleus size, irrespective of context, will be sufficient to detect errors. Judging from this figure alone, it appears as if the unigrams are the worst category, but if we remove the null elements of length one, we find that the precision jumps to 75% (27/36), and it is difficult to say whether the size of the nucleus has any effect on the precision of the method.

For example, in (18) we find a variation nucleus of size 5, *chairman and chief executive officer* surrounded by commas. In (18a) this is an NP, while in (18b) and 19 other examples it is part of a bigger nominal constituent and thus is assigned the category NIL. Despite the relatively long nucleus, this is still an ambiguity because the surrounding context beyond the n-gram is informative as to the nature of this nucleus.

(18) a. Sam Ginn , [$_{NP}$ [$_{NP}$ chairman] and [$_{NP}$ chief executive officer]] , told securities analysts in New York that the company expects somewhat slower per-share earnings growth in 1990 , although annual growth should return to the traditional figure of about 7 % thereafter .

[53] Considering the similarity in precision between the two evaluation methods and the relative computational ease of this second method of evaluation, we will use the shortest n-grams in future evaluation. If we were to similarly re-test with the POS method in chapter 2, this would correspond to looking at only the trigrams.

Size	Errors	Total	Precision
1	28	49	57.14%
2	18	22	81.81%
3	8	13	61.54%
4	5	5	100%
5	2	4	50%
6	2	2	100%
7	1	1	100%
8	1	1	100%
11	1	1	100%
12	1	1	100%
24	1	1	100%

Figure 3.9: Number and precision of errors for each nucleus size with word context

b. Jack Davis , Dataproducts ' [$_{NX}$ [$_{NX}$ president] **,** [$_{NX}$ chairman] and [$_{NX}$ chief executive officer]] **,** said 0 the company " is at a loss * to understand DPC 's intentions . "

Something further should be said about contexts of coordination. Even though coordinating conjunctions are not good indicators of errors, it is possible to simply extend the context further to see if the nucleus still matches. For example, the sentence in (19) appears twice in the WSJ with the variation nucleus as underlined. Here, coordination does not hinder the method because the context beyond the coordinating conjunction is exactly the same in both cases.

(19) Also shown * is *T* the closing listed market price **or/CC** a dealer-to-dealer asked price of each fund 's shares , with the percentage of difference .

Context generalization We experimented with using contexts composed of POS tags, in an effort to increase recall. Indeed, the POS-generalization gave us 9074 shortest non-fringe variation nuclei, more than twice as many as the method with word contexts. Again, over half of these (4653) have nuclei of length one. We sampled

100 of the 9074 cases and found that 52 pointed to an error. We broke this down by nucleus size and found similar trends as with the word contexts, as shown in figure 3.10.

Size	Errors	Total	Precision
1	19	48	39.58%
2	17	30	43.59%
3	4	9	44.44%
4	2	2	100%
5	1	2	50%
8	1	1	100%
9	2	2	100%
10	2	2	100%
12	1	1	100%
13	1	1	100%
17	1	1	100%
20	1	1	100%

Figure 3.10: Number and precision of errors for each nucleus size with POS context

The 95% confidence interval for the point estimate of .52 is (.4221, .6179), which means that we estimate between 3830 and 5607 errors with 95% confidence. The POS generalization, then, detects upwards of 2500 more errors than the regular method with word contexts, and the precision remains above 50%.

As with the regular method, we can also remove null element nuclei (359 examples) and contexts of coordination (1229 examples) from consideration. We only saw one null element nucleus, so it is hard to generalize, but removing null elements gives us a precision of 52.52% (52/99), removing contexts of coordination gives 57.65% (49/85),

and removing both gives 58.33% (49/84), while narrowing down the search to 7486 total shortest variation nuclei. The results for the WSJ are summarized in figure 3.11.

Context	Condition	Precision	Errors
Word	Original	67%	2291–3022
	Without nulls	76%	2420–3070
	Without CC	69%	1999–2632
	Without both	79%	2106–2647
POS	Original	52%	3830–5607
	Without nulls	53%	3719–5435
	Without CC	58%	3698–5346
	Without both	58%	3577–5155

Figure 3.11: Accuracy rates for the WSJ using shortest variation nuclei

A few notes about the POS context generalization are in order. First, we should mention that sometimes the task is exactly the same as with word contexts. Not surprisingly, we sometimes see the same parts of speech in the context because the words in the context are exactly the same. For example, we find the variation n-gram in (20a), where *TO* and . are POS labels, but this amounts to the same thing as (20b), with lexical items in context.

(20) a. TO 1/2 point .
b. to 1/2 point .

This is of course not to say that the context generalization produces trivial results; using POS labels for context turns up many new errors. For example, with the nucleus *little or no* varying between ADJP and NIL, it is not important whether it is followed by *effect* or *loss*; the fact that they are both nouns (NN) helps this error to be located.

Secondly, some of the errors that were found were not errors in structure, but errors in the surrounding context. For example, we have the trigram in (21a) where *competition* varies between NP and NIL. When we examine the NP case, however, we find it as in example (21b).

(21) a. NN competition .
b. forcing the bank to **face/NN** competition **.**

Here, *face* has clearly been wrongly annotated, and this error in the context causes us to detect variation in a context which is not really the same. Since we still discovered an error, even if not a structural one, we grouped these six cases into the 52 erroneous cases.

Thirdly, there were four cases which could not properly be called an error, but which appeared to be typographical errors in the original WSJ text or textual errors introduced when putting the Wall Street Journal into the Penn Treebank format. For example, the string ** beginning in* varies between S and NIL in the context in (22a). The NIL case is clear, as *in* requires a following noun (or indication of such in the Penn Treebank scheme) and thus forms a unit only when that noun phrase is added, as in (22b). The S case is a sentence which abruptly ends with the word *in*, as if the data was chopped off prematurely. We did not group these cases with the errors, but note that we have detected a pattern which violates the rules of English.

(22) a. NNS * beginning in DT
b. * beginning in April 1990

3.3 Grammar rule error detection

Turning to a technique more driven by the annotation but still motivated by a desire for consistency, one can also examine the properties of local trees assigned within a treebank. The approach presented here also focuses on detecting inconsistencies, but instead of focusing on the consistent assignment of a label to a string, as in the approach presented in sections 3.1 and 3.2, we here investigate the consistency of labeling within local trees. Eliminating inconsistencies can have a practical effect on parser performance, in addition to the qualitative improvement of the corpus, and we show in section 3.3.4 that eliminating the detected errors from the training data of a probabilistic context-free grammar (PCFG) parser improves its performance.

3.3.1 Procedure

Most natural language expressions are linguistically analyzed as endocentric, i.e., a category projects to a phrase of the same general category. For example, an adjective will project to an adjectival phrase. This assumption is directly encoded in the widely adopted X-bar schema (Jackendoff, 1977), a generalization over phrase structure rules, and similar generalizations are encoded in virtually all constituency-based syntactic frameworks. An interesting effect of this organization of constituent structure is that one can generally determine the syntactic category of the mother based on the categories of the daughters. We thus propose to systematically search for variation in the mother categories dominating the same daughters in order to find erroneous annotation in local trees.

More concretely, to identify potentially erroneous rules, we extract all local trees from a treebank and index the resulting rules by the daughters list. For each list of daughters (consisting of part-of-speech labels for lexical daughters and syntactic category labels for phrasal daughters), the set of immediately dominating mothers is determined. If this *immediate dominance (ID) set* has more than one element, we will say that a daughters list shows *ID variation* and interpret it as an indication of a potential error. Note that this approach turns the usual conceptualization of local trees as rules on its head: instead of looking for which daughters can expand a fixed mother category, we are fixing the daughters and asking which mother categories can dominate these daughters.

The statistical parsing literature (see, e.g., chapters 11 and 12 of Manning and Schütze (1999) and references therein), often examines the multiple possible daughters of a given mother in order to calculate the probabilities for the rules. The ID variation errors we are interested in affect the probabilities of the daughters of a certain mother. If a daughters list β has mothers A and B and if B is wrong, then the probability for B generating β is incorrect, which affects the probabilities for B generating any other daughters list since all probabilities sum to one.

Let us illustrate the idea of ID variation with an example from the Wall Street Journal (WSJ) as annotated in the Penn Treebank 3 (Marcus et al., 1993). The daughters list ADVP VBN NP (adverbial phrase, past participle, noun phrase) occurs 167 times in the WSJ, with two distinct mother categories: verb phrase (VP, 165 times) and prepositional phrase (PP, 2 times). Based on the endocentricity consid-

erations mentioned above, a past participle verb (VBN) is expected to project to a verbal category, such as VP, but not to a PP—and indeed the two trees dominated by PP turn out to be incorrect.

Note that, like the variation n-gram method, we are here also looking for consistency, but unlike the variation n-gram method, we are driven by the annotation (and not the lexical items) and we look for errors/variations independent of context. The mother category of non-lexical items should in general not vary, irrespective of context.

3.3.2 ID variation in the WSJ

Running the algorithm sketched above on the entire WSJ corpus, we obtain 844 lists of daughters which are assigned more than one mother category in the corpus.

To investigate how many of these variations in mother category point to errors, we randomly sampled 100 of these daughters lists and manually evaluated for each whether it points to an error, a genuine ambiguity, or whether it was unclear according to the annotation guidelines. We only count a daughters list as pointing to an error if for (at least) one of the mothers in the ID set, every occurrence of the daughters with that mother is an incorrect annotation. Of the 100 daughters lists, 74 pointed to an error, 24 were genuine ambiguities, and 2 were unclear. For the total of 844 daughters lists obtained for the full WSJ this means that, based on the 95% confidence interval for the point estimate of .74 (.6540,.8260), between 552 and 697 of the 844 variations point to errors.

Since each pairing of a daughters list and a mother category constitutes an instance of a phrase structure rule, it is interesting to know how many rules the 100 daughters lists with their possible mother categories correspond to. For our sample of 100 daughters lists, there are 291 such rules, i.e., each ID set on average contains 2.91 categories.

Zooming in on the 74 cases pointing to at least one erroneous rule, we want to know how many of the elements of the ID set are errors (it could be anywhere between one and the entire set, in which case none of the annotations are correct). The 74 daughters lists correspond to 223 different rules, and examining all instances shows that 127 of the rules are errors. For example, the daughters list IN NP (preposition/subordinating conjunction, noun phrase) has a nine element ID set (i.e., nine different mothers in the corpus). Three of the elements (PP, FRAG, X) can indeed occur as mothers for the daughters list IN NP. Six of the elements (ADJP, ADVP, NP, SBAR, VP, WHPP) are never correct. In sum, the daughters list IN NP corresponds to six erroneous rules.

Analyzing the nature of the 74 daughter lists pointing to errors, the errors fall into three groups: daughter label errors (38), mother label errors (41), and bracketing errors (13), with some errors having multiple causes. Note that this overlaps very little with the variation n-gram method. In that method, 4813 out of 6277 variations were NIL-based, i.e., highlighted discrepancies in the bracketing of the strings. Thus, it seems that this grammar rule error detection helps locate errors we are not otherwise able to find very robustly.

As an example of a daughter error, we have the daughters DT DT NN, with variation between the mothers NP and QP. One example of this rule annotated as NP is in (23). In this situation, according to the POS guidelines (Santorini, 1990), *all* should be a predeterminer (PDT), not a determiner (DT). We should also note that, while many variation local trees have relatively few token occurrences, for this example we have 19 occurrences (token counts) of the rule NP → DT DT NN and the number of incorrect local trees is thus likely to alter any method training on the corpus. We will return to the issue of token counts shortly.

(23) all the hoopla
DT DT NN

A mother labeling error can be seen with the daughters JJ , NN CC JJ. The mother category varies between ADJP (adjective phrase) and UCP (unlike coordinating phrase). Because we have a noun coordinating with an adjective, it should be UCP in all cases, and thus we have a mother labeling error in example (24), which is annotated ADJP.

(24) the [$_{ADJP}$ scientific , engineering and academic] communities
JJ , NN CC JJ

Although both mother and daughter labeling errors are common, sometimes it may not be clear exactly what kind of error it is. All we can determine is that there is a mismatch between the mother and the head daughter, i.e., the rule is non-projective and violates the principles of endocentricity. For example, we find the single (unary) daughter RP dominated by PRT, ADVP, PP, and ADJP in different corpus occurrences. The guidelines state that PRT is the "[c]ategory for words that should be tagged RP" (Bies et al., 1995). Thus, we can deduce that ADVP, PP, and ADJP are mismatches with the RP POS tag. For example, we find both *up* and *down* in example (25)

annotated with RP POS tags and ADVP category labels. Which labeling is correct may be debatable (RB dominated by ADVP seems most likely here), but it is clear that either the mother or the daughter must change.

(25) may blip [$_{ADVP}$ up] again before they blip [$_{ADVP}$ down]
RP RP

An example of a bracketing error can be seen in the single daughter RBS (superlative adjective). We find examples like (26), where *least* is marked as RBS dominated by NP. The guidelines state that *at least* should be bracketed as a flat ADVP. Thus, by making *least* into a constituent, we have an extra, unnecessary level of structure in the tree.

(26) But Hertz points out that at [$_{NP}$ least] it 's now charging only $ 3.95 *U* a
RBS
day in Texas , while some competitors are charging $ 6.99 *U* .

We mentioned that an example could have multiple kinds of errors, and the daughters list IN NP when it is dominated by VP is a good example. In (27a), we see a bracketing error, where all of *runs*, *up*, and *high commission costs* should have been bracketed as sisters of a mother VP. (The part-of-speech for *up* is also wrong; it should be RP.) In (27b), we see a mother category error, in that *past it* is not a verb phrase, but a prepositional phrase (or perhaps adverbial). And in (27c), we find a daughter error in the clearly erroneous tagging of *like*, tagged as a preposition (IN) when it is obviously a verb (VB).

(27) a. Frequent trading runs [$_{VP}$ up [NP high commission costs]] .
IN

b. Turkey in any event is long [$_{VP}$ past [NP it]] .
IN

c. It turns out 0 Mr. Friend 's client , Machelle Parks of Cincinnati , did n't

[$_{VP}$ like [NP the way 0 defense attorney Tom Alexander acted during the
IN
legal proceedings *T*]] .

In addition to examining the number of types of local trees affected, we also counted the number of local tree tokens which are erroneous in the corpus. Of the 127 tree types (rules) which are erroneous, we find 847 occurrences in the treebank.

Treebank effects Part of the reason that this method of detecting errors in grammar rules implicit in a treebank works well is because of the particular theory used in this treebank. Instead of representing the text exactly as it is, null elements have been inserted, and thus phrasal rules retain their daughters, even if "missing." For instance, in example (28a), we find an SBAR dominating an S and nothing else. Normally, an SBAR will have a complementizer, and—because this treebank uses null elements—it should have a null complementizer even when there is not one overtly. That is, the sentence should look as in (28b), which would mean that SBAR now has a different daughters list, namely -NONE- S.

(28) a. He thought [$_{SBAR}$ [$_{S}$ the moves in the metals last week were most influenced * by the uncertainty in the equity and other financial markets]] .

b. He thought [$_{SBAR}$ 0/-NONE- [$_{S}$ the moves in the metals last week were most influenced * by the uncertainty in the equity and other financial markets]] .

The theoretical generalizations, however, have to account for the data actually found in the corpus, and null elements are not always called for. The guidelines for VP gapping and the like often do not use null elements and could in principle be a

source of variation, yet we obtained quite good results. Thus, it is possible that this work will be also useful for treebanks which maintain a closer connection between the structure and the surface realization.

Regardless of inserted elements or not, there are some properties of the distinctions made in the treebank which can cause problems for any method searching for endocentricity violations, namely properties of the annotation scheme which directly violate endocentricity. The guidelines for proper nouns (words annotated with the POS tag NNP or NNPS) are a good example of this. The rule for POS annotation seems to be that capitalized words which appear in a title should be tagged NNP, as in example (29a), which means that a title like *Saved by the Bell* gets annotated as in (30a). In the syntactic guidelines, however, titles are specified to be annotated like running text (and with a TTL (title) function tag), e.g., as in (29b). Thus, we find the structure for (30a) as given in (30b) for the WSJ. The NNP and its VP mother seem to be unrelated from an endocentric perspective, and it is caused by the fact that the POS and syntactic guidelines diverge in terms of what linguistic facts they are capturing. Using knowledge about the TTL function tag in conjunction with an NNP daughter might be useful in this context, but we do not explore this here.

(29) a. A Tale of Two Cities (Santorini, 1990, p.32)
NNP NNP IN NNP NNP

b. [S-TTL [NP-SBJ *] [VP Driving [NP Miss Daisy]]] (Bies et al., 1995, p. 207)

(30) a. Saved By The Bell
NNP NNP NNP NNP

b. [VP Saved [PP By the Bell]]
NNP

Discussion An important question raised by the presence of errors we have detected is: how does it affect NLP technology? Discussing how incorrect grammar rules cause

problems for treebank grammars, Charniak (1996) lists five reasons why parsers might fail to return the correct analysis for a sentence, and two of these are directly relevant here. In terms of training, parsers can fail to give the correct parse because "the rules are there, but their probabilities are incorrect" (Charniak, 1996, p. 6). As we have shown, with variation between mothers, the probabilities of rules will be incorrect. In terms of testing, as we showed in section 1.2.1, a parser can fail because "it found the correct parse, but the tree-bank 'gold standard' was wrong" (Charniak, 1996, p. 6). The errors we have presented here clearly are problematic for any accurate evaluation.

It is also noteworthy that the method pursued in this section catches three different kinds of errors. Krotov et al. (1998) are able to catch some bracketing errors during grammar compaction, namely by parsing the RHS of rules using other rules and thus adding structure where there (often) should have been structure before. And methods like that of Charniak (1996) use right-bracketing preferences to avoid some bracketing errors. But grammar compaction methods do nothing to remove mother or daughter labeling errors, which together occurred in nearly all of the variation error examples.

Given the potential negative impact of errors detected by our method, we want to know to what degree they are problematic. We will thus examine the task of training different models for supervised grammar induction in section 3.3.4, with and without erroneous rules included. To set up this experiment, however, we first need to detect which specific erroneous rules are wrong since up to this point we have only identified erroneous ID variation sets, and we will do this in section 3.3.3.

3.3.3 Automatic detection of erroneous rules

We have established that ID variation is useful for finding incorrectly annotated local trees and, by extension, the rules licensing these trees. To make this observation practically useful for a large corpus, where hand validation often may not be feasible, we now need to define a heuristic for automatically detecting which of the elements in the ID set of a given daughters list are errors and which are a part of a legitimate ambiguity.

These results can then be used to remove rules from a treebank grammar (as we do in section 3.3.4), if one wishes, but ultimately how these erroneous rules are accounted for in a parsing regime depends on the details of the system.[54]

Frequency-based heuristics For PCFG parsing, it is often assumed that one can prune low-frequency rules without a degradation in parsing performance (Gaizauskas, 1995; Charniak, 1996; Cardie and Pierce, 1998) (although there is some indication that this is not always the case (Bod, 2003)). Based on this idea, one can create a heuristic counting the low-frequency ID variation categories as errors and the frequent ones as genuine ambiguities.

Thus, we first isolated all rules which occurred only once in our sample set, in order to gauge how well this simple and commonly-used method of elimination works. The results are given in figure 3.12, for both type and token counts, where we report how many of the erroneous trees the heuristic identifies out of all the trees it identifies

[54]One could assess penalties for a parse tree which uses a flagged penalty, for example, thereby not eliminating a rule completely but discouraging its use.

(precision) and out of all the trees it should have identified (recall). Since each rule we detect here occurs only one time, the type and token precision figures are exactly the same.

	Precision	Recall
Types	74.75% (74/99)	58.27% (74/127)
Tokens	74.75% (74/99)	8.74% (74/847)

Figure 3.12: Evaluating the single rule occurrence detection heuristic

Figure 3.12 shows high precision, but, because these rules occur once, the token recall is quite low. There are two main reasons that we find this predictor to be insufficient. On the one hand, we find examples of frequently-occurring rules which are incorrect, such as the rule NP $\rightarrow$ VBG which appears 177 times, despite being wrong. On the other hand, there are examples of infrequently-occurring rules which are correct; e.g., NX $\rightarrow$ VBG NN is correct even though it occurs only three times, in comparison to the same daughters occurring 156 times with NP as mother. Of the 99 rules in our set which occur once, a full 25 of them are correct.

The first heuristic looks only at token frequency counts and does not use information found within the variations. To identify more rule occurrences which are wrong, we can use the properties found in a variation to define a second frequency-based heuristic. Thus, for each variation we took the total number of token occurrences of the daughters list and extracted all rules whose token occurrences were less than 10% of the total. For example, there are 86 total times in the corpus where the daughters

list NNP CC NNP NNP appears bracketed as a constituent. NP (noun phrase) is the mother 83 times and thus is likely correct, but UCP (unlike coordinated phrase) is only a mother twice, or in 2.33% of the cases. Thus, UCP is flagged as a likely error, and indeed it is erroneous.

As we can see in figure 3.13,[55] detecting erroneous rules with this 10% metric identifies approximately 60% of the erroneous local tree types which we want to remove (recall). However, in terms of types of local trees, almost half of what it detects is correct data (precision).

	Precision	Recall
Types	60.95% (78/128)	61.42% (78/127)
Tokens	44.28% (499/1127)	58.91% (499/847)

Figure 3.13: Evaluating the under 10% heuristic

The token results have even worse precision, being below 50%. Continuing with the NNP CC NNP NNP example, for instance, we find one occurrence of NX as a mother, but this is correct. And, having more of an impact, we find rules under the 10% threshold despite being quite frequent; for example, the label NX (certain complex noun phrases) appears 102 times as the mother of JJ NN, but NP appears 5972 times as the mother of the same daughters, so the correct label NX is ruled out

[55]One variation, for the daughters list NP VP, had a mother (NP) with 4297 occurrences, yet was below the 10% threshold. Because of its extreme nature, we have removed this data point for these results and more fully explore a heuristic with absolute counts below. The most significantly affected number presented in 3.13 by this removal is, of course, the token precision, which would drop to 8–9%.

by being under the 10% mark. We have dramatically improved the recall from the first heuristic to the second, but for the reasons given here the precision has suffered as a result.

To improve the precision, our third frequency-based heuristic also detects rules using a 10% threshold but now uses an absolute threshold in conjunction with it, to account for the problems stated above. Specifically, we now flag as errors those local tree types below the 10% threshold and which also occurred less than 20 times in the corpus. The results are given in figure 3.14, where we can see that the precision is now higher, but the recall is again much worse.

	Precision	Recall
Types	64.35% (74/115)	58.27% (74/127)
Tokens	89.05% (187/210)	22.08% (187/847)

Figure 3.14: Evaluating the under 10%/under 20 heuristic

The results for these three frequency-based heuristics seem to indicate that we can get high precision or high recall but not both for the task of automatically detecting erroneous rules. The problems we encountered with frequency-based methods were that there are both infrequent correct rules and frequent incorrect rules. Using only frequencies to identify erroneous rules, therefore, can never provide us with 100% results.

Ambiguity-based heuristic Instead of relying solely on low token frequency, we need another property which makes detecting erroneous rules more accurate. Thus,

we combine a token frequency measure with a measure of how likely it is for two categories to be involved in a genuine ambiguity. For example, regardless of the number of token occurrences within a given variation, NP and NX can generally vary between one another, since their distribution is dependent on what comes outside the constituent. The label NX is used for noun phrases which share a modifier with another noun phrase; if there is no shared modifier, then the identical-looking constituent is labeled NP. Both are nominals, and both often have the same set of daughters.

To use a measure of ambiguity, we need to automatically deduce which variation pairs are acceptable variation pairs, and we do so by looking at their distribution over all 844 variations, which the ID variation algorithm provided for the WSJ. In the case of the NP-NX pair, for instance, we see that out of the 844 variations obtained, 114 have variation between NX and NP, the second-highest variation pair, as shown in figure 3.15.

Variation pair	Frequency
ADJP-NP	163
NP-NX	114
ADJP-ADVP	111
ADVP-NP	100
NP-S	95

Figure 3.15: The five most frequent variation pairs

The numbers in figure 3.15 were derived by counting all occurrences of the pair in question across all variations. So, for example, the daughters list NNP CC NNP NNP has the ID variation set of NP, NX, and UCP. We count each of the following pairs once for this variation: NP-NX, NP-UCP, and NX-UCP.

We find that the most frequent variation pairs are indeed pairs of mothers which can often have the same daughters list. Thus, using a frequent pairing as an indication of a potentially acceptable variation provides a way to improve erroneous rule detection.[56]

For our ambiguity-based heuristic, then, we start out with the token frequency measure, where for each variation we take the total number of occurrences of a daughters list and mark as errors all ID set elements whose token occurrences make up less than 10% of the occurrences in their ID set. Then we further restrict the set of potential errors by eliminating all ID set categories which are among the top five categories in variation with the most frequent category. For example, for the daughters JJ NN, because the mother label NX varies with NP throughout the corpus and the two are among the five most frequent mother categories occurring together in ID variation sets, it is not flagged as an error.

A note is in order about how we select the pairings for a given variation—since there are usually multiple possible pairings for a variation, for each mother we have to systematically select which other mother to pair it with. We choose to pair every mother in the ID variation set with the most frequent mother from that set, using the rationale that the most frequent mother is most likely correct and thus provides the most important pairing. For instance, in the example of the daughters list JJ

[56]Another possibility would be to additionally calculate the probability of one mother given another mother.

NN, we have the ID variation set of ADJP, INTJ, NP, NX, and QP. The mother NP occurs most often, and thus the relevant pairings are: ADJP-NP, INTJ-NP, NP-NX, and NP-QP.[57]

The resulting precision and recall figures for automatically detecting which variations are errors with this heuristic is shown in figure 3.16. The results show a better (type and token) precision than with error detection with a 10% threshold (3.13) and a better token recall than with absolute thresholding (figure 3.14).

	Precision	Recall
Types	73.03% (65/89)	51.18% (65/127)
Tokens	65.59% (364/555)	42.98% (364/847)

Figure 3.16: Evaluating the ambiguity pair heuristic

We see this general improvement in token precision and recall because the ambiguity heuristic lets us sort out highly frequent rules based on something other than frequency. For instance, in the example of the JJ NN daughters, ADJP occurs 25 times as a mother but less than 10% of the time within the variation. We ignore the token frequency count of 25, but the pairing ADJP-NP is the most frequent overall pairing, so the rule is (correctly) not detected as an error. On the other hand, with the daughters list IN NP, the mother ADVP occurs 170 times, but the pairing ADVP-PP is not one of the five most frequent, so this is still flagged as an error, and correctly so.

[57]Since the most frequent label (in this case NP) is never flagged as erroneous in these experiments (because it is always above the 10% threshold), it does not matter what we pair the most frequent label with.

Overall, these experiments have shown that it is difficult to decide automatically whether a given rule used in the corpus annotation is correct or not. This fact serves to highlight the necessity of error correction for training data for a task such as supervised grammar induction (e.g., Charniak, 1996; Krotov et al., 1998; Klein and Manning, 2001). However, while the ambiguity-based heuristic for spotting erroneous rules in ID variation sets can clearly be improved, the results are encouraging enough to try to measure the impact of removing all rules detected by this method.

3.3.4 Impact on PCFG parsing

To test the impact of erroneous rules on PCFG parsing, we used the left-corner parser LoPar (Schmid, 2000),[58] which has the nice property that, after training, one can remove rules from the set of pairs of rules and their number of occurrences. For these experiments, we split the WSJ into training (sections 2-21) and test (section 23) data. We did not use the training features of LoPar, but generated the lists of rules ourselves from the training data for parsing, as it was more transparent to remove rules this way and LoPar allows any grammar rule files in the proper format to be used for testing.

We used three different treebank grammars derived from the training data. The first rule file was composed of all grammar rules from the treebank without modification (*All*). The second took that set of rules and removed all rules which occurred only once (*Twice or More*). And the final set of grammar rules took the original set of all rules and removed the rules flagged by the ambiguity heuristic, as described for

[58] We used the unlexicalized, non-headed version. LoPar is available from http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/LoPar-en.html

figure 3.16 (*Pairs*). The resulting number of rules, as well as how many sentences out of the 2132 sentences of 40 words or less LoPar parsed using each rule set, is given in figure 3.17.[59]

	Rules	Sentences
All	15,246	2132
Twice or More	6470	2129
Pairs	14,798	2132

Figure 3.17: The number of rules and number of sentences LoPar was able to parse for the three grammar rule sets

The results of parsing using these three PCFG models on the test data are given in figure 3.18, using the standard PARSEVAL measures (Black et al., 1991), i.e., values for bracketing precision, recall, and F-measure, for both labeled and unlabeled evaluation.[60] Note that the values for *Twice or More* are after removing the 3 sentences which LoPar could not parse.

We see an improvement in precision and recall for both heuristic methods (*Twice or More* and *Pairs*). To see whether the improvement was due to chance or not, we also computed the significance of the precision and recall changes using stratified shuffling.[61] Significant precision and recall changes, as compared to the *All* model,

[59]We selected sentences of 40 words or less in order to make the runtime feasible. Note that an additional 28 sentences—all of them very short—were removed from the testing data because LoPar stopped running while processing them. We have no explanation for why this happened, but as we keep the data constant for all our experiments, the comparison of methods should be valid.

[60]Scores were computed using "evalb" by Satoshi Sekine and Michael John Collins, available at: http://nlp.cs.nyu.edu/evalb/

[61]Specifically, we used the Randomized Parsing Evaluation Comparator written by Dan Bikel and available at: http://www.cis.upenn.edu/ dbikel/software.html.

	Precision		Recall		$F_{\beta=1}$	
	Lab.	Unl.	Lab.	Unl.	Lab.	Unl.
All	70.39%	74.73%	67.31%	71.46%	68.82%	73.06%
Twice or More	70.97%	75.18%	**68.39%**	**72.45%**	69.66%	73.79%
Pairs	***71.48%**	***75.68%**	***68.40%**	***72.42%**	69.91%	74.01%

Figure 3.18: LoPar results using different rule sets

at the $\alpha = 0.05$ level are in bold ($^* =$ at the 0.001 level), so from figure 3.18 we see that the method which removes rules based on information present in their variations (*Pairs*) performs the best, providing significant changes in both precision and recall at the $\alpha = 0.001$ level. In comparison, removing low-frequency rules (Gaizauskas, 1995; Charniak, 1996; Cardie and Pierce, 1998) by eliminating all rules which occur only once resulted in some improvement over the *All* baseline, but not significantly in the way the *Pairs* method does.

Removing rules in the way we have suggested gives the best performance, and this result indicates a few things. Removing rules in a way which accounts for erroneous data can outperform a removal of rules wherein only low-frequency rules are removed. Since we see this increase in parsing performance by removing likely erroneous rules, we might conclude that erroneous rules are harmful for training data. While this result certainly gives a good indication that erroneous rules in the training data are harmful for a parser using an induced grammar, there are a few reasons to be cautious about this conclusion and which offer some avenues of further testing.

The first word of caution is that we are testing on data (section 23 of the WSJ) which has not been cleaned and thus likely contains errors. As argued in section 1.2.1, this means that we cannot be certain that these are the true results. In lieu of the

larger task of cleaning the evaluation data, one could hand-examine a subset of the differences between the parser output, thus allowing us to ignore sentences where both methods were equally correct. Indeed, of the 2132 sentences, the *All* rules and the *Pairs* rules completely agree in 1646 sentences, and only disagree in 486 cases.

The second caveat is that we ran this experiment on a subpart of the WSJ, in order to have distinct training and testing data. The WSJ is the same corpus we used in finding which detection methods work best in the previous section. Even though we are now only using sections 2–21 of the WSJ for training, we have a good idea of what to expect. While this means that this experiment was not truly blind, it also speaks to the fact that it helps to know the nature of one's data in order to separate noise from exceptions. Experimenting on a variety of corpora can better demonstrate how robust the erroneous rule identification methods are.

Related to that point, because we have switched from the full WSJ to a subset, we also have the problem that we have not proven that the removed rules are indeed erroneous. What we can say at this point is that removing some set of rules results in better performance, and that the set we have removed is likely to be mostly composed of erroneous rules. As we saw in figure 3.16, however, the error detection heuristic using ambiguity pairs only has a type precision of 73%, so it is clear that in this set-up we have likely removed some correct rules. In the future, one can examine the particular removed rules to find out more about their properties.

Fourthly, it is not clear if using a different parser for this experiment would result in better performance or not. It might be the case that these rules work well with this particular implementation because of all the different decisions that have to be made for each parser, e.g., the handling of unknown words, the parameter smoothing

algorithm, and the particular search techniques. For example, because LoPar uses a beam search technique to maintain its parsing chart, it is possible that small changes in the rule set will have large effects, regardless of what kind of rules those are. If a low-probability active edge is pruned from the chart, an edge it predicts might actually have a high probability, but we will never see it. By removing rules (and thus potential edges), we may alter/decrease the chances of this happening, but the fact that it does happen is an effect of the parsing regime acting in conjunction with the set of rules it has. To address this concern, one can try the same experiment with a range of different parsers, using different search techniques. One can also experiment with equal-sized sets of randomly-removed rules, to get a feel for whether the differences we are seeing are attributable to the particular qualities of the rules we have removed.

Finally, the PARSEVAL measures are imperfect and can penalize parsers very strongly for a single attachment mistake which proliferates up the tree (Manning and Schütze, 1999, ch. 12). Given this and other well-known problems with the standard PARSEVAL measures (e.g. Carroll et al., 2002), we would also like to explore an evaluation with other methods for comparing parser output.

3.4 Summary for syntactic annotation

We have demonstrated the effectiveness of error detection for treebanks using two related methods: 1) an extension of the variation n-gram method, and 2) a method based on variation in local tree annotation. Both of these methods search for errors by systematically looking for inconsistencies in the data.

We have shown how an approach to treebank error detection based on variation n-grams can be defined and have illustrated with a case study based on the WSJ treebank that it successfully detects inconsistencies in syntactic category annotation. Since such inconsistencies are generally introduced by humans, our method works best for large corpora that have been annotated manually or semi-automatically, which is generally the case for current syntactic annotation and other high-level annotations.

We have described a symmetric, data-driven method that starts from the occurrence of recurring strings and searches for non-terminals that can cover these strings, in contrast to some interannotator agreement methods (e.g. Brants and Skut, 1998). The method handles comparisons between multiple sentences (not just two, as with interannotator agreement) since it looks at all occurrences of a given string in parallel. Furthermore, the approach we have presented automatically determines comparable strings, which can be smaller or larger than a single sentence. (See section 1.3.2 for more details on interannotator agreement.)

The second method of error detection for syntactic annotation, that of finding inconsistent local trees, enjoys some of the same benefits. It too is symmetric, and compares annotations over all sentences, and requires no sophisticated language technology. Furthermore, as we have shown, such a method calls into question the validity of training on erroneous data.

Taken together, these methods serve two main purposes for treebank improvement. They are a means for finding erroneous variation in a corpus, which can then be corrected. And they provide feedback for the development of empirically adequate standards for syntactic annotation, showing which distinctions are difficult to maintain over an entire corpus.

In the next chapter, we will see how the variation n-gram method can be extended to treebanks which allow constituents to be non-contiguous. The method worked for continuous syntactic annotation because for the most part such annotation relies on local distinctions. But, as we saw in example (16), elements which are inherently non-local, such as null elements referring to material an arbitrary distance away, can cause problems for error detection. Discontinuous constituents inherently contain less strict locality, through the segments of the constituent being separated over parts of a sentence, and so it is not certain that the method will be as effective for such an annotation. The next chapter will show that it is indeed effective.

CHAPTER 4

DISCONTINUOUS CONSTITUENTS

4.1 Introduction

We started our error detection research in chapter 2 with annotation which is positional in nature: each corpus position is associated with a part-of-speech tag. We then turned to structural annotation such as that used in syntactic treebanks in chapter 3, which assigns syntactic categories to contiguous sequences of corpus positions. The next step in the variation n-gram research is to extend the method to handle annotation which applies to corpus positions that are non-contiguous. This task is closely related to the previous chapter—find variation for the annotation of multiple positions, as with a constituent—with the added complication of discontinuity. Thus, as with the last chapter, we will examine the effectiveness of the method on syntactic annotation, but this time allowing for crossing branches. For languages with relatively free constituent order, such as German, Dutch or the Slavic languages, the combinatorial potential of the language encoded in constituency cannot straightforwardly be mapped onto the word order possibilities of those languages. As a consequence, the treebanks that have been created for German (NEGRA, Skut et al., 1997; Verbmobil, Hinrichs et al., 2000; TIGER, Brants et al., 2002) have relaxed the requirement that

constituents have to be contiguous. This makes it possible to syntactically annotate the language data as such, i.e., without requiring postulation of empty elements as placeholders or other theoretically motivated changes to the data.

Not only are discontinuous constituents used in treebanks, they are well-motivated theoretically; the concatenation of strings underlying traditional constituency has been rejected by researchers in a wide range of linguistic frameworks that place a premium on the empirical reality of languages other than English, such as Dependency Grammar (Bröker, 1998; Plátek et al., 2001), Tree Adjoining Grammar (Kroch and Joshi, 1987; Rambow and Joshi, 1994), Categorial Grammar (Dowty, 1996; Hepple, 1994; Morrill, 1995), linearization-based Head-Driven Phrase Structure Grammar (Reape, 1993; Kathol, 1995; Richter and Sailer, 2001; Müller, 1999; Penn, 1999; Donohue and Sag, 1999; Bonami et al., 1999), and approaches positing tangled trees (McCawley, 1982; Huck, 1985; Ojeda, 1987; Blevins, 1990). In addition to the empirical and theoretical linguistic motivation, Müller (2004) has argued that grammars which license discontinuous constituents for languages like German should also be preferred on computational grounds. The idea underlying his argument is that in order to license the many word order possibilities (as, for instance, those found in the so-called German *Mittelfeld*) using only continuous constituents, a large number of rules, or equivalent specifications, are needed, resulting in a large number of passive edges during parsing. Since there is no need to distinguish these different word orders in terms of the resulting semantics, positing different rules for each order results only in wasted computational effort (Daniels and Meurers, 2004).[62] In sum, the use

[62]It has been argued that such different word orders correspond to (subtle) semantic differences (see, for instance, Lenerz, 2001). However, until a theory of these differences has been worked out, the best option is to license the indistinguishable word order variations as instances of the same semantic form. For most computational purposes this is also likely to be sufficient in general.

of discontinuous constituents such as found in the NEGRA (Skut et al., 1997), the TIGER (Brants et al., 2002), and the Verbmobil (Hinrichs et al., 2000) treebanks of German seems to be well-motivated for the syntactic annotation of the wide range of languages with relatively free constituent order.

Discontinuous constituents are strings of words which are not necessarily contiguous, yet form a single constituent with a single label, such as the noun phrase *Ein Mann der lacht* in the German extraposition example (31) (Brants et al., 2002).[63]

(31) **Ein** **Mann** kommt , **der** **lacht**
a man comes , who laughs
'A man who laughs comes.'

Treebanks allowing for crossing branches annotate *Ein Mann der lacht* as a single constituent, meaning that there is intervening material which is not part of the constituent. In a treebank only allowing for continuous constituents (as covered in chapter 3), a null element would be inserted, allowing for a non-local reference, or edge labels or some similar convention would be used (see section 4.2 for a discussion of representing discontinuities in treebanks). When represented with crossing branches, the discontinuous constituent can be spread out over a sentence. With such non-locality, it is yet to be proven that an error detection method which relies on local context for information, such as the variation n-gram method, will be useful.

In this chapter, building on Dickinson and Meurers (2005b), we present an approach to the detection of errors in discontinuous structural annotation. We focus on syntactic annotation with potentially discontinuous constituents and show that, once the necessary adjustments are made, the approach successfully deals with the

[63]The ordinary way of marking a constituent with brackets is inadequate for discontinuous constituents, so we instead boldface and underline the words belonging to a discontinuous constituent.

discontinuous syntactic annotation found in the TIGER (Brants et al., 2002) and Verbmobil (Hinrichs et al., 2000) treebanks. In section 4.2 we discuss how discontinuous constituents are represented in treebanks, outlining the scope of treebanks to be covered by this method. In section 4.3 we turn to how the variation n-gram method is extended to deal with discontinuities, making several adjustments for both variation nuclei and variation n-grams. Sections 4.4 and 4.5 present results on the TIGER and Verbmobil treebanks, respectively, the latter showing the applicability of this method to a spoken language corpus.

There are several efficiency issues to deal with in covering discontinuous constituents, but it is important to remember from the outset that finding errors in a treebank with discontinuities is not the same task as parsing discontinuous constituents. In parsing discontinuous constituents, efficiency issues are of an even greater concern (Daniels, 2005), but here we are grounded by the analysis given in the treebank. Although we will at times borrow techniques from the literature on parsing discontinuities (e.g., for representing strings), the tasks are quite different.

4.2 Discontinuous relations in treebanks

A technique such as the variation n-gram method is applicable to corpora with a one-to-one mapping between the text and the annotation. For corpora with positional annotation—e.g., part-of-speech annotated corpora—the mapping is trivial given that the annotation consists of one-to-one correspondences between words (i.e., tokens) and labels. For corpora annotated with more complex structural information—e.g.,

syntactically-annotated corpora—the one-to-one mapping is obtained by considering every interval (continuous string of any length) which is assigned a category label somewhere in the corpus.

While this works for treebanks with continuous constituents, a one-to-one mapping is more complicated to establish for syntactic annotation involving discontinuous constituents (NEGRA, Skut et al., 1997; TIGER, Brants et al., 2002). In order to apply the variation n-gram method to discontinuous constituents, we need to develop a technique which is capable of comparing labels for any set of corpus positions, instead of for any interval.

We should first delineate the space of treebanks which we are setting out to cover. Different encodings of treebanks require different methods for finding variation; for this chapter, we will want to focus on treebanks with explicit crossing branches.

Null elements. Taylor et al. (2003, p. 13) discusses the handling of non-contiguous structures in the Penn Treebank. Working with a formalism which has a context-free backbone, they treat discontinuous constituents by means of null elements (Bies et al., 1995, p. 107-111). Null elements are co-indexed with a non-adjacent constituent, the idea being that in the predicate argument structure the constituent should be interpreted where the null element is. To this end, a class of null elements, referred to as "pseudo-attached" elements, were developed.[64] With these, the context-free structure is maintained, at the cost of inserting null elements as markers of discontinuous relations into the text.

[64]One of the four null elements, *PPA* (permanent predictable ambiguity), was dropped in later annotation projects for the Penn Treebank, namely the Switchboard corpus, since "annotators did not reliably detect these ambiguities" (Taylor et al., 2003).

Our error detection method has two main options for handling these kinds of discontinuities. One option for dealing with a corpus with discontinuous constituents realized through null elements is to handle them as we did with the regular syntactic variation n-gram error detection method, as shown in chapter 3. However, this can amount to ignoring all variation that could be detected through the dislocated part of a constituent: as we mentioned in section 3.2.2, null elements need to be ignored as variation nuclei because the variation in the annotation of a null element as the nucleus is largely independent of the local environment. Accounting for such non-local dependencies (represented by null elements), as we showed, increases the precision of our method and so these "discontinuities" cannot simply be disregarded and treated like continuous constituents.

Another option is to establish a mapping between an annotation scheme with null elements and an annotation scheme which allows crossing branches. Then, one can treat these structures the same way that we treat discontinuous constituents for unordered trees (see section 4.3).

Edge labels. Another kind of approach with a context-free backbone is a treebank annotated with topological fields. For example, the Verbmobil Treebank (Hinrichs et al., 2000) mostly treats discontinuous constituents via edge labels, in order to maintain a topological field analysis in which categories can be split across several fields. To account for these edge labels in the variation n-gram error detection method, one can define a larger set of categories, where each new category is composed of the original category label and any possible function labels such a category can have (e.g., NP-SUBJ, NP-OBJ, etc.). Alternatively, one could use the edge labels to create

an annotation scheme with crossing branches for the discontinuous constituents and then use the method we develop in section 4.3. The simplest possibility is to ignore edge labels completely and run the method as before, to at least catch the errors in continuous strings. As the Verbmobil treebank also contains some discontinuous structures, we will run the method developed in this chapter on it, ignoring edge labels, in section 4.5.

Trees with crossing branches. We now turn our attention to treebanks which encode discontinuities directly, i.e., by means of crossing branches. The most prominent treebanks are the TIGER treebank (Brants et al., 2002), the NEGRA treebank (Skut et al., 1997; Brants et al., 1999), and the analytic layer of the Prague Dependency Treebank (PDT) (Hajičová et al., 1998; Hajič, 1998; Böhmová et al., 2003). In these treebanks, argument structure is represented more transparently by relaxing the definition of a linguistic tree: if a constituent is a daughter of another constituent, it is represented directly as a daughter, whether or not it is contiguous with its sisters. A method applied to such graph annotations will have to be able to deal with constituents discontinuously realized over a sentence. We can see this graphically in figure 4.1, where the discontinuous constituent *kein Arzt der sich auskennt* ('no doctor who SELF is_knowledgable') crosses over several corpus positions. Given that constituents in this representation are no longer contiguous, it is this representation of discontinuous constituents which poses a challenge to the variation n-gram method and will be the focus of our attention for the remainder of the chapter.

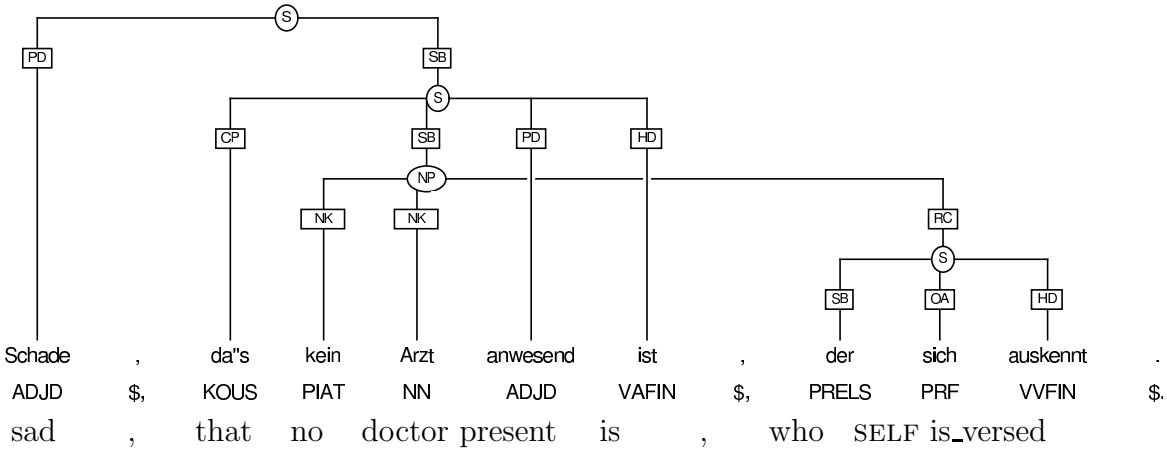

Figure 4.1: An example of crossing branches in the NEGRA corpus

4.3 Extending the variation n-gram method

To extend the variation n-gram error detection method for treebanks with discontinuous constituents, we first have to define the characteristics of a constituent (section 4.3.1), in other words our units of data for comparison. Exploring issues which arise in the literature on parsing with discontinuous constituents (cf. Daniels and Meurers, 2002, 2004 and references cited therein) will help us better define constituents, and by extension all variation nuclei. On the basis of this definition, we can find identical non-constituent (NIL) strings (section 4.3.2) and expand the context into variation n-grams (section 4.3.3).

4.3.1 Variation nuclei: Constituents

Nucleus size For contiguous syntactic annotation, a variation nucleus is defined as a contiguous string with a single label; this design decision allows the variation n-gram method to be broken down into various runs of different constituent sizes. For

discontinuous syntactic annotation, we are still interested in comparing cases where the nucleus is the same. We thus will treat two constituents as having the same size if they consist of the same number of words, regardless of the amount of intervening material. The intervening material is then accounted for when expanding the context into n-grams (see section 4.3.3), and we can again break the method down into runs of different sizes.

Word order A question arises concerning the word order of elements in a constituent. Consider the German subordinate clause example (32) (Müller, 2004).

(32) weil der Mann der Frau das Buch gab.
because the man$_{nom}$ the woman$_{dat}$ the book$_{acc}$ gave
'because the man gave the woman the book.'

The three arguments of the verb *gab* ('give') can be permuted in all six possible ways and still result in a well-formed sentence. It might seem, then, that we would want to allow different permutations of nuclei to be treated as identical. If *das Buch der Frau gab* ('gave the book to the woman') is a constituent in another sentence, for instance, it should have the same category label as *der Frau das Buch gab*, regardless of the ordering of the elements.

Putting all permutations into one equivalence class, however, amounts to stating that all orderings are always the same. But even "free word order" languages are more appropriately called free constituent order; for example, in (32), the argument noun phrases can be freely ordered, but each argument noun phrase is an atomic unit, and in each unit the determiner precedes the noun.

Since we want our method to remain data-driven and order can convey information which might be reflected in an annotation system, we keep strings with different orders of the same words distinct, i.e., ordering of elements is preserved in our method.[65]

String representations In terms of representing the discontinuous constituent string, Johnson (1985) presents two logically equivalent representations in the realm of parsing: interval lists and bitvectors. Interval lists are lists of the locations of the parts of the constituent, and bitvectors represent positions with either a one for an occupied position or a zero for a position not occupied by this constituent. For example, the Guugu Yimidhirr sentence in (33),[66] taken from Johnson (1985), contains a discontinuous constituent, *Yarraga-aga-mu-n biiba-ngun* ('the boy's father'). Assuming the sentence spans from position 0 to position 4, this noun phrase can be represented by an interval as in (34a) or as a bitvector as in (34b). The advantage of the bitvector notation is that many bitvector operations—which are needed for parsing operations—can be calculated in constant time (Daniels and Meurers, 2002).

(33) **Yarraga-aga-mu-n** gudaa gunda-y **biiba-ngun**
boy-GEN-mu-ERG dog-ABS hit-PAST father-ERG
'The boy's father hit the dog.'

(34) a. $[[0, 1], [3, 4]]$
b. 1001

For our purposes, a version of a bitvector encoding is convenient to store the coverage of a variation nucleus within a sentence, and so we choose this representation. Since we have to account for the coverage of the nucleus and of the n-gram, in addition

[65]Preliminary inspection of the nuclei of size two shows that by preserving word order we lose very few examples. Of 26 nuclei pairs which appear in both orders, five would add new variation with free word order, and none of these five provide any new non-fringe variation n-grams.

[66]GEN = 'genitive', ERG = 'ergative', ABS = 'absolutive'.

to positions not yet covered within the span of the string, we distinguish three values for each position in our vector: *1* denotes a position in the nucleus, *X* a position in the context, and *0* a position not (yet) included in the *n*-gram.[67] An example is given in (35), where *sich einig* is the nucleus, *seien sich Bonn einig* is the full *n*-gram (4-gram), and *und London nicht* is not yet in the context. (See (36) for the full context of this example.)

(35) a. **seien** sich **Bonn** und London nicht einig
are SELF Bonn and London not agreed
b. X1X0001

4.3.2 Variation nuclei: Non-constituents

The basic idea is to compare a string annotated as a constituent with the same string found elsewhere—whether annotated as a constituent or not. So we need to develop a method for finding all string occurrences not analyzed as a constituent (and assign them the special category label NIL). Following the discussion in chapter 3, we only look for non-constituent occurrences of those strings which also occur at least once as a constituent. But do we need to look for discontinuous NIL strings or is it sufficient to assume only continuous ones? Consider the TIGER treebank examples (36).

(36) a. in diesem Punkt seien **sich** Bonn und London nicht
on this point are SELF Bonn and London not
einig .
agreed .
'Bonn and London do not agree on this point.'

[67] We only explicitly use *0* for positions within the span of the *n*-gram, i.e., all material between the leftmost and rightmost elements of the *n*-gram (inclusive).

b. in diesem Punkt seien **sich** Bonn und London offensichtlich **nicht einig**
on this point are SELF Bonn and London clearly not agreed
.
.

In example (36a), *sich einig* ('SELF agree') forms an adjective phrase (AP) constituent. But in example (36b), that same string is not analyzed as a constituent, despite being in a nearly identical sentence. We would like to assign the discontinuous string *sich einig* in (36b) the label NIL, so that the labeling of this string in (36a) can be compared to its occurrence in (36b).

In consequence, our approach should be able to detect NIL strings which are discontinuous—an issue which requires special attention to obtain an algorithm efficient enough to handle large corpora.

Use sentence boundary information The first consideration makes use of the fact that syntactic annotation by its nature respects sentence boundaries. In consequence, we never need to search for NIL strings that span across sentences. This restriction clearly is syntax specific and other topological domains need to be identified to make searching for NIL strings tractable for other types of discontinuous annotation.

Use tries to store constituent strings The second consideration concerns how we calculate the NIL strings. To find every non-constituent string in the corpus, discontinuous or not, which is identical to some constituent in the corpus, a basic approach would first generate all possible strings within a sentence and then test to see which ones occur as a constituent elsewhere in the corpus. For example, if the sentence is *Nobody died when Clinton lied*, we would see if any of the 31 subsets of

strings occur as constituents (e.g., *Nobody*, *Nobody when*, *Clinton lied*, *Nobody when lied*, etc.). But such a generate and test approach clearly is intractable given that it generates generates $2^n - 1$ potential matches for a sentence of n words.

Instead of searching for constituents based on strings, we instead split the task of finding NIL strings into two runs through the corpus and look for NIL strings based on constituents. In the first, we store all constituents in the corpus in a trie data structure (Fredkin, 1960), with words as nodes. In the second run through the corpus, we attempt to match the strings in the corpus with a path in the trie, thus identifying all strings occurring as constituents somewhere in the corpus. At each node of the trie, either a path is found to a node for the next word in the sentence or the search is stopped. Continuing our example, if in searching through the trie we look for *Nobody when* and find no path from *Nobody* to *when*, then we stop the search without having to consider *Nobody when lied*. Using a trie dramatically increases the speed of computation; for a corpus of 40,000 sentences, generating all NIL strings took approximately fifteen minutes.

Filter out unwanted NIL strings The final consideration removes "noisy" NIL strings from the candidate set. Certain NIL strings are known to be useless for detecting annotation errors, so we should remove them to speed up the variation n-gram calculations. Consider example (37) from the TIGER corpus, where the continuous constituent *die Menschen* is annotated as a noun phrase (NP).

(37) Ohne diese Ausgaben, so <u>die</u> Weltbank, seien **<u>die Menschen</u>**
without these expenses according to the world bank are the people
totes Kapital
dead capital
'According to the world bank, the people are dead capital without these expenses.'

Our basic method of finding NIL strings would detect another occurrence of *die Menschen* in the same sentence since nothing rules out that the other occurrence of *die* in the sentence (preceding *Weltbank*) forms a discontinuous NIL string with *Menschen*. Comparing a constituent with a NIL string that contains one of the words of the constituent clearly goes against the original motivation for wanting to find discontinuous strings, namely that they show variation between different occurrences of a string.

To prevent such unwanted variation, we eliminate occurrences of NIL-labeled strings that overlap with identical constituent strings from consideration. Two identical NIL strings which overlap, on the other hand, are not ruled out because we do not know in advance which NIL value should correspond to some constituent string in another sentence, and with multiple NIL values in the same sentence, there is no variation within that sentence.[68] Removing these noisy non-constituent strings from consideration will improve the efficiency of calculating variation n-grams, as will be described shortly.

4.3.3 Variation n-grams

The more similar the context surrounding the variation nucleus, the more likely variation in the annotation of a nucleus is an error. For syntactic annotation with proper tree structures (see section 3.1), the context was defined simply as being elements to the right and the left because the nuclei were contiguous units. When nuclei can be discontinuous, however, there can also be *internal context*, i.e., elements which appear between the words forming a discontinuous variation nucleus.

[68] This filter may be less desirable for other kinds of discontinuous annotation, such as co-reference annotation, where a word may be in relation to more than one constituent.

Parsers for continuous constituents often attempt to tackle the input string in a left-to-right or right-to-left fashion. As Kasper et al. (1998) point out, for grammars with discontinuities, "a strict left-to-right or right-to-left approach may be less efficient than a bidirectional or non-directional approach." In like fashion, as in our earlier work, an instance of the a priori algorithm is used to expand a nucleus into a longer n-gram by stepwise adding context elements. Where previously it was possible to add an element to the left or the right, we now also have the option of adding it in the middle—as part of the new, internal context. But depending on how we fill in the internal context, we can face a serious tractability problem. Given a nucleus with j gaps within it, we need to potentially expand it in $j + 2$ directions, instead of in just 2 directions (to the right and to the left).

For example, the potential nucleus *was werden* ('what becomes') appears as a verb phrase (VP) in the TIGER corpus in the string *was ein Seeufer werden* ('what a coast of a lake becomes'), as in (38a); elsewhere in the corpus *was* and *werden* appear in the same sentence with 32 words between them, as in (38b).

(38) a. was ein Seeufer werden
what a coast of a lake becomes

b. was in . . . ein . . . mittragen werden
what in . . . a . . . jointly carry will

The chances of one of the middle 32 elements matching something in the internal context of the VP is relatively high, and indeed the twenty-sixth word is *ein*. However, if we move stepwise out from the nucleus in order to try to match *was ein Seeufer werden*, the only options are to find *ein* directly to the right of *was* or *Seeufer* directly to the left of *werden*, neither of which occurs, thus stopping the search.

In conclusion, we obtain an efficient application of the a priori algorithm by expanding the context only to elements which are adjacent to an element already in the n-gram. Note that this was already implicitly assumed for the left and the right context, used in chapter 3 for ordinary syntactic annotation.

There are two other efficiency-related issues worth mentioning. Firstly, as with the variation nucleus detection, we limit the n-grams expansion to sentences only. Since the category labels do not represent cross-sentence dependencies, we gain no new information if we find more context outside the sentence, and in terms of efficiency, we cut off what could potentially be a very large search space.[69] In the TIGER corpus, for example, stopping n-grams at sentence boundaries prevents expanding a nucleus to a variation 1705-gram.

Secondly, the methods for reducing the number of variation nuclei discussed in section 4.3.2 have the consequence of also reducing the number of possible variation n-grams. For example, in a test run on the NEGRA corpus, we allowed identical strings to overlap; this generated a variation nucleus of size 63, with 16 gaps in it, varying between NP and NIL within the same sentence. Fifteen of the gaps can be filled in and still result in variation. The filter for unwanted NIL strings described in the previous section eliminates the NIL value from consideration. Thus, there is no variation and no tractability problem in constructing n-grams.

[69]Note that similar sentences which were segmented differently could potentially cause varying n-gram strings not to be found. For example, in one part of the TIGER corpus *FRANKFURT A. M. (dpa) .* is a sentence, while in another part *FRANKFURT A. M.* and *(dpa) .* are separate sentences. We think that such errors are best treated in a separate sentence segmentation error detection phase.

Generalizing the n-gram context

So far, we assumed that the context added around variation nuclei consists of words. As with syntactic annotation, given that treebanks generally also provide part-of-speech information for every token, we experimented with part-of-speech tags as a less restrictive kind of context. The idea is that it should be possible to find more variation nuclei with comparable contexts if only the part-of-speech tags of the surrounding words have to be identical instead of the words themselves. As we will see in section 4.4, generalizing n-gram contexts in this way indeed results in more variation n-grams being found, i.e., increased recall.

4.3.4 Adapting the heuristics

To determine which nuclei are errors, we can build on the two heuristics discussed in previous chapters—trust long contexts and distrust the fringe—with some modification, given that we have more fringe areas to deal with for discontinuous strings. In addition to the right and the left fringe, we also need to take into account the internal context in a way that maintains the non-fringe heuristic as a good indicator for errors. As a solution that keeps internal context on a par with the way external context is treated in previous chapter, we require one word of context around every terminal element that is part of the variation nucleus. As discussed below, this heuristic turns out to be a good predictor of which variations are annotation errors; expanding to the longest possible context, as in chapter 2, is not necessary.

4.4 Results on the TIGER Corpus

We ran the variation n-gram error detection method for discontinuous syntactic constituents on the TIGER corpus, version 1.0 (Brants et al., 2002), a corpus of 712,332 tokens in 40,020 sentences. The method detected a total of 10,964 variation nuclei. From these we sampled 100 to get an estimate of the number of errors in the corpus which concern variation.[70] Of these 100, 13 variation nuclei pointed to an error; with this point estimate of .13, we can derive a 95% confidence interval of (0.0641, 0.1959), which means that we are 95% confident that the true number of variation-based errors is between 702 and 2148. The effectiveness of a method which uses context to narrow down the set of variation nuclei can be judged by how many of these variation errors it finds.

Based on the usefulness of the non-fringe heuristic in chapters 2 and 3, we selected the shortest non-fringe variation n-grams to examine. Occurrences of the same strings within larger n-grams were ignored, so as not to artificially increase the resulting set of n-grams. (See section 3.2.2 for a full discussion of evaluation methods.)

When the context is defined as identical words, we obtain 500 variation n-grams. Sampling 100 of these and labeling for each position whether it is an error or an ambiguity, we find that 80 out of the 100 samples point to at least one token error. The 95% confidence interval for this point estimate of .80 is (0.7216, 0.8784), so we are 95% confident that the true number of error types is between 361 and 439. Note that

[70]Special thanks to George Smith and Robert Langner of the TIGER team at the University of Potsdam for evaluating our results.

the precision of 80% is better than the estimates for continuous syntactic annotation of 67% (with null elements) and 76% (without null elements), obtained for the shortest variation n-grams in figure 3.11. The results are summarized in figure 4.2.

	Context	Precision	Errors
WSJ (Continuous)	Word	67–76%	2291–3022
	POS	52%	3830–5607
TIGER (Discontinuous)	Word	80%	361–439
	POS	52%	632–926

Figure 4.2: Accuracy rates for the different contexts

When the context is defined as identical parts of speech, we obtain 1498 variation n-grams. Again sampling 100 of these, we find that 52 out of the 100 point to an error, and the 95% confidence interval for this point estimate of .52 is (0.4221, 0.6179), giving a larger estimated number of errors, between 632 and 926. It is noteworthy that 483 of the examples were supersets of examples in the word context cases; thus, by running only the POS context method, one can find most of the variations that were found with the word contexts, namely 483 of the 500 variations.[71]

Words can convey more information than part-of-speech tags, and so we see a drop in precision when using part-of-speech tags for context, but these results highlight a very practical benefit of using a generalized context. By generalizing the context, we maintain a precision rate of approximately 50%, and we substantially increase the

[71]The reason that 17 cases are not found by the POS context method is that there is variation in how the surrounding context is tagged; we have not tested whether this is more or less indicative of an error.

recall of the method. There are, in fact, likely twice as many detected errors when using POS contexts as opposed to word contexts. These results, then, support the trends observed for regular syntactic annotation in section 3.2.2 and summarized in figure 4.2: a drop in precision to approximately 50% accompanied a huge gain in the number of errors found. Corpus annotation projects willing to put in some extra effort thus can use this method of finding variation n-grams with a generalized context to detect and correct more errors.

Given that the results were so similar to those obtained for continuous syntactic annotation in the general percentages,[72] one can ask if the extra work for discontinuities was necessary. We have to ask if the examples we derived were actually discontinuous or not. Of the 500 word context cases, 125 (25%) involved a discontinuity in at least one of the occurrences, and 157 (10%) of the 1498 POS context cases involved a discontinuity. Of the 100 samples of each, 15 of 19 examples involving discontinuities for the word context and 10 of 11 examples for the POS context were erroneous. While there are certainly more continuous constituents than discontinuous in the corpus, these numbers show that accounting for discontinuous labeling is important in such a corpus.

[72] As shown in figure 4.2, the absolute number of errors found is different, but there are several factors we can attribute this to: difference in corpus sizes (1.2 million (WSJ) vs. 700,000 (TIGER), differences between English and German, and, perhaps most importantly, differences in the original quality of the annotation.

4.5 Spoken language corpora

Spoken language differs in many respects from written language, but the issue of error detection in spoken language corpora has not been well addressed.[73] This is significant since spoken data is increasingly relevant for linguistic and computational research, and such corpora are starting to become more readily available, as illustrated by the holdings of the Linguistic Data Consortium (http://www.ldc.upenn.edu). We address this issue here in a pilot study based on Dickinson and Meurers (2005a), using the German Verbmobil treebank (Hinrichs et al., 2000) as an exemplar of a spoken language corpus and one which contains discontinuous constituents. We discuss the properties of such corpora which are relevant when adapting the variation n-gram approach to spoken language corpora. We also use this smaller corpus to test the utility of using sentence boundaries for detection.

4.5.1 The Verbmobil corpus

For our experiments, we used 24,901 sentences (248,922 tokens) of the German Verbmobil corpus (Hinrichs et al., 2000).[74] This corpus is domain-specific, consisting of transcripts of appointment negotiation, travel planning, hotel reservation, and personal computer maintenance scenarios.The speech was segmented into *dialog turns*, in order to take into account repetitions, hesitations, and false starts; these are akin to sentences found in written language, but dialog turns "may consist of one or more sentences in the grammatical sense" (Stegmann et al., 2000, p. 22).

[73]See section 2.3.1, though, for some issues arising for error detection regarding spoken language corpora with respect to POS annotation.

[74]Specifically, we used the treebank versions of the following Verbmobil CDs: CD15, CD20, CD21, CD22, CD24, CD29, CD30, CD32, CD38, CD39, CD48, and CD49.

The annotation of the Verbmobil corpus consists of tree structures with node and edge labels (Stegmann et al., 2000). The node labels refer to one of four different levels of syntactic annotation: the highest level of nodes specify the turn type; field-level nodes give topological field names; phrase-level nodes indicate syntactic categories; and lexical-level nodes encode the part-of-speech using the Stuttgart-Tübingen TagSet (STTS, Schiller et al., 1995; Thielen and Schiller, 1996). Thus, the node labels in the tree encode both syntactic category and topological field information. Edge labels on the phrase level encode grammatical functions.

While many structures annotated using crossing branches in other corpora, such as TIGER (Brants et al., 2002) are encoded in the Verbmobil corpus using edge labels, the Verbmobil corpus does contain some discontinuous structures, i.e., category labels applying to a non-contiguous string. The discontinuities were often over punctuation, which is unattached in the corpus. Thus, we ran the version of the variation n-grams method for syntactic annotation that is suitable for handling discontinuous constituents, as described in section 4.3 above.

Before turning to a discussion of the results of running the resulting algorithm on the Verbmobil corpus, there are two interesting aspects of the corpus that should be discussed, given that they are typical for such a spoken language corpus and directly affect the variation n-gram error detection method. The first is repetition, arising because people engaged in a dialogue on a specific topic tend to express the same contents; thus, one encounters the same strings again and again in a corpus. For example, one finds 35 instances of (39) in the Verbmobil corpus, with *guten Tag*

labeled 33 times as DM/NX (discourse marker dominating a noun phrase)[75] and twice as NIL (non-constituent). This kind of repetition is readily exploited by the variation n-gram approach.

(39) , guten Tag , Frau
, good day , woman
'Hello, Ms. ...'

A different kind of recurrence, however, is that of identical words appearing next to each other, often caused by hesitations and false starts. For example, we find the unigram *und* 'and' in the middle of the trigram *und und Auto*, as in (40).

(40) und und Auto
and and car

The problem with such examples is that with the same word being repeated, the surrounding context is no longer informative: it results in differences in context for otherwise comparable variation n-grams, as well as the opposite, making contexts comparable which otherwise would not be. False starts and hesitations involving single words can be identified and filtered out prior to error detection, but longer false starts are difficult to detect and can be confusable with ordinary sentences that should not be filtered out, such as in the English *he said he said hello*.

4.5.2 Results

Turning to the results, in our first experiment we started with the version of the variation n-gram algorithm for discontinuous constituents that uses the boundaries of the largest syntactic units as stopping points for n-gram expansion to ensure efficient processing. As shown in Figure 4.3, this resulted in 9174 total variation nuclei.

[75] As discussed at the end of section 4.5.2, we collapse unary branches into a single label.

From this set, we extract the shortest non-fringe variation nuclei, thereby ignoring occurrences of the same strings within larger n-grams. This resulted in 1426 shortest non-fringe variation nuclei.

size	nuclei	non-fringe nuclei	size	nuclei	non-fringe nuclei
1	1808	897	8	47	2
2	2777	252	9	26	1
3	2493	135	10	12	1
4	1223	80	11	6	0
5	482	35	12	3	0
6	200	13	13	1	0
7	95	10	14	1	0

Figure 4.3: Number of variation nuclei in the Verbmobil corpus

It is useful to compare this result to that obtained for the newspaper corpus TIGER (Brants et al., 2002), as described in section 4.4. The Verbmobil corpus is roughly one-third the size of the TIGER corpus, but we obtained significantly more shortest non-fringe variation nuclei for the Verbmobil corpus (1426) than for TIGER (500), indicating that the Verbmobil corpus is more repetitive and/or includes more variation in the annotation of the repeated strings. This supports the reasoning in section 4.5.1 that the variation n-gram approach is well-suited for domain-specific spoken language corpora, such as the Verbmobil corpus.

The effect of dialog turn boundaries

In another experiment, we explored the effect of using the boundaries of the largest syntactic units in the corpus, i.e., the dialog turn boundaries, as stopping points for n-gram expansion. Allowing variation n-grams to extend beyond a dialog turn resulted in 1720 cases, i.e., 20% more than in our first experiment, where variation detection was limited to a single dialog turn; a complete breakdown is given in Figure 4.4. In conclusion, the second experiment shows that repeated segments frequently go beyond a dialog turn so that error detection for spoken language corpora should ignore dialog turn boundaries.

size	nuclei	non-fringe nuclei	size	nuclei	non-fringe nuclei
1	1808	1081	8	47	2
2	2777	307	9	26	1
3	2493	169	10	12	1
4	1223	95	11	6	0
5	482	39	12	3	0
6	200	14	13	1	0
7	95	11	14	1	0

Figure 4.4: Number of variation nuclei, ignoring dialog turn boundaries

The effect of punctuation

Finally, in a third experiment, we investigated the role of punctuation, which had been inserted into the transcribed speech of the Verbmobil corpus. We removed all punctuation from the corpus and reran the error detection code (in the version ignoring dialog turn boundaries). This resulted in 1056 shortest non-fringe variation

nuclei, a loss of almost 40% of the detected cases compared to the second experiment. The punctuation inserted into the corpus thus seems to provide useful context for detecting variation n-grams.

However, even though punctuation appears to be useful in finding more variations, punctuation symbols are not always reliable indicators of identical context. To illustrate this, let us examine some different uses of the comma. In (41), we find commas delimiting elements of an enumerated list, and thus *Freitag* ('Friday') forms a noun phrase (NX) by itself.

(41) das wäre Donnerstag , Freitag , Samstag .
that would be Thursday , Friday , Saturday .

In example (42), on the other hand, we have a comma being used in a date expression. In this case, *Freitag* correctly forms part of the larger noun phrase *Freitag den achten Mai* ('Friday, the eighth of May'), where the comma is used to separate the day from the month.

(42) ab achten Mai , Freitag , den achten Mai , hätte ich für vier Tage Zeit
from eighth May , Friday , the eighth May , would've I for four days time

The comma and other punctuation thus are potentially ambiguous tokens, with different uses and meanings, essentially on a par with ordinary word tokens—an observation which is not specific to spoken language but equally applies to written language corpora.

The effect of the annotation scheme

Turning to the annotation scheme used in the Verbmobil corpus and its effect on error detection using the variation n-gram detection method, there are two issues that deserve some attention: non-local category distinctions and the role of topological fields.

Non-local distinctions Clearly the most serious problem for our error detection method (and for most algorithms for corpus annotation) are category distinctions which are inherently non-local. We use the non-fringe heuristic to isolate variation n-grams for analysis, but some nuclei cannot be reliably disambiguated using the local context. For example, the nucleus *fahren* ('drive') in the 4-gram given in (43) is ambiguous; in (44) it is a finite verb (VXFIN), but it is a non-finite verb (VXINF) in (45).

(43) nach Hannover fahren .
to Hannover drive .
'to drive to Hannover'

(44) daß wir am Mittwoch und Donnerstag nach Hannover fahren .
that we on Wednesday and Thursday to Hannover drive .
'that we drive to Hannover on Wednesday and Thursday.'

(45) Wir wollten nach Hannover fahren .
we wanted to Hannover drive .
'We wanted to drive to Hannover.'

In (44), *fahren* ('drive') is a finite verb in third person singular, occurring in a dependent clause. In (45), on the other hand, the verb *wollten* ('wanted') is the finite verb in a declarative sentence and it selects the infinitival *fahren* ('to drive'). The problem is that looking solely at *fahren* and its local context, it is not possible to determine whether one is dealing with the finite or the non-finite form since the finite verb in a declarative sentence like (45) typically occurs as the second constituent of a sentence, which is arbitrarily far away from the non-finite verbs typically found at the right edge of a sentence. To be able to distinguish such cases, the variation n-gram detection method thus would need to be extended with a more sophisticated notion of disambiguating context that goes beyond the local environment.

Topological fields When we introduced the annotation of the Verbmobil corpus in section 4.5.1, we mentioned that the annotation includes topological field information. The topological field labels encode the general word order properties of the entire sentence, not the properties of a word and its local context. As a result, they can cause problems similar to the just discussed non-local category distinctions. For example, the complementizer field C is described as follows in the manual: "The C-position only occurs in verb-final clauses" (Stegmann et al., 2000, p. 11), but whether a clause is verb-final or not is a property of the sentence, not of the C field itself.

A second issue involving the topological fields arises from the fact that the annotation scheme includes two kinds of non-terminal nodes: field-level nodes that bear topological field labels and phrase-level nodes with syntactic category labels. This has the effect that some phrases are dominated by phrases, whereas others are dominated by fields—but clearly one does not want to compare field labels with category labels.

A case where this becomes directly relevant to the detection of annotation errors is the treatment of unary branches in the syntactic annotation. Since both nodes in a unary branching tree dominate the same terminal material, we proposed in section 3.2.1 that such unary branches are best dealt with by folding the two category labels into a single category. For example, an NP (noun phrase) dominating a QP (quantifier phrase) in the WSJ corpus is encoded as a node of category NP/QP. Such folding can involve any non-terminal node dominating a single non-terminal daughter, so that in the Verbmobil corpus it can also combine a topological label with a syntactic category label. For instance, we find NF/NX for a Nachfeld (NF, used for extraposed material) dominating a noun phrase (NX).

Such labels combining field and syntactic category information can cause problems by introducing artificial variation. Consider, for example, a variation nucleus involving *je nachdem* ('depending on') in (46), which varies between PX (prepositional phrase) and VF/PX (Vorfeld, i.e., fronted material, dominating a prepositional phrase).

(46) **und** je nachdem **,**
and depending on ,

This sort of variation is perfectly acceptable, i.e., not an error, since the topological field (Vorfeld) refers to where the prepositional phrase is placed in the sentence and thus is only indirectly related to the internal properties of the PX. For the purpose of our error detection approach, we thus need to keep the topological field nodes clearly distinct from the syntactic category nodes.

Finally, the issue of the topological field information included in the annotation of the Verbmobil corpus highlights that it is important to understand the nature of the corpus annotation scheme when porting the variation n-gram method to a new type of annotation scheme.

4.5.3 Summary for spoken language

As our pilot study on the German Verbmobil corpus indicates, the variation n-gram method seems well-suited for detecting errors in the annotation of such corpora given that repetitions are prevalent in domain-specific speech. At the same time, error detection in spoken language corpora requires special attention to the role of segmentation, inserted punctuation, and particularly the nature of repetition and its causes.

In the future, we plan on fully evaluating the number of errors detected by the method, after identifying and removing the problematic patterns mentioned above. We would also like to apply the method to other layers of annotation in the German Verbmobil corpus, such as part-of-speech annotation, and to test the general applicability of the insights we gained from working with the Verbmobil corpus by applying the method to other spoken language corpora, e.g., the ATIS corpus (Hemphill et al., 1990).

4.6 Summary for discontinuous annotation

In this chapter we have described a method for finding errors in corpora with graph annotations. We showed how the variation n-gram method can be extended to discontinuous structural annotation. We showed how this can be done efficiently and with as high a precision as we obtained for continuous syntactic annotation. Our experiments with the TIGER corpus confirm the results from chapter 3 that generalizing the context to part-of-speech tags can increase recall and that this method can have a substantial practical benefit when preparing a corpus with discontinuous annotation. Furthermore, our work on the Verbmobil corpus indicates that the variation n-gram method can successfully be applied to spoken language corpora.

By applying the method to three different kinds of annotation—positional, structural, and graph—we are now in a position to state what must be true of a corpus for the method to work. First, there must be some discrete element or group of elements which maps to a single annotation; these elements can then be defined as the variation nucleus.

Secondly, there must be some sort of disambiguating context. We initially defined this as identical words and then backed off to identical parts of speech, as both words and parts of speech are relevant for disambiguating the morphosyntactic annotations we have examined. But the context need not be so restrictive. We could have used features akin to those used in part-of-speech taggers, such as the preceding and following two tags and defined a distance metric threshold where two elements are considered the "same," using techniques developed for, e.g., memory-based learning (Daelemans et al., 1996, 1999). By using more general features, which approximate the features used in the NLP task the corpus is generally used for, we will better be able to extend the method to other forms of annotation.

Extending the error detection method to handle discontinuous constituents, as we have done in this chapter, has significant potential for future work given the increasing number of free word order languages for which corpora and treebanks are being developed. In addition to their use in syntactic annotation, discontinuous structural annotation is also relevant for semantic and discourse-level annotation—essentially any time that graph structures are needed to encode relations that go beyond ordinary tree structure. Such annotations are currently employed in the mark-up for semantic roles (Kingsbury et al., 2002) and multi-word expressions (Rayson et al., 2004), as well as for the annotation of spoken language corpora or corpora with multiple layers of annotation (Blache and Hirst, 2000). Future explorations of the variation n-gram method can be applied to these kinds of annotation, but the method will only be effective if the heuristics and the notion of context are adapted. The variation n-gram notion of context works for POS and syntactic annotations because the

immediately surrounding context gives an indication of the morphosyntactic properties of the focus word, or string, and it is these properties which are being labeled. The context also serves to disambiguate the semantic properties of a string, but it is less clear how this is to be exploited.

Having explored the variation n-gram method for detecting errors in three kinds of annotation with varying levels of complexity, we turn in the next two chapters to methods for automatically correcting these errors. With the precision for detection as high as it is, there is less of a need for automatic correction, given that manual correction of errors detected with high precision is feasible for the gold-standard annotation we have been focusing on. However, by exploring methods for correction we can gain insight into the properties of the corpus and of the methods, and we can speed up the correction process.

CHAPTER 5

AUTOMATIC CORRECTION OF VARIATION ERRORS

5.1 Introduction

Having successfully detected errors for various levels of annotation in chapters 2, 3, and 4, we turn now to methods for correcting these errors. Since we detected *types* of strings which are wrongly annotated, the process of correction involves two steps: 1) detecting the exact tokens which are erroneous, and 2) assigning a correct label for erroneous tokens. We will conflate the two steps into one, but we will return to this issue in section 6.5.

Errors which are variations in the annotation of a corpus, as we have detected, stem from the property of natural language being ambiguous. The problem of ambiguity is that more than one label may in general be possible for a given string (word or sequence of words). The context narrows down the possibilities of labels to only one (or a few). For part-of-speech annotation, the usage of a word in a sentence generally determines the label, and so classifiers[76] trained on the surrounding context of tags are able to automatically disambiguate the annotation to a single label with high

[76]We will use the term *classifier* to refer to any technology which assigns a class to a data point (e.g., a word).

accuracy. With that in mind, we can use classifiers to aid in automatic or semi-automatic corpus correction, using the variation n-gram output for error detection as our starting point.

Because we are looking at classifier-based correction, this is in principle applicable to any annotation for which a label can be given to a sequence of text. We choose to use the simplest kind of annotation discussed so far, that of part-of-speech annotation (see chapter 2), for our experiments in correction; we expect non-trivial modifications for non-positional annotations, but some aspects are easily extendible (see section 6.6).

Before setting out to correct, we need to address the motivation for automatic or semi-automatic correction. It can be argued that only manual correction is desirable for fixing a corpus, at least for a gold standard corpus. In the first place, we could lose the original corpus information, which could be useful to researchers. Some researchers may want to use data which contains errors to ensure that their methods are robust enough to handle such errors. In the second place, and more importantly, anything less than 100% accuracy in correction of the data is insufficient. Having new, consistent errors may be worse than the original situation of having errors which could at least be found using their inconsistent properties. Why, then, should we bother to automatically correct a corpus?

The first point to make is that the information about errors originally present in the corpus does not have to be lost. With annotation setups such as XML in wide use, it is easy to incorporate multiple layers of annotation within the same corpus, or to use standoff annotations (e.g., Bird and Liberman, 2000). Thus, both the original annotation and the corrected annotation can co-exist in the corpus, and researchers

can choose which to use. If they do not trust the corrected annotation or if they wish to use erroneous data to test the robustness of their methods, they can use the original data.

The second objection, that of not being able to obtain 100% accuracy, is a more serious one. It is likely that our experiments will fall short of 100% accuracy, but that does not negate their value. It is true that automatic correction software must be used with care, but this is true of any software in use for corpus creation. Large corpora require semi-automated methods of annotation to begin with, and automatic tools must be used sensibly at every stage in the corpus building in order to ensure an accurate corpus. Automated annotation methods are not perfect, but humans also add errors, from biases and inconsistent judgments; a benefit of the variation n-gram method for error detection is that it shows where humans had a hard time making consistent decisions and where corpus annotation standards need strengthening.

Automatic correction can be used in conjunction with manual checking, but in this chapter we will see how far automatic methods for correction can get us. The problem that the variation n-gram error detection shows us is one of inconsistency, and using classifiers to correct such errors is in keeping with the idea in van Halteren (2000) that automatic tagging methods can be used to enforce consistency. Automatic correction can also inform us about the task of classification; adapting POS tagging technology for correction offers insights into the general process of tagging. Pushing the limits of automatic correction, then, will be the goal of this chapter.

In the next chapter, we will explore sorting the output of the automatic correction methods developed in this chapter. If we can distinguish automatically correctable tags from tags needing human assistance to correct, then we can obtain some degree

of automaticity. Furthermore, since we here conflate the tasks of token error detection and correction, we can also view the work in this chapter as token error detection. That is, the variation n-gram method tells us the type of error which is wrong, pointing to at least two corpus positions, only one of which might be wrong. By identifying the corpus positions in the benchmark corpus that a part-of-speech tagger disagrees with, we can pinpoint the specific positions which are erroneous.

The outline of the rest of the chapter is as follows: in section 5.2 we first discuss our methodology, specifically how to evaluate corpus correction work in light of the fact that we have no benchmark corpus to compare to—we are correcting the so-called "gold standards." In section 5.3 we turn to the actual work of correction, using a variety of different methods and using the WSJ corpus as our data. Building on this work, in section 5.4 we modify the tagging model in order to better account for ambiguities and enhance performance. In section 5.5 we will attempt to validate the work on the BNC-sampler and point out problems which arise in the process.

5.2 Methodology

When evaluating a correction method, we face an immediate problem: since we are correcting the "gold standard," we have no benchmark by which to gauge the accuracy of the corrected corpus; in other words, we lack a true gold standard. Thus, we have to do some evaluation by hand. As with the error detection results, we would like to have had multiple evaluators, ideally annotators from the original corpus project, doing the evaluation. Due to limitations of time and money, however, only the author was able to evaluate the results.

Instead of evaluating the output of each method separately by hand, we sampled 300 positions flagged by the variation n-gram method from the Wall Street Journal corpus as potential errors, out of the 21,575 possible flagged positions. Based on the error detection work, the set of flagged positions was derived by taking the position of every non-fringe variation nucleus (i.e., every trigram nucleus). Every token position associated with a variation type was included; we expect less than half of the flagged positions to be erroneous since many variations occur frequently with one tag and rarely with another tag, and the majority tag is likely correct.[77]

For these 300 samples, we removed the tag for each position and hand-marked what the correct tag should be. That is, in order to prevent any bias in this re-annotation phase, we had no access to the original tag or the set of variation tags, but had to decide the correct tag based solely on the tagset distinctions, as given in the manual (Santorini, 1990).

This same sample can then be used to evaluate the output of any method, thereby minimizing the amount of human effort required. We can use it to tell how well each method is doing overall and how well each is doing with respect to the corpus positions where the correction method has changed the tag. The former tells us how accurate the flagged portion of the corpus is if we replace every original tag with a tag from the correction method. The latter tells us how accurate the changes to the corpus are, which is an indication of how much the corpus is being improved.

An additional complication is that, because some of the tagset distinctions were not defined clearly enough in the WSJ tagging guidelines (Santorini, 1990), we could not decide for every corpus position what the exact tag should be. (See discussion

[77]We will show that the majority tag is more often correct than not in section 6.3.

in section 2.2.3 under the heading *Problematic tags.*) We distinguished the following cases: $A|B$ means that we could not decide at all between tag A and tag B; there were 29 such cases. A/B means that we favored tag A over tag B, but there was insufficient evidence to say whether B was absolutely not viable; there were 73 such cases. For the purposes of comparison, in the A|B case, matching either tag results in a correct designation, while in the A/B case, matching A is marked as correct, but matching B is marked as possibly correct.

For the original corpus, we find that 202 samples are definitely correct; 28 may be correct, in that they matched the second tag; and 70 are incorrect. Since a human could not decide which tag was correct for many of these cases, we cannot expect our technology to do better. Thus, we will use the more lenient precision figure of folding the "maybe" cases into the correct cases, making our precision figure an indication of, on the one hand, how often the data is clearly not wrong, and, on the other, the best performance we can expect to achieve. Keeping this evaluation metric consistent across all experiments will provide a way to compare whether a correction method is improving the corpus or not. For the benchmark, combining the correct and maybe cases results in a precision of 76.67% (230 out of 300). A correction method must then surpass this precision figure in order to be useful.

5.3 Methods for correction

We can now discuss different classification techniques to employ in correcting a corpus. As van Halteren (2000) points out, taggers can be used to enforce consistency, and thus can potentially correct errors arising from inconsistency.

Classifiers fall into two main classes: supervised and unsupervised. Supervised methods for classifying assume the existence of annotated training data; unsupervised methods do not assume the existence of such data. The situation with error correction is that of wanting to correct a flawed corpus, so one must wonder if using a supervised method will work effectively. There is a chance that the classifier will learn the wrong patterns from the corpus. However, we need to be aware of the patterns in the corpus. If we assume humans are more often right (consistent) than not, it seems likely that training a classifier on a corpus and using the result to generate the correct tags has some promise. Additionally, unsupervised methods generally perform worse than supervised (see, e.g., Brill, 1995b). Thus, we will discuss how we can use supervised methods of classification for error correction.

The procedure for using a supervised tagger for error correction will be as follows:

1. Train the tagger on the entire corpus.

2. Run the trained tagger over the entire corpus.

3. Isolate the positions that the variation n-gram error detection method flags as potentially problematic.

4. Choose the tagger's label obtained in 2 for those positions.

It is hoped that the tagger will learn the consistent patterns from the corpus and will generalize these patterns over the problem parts of the corpus. Items which violate these generalizations will be corrected.

The task of automatic correction is different from using POS taggers to annotate a corpus. As stated in step 3 above, we are only concerned with the potential error sites that the variation n-gram method flagged, not the entire corpus. With the high precision of error detection (see section 2.2.3), we are able to focus on spots which are more likely to need correction.

We now turn to the classification techniques which we use for error correction. The task here is somewhat different than the original task of tagging a completely unseen corpus, and so some issues, such as how to tag unknown words, do not concern us. For each method, there are advantages and disadvantages for our particular task. All of the techniques we will examine have in common that they use the surrounding local context (typically, a window of two or three words and/or tags) to determine the proper tag for a given corpus position, but they use this information in different ways. The results will show that there is no clear best method, but in section 5.4, we will show how to improve upon the results by adding information about ambiguity classes into the tagging model.

In each of the following subsections, a brief description of the underlying technology is given; those who are familiar with the technology or who wish to skip the details can go directly to the **Results** paragraphs.

5.3.1 N-gram Taggers/Markov Models

N-gram taggers (Church, 1988; DeRose, 1988; Charniak et al., 1993; Weischedel et al., 1993; Brants, 2000b) use probability information about the surrounding n tags to indicate what the most likely label for a given word is. They maximize two probabilities in selecting a tag: 1) the probability of a tag given the surrounding tags

(often only the preceding tags) and 2) the likelihood of a tag given the word. These are what Church (1988) refers to as the contextual probability and the lexical probability, respectively. The goal is to find the most likely sequence of tags T ($\{t_1, ..., t_n\}$) with these two probabilities, given a particular word sequence W ($\{w_1, ..., w_n\}$). In other words, we try to find the value of T which maximizes $P(T|W)$.

Most statistical classifiers use a Markov model for determining the best sequence of tags, and good descriptions of using a Markov model for part-of-speech tagging can be found in Charniak et al. (1993) and chapters 9 and 10 of Manning and Schütze (1999). We will follow Weischedel et al. (1993) in describing this process.

Because of Bayes' law, we can rewrite $P(T|W)$, the probability of a tag sequence given a word sequence, as below:

(47) $P(T|W) = \frac{P(T)P(W|T)}{P(W)}$

However, since the sequence of words is always the same, regardless of what sequence of tags we select, we can ignore the denominator. Thus, we are interested in calculating and maximizing $P(T)P(W|T)$.

As stated above, the contextual probability of a tag depends on the previous tags, so $P(T)$ will be equivalent to finding the probability of each tag given all the previous tags. Likewise, the lexical probabilty depends on the previous words and tags. Thus, we derive the equation in (48).

(48) $P(T)P(W|T) = \prod_{i=1}^{n} P(t_i|t_1, ..., t_{i-1})P(w_i|t_1...t_{i-1}, w_1...w_{i-1})$

However, in practice, it is very costly to calculate these probabilities. To reduce the amount of contextual information, only the previous n tags are used; for a tag's lexical probability, only the current tag is used. For a trigram (3-gram) model, for instance, the following independence assumptions are made:[78]

(49) a. $P(t_i|t_1, ..., t_{i-1}) = P(t_i|t_{i-2}t_{i-1})$
b. $P(w_i|t_1...t_{i-1}, w_1...w_{i-1}) = P(w_i|t_i)$

Instead of examining all of the previous context, we only examine a small portion of it. Markov models can be viewed as a series of states (represented by tags) and of transitions between states. The probability in (49a) is called the transition probability, as it gives the likelihood of transitioning from one state to the next one. The probability in (49b) is called the emission probability, since it gives the likelihood of a word being emitted at the current state. In addition to specifying the transition probabilities and the emission probabilities, the third component of the Markov model is to specify the initial probabilities, i.e., the probability of a tag starting a sequence.

Putting this all together, to find the most likely sequence of tags, we want to find the sequence which maximizes the equation in (50).

(50) $P(T|W) = \prod_{i=1}^{n} P(t_i|t_{i-2}t_{i-1})P(w_i|t_i)$

In short, we use information from the preceding n tags and information about how likely a word is given a tag to figure out what the best sequence of tags is. Using the previous two tags might be an advantage for correcting the variation n-gram method; in the exact same context of words, we get two different taggings, so looking at a more general context might better reveal the true pattern.

[78]These obviously are only an approximation of natural language, as dependencies exist between words over the length of entire sentences.

To obtain these probabilities during training, an annotated corpus is used to calculate the relative frequencies. For every tag (t_k), we count up how many times it occurs preceded by the particular tags (t_i, t_j) and divide that by how many times the preceding bigram occurs in isolation. This gives us the likelihood of tag t_k given that the previous tags were t_i, t_j, as in (51a) (where $C(X)$ is the count of occurrences of X). Likewise, we can easily calculate the probability of a word given a tag from a tagged corpus, as in (51b).

(51) a. $P(t_k|t_i, t_j) = \frac{C(t_i,t_j,t_k)}{C(t_i,t_j)}$

b. $P(w_j|t_j) = \frac{C(w_j,t_j)}{C(t_j)}$

Calculating probabilities is done during training; for actual tagging, one must find the best possible path through the Markov model of states and transitions, based on the transition and emission probabilities. This can be extremely costly, as multiple ambiguous words means that there will be a rapid growth in the number of transitions between states. Thus, the Viterbi algorithm (Viterbi, 1967) is commonly used to reduce complexity. Instead of calculating the costs for all paths at each state, we only keep the k-best paths.[79]

One often-discussed important issue for tagging is that of smoothing: if a word has not been seen with a tag before, or if a tag sequence has not been seen before, that does not mean it should have a probability of zero. Rather, some non-zero probability should be assigned to unseen possibilities, so that they are not completely ruled out. Brants (2000b) demonstrates that the particular smoothing technique is important for the accuracy of one's tagger; with good smoothing, his tagger achieves an accuracy rate of 96.7%.

[79]Note that k may equal one and note also that another option would be to keep every path which is above a certain threshold of probability.

For the task of error correction, where the training and testing are done on the same data set, smoothing is highly unlikely to be useful since there will be no unknown words. It is possible that new tag sequences or new tags for words will be needed, but we are trying to add consistency to the corpus, and so we expect a pattern seen elsewhere to be applied to a new position. For these reasons, we will experiment with using a tagger without any smoothing.

Brants (2000b) also points out that the handling of unknown words affects a tagger's importance. However, since we are training and testing on the exact same text, we need not concern ourselves with the methods for dealing with unknown words.

Results

For our purposes, we used the TnT ("trigrams and tags") tagger (Brants, 2000b), a Markov model tagger, trained on the entire original corpus and then tested on the same corpus. As shown in figure 5.1, we found that 217 tags were correct, and 24 were possibly correct, giving an accuracy of 80.33%.

	Changed	Unchanged	Total
Default	68.47% (76/111)	87.30% (165/189)	80.33% (241/300)
No smoothing	70.64% (77/109)	89.01% (170/191)	82.33% (247/300)

Figure 5.1: Results of using TnT to correct the 300 samples

Here and throughout we report not only precision figures for all 300 samples, but also precision figures for only those corpus positions which the tagger changed and the complement set, the positions the tagger did not change. In this case, of the 111 changed positions, we find in figure 5.1 a much lower accuracy than overall, that of 68.47%. It seems, then, that this method does a fairly decent job of not changing what it should not change, but those tags which it does changes are only correct 68.47% of the time.

As mentioned before, it might be the case that smoothing is detrimental to our results since we know that no new words will be seen. Indeed, removing smoothing (using the option "-d1/0" in TnT) seems to help somewhat, giving 82.33% overall accuracy and 70.64% accuracy on the changed positions, but the numbers are still fairly similar, and we cannot say that this change is significant.

We will return to the TnT tagger and some of the problems it faced in section 5.4 when we explore ways of modifying the tagging procedure in order to improve the accuracy.

5.3.2 Decision trees.

A probabilistic decision tree tagger is developed in Schmid (1997). This classifier is akin to an n-gram based tagger, in that the determining context for deciding on a tag is the space of the previous two tags. However, instead of calculating probabilities of sequences of tags from their frequencies during training, a binary-branching decision tree is constructed. The nodes of the tree refer to one of the previous two tags and ask whether that tag has a certain value. For example, the node might be $tag_{-1} = ADJ?$, and the branches correspond to either a yes or a no answer. By following the

path down to the terminal elements of the tree—sets of (tag, probability) pairs—one can calculate what the most likely tag is. These decision trees are created during a training phase and used during tagging to calculate transition probabilities for sequences of tags (see below).

Each node should divide the data into two maximal subsets, i.e. should ask the question which provides the most information about a tagging decision. To do this, a metric of information gain is used, which determines how much information is gained by using this node. By maximizing the information gain, the average amount of information still needed *after* the decision is made is minimized.

Once a decision tree is constructed, it can be used to derive transition probabilities for a given state in a Markov model. As with other probabilistic classifiers utilizing a Markov model, the Viterbi algorithm is then used to find the best sequence of tags. With this, Schmid (1997) reports accuracy results up to 96.36%.

As the motivation for using decision trees over n-gram taggers, Schmid (1997) cites the problem of n-gram taggers in estimating small probabilities from sparse data, which decision trees should do a better job of obtaining. Another possible advantage that the decision tree model has is that the appropriate context size is determined by the decision tree. For example, just the preceding or the following tag may be enough to determine the correct tag. In (52), taken from the WSJ, for instance, to know that *such* should be changed from adjective (JJ) to pre-determiner (PDT), one only need look at the following determiner *an*, and that provides enough context to disambiguate.

(52) Mr. Bush was n't interested in such/JJ an informal get-together .

Another possible advantage is that decision trees have been applied to parsing (Magerman, 1995) using a positional encoding (features are assigned to individual words) which may assist in extending this error correction work to the classification of the output of the variation n-gram method for syntactic annotation. Additionally, Mitchell (1997) states that, because decision tree algorithms use all of the training data at once to make a decision, "Decision tree learning methods are robust to errors"; if that is true, then we should find fairly successful correction. However, the claim about the robustness towards errors is true if the errors are local to a single training example, or only a small number of examples (Mitchell, 1997). Frequently occurring errors, or types of errors, will still cause problems. Also, the robustness towards errors is dependent upon what method of pruning is used.

Results

We used Schmid's Decision Tree Tagger (which we will occasionally refer to as DTT), and what we find are results which are slightly higher than with n-gram tagging. In figure 5.2, we see that we are able to obtain accuracies of around 84% total, but again much lower on changed positions (73-74%). We tried two different methods: in the *default* case, the tagger was given a corpus without any annotation. In the *using tags* case, every non-variation tag was assigned its original tag, in the hopes that by only having to guess at the variation tags, the tagger would be using more reliable contextual information. Somewhat surprisingly, we see very little change between the two methods of tagging, indicating that the context of tags the tagger is able to generate on the fly is as sufficient for these purposes as using the original tags as context, if not better.

	Changed	Unchanged	Total
Default	73.83% (79/107)	90.15% (174/193)	84.33% (253/300)
Using tags	72.82% (75/103)	89.34% (176/197)	83.67% (251/300)

Figure 5.2: Results of running the Decision Tree Tagger on the 300 samples

Aside from the possible robustness to errors, one reason DTT outperforms TnT is likely that it has a more flexible context. As mentioned, in example (52)—which DTT correctly changes and TnT does not—the fact that *interested* is tagged JJ two words back is irrelevant to the tagging decision at hand. Such information could indeed mislead a tagger, if a preceding context of JJ IN happens to be an accidental predictor of JJ for the current word. TnT uses a fixed context of trigrams, and so is swayed by tags appearing two words back. DTT is more flexible and can in principle focus on smaller windows of context when that is the only pertinent information, thereby allowing it to ignore misleading information. As DTT does not provide a way of viewing output trees, however, we cannot confirm that this is the reason for improvement.

5.3.3 Transformation-Based Error-Driven Learning (TBL)

Error-driven Transformation-Based Learning (TBL) (Brill, 1994, 1995a) is a technique which attempts to automatically derive rules for classification from the corpus. The advantage over statistically-based tagging—as detailed in sections 5.3.1 and 5.3.2—is that the rules are more linguistic than probabilities and are thus more easily interpretable.

The first component of TBL is the initial state annotator. For both training and tagging, the first step is to run an initial state annotator over the text. For training, this text is the original, but now unannotated, corpus; for testing, this text is whatever text needs to be annotated. Note that these are identical for our purposes. The initial state annotator can be anything from a random assignment of tags to a different sophisticated classifier.

The second component is the set of allowable transformations. The goal of the training phase is to produce a list of ordered rules to be applied to a text when tagging, but we have to specify the kinds of rules which can be learned. Every rule, or transformation, is of the form in (53), which specifies that Tag_i should be changed to Tag_j in a particular context C.

(53) $\text{Tag}_i \rightarrow \text{Tag}_j$ in context C

The set of allowable transformations defines what kinds of contexts can be specified by using rule templates. One rule template is given in (54).

(54) $\text{Tag}_i \rightarrow \text{Tag}_j$ when the preceding (following) word is tagged z.

An instance of (54) cited in Brill (1994) is "change the tagging of a word from **noun** to **verb** if the previous word is tagged as a **modal**." The set of allowable transformations used in Brill (1994) allows tags to be changed depending on the previous (following) three tags and on the previous (following) two word forms. Note that this gives the tagger a slightly larger window and in some respects more flexibility than the previous taggers we have looked at. For example, we find the following rule in our experiments below: *Change VBP (3rd person present tense verb) to VB (base form verb) if one of the previous three tags is MD (modal).* This rule (which is the second-highest ranked

rule in all the experiments below) allows the tagger to pick any one of the three tags and thus is not dependent on a particular tag in a particular location, and it correctly parallels a linguistic generalization—base form verbs are arguments of modals.

The third component, the objective function, specifies when a rule should be learned. The usual metric is tagging accuracy: we want our rules to result in a more accurate text, as determined by the benchmark corpus, here called the "truth." The training phase is iterative: the learner runs over the text multiple times and tries out all possible transformations. It compares the output of that rule application with the "truth." A rule is learned on each iteration when it results in the greatest error reduction of the tagged text, i.e., maximizes the objective function. When the scores of rules—the error reduction scores—drop below a pre-defined threshold, the training phrase is completed. The threshold is defined as the number of corrections a rule makes minus the number of errors it introduces, i.e., the net error reduction in the text.

The emphasis on reducing errors is the impetus for the name "error-driven," and as Brill and Pop (1999) point out, the attempt to minimize the number of errors distinguishes the transformation-based tagger from Markov-model based taggers, which maximize the string probability. With such a set-up, Brill (1994) reports accuracies of up to 97.2% for text with no unknown words.[80]

[80]Volk and Schneider (1998) report that the Brill tagger outperforms a decision tree tagger with respect to the tokens which are ambiguous in the lexicon. It is unclear whether this performance will carry over to the kinds of variations we are examining.

One could experiment with adjusting the objective function, especially since we know the "truth" is flawed in some places in the corpus. However, in the absence of a solid objective function to replace Brill's, we do not pursue that here.[81]

Results

For our experiments, we used the original transformation-based learner written by Brill[82] and trained our contextual rules on the entire WSJ corpus,[83] using the 26 rule templates provided with the release and described in Brill (1994). The initial state annotator simply assigns a word's most frequent tag to the corpus position. The training phase produces a list of ordered rules, and one can use the whole list or only a subset of the rules for tagging. For our purposes, we used different subsets, based on particular thresholds, i.e. using only the rules at or above a certain value. In figure 5.3, we present the results of the Brill tagger on the 300 samples with thresholds of 7, 10, 15, and 20.

Two items are noteworthy about these results. First, the overall precision slightly degrades in going from a threshold of 15 to a threshold of 10, even though a lower threshold generally means that the tagger will be more accurate. However, the changed results are the highest we have seen, and they get better as the thresholds get lower and more rules are used. The tagger does better on changed positions than either TnT or DTT because it adds rules by compaing to the corpus and maximizing

[81]One might consider a function which maximizes the corpus consistency, but it is not clear how to exactly quantify consistency, and to detect a single value over an entire corpus would be a time-consuming effort.

[82]Available at: http://www.cs.jhu.edu/~brill/

[83]The training corpus is usually split into a section for unknown word rule learning and a section for contextual rule learning, but since unknown words were not a problem, we used the entire corpus for the learning of contextual rules.

Thresh.	Rules	Changed	Unchanged	Total
7	665	75.79% (72/95)	83.41% (171/205)	81.00% (243/300)
10	478	75.00% (69/92)	84.13% (175/208)	81.33% (244/300)
15	346	74.49% (73/98)	85.64% (173/202)	82.00% (246/300)
20	270	70.59% (72/102)	84.85% (168/198)	80.00% (240/300)

Figure 5.3: Results of running the Brill Tagger on the 300 samples with all 26 templates

the reduction in error rate. When a new rule is added and subsequently used, the tagger should match the corpus better. If it does not match the corpus, despite this closer fit, then the evidence must be strong elsewhere in the corpus for the changed tag.

By examining a subset of the rules more closely, we can see that one problem the tagger has is that it is lexicalized, in that it can learn rules for specific words. While this matches a corpus well, it might do so at the cost of losing general patterns. For example, the tagger six times guessed common noun (NN or NNS) when it should have assigned adjective (JJ) (and only once guessed JJ when it should have assigned NN). We can see two examples in (55) where the hyphenated modifier should be JJ and appears in an environment which is prototypically an adjectival environment. Granted, because of noun-noun compounds, this is not always an adjectival spot, but for a word which varies between noun and adjective, the chances of an adjectival use should increase when between a determiner and a noun. However, as shown in figure 5.4, we see no such indication that this is an adjective slot; what we find instead is a list of rules conditioned for the most part on the particular lexical item involved.[84]

[84]The notation *WDNEXTTAG executive NN* for *NN JJ* should be read as: *Change NN to JJ when word = executive and* tag_{+1} *= NN.*

(55) a. an interest-rate environment
b. a deficit-reduction bill

Rule number	From	To	Context
72	NN	JJ	WDNEXTTAG executive NN
132	NN	JJ	CURWD first
144	NN	JJ	WDNEXTTAG official NN
167	NN	JJ	WDNEXTTAG right NN
244	NN	JJ	WDNEXTTAG giant NN
250	NN	JJ	WDNEXTTAG official NNS
251	NN	JJ	WDPREVTAG DT third
263	NN	JJ	WDNEXTTAG firm NN
285	NN	JJ	WDNEXTTAG close NN
286	NN	JJ	WDNEXTTAG half NN
301	NN	JJ	NEXTWD oil
302	NN	JJ	RBIGRAM equivalent to
310	NN	JJ	SURROUNDTAG DT NNP

Figure 5.4: Rules for changing NN to JJ in the Brill tagger (threshold = 15)

Likewise, the tagger 13 times guessed IN when it should have guessed RB (and never guessed RB instead of IN, which is generally the more frequent tag). We see in the six rules changing IN to RB in figure 5.5 that *as/As*, *before*, and *about* are essentially the only words which get their tags changed in particular contexts; otherwise, if a word's most common tag is IN, it will stay IN.

To test whether removing some lexical information improves performance, we removed the 12 templates in the tagger which refer to surrounding words and completely retrained the tagger to obtain a new list of rules; templates referring to the current

Rule number	From	To	Context
12	IN	RB	WDAND2AFT as as
116	IN	RB	SURROUNDTAG , ,
168	IN	RB	WDNEXTTAG as RB
252	IN	RB	WDAND2AFT As as
271	IN	RB	WDNEXTTAG before .
328	IN	RB	WDAND2AFT about %

Figure 5.5: Rules for changing IN to RB in the Brill tagger (threshold = 15)

word were kept in the set. As we can see in figure 5.6, this did not help at all—the overall precision is the same for almost every threshold level. It does show that we can obtain similar rules using higher thresholds and fewer templates, making the processing time somewhat faster.

Thresh.	Rules	Changed	Unchanged	Total
7	605	73.73% (73/99)	84.58% (170/201)	81.00% (243/300)
10	457	74.19% (69/93)	84.54% (175/207)	81.33% (244/300)
15	325	73.47% (72/98)	86.14% (174/202)	82.00% (246/300)
20	256	71.00% (72/102)	85.50% (168/198)	80.67% (242/300)

Figure 5.6: Results of running the Brill Tagger on the 300 samples with 14 templates

5.3.4 Memory-Based learning

Different from the previous methods, the next method, memory-based learning (MBL) (Daelemans et al., 1996, 1999), is based on the notion that keeping all training instances in memory will aid in the classification task. MBL has been used for a variety

of classification tasks, including part-of-speech tagging, base noun phrase tagging, and parsing (see, e.g., Daelemans et al. (1999); Kübler (2003) and references therein). Little abstraction is done in storing the training data, and the testing data is classified based on the similarity to the examples in memory, using a similarity metric.

For the task of part-of-speech tagging, Daelemans et al. (1996) define the data to be stored as cases: each case consists of a word, the preceding and following context, and the category label in that context. Training is thus a matter of selecting the size and nature of the context and then simply storing these cases.

When tagging, one finds the most similar case(s) in memory to the current situation.[85] The similarity metric is calculated by examining the distance between feature values. The distance, or overlap, metric is defined in (56) for two feature values (x_i and y_i) and is basically an identity check.

(56) $\delta(x_i, y_i) = 0$ if $x_i = y_i$, else 1

So, the similarity metric is obtained by summing over all n features, as in (57).[86]

(57) $\Delta(X, Y) = \sum_{i=1}^{n} \delta(x_i, y_i)$

Results

We ran some experiments with the Tilburg Memory-Based Learner (TiMBL), version 5.0 (Daelemans et al., 2003), which attempts to make the best match between the current data point with a data point stored in memory, without abstraction. As we will see, without modifying the default algorithms, this matching results in a close match to the original data. In fact, when training on the entire corpus and then

[85]This is essentially a form of *k-nn* (*k*-nearest neighbor) classifying.

[86]As it is, the similarity metric treats each feature equally. The final formula used in Daelemans et al. (1996) weights each feature. We will experiment with both methods.

testing on it again, we see relatively few changes. We used the following features: word, tag_{-2}, tag_{-1}, tag_{+1}, tag_{+2},[87] and we found that, out of the 1,289,201 tokens in the corpus, TiMBL matched the tags for 1,283,256 of them, or 99.54%. For our sample of 300 items, TiMBL only made 43 changes out of the 300.[88] The precision results were also fairly poor: 78.33% (235/300), with 74.42% (32/43) for changed positions and 78.99% (203/257) for unchanged positions, as shown in the first line of figure 5.7. This is barely above the baseline accuracy rate we established with the original corpus. But these results are obtained with the defaults and are rather blind to the qualities of TiMBL as a tagger.

Modifying the TiMBL options

Changing the size of k to find the k-nearest distances has a useful effect. The default is to find the one nearest distance and select the majority label from all neighbors which match that distance. But because the data is being matched too well (i.e., we are overtraining), we would like to expand the candidate set from which to draw a correct tag. Thus, we experiment with using different sizes of k.

Without changing the voting scheme, this means that for k=3, for example, the items with the three closest distances all vote once for their tag. The effect of this is to draw more candidates into the voting scheme and hopefully to overtrain less.

[87]Note that, in principle, this is better information than that used in the cases for Daelemans et al. (1996) since their tagger proceeds in an iterative fashion and only has access to disambiguated tags for the previous words. Instead of our tag_{+1} and tag_{+2} features, they use the ambiguity class of the focus word and of the following word.

[88]Note, though, that changing 14.33% of the 300 items is much higher than the average corpus-changing rate of 0.46%.

We tried this by using our original features and leaving all other defaults unchanged. The results are given in figure 5.7. We obtain the best results for k=3 and k=4. Precision starts to drop dramatically with k=5, and we even see some very strange taggings, such as a comma tag (,) when VBN/JJ was correct. This is because everything in the candidate set gets an equal vote, so once we start expanding out the candidate set, if some features happen to coincide, that instance gets an equal vote.

k	Changed	Unchanged	Total
1	74.42% (32/43)	78.99% (203/257)	78.33% (235/300)
2	73.08% (57/78)	83.78% (186/222)	81.0% (243/300)
3	76.19% (64/84)	83.80% (181/216)	81.67% (245/300)
4	73.33% (66/90)	85.24% (179/210)	81.67% (245/300)
5	65.31% (64/98)	85.64% (173/202)	79.0% (237/300)

Figure 5.7: Adjusting the size of k in TiMBL

We can change the weighting so that the closer a candidate is, the more weight it receives, using the distance-weighted class voting option of TiMBL. This weights the votes based on the distance from the center. For example, when k is equal to 3, instead of all items (k = 1, 2, 3) getting the same weight, items one feature away have a higher weight than items three features away. Trying this, however, gives us worse results, as shown in figure 5.8. Once again we seem to be training too much on the erroneous data: TiMBL makes few changes and increasing k does very little to improve the quality of tagging.

k	Changed	Unchanged	Total
1	74.42% (32/43)	78.99% (203/257)	78.33% (235/300)
2	76.74% (33/43)	79.38% (204/257)	79.0% (237/300)
3	77.78% (35/45)	79.61% (203/255)	79.33% (238/300)
4	77.27% (34/44)	79.69% (204/256)	79.33% (238/300)
5	75.0% (33/44)	79.30 (203/256)	78.67% (236/300)

Figure 5.8: Adjusting the size of k in TiMBL with weighted voting

5.3.5 Summary for correction methods

The best results obtained so far for each automatic correction method are given in figure 5.9. We can see that the overall precision levels are around 82-84%, but that the precision for the changed positions is at or below the baseline of 76% for the WSJ corpus.

Method	Changed	Unchanged	Total
TnT	70.64% (77/109)	89.01% (170/191)	82.33% (247/300)
DTT	73.83% (79/107)	90.15% (174/193)	84.33% (253/300)
Brill	74.49% (73/98)	85.64% (173/202)	82.00% (246/300)
TiMBL	76.19% (64/84)	83.80% (181/216)	81.67% (245/300)

Figure 5.9: The best correction results

We can conclude a few things from figure 5.9. First, fully automatic correction is not feasible using the tagging models as given; the corpus positions we change are not sufficiently accurate. Secondly, as evidenced from the fact that up to 90% of the

unchanged tokens did not need to be changed, the task of detecting erroneous tokens is fairly successful, in that the changed tokens contain the tokens which need to be changed. This is more fully discussed in section 6.5.

5.4 Making taggers aware of difficult decisions

We tried four very different classification methods for correction, and at best, we obtained 84% accuracy, but the results on the changed positions were below the baseline of 76.7%. Note that because of good error detection, those percentages of 70-75% are still very high—by way of comparison, van Halteren (2000) found that his tagger was right no more than 20% of the time when it disagreed with the benchmark—but are still not sufficient. Given the variety of methods but similarity of results, the answer for error correction seems to lie less in what tagging method is used and more in how we use those methods.

The errors we detect through the variation n-gram method arise from variation in the corpus. Such variation can reflect decisions which were difficult for annotators to maintain over the entire corpus, for example, the distinction between preposition (IN) and particle (RP) for certain words. In order to improve our correction results, we intend to exploit the fact that these difficult distinctions recur often and can be added to a tagging model. That is, we intend to make our correction classifiers aware of the difficult distinctions, so that they might more correctly assign tags to variation positions.

But how do we make a classifier "aware" of a problematic distinction? To answer this question, we will first discuss the idea of a lexicalized tagger and how taggers have been lexicalized in previous work. This will help us to understand how to inform

a tagging model about prominent tagging distinctions. By discussing the motivation for and drawbacks to tagger lexicalization in section 5.4.1, we will be in a position to develop a model based on the difficult distinctions found in variations. Thus, to improve correction, we propose in section 5.4.2 a modification to the tagging model based on using a word's ambiguity class.

5.4.1 Lexicalizing the tagger

A tagger enforces uniform decisions for (classes of) words over the entire corpus; as the variation n-gram error detection method shows, such consistency in decision-making can be missing in a corpus. One way to enforce consistency for a particular word is to "lexicalize" the tagging model. Lexicalizing the tagger means that classes (tags) are subdivided into lexically-specific classes; such subdividing captures the intuition that some words have different distributions than other words within the same class.

By lexicalizing a tagger, we will enforce uniform decisions for a given word. Lexicalization will also have the effect that the tags assigned are a much better fit to the benchmark. That is, the tagging model will be more specific—it will now have a greater number of classes with probabilities specific to those word-specific classes. With a better match to the data, one might gather that the deviations from the uniformity it then enforces are caused by errors in the corpus. This was not true, however, with TiMBL (see section 5.3.4), and we will also see here that fitting the data well does not mean the tagger always is correct.

To see if a better match of the data results in a better correction, we attempted to lexicalize a tagger. To do this, we replaced the tag of every word which serves as a variation nucleus somewhere with a complex tag composed of the word and its original label; all other words retained their original tags. Then, following Pla and Molina (2004), we ran TnT on it. Since we are ultimately just creating new tags, we are in essence not really modifying the tagger so much as we are modifying the tagset. An example is provided in (58); the tag of RB (adverb) in (58a) is changed to <ago,RB> in (58b); i.e., *ago* now has its own tag.

(58) a. ago/RB
b. ago/<ago,RB>

The results, as shown in figure 5.10, indicate that a lexicalized tagger does not work well for our purposes because it enforces uniform decisions for a single type of word, thus giving it less information to make a general decision. The lexicalized tagger makes very few changes: only 48 out of 300 tags were changed from the original corpus, compared to 111 changes made by an unaltered TnT. (Overall, according to the program tnt-diff, which compares the tagged corpus with the original, only 0.93% of the corpus was changed.) When it changes, it does so with more precision than the regular TnT model did, but still only with 75% precision, and it does considerably worse on the unchanged positions.

	Changed	Unchanged	Total
Regular TnT	68.47% (76/111)	87.30% (165/189)	80.33% (241/300)
Lexicalized TnT	75% (36/48)	78.17% (197/252)	77.67% (233/300)

Figure 5.10: Results of using a tagger lexicalized for variation words

The problem with lexicalizing taggers for this task is that the benefits of lexicalization do not match our goals. Lexicalization works well when the tagger encounters "words whose probability distribution within a certain category is different from the rest" (Pla and Molina, 2004). For example, *that* behaves differently from other IN (preposition/subordinating conjunction) words, so splitting it into its own node provides a more accurate representation of its distribution. However, there is no reason to believe that the variation words we are examining have their own distribution.

Our goal, on the other hand, is to make a tagger aware of the difficult distinctions that the variation n-gram method turned up. Lexicalizing tries to capture the intuition that some words have different distributions than other words (or than a whole class of words). What we want to test is whether some *variations* have different distributions than other variations. We will explore this in the next section.

As a side note, we want to point out that a lexicalized tagger might actually be modeling noise: simply making sure that *ago* has its own distribution does not necessarily mean we will tag it correctly, even though it will conform more to the original corpus. Thus, errors in the training data can throw off a lexicalized tagger even more than normal, despite an apparent gain in accuracy over the whole corpus.

5.4.2 Using complex ambiguity tags

Lexicalizing the tagger for some words treats those words as distinct classes, but a word sometimes behaves like an instance and sometimes like a class. Classes have certain properties, but individual lexical items can belong to many different classes—

comprising what we will call an *ambiguity class*—and it might be the case that these ambiguity classes behave in their own way. For example, consider the words *away* and *aboard*, both of which can be adverbs (RB).

In example (59), we find that *away* can also be a particle (RP), thus making it a part of the ambiguity class RB/RP. On the other hand, as we see in (60), *aboard* can be an preposition (IN), as part of the ambiguity class IN/RB, but not a particle. Not only do *away* and *aboard* belong to different classes—*away* can appear in a particle-verb construction (59a) and *aboard* as the head of a preposition (60a)—but even their adverbial uses are distinguished. The use of *away* is followed by *from*, a construction forbidden for *aboard* (**aboard from*[89]).

(59) a. A lot of people think 0 I will give away/RP the store
b. the Cray-3 machine is at least another year away/RB from a fully operational prototype

(60) a. These are used * aboard/IN military helicopters
b. Saturday 's crash of a Honduran jetliner that *T* killed 132 of the 146 people aboard/RB

By using ambiguity class information, we will then be able to distinguish adverbs which can be followed by *from* (e.g., *away*) from those which cannot (e.g., *aboard*). Furthermore, when we examine the RB/RP words, we find that they form a natural class: *apart*, *aside*, and *away*, all of which can be followed by *from*. Not every ambiguity class is so cleanly delineated, but this demonstrates that such classes can indeed provide more unified groupings.

[89]An asterisk before an example indicates ungrammaticality, whereas an asterisk in the WSJ corpus indicates a null element, as in example (60a)

By deriving such an ambiguity class for each word, we are better able to capture the distinctions which annotators faced in annotating that word. In essence, then, we propose splitting a class such as RB into subclasses, e.g., into JJ/RB, NN/RB, IN/RB, etc. This reflects the fact that a whole class—as determined by the tagset—may be fractured, with one subpart behaving one way and another subpart another way. For example, consider how the Penn Treebank uses IN for both subordinating conjunction and prepositional uses, even though these subparts behave differently. In (61a) (Santorini, 1990, p. 31), for instance, *when* is a subordinating conjunction, but a host of prepositions are ungrammatical in the same slot (61b).

(61) a. I like it when/IN you make dinner for me.
b. I like it *in/*of/*on/*up you make dinner for me.

This proposal is akin to work on splitting labels, or states in a Markov model, in order to obtain better statistics (e.g. Brants, 1996; Ule, 2003) for situations with "the same label but different usage" (Ule, 2003, p. 181). The approach also falls in line with taggers that were trained using class distinctions based on ambiguity classes. For example, the decision tree tagger described in Marquez, Padro, and Rodrigues (2000) bases its decision nodes on particular tag pairings.

In like fashion, we will make the tagger informed about which tag pairings are difficult to disambiguate. If we know what decision was involved in the selection of a tag, then we can derive a better model. In other words, we are trying to model what the annotators did, but to apply their decisions in a consistent fashion. We return to the rationale for using ambiguity class information after we first fully define what *complex ambiguity tags* are. Then we describe how they are automatically assigned.

Complex ambiguity tags

In terms of implementing this idea, we can view the task as akin to lexicalizing the tagger, wherein all words involved in a variation trigram are given a new (complex) tag. The complex tag is composed of the word's ambiguity class and the resolved tag for that position. Let us take the example of *ago*, which varies between preposition (IN) and (RB) throughout the corpus. Its ambiguity class is thus IN/RB, and *ago* is assigned this ambiguity class throughout the corpus. At a particular position, however, it either resolves to IN or RB, and in example (62), it resolves to RB. Thus, we assign *ago* the complex ambiguity tag <IN/RB,RB>, as shown in (62b).

(62) a. ago/RB
b. ago/<IN/RB,RB>

We replace every word in the corpus with a tag representing the ambiguity class of that word, assuming we can determine what the appropriate ambiguity class is. Thus, we have the whole corpus from which to draw generalizations

A key question is: what ambiguity class should every word be assigned? In the tagging literature (e.g. Cutting et al., 1992) a class is composed of the set of every possible tag for a word. We could do likewise, but this could result in too many classes to be of practical use, for two reasons: 1) there are erroneous tags which should not be part of the ambiguity class, and 2) some tags and some classes are irrelevant for disambiguating variation positions.

Assigning complex ambiguity tags

To assign complex ambiguity tags to all words in the corpus, we used the following procedure, based on whether a word was flagged as a potential error by the variation n-gram method (1) or not (2).

1. Every word which is a variation word (i.e. nucleus of a non-fringe variation) or type-identical to a variation word is assigned:

   (a) a complex tag reflecting the ambiguity class of all relevant ambiguities (defined below) in the 3-grams.

   (b) a simple tag reflecting no ambiguity, if tag is irrelevant.

2. Based on their unigram tags, non-variation words are assigned:

   (a) a complex tag, if and only if this word's ambiguity tag also appears as a variation ambiguity.

   (b) a simple tag otherwise.

Variation words The whole point of assigning complex ambiguity tags is to make prominent the variations in variation words, and that is what choice 1a does. An example of choice 1a is in (62b), where *ago* varies between IN and RB in the trigrams, and so receives the tag <IN/RB,IN> when it resolves to IN and <IN/RB,RB> when it resolves to RB.

The choice can actually be a little more complicated in some cases. Instead of simply assigning all tags occuring in an ambiguity, we filter out ambiguities which we deem irrelevant. Following Brill and Pop (1999) and Schmid (1997), we do this by examining the unigrams (in other words, an automatically-generated dictionary of tags for this corpus) and throwing out tags which occur less than 0.01 of the time for a word (unless they appear a minimum of 10 times). This eliminates variations like ,/DT where DT appears 4210 times for *an*, but the comma tag (,) appears only once. Doing this means that *an* can now be grouped with other unambiguous determiners.

Admittedly, we may lose some information this way, in the form of correct but rarely-occurring cases,[90] but we remove some erroneous classes, and we gain generality and avoid data sparseness since fewer ambiguity classes are now being used.

For variation words, choice 1b is chosen when the current tag is not in the ambiguity class. For instance, if the class is IN/RB and the current tag is JJ, it gets JJ instead of <IN/RB,JJ> because this latter option does not really make sense (a word varying between IN and RB should not resolve to JJ). This situation arises in two different scenarios.

In the one case, because we are taking the ambiguities only from the 3-grams and not from all the possible tags throughout the corpus, words which are involved in a variation may have tags which are never involved in a variation. For example, *Advertisers* shows up as a non-fringe nucleus varying between NNP and NNPS (*National Advertisers .*). In non-variation positions, it appears as a plural common noun (NNS). When it appears as NNS, it receives the tag NNS because NNS is not relevant to the variation we wish to distinguish.

In the second case, because some tags have been removed from the ambiguity classes, as described above, there are tags no longer relevant for the variation. For example, *an* receives the comma tag (,) instead of <,/DT,,> because the comma was filtered out as a possible ambiguity tag, i.e. was deemed irrelevant.

One more note is needed explain how we handled the vertical slashes that are used in the Penn Treebank annotation. Vertical slashes represent uncertainty between two tags (e.g., JJ|VBN means the annotator could not decide between JJ (adjective) and VBN (past participle)); this is a case of two tags in one. When there is variation

[90]See the discussion surrounding Daelemans et al. (1999) in section 1.2.1.

between JJ, VBN, and JJ|VBN, this is simply variation between JJ and VBN, and we will uniformly represent it by the class JJ/VBN. Note that this decision has non-trivial effects. Because JJ/JJ|VBN/VBN is converted to JJ/VBN, JJ/VBN words now have a larger class and thus more data.

Non-variation words In order to group more cases together, non-variation words can also take complex ambiguity tags. As described in 2, for words which are not a part of a variation trigram nucleus, we assign a complex ambiguity tag if the ambiguity is also involved in some non-fringe trigram (choice 2a). For instance, in the first sentence of the treebank, *join* gets the tag <VB/VBP,VB>, even though *join* is never a non-fringe variation nucleus. It gets the tag <VB/VBP,VB> because its ambiguity class VB/VBP is represented in the non-fringe trigrams.

On the other hand, we ignore ambiguity classes which have no bearing on the variation trigrams (choice 2b). For example, *ours* varies between JJ (adjective) and PRP (personal pronoun), but there are no non-fringe variation nuclei which have this same ambiguity class, and therefore we assign the three cases of PRP to PRP and the three cases of JJ to JJ. We treat non-variation words as we do in order to increase the amount of relevant data (choice 2a), and still to put all non-varying data together (choice 2b).

Uniform assignment of tags It might be wondered why we only allow one possible ambiguity class per word over the whole corpus, instead of dividing up the class for more specific instances. For example, in *publicly traded investments*, *traded* varies between JJ (adjective) and VBN (past participle), but in *contracts traded on*, it varies between VBD (past tense verb) and VBN. It seems like it would be a good idea to

keep the JJ/VBN cases separate from the VBD/VBN cases, so that a tagger can learn one set of patterns for the JJ/VBN cases and a different set for the VBD/VBN cases (i.e., maximize the distinctiveness between distinct cases). While that is in principle a good idea and is an avenue of further study, there are several reasons why restricting words to a single ambiguity class is a good idea for present purposes, i.e., why we assign *traded* the ambiguity class JJ/VBD/VBN.

First, akin to the task of lexicalization, we want to group as many of the word occurrences as possible together into a single class. That is, using JJ/VBN and VBD/VBN as two separate ambiguity classes would mean that *traded* as VBN lacks a pattern of its own. To give us more data applicable for that class and for that word, we allow a word to have no more than one ambiguity class.

Secondly, multiple ambiguity classes for a word would also potentially increase the number of possible tags for a word. For example, instead of having only the tag <JJ/VBD/VBN,VBN> for when *traded* is VBN, we would have both <JJ/VBN,VBN> and <VBD/VBN,VBN>. With a greater increase in the number of tags, we begin to have a problem with data sparseness; everything not designated as an instance of a particular class pulls information away from that class. For a technology such as Markov model tagging, we are already splitting states in the tagging model (see discussion below) by creating these new tags (e.g., instead of JJ, we now have <JJ/NN,JJ>, <JJ/VBN,JJ>, and so on). As much as possible, we would like to minimize the amount of splitting of states in the HMM tagger. If we split too often, we will have classes which barely ever appear. For example, as things stand right now, <JJ/VB/VBN,VBN> appears only 3 times, which does not give us much information. Increasing the number of tags would only create more such sparse classes.

Finally, a word has only one ambiguity class throughout the corpus because, although we know what the exact ambiguity in question is for a variation trigram, it is simply too difficult to go through position by position to guess the correct ambiguity for every other spot. If we encounter a JJ/VBD/VBN word like *followed* tagged as VBN, for example, we cannot know for sure whether this is an instance where JJ/VBN was the decision which had to be made or if VBD/VBN was the difficult choice.

In short, each word has only one ambiguity class, and we assign every word in the corpus a complex ambiguity tag when that ambiguity is relevant for disambiguating a variation. When the ambiguity will not aid in disambiguation, the word is given a simple tag.

The rationale for complex ambiguity tags

Having defined how we assign complex ambiguity tags, we can now return to the purpose that they serve. As mentioned earlier, we use complex ambiguity tags because they can provide better distinctions than we had with unaltered tags. For example, if <IN/RB,IN>—a word varying between IN and RB which resolves to IN at this position—has a different distribution than <DT/IN,IN>, then they should have different representations in the tagging model, as opposed to being conflated into the single tag IN. Words which can be <IN/RB,IN> (e.g., *ago*) but not <DT/IN,IN> can ignore the contextual information that <DT/IN,IN> words like *that* provide.

To take an example, we will work through the 5-gram *revenue of about $ 370* as it is tagged by an n-gram tagger. The 5-gram in the WSJ is annotated as in (63) (starting at position 1344 in the WSJ).

(63) revenue/NN of/IN about/IN $/$ 370/CD

Between *of* and *$*, the word *about* varies between preposition (IN) and adverb (RB): it is IN 67 times and RB 65 times. When training TnT on the whole corpus without any modifications, we find that RB is a slightly better predictor of the following $ tag based on the previous two tags (1.35 times more likely), as shown in (64).

(64) a. $p(\$|IN, RB) = .0859$
b. $p(\$|IN, IN) = .0635$

However, due to the surrounding probabilities, IN is the tag that TnT assigns. Note that this is the tag found in the original corpus, but that tag is incorrect with respect to the manual since "*about* when used to mean 'approximately' should be tagged as an adverb (RB), rather than a preposition (IN)" (Santorini, 1990, p. 22). Thus, as part of error correction, we would like to see it tagged RB—even though IN is slightly more likely in this particular lexical trigram (*of about $*).

The word *about* generally varies between three tags: IN, RB, and RP (particle); thus, it receives the ambiguity class IN/RB/RP (as does *of*). With the assignment of complex ambiguity classes, the probabilities in the altered model are quite different. We see a much stronger probability for $ conditioned on RB (technically, on <IN/RB/RP,RB>) when we have complex ambiguity tags (4.79 times more likely), as shown in (65).

(65) a. $p(\$| < IN/RB/RP, IN >, < IN/RB/RP, RB >) = .6016$
b. $p(\$| < IN/RB/RP, IN >, < IN/RB/RP, IN >) = .1256$

What we witness is that RB applied to a word which has an ambiguity class of IN/RB/RP behaves differently than the general class of RB words. We are given an indication that, for IN/RB/RP words, RB is a significantly more probable in this context than IN.

We have just shown that the contextual probabilities of an n-gram tagger are affected when using ambiguity tags as part of a complex tag, but we also gain information about lexical probabilities in the process. The original pertinent emission probabilities were as in (66), but for the transformed corpus, we have the probabilities as in (67).

(66) a. $p(about|IN) = 2074/134926 = .0154$
b. $p(about|RB) = 785/42207 = .0186$

(67) a. $p(about| < IN/RB/RP, IN >) = 2074/64046 = .0324$
b. $p(about| < IN/RB/RP, RB >) = 785/2045 = .3839$

The most striking difference between these two sets of emission probabilities is how much more likely RB is in the complex ambiguity model (67) than in the unaltered model (66). This difference comes about in the following way: from (67), we can deduce that, in comparison to all other words out there which are IN/RB/RP, *about* has a much greater chance of resolving to RB. That is, IN is generally about 31 (64046/2045) times more likely than RB for IN/RB/RP words, but for *about*, IN is only 2.6 (2074/785) times more likely. So, when calculating the conditional probability of *about*, conditioning on RB (i.e., <IN/RB/RP,RB>) results in a much higher probability than conditioning on IN: within this ambiguity class, *about* is much more likely to be RB. Thus, we are able to derive information similar to that found in a lexicalized tagger—i.e., *about* behaves differently than the rest of its class—but we still are able to bring general IN/RB/RP class information to bear on this tagging situation.

The information given by using ambiguity classes is able to help correctly assign a tag for a word, but it is only helpful as long as two conditions are met: 1) we have the correct ambiguity class for that word, and 2) the patterns for these (ambiguity) classes

were followed more often for the correct case than the incorrect case(s). The hope is that, since variation errors are errors for words with prominent ambiguity classes, zeroing in on these ambiguity classes will provide more accurate probabilities. By adopting this approach, we are mimicking what annotators were instructed to focus on, namely "difficult tagging decisions," or "confusing parts of speech" (Santorini, 1990, p. 7).

Results of using complex ambiguity tags

We will show that the results of using complex ambiguity tags for correction are quite promising, but to begin with we can look at some basic numbers. Whereas there were originally 80 tags in the corpus,[91] there are now 418 tags, 53 of which are simple (e.g. IN) and 365 of which are complex (e.g. <IN/RB,IN>).

TnT When we look at the 300 hand-annotated samples of variation positions from the Wall Street Journal corpus for the TnT tagger, we find that 260 spots are indeed correctly tagged, for a precision of 86.67%. Breaking this down a bit, as shown in figure 5.11, we find 86.36% precision for tags which have been changed from the original corpus and 86.79% precision for unchanged tags. Precision values are slightly lower but comparable when no smoothing is used; note, too, that fewer positions are changed (79 changes without smoothing, 88 with smoothing).

[91] The number of tags here includes tags with vertical slashes in the original corpus.

	Changed	Unchanged	Total
Regular TnT	68.47% (76/111)	87.30% (165/189)	80.33% (241/300)
No smoothing	70.64% (77/109)	89.01% (170/191)	82.33% (247/300)
Complex TnT	86.36% (76/88)	86.79% (184/212)	86.67% (260/300)
No smoothing	86.08% (68/79)	85.07% (188/221)	85.33% (256/300)

Figure 5.11: Results of using TnT with complex ambiguity tags

With the 86.36% precision for changed positions, what this means for error correction is that for the 5373 changes that the tagger makes, out of 21,575 flagged positions, we expect approximately 4640 of those changes to be correct changes. (The 95% confidence interval predicts between 4255 and 5025 changes.) We will return to these numbers in section 6.5.

Note that this corrected corpus is much better than the original "gold standard" corpus, which had a precision of only 76.67% (230/300) for the 300 samples. And we have also drastically improved upon the output of running TnT with no adjustments to the corpus, especially for changed positions. That method gave us a precision of 80.33% vs. 86.67% here. An even greater improvement is seen in the changed positions, where the unaltered TnT had 68.47% precision vs. 86.36% here.

One might wonder whether we are just doing a better job overall of tagging the corpus, and this is to some extent true. Training TnT on the original corpus without making any transformations and then testing on the same corpus results in a corpus which is 97.37% the same as before. However, with the complex tags which include ambiguity class information, the resulting corpus is now 98.49% similar to the original. So, clearly, the model is a closer fit to the original corpus.

It is not necessarily the case, however, that it is a better general tagging model. To test this, we split the corpus into 90% training data and 10% testing data and found that the original TnT gets a 96.54% match with the testing data, while the altered TnT is slightly better, matching 96.69% of the testing data. It might be the case that the altered TnT is tagging erroneous positions correctly, but without further testing, we can draw no firm conclusions.[92]

What we can say, however, is that this work lends empirical evidence to the theoretical result in Padro and Marquez (1998) that two taggers which have similar precision scores on a benchmark corpus have different precision scores on the actual truth. This conclusion is further confirmed by other results we have seen in this chapter: we saw very high precision scores for the whole corpus with TiMBL (section 5.3.4), but the precision with respect to the corrections was very poor.

Taking the results in another direction, we can use information about when the altered model disagrees with the original model to improve our recall. More specifically, we can sort the complex tags into groups, based on whether they agree with the unaltered TnT or not. To improve recall, we need to look at the corpus positions which the tagger did not change and find out which ones it should have changed. A suitable set of examples to examine is the set of unchanged positions which the original unaltered tagging model changed—the unchanged positions where the two models disagree in figure 5.12. This is indeed the space of corrections which has the worst precision and likely needs manual correction. Examining these 44 instances by hand would result in 13 more corrections. If these proportions remained true over all

[92]Note that this experiment does not show that we can necessarily use a tagger with complex ambiguity tags to improve the tagging process in general, as our complex ambiguity tags were derived from 100% of the corpus. Further work needs to go into deriving complex ambiguity tags to apply them to unseen text.

21,575 flagged positions, that would give us 935 more corrections out of 3164 more possibilities. One would have to consider the tradeoff between such a low precision (29.54%) and the gain in recall, but in some applications this would be worthwhile.

	Changed				Unchanged			
	yes	maybe	no	Precision	yes	maybe	no	Precision
Altered	66	10	12	86.36%	171	13	28	86.79%
Agree	47	8	9	85.94%	144	9	15	91.07%
Disagree	19	2	3	87.50%	27	4	13	70.46%

Figure 5.12: Fine-grained results of TnT with complex ambiguity tags

Looking at this issue slightly differently, we can first note that of the 68 spots of disagreement, at least one of the taggers is right in 64 spots, showing that tagger combination is possible. Examining the disagreements more closely to find where one tagger is right to the exclusion of the other, we find that the complex tagger is correct 31 times and the simple tagger is correct only 12 times. Thus, a voter which is weighted favorably towards the TnT tagger with complex ambiguity tags could in principle do quite well. We tested no such voting scheme here, however, and note that the distribution of correct tags between simple and complex taggers is more even in the case of the decision tree tagger (see next section).

On a different note, we mentioned earlier that a key question for using complex ambiguity classes is: what ambiguity tag (or ambiguity class) should every word get? We experimented with different methods for ambiguity class assignment for TnT: assigning tags to variation positions and randomly selecting at other times; not

giving non-variation words ambiguity tags; limiting the number of tags in a class to two; etc. Interestingly, regardless of the particular technique, we always have results in the 82-87% precision range for TnT. The reason, to be discussed in section 5.4.4, is that the different methods seemed to always encounter the same problems.

Decision Tree Tagger We also used complex ambiguity tags with the Decision Tree Tagger, and the results are given in figure 5.13. As with the n-gram tagger, we see an improvement in tagging—from 84% to 86%—but not as great an improvement as with the TnT tagger. There is, however, a noticeable difference for the positions which the tagger changed: whereas before the accuracy was around 73-74%, we now see results around 86%.

	Changed	Unchanged	Total
Default	86.21% (75/87)	85.92% (183/213)	86.00% (258/300)
Using tags	85.88% (73/85)	86.05% (185/215)	86.00% (258/300)

Figure 5.13: Results of running the Decision Tree Tagger with ambiguity tags on the 300 samples

And we also see less changes overall. So, the tagger is changing fewer items but changing roughly the same amount of items correctly, thus resulting in a higher precision. In fact, of the 87 changes the tagger now makes, 65 are the same changes that it made with regular tags (and 57 of the 65 are correct changes, or 87.69%), so it really is a case where the tagger is just now being more selective in its changes.

The decision tree taggers trained on different tagging models (unaltered vs. altered) disagree in 64 out of 300 cases. Of these 64 spots where the taggers disagree (42 where the simple tagger changed the tag; 22 where the complex tagger changed it), in 60 spots at least one tagger got it right. This tells us that there is hope in combining these methods. But combination must be done carefully, as neither tagger significantly outperforms the other on the disagreement data: when one tagging method is correct to the exclusion of the other, the method with complex tags is correct 20 times, while the regular DTT method is correct 17 times.

It is useful to examine their differences, with an eye towards possibly being able to combine the methods. For example, of the 21 cases where at least one tagger made a change involving a JJ/NN ambiguity, the simple tagger chose JJ 19 times. With more information about what kind of distinction is relevant, the complex tagger has less of a bias and is able to select NN 10 times.

There is also hope in combining taggers with completely different algorithms: where one algorithm fails, another can succeed. Of the 300 samples, when using complex ambiguity classes, the decision tree tagger (DTT) and TnT disagree in 25 spots. We should note several things from this.

The first is that there are only 25 differences, whereas there were 64 differences between the simple and complex ambiguity tag-trained taggers using the same decision tree algorithm. This shows that the division of the tagset makes more of an impact than the algorithm underlying the method, at least for these purposes. Thus, using a combination of classifiers for correction most likely depends not on which algorithm we select, but on how we alter that algorithm for our purposes.

Another noteworthy item is that it is always the case that one of the taggers gets the tag right, but one tagger is not necessarily better than the other. Of the 14 times when one tagger is right to the exclusion of the other, TnT gets 7 right and DTT gets 7 right. Furthermore, we can see certain biases in what the algorithms are doing. For example, in the 10 cases where JJ/NN (adjective/noun) is the relevant ambiguity, TnT always selects JJ, while DTT always selects NN. Although this might seem to indicate that combining tagging algorithms will simply result in indecision, this is not necessarily a bad thing. Of these 10 cases, human evaluation could not completely determine whether JJ or NN was correct for 9 of them.

TiMBL Building on the success of using ambiguity class information for HMM and decision tree tagging, we attempted to use this information in TiMBL.[93] However, because cases in TiMBL are vectors of features, we had more options to work with.

We tried three methods, as exemplified in figure 5.14: 1) using complex ambiguity tags for the features and for the class TiMBL learns, akin to the HMM task (*Complex*); 2) using a complex ambiguity tag for the class TiMBL learns, but using simple tags for all the features (*Mixed*); and 3) using simple tags throughout, but adding an extra feature which is the ambiguity class (*Extra*).

	w	t_{-2}	t_{-1}	t_{+1}	t_{+2}	(amb.)	tag
Complex	as	DT	<NN/VB,NN>	<DT/SYM,DT>	JJ		<IN/RB,IN>
Mixed	as	DT	NN	DT	JJ		<IN/RB,IN>
Extra	as	DT	NN	DT	JJ	IN/RB	IN

Figure 5.14: Examples of different methods employing ambiguity classes

[93]For efficiency reasons, we did not test the Brill tagger with complex ambiguity classes.

The results of using each method are given in figure 5.15. As we can see, none of these different methods seems to dramatically affect the precision results. It is possible that increasing k to 3 or 4 would improve the results, similar to the patterns observed for the unaltered results; but given the similarity here to the unaltered $k = 1$ case and the drastic increase in time for tagging with complex tags and larger values of k, we do not pursue that here.

	Changed	Unchanged	Total
Complex	75.0% (27/36)	78.79% (208/264)	78.33% (235/300)
Mixed	75.56% (34/45)	79.61% (203/255)	79.0% (237/300)
Extra	75.61% (31/41)	78.38% (203/259)	78.0% (234/300)

Figure 5.15: Results of using different methods employing ambiguity classes

5.4.3 Summary for correction methods

Figure 5.16 gives the best results obtained for the three different classification approaches after replacing each corpus tag with a complex ambiguity tag. We see that, although the results for TiMBL are no better than before, we now have 86% precision on both the changed positions and for the overall accuracy on all 300 positions for both TnT and the Decision Tree Tagger. This is better than the previous best of 84% and also quite a deal better than the previous best of 76% on the changed positions.

Method	Changed	Unchanged	Total
TnT	86.36% (76/88)	86.79% (184/212)	86.67% (260/300)
DTT	86.21% (75/87)	85.92% (183/213)	86.00% (258/300)
TiMBL	75.56% (34/45)	79.61% (203/255)	79.0% (237/300)

Figure 5.16: The best correction results with complex ambiguity tags

The results are highly improved, allowing for better correction, in the form of a corpus which is more correct than it was originally, but they are still obviously not 100% accurate. In the next section, we discuss the reasons why we cannot obtain completely automatic correction, and in the next chapter we propose ways of using this classifier-based correction as part of a semi-automatic correction process.

5.4.4 Remaining problems

Despite the promising results obtained with complex ambiguity tags, there are still several issues to consider in order to improve upon these results. For some of these problems, we can attempt some solutions at this time, but for others, the problem is much more persistent and seems to require manual intervention or much more sophisticated processing.

The first point is that some distinctions simply cannot be handled by most any automated system of the kind we have been using, i.e., without semantic or non-local information. As Marquez and Padro (1997) point out, distinctions such as that between an adjective (JJ) and a past participle (VBN) are essentially semantic distinctions and do not have any structural basis. Since our method uses no external

semantic information, we have no way to know how to correct this.[94] For example, in the phrase *proposed offering*, the reason that *proposed* should be VBN is that it can be rephrased as an *offering that was proposed* and indicates a specific event, facts not easily deduced in a data-driven, automatic way.

Other distinctions, such as the one between a past tense verb (VBD) and a past participle (VBN), require some form of non-local knowledge in order to disambiguate because it depends on the presence or absence of an auxiliary verb, which can be quite far away. (See 2.2.1 under *Distrust the fringe* for a more complete discussion of this tag distinction.)

Secondly, sometimes the corpus was more often wrong than right for a particular pattern. This can be illustrated in the following example, focusing on the word *later*: at position 12,136 of the corpus, we find the following: *Now , 13 years later , Mr. Lane has revived his Artist ...* On page 25 of the tagging manual (Santorini, 1990), we find the description of *later*, as shown in (68).

(68) **later** should be tagged as a simple adverb (RB) rather than as a comparative adverb (RBR), unless its meaning is clearly comparative. A useful diagnostic is that the comparative *later* can be preceded by *even* or *still*.
EXAMPLES: I'll get it [sic] around to it sooner or later/RB.
If you don't hurry, we'll arrive (even) later/RBR than your mother.

In this particular case, along with the fact that this is 13 years later as compared to now (i.e. comparative), one can say *Now, (even) 13 years later, Mr. Lane has revived his Artist ...*, favoring RBR as a tag.[95] But the trigram *years later ,* occurs

[94]Note, though, that detecting an error between the two tags works quite well. It could be argued that this lack of a structural distinction contributed to the inconsistency among annotators in the first place and thus made error detection successful.

[95]Note that there may be some level of disagreement between potential annotators here.

16 times, 12 as RB and 4 as RBR. So, assuming RBR is correct, we clearly have a lot of wrong annotations in the corpus. And we can see in (69) that in the context of following CD and NNS, RBR is much less likely than either RB or JJ for TnT.

(69) a. $p(JJ|CD, NNS) = .0366$
b. $p(RB|CD, NNS) = .0531$
c. $p(RBR|CD, NNS) = .0044$

As shown in (70), even when we use complex ambiguity tags, we still find this favortism for RB because of more wrong data in the corpus. As a side point, however, we can note that although RB is now favored, its next closest competitor is now RBR—not JJ—and RB is no longer favored by as much as it was over RBR. For words with a JJ/RB/RBR ambiguity class, JJ is not as much of a serious option in this context. It seems that we have more appropriately narrowed down the list of proper tags for this position by using complex ambiguity tags, but because of an overwhelming incorrect use of RB, we still generate the wrong tag. For this kind of error, there seems to be no way to automatically correct it.

(70) a. $p(< JJ/RB/RBR, JJ > |CD, NNS) = .0002$
b. $p(< JJ/RB/RBR, RB > |CD, NNS) = .0054$
c. $p(< JJ/RB/RBR, RBR > |CD, NNS) = .0017$

Finally, certain tags are simply too problematic to reliably disambiguate. Despite getting 86.67% overall precision with this altered tagging model, certain tags such as IN (preposition) were wrong much more often than other tags. We will turn to a solution for automatically identifying these problematic tags in section 6.2.

5.5 BNC-sampler

Using the best results obtained from our experiments with the WSJ, we also ran these classification methods on the BNC-sampler. Recall from chapter 2 that the overall precision of the variation n-gram error detection method was lower than for the WSJ, so we expect that correction precisions will also be lower.

We again annotated 300 samples, randomly obtained from the set of all tokens corresponding to non-fringe variation trigrams in the BNC-sampler. The first thing to note is that, of the hand-annotated 300 samples, 258 are correct, 8 are possibly correct, and there are only 34 of them which are incorrect in the original BNC-sampler annotation; compare the 70 incorrect cases in the parallel WSJ 300 samples. The precision figure for the 300 variation-flagged samples is 88.67% (266/300), providing a baseline for our correction methods to beat.

The lower precision is attributable to a couple of different factors. Firstly, as discussed in section 2.3.1, we have lower precision in error detection in the BNC-sampler than in the WSJ (due to tagset distinctions and spoken language). Secondly, the BNC-sampler has a much lower estimated error rate than the WSJ. Note, though, that there is a tenfold difference in reported error rates between the 3% in the WSJ and the 0.3% in the BNC-sampler. The difference in error rates of the 300 samples is only twofold. There are two possible interpretations for this: either the difference between the two corpora is not as great as reported or the variation n-gram method has successfully identified a higher percentage of positions in the BNC-sampler than in the WSJ which are problematic.

TnT First, we ran TnT, with normal tags and also with complex ambiguity tags. Complex ambiguity tags were derived for the BNC-sampler in exactly the same way as for the WSJ, as described in section 5.4.2, p. 174ff. Results are shown in figure 5.17.

	changed	unchanged	total
Regular	46.00% (23/50)	93.20% (233/250)	85.33% (256/300)
Ambclass	43.75% (14/32)	91.79% (246/268)	86.67% (260/300)

Figure 5.17: Results of running TnT on the BNC-sampler

Overall, we get about the same results as with the WSJ, if not better, with approximately 85-85% precision (but note that we are now below the baseline of 88.67%). However, we change very few tokens (32-50) and only get a precision in the mid-40% range for these changes. There are two things to note, though, before continuing: 1) The precision on the unchanged results is actually higher than the baseline, i.e. we are doing a good job of not changing when we do not need to. 2) Of the 23 changes we made which are acceptable (*yes* or *maybe*), 17 of those were unacceptable in the original corpus (the other 6 being *maybe*s). This means that the recall is 50% (17/34) and furthermore, that unclear/maybe cases are falling into the changed category.

Decision Tree Tagger We then ran the Decision Tree Tagger. As with our results from the WSJ, the results in figure 5.18 follow the TnT results closely, except for being slightly higher, and so we do not comment on them here.

	Changed	Unchanged	Total
Regular	47.73% (21/44)	92.19% (236/256)	85.67% (257/300)
Ambclass	50.0% (17/34)	92.11% (245/266)	87.33% (262/300)

Figure 5.18: Results of running the Decision Tree Tagger on the BNC-sampler

TiMBL Finally, we ran the method with TiMBL, using values of 1, 3, and 4 for k. We again obtained results which were less accurate than for the WSJ method, as shown in figure 5.19. A few comments are in order. As with the WSJ results, TiMBL makes very few changes when k is equal to one and so seems to be of little use for our purposes. Forcing TiMBL to make more changes, however, brings with it a much lower accuracy.

One positive side—which we will return to in section 6.5—is that increasing k gives us more recall of the tokens which absolutely needed to be changed. For example, with k equal to four, we have 64 cases to sort through, and even though we only get 19 of them correct, there are 20 cases which needed to be corrected in this set of 64 (58.8% of the 34 which needed correction).

k	Changed	Unchanged	Total
1	53.33% (8/15)	90.88% (259/285)	89.00% (267/300)
3	31.48% (17/54)	93.50% (230/246)	82.33% (247/300)
4	29.69% (19/64)	94.07% (222/236)	80.33% (241/300)

Figure 5.19: Results of running TiMBL on the BNC-sampler

5.6 Summary for automatic correction

In this chapter, we have demonstrated the efficacy of using automatic methods of classification for correcting a corpus, by isolating those spots identified by the variation n-gram method as potential errors. Combining the process of token error detection and correct label assignment into a single step, we first showed that simply using a tagger as is provides only moderate results, but adapting a tagger to account for common ambiguities in the data—i.e., using complex ambiguity tags—can perform much better than an unmodified tagger and reduce the true error rate within a corpus.

Despite the gain in accuracy, we pointed out that there are still several residual problems which would be difficult for most any automatic tagger to overcome. Thus, in the next chapter we pursue two lines of thought. One is to automatically sort the tags so that the difficult tagging decisions can be dealt with differently from the easily disambiguated corpus positions. The second line of thought is to pull apart the task of token error detection from automatic correction.

The method of adapting a tagging model by using complex ambiguity tags originated from an understanding of how the general tagging process works, namely that taggers attempt to disambiguate items which behave similarly. Based on this notion, the classification work described in this chapter can be extended to the general task of part-of-speech tagging. Our results indicate that a tagger using complex ambiguity classes could be better at tagging than its counterparts, as it attempts to tackle the difficult distinctions in a corpus. Such a tagger could also be useful as a part of a combination of taggers, as it provides different information. To pursue these avenues of research and to determine if this is a generally applicable method, work has to go

into defining ambiguity classes for words in a new text, given that we do not have the same unigram/trigram information in new text that we have in the variation n-gram output.

CHAPTER 6

IDENTIFYING AUTOMATICALLY CORRECTABLE ERRORS

6.1 Introduction

Continuing with the goal introduced in the last chapter of automating the process of correcting corpus annotation errors, we here attempt to automatically sort such errors to assist in correction. We have gone from the variation n-gram method of error detection for POS annotation in chapter 2 directly to automatically correcting those errors in chapter 5. Aside from using the variation n-gram output to identify which corpus positions to focus on, we initially used no other information provided by the variation n-gram method. When we began to use more information, in the form of deriving ambiguity classes to alter the tagging model, as described in section 5.4, we saw an improvement in correction.

But the amount of information from the variations that we used in section 5.4 was very minimal, and our best results were 86.67%, leaving much room for improvement. Thus, we want to do at least two things: bring more information from the variation n-gram output to bear on the issue of what the correct tag is for a corpus position, and use the classifier-based correction as part of a larger semi-automatic corpus correction effort, similar to the design of corpus annotation efforts (see chapter 1).

To fold the automatic correction stage into a semi-automatic corpus correction process, we can sort the classifier correction output into different classes of reliability, in order to prioritize the most reliable corrections first and to know which corrections need manual verification. But how are we to sort the correction output? The methods we will discuss in this chapter all share in common the property of using more information from the variation n-gram output in order to gauge what kinds of errors can reliably be fixed.

Ideally, we can sort corpus positions into two groups: those which need human assistance and those which can reliably be automatically corrected. But sorting the correction output is only one way to use the correction output. In the last chapter, we alluded to using the automatic correction output completely as a form of token error detection. Isolating those spots identified by the tagger as needing to be changed, we can then back off to complete manual correction.

Thus, in this chapter we first explore various options for sorting, based on information present in the variation n-gram output. These sorting methods include: using n-gram information to identify particularly problematic distinctions in section 6.2; using information from the label with a majority of occurrences in a trigram in section 6.3; and using information from the distribution of tags in a trigram in section 6.4. Finally, in section 6.5, we back off from fully automatic correction to using our developed methods for token error detection with suggestions for corrections, akin to work in improving annotation efforts more broadly (cf. Brants and Plaehn, 2000).

6.2 Identifying problematic distinctions

It would be very useful if we could identify tags that a tagger does not consistently use correctly. As mentioned in section 5.4.4, a tag like preposition (IN) or adverb (RB) is often hard for a classifier to disambiguate in our sample set because these tags vary so much in the original data. Since this problem arises due to properties found in the data—unlike some other problems mentioned in that section—there is a chance that we could find some way to automatically identify these problematic tags.

One idea for identifying problematic tags is based on determining how often a particular tag varies within a context. Words are generally ambiguous, but if a variation between two or more tags persists within a context, it may be the case that annotators had difficulty making this distinction. The variation n-gram method provides the information we need to find problematic tags. The way we identify problematic tags is described below and is given in an overview in figure 6.1.

1. Calculate the set of unigram variation pairs and their (type) frequencies.
2. Calculate the set of non-fringe variation pairs and their (type) frequencies.
3. Calculate the relative frequency of variation pairs.
4. Extract individual tags and add up their frequencies.

Figure 6.1: The procedure to find the most problematic tags in the corpus

In step one, we calculate the set of variation pairs over the whole corpus: for every variation unigram, we count up how often each variation type occurs. The variation n-gram approach provides sets of tags, but we are interested in pairs of tags in variation because the Penn Treebank tagging guidelines (Santorini, 1990) and others represent difficult decisions as choices between two tags. Following this practice also prevents data sparseness, e.g., in the case of a variation between four tags. Thus, we separate variation between multiple tags into unordered pairs: e.g., variation between three tags becomes three variations between two tags. For example, if a word varies between IN, RB, and RP, we count one of each of the following pairs: IN and RB, IN and RP, and RB and RP.

Step two involves calculating a second set of pairs, namely the set of varying tags which remain in context—the non-fringe variation tags. This set is taken directly from the variation trigrams. We count how often each variation type occurs in the same way as the unigrams.

From these two sets, in step three we can derive a simple metric which indicates how often a particular tag pairing is not disambiguated. To do this, we take a particular tag pairing and divide its count of contextual occurrences by its total number of occurrences, as given in (71).

(71) $\frac{\#\ \textit{non-fringe types}}{\#\ \textit{total types}}$

Generally speaking, the higher the value, the harder it was to disambiguate those tags in different contexts, either because the distinction is non-local or the distinction is difficult to make reliably. If a pairing gets a score of zero, it is always disambiguated in context. The highest score is unbounded since, due to the fact that we count by types, and that a single unigram type can result in multiple trigram types, scores

often exceed 1.[96] The highest score is 17.5 for POS (possessive ending) and VBZ (third person singular present verb), which reflects a tag distinction that is often non-local.

Outside of this one variation, the individual tags POS and VBZ are not as difficult to resolve in context. Furthermore, the metric we have above heavily favors distinctions which only apply to a few number of word types—in this case, only two (*'s* and *'S*)—since the denominator will be low. One way to overcome this limitation of the metric is to instead focus on individual tags which frequently contribute to problematic distinctions.

And so in step four, we find out which particular tags are likely to be the most problematic for a tagger to assign. To do this, we pull out individual tags from each tag pairing and add up their scores. For example, for IN we add 12.5 for DT/IN, 7.0 for IN/RP, 6.53 for IN/RB, and so on down to 0.048 for IN/NN, giving a total of 35.82. The intuition here is that if a single tag is often not disambiguated for a variety of words and with a variety of other tags, that tag is likely to be difficult to reliably assign. The five highest-ranked and likely most problematic tags are given in figure 6.2,[97] and we will show that the top-marked ones are indeed difficult for taggers to assign.

[96] 31 of the 327 tag pairings have values greater than 1; 169 have values of zero and so are not problematic at all. They might, however, be worth investigating to see if their low frequency variations should vary at all.

[97] The tags have the following meanings (Santorini, 1990): IN = preposition or subordinating conjunction, DT = determiner, RB = adverb, VBZ = third person singular present verb, POS = possessive ending.

Tag	Score
IN	35.82
DT	35.78
RB	28.16
VBZ	18.63
POS	18.5

Figure 6.2: The most problematic tags in the WSJ

6.2.1 Filtering problematic tags for TnT

In chapter 5 we tried various automatic correction methods for POS annotation by examining how different tagging models performed on positions flagged by the variation n-gram method as potential errors. To find out how problematic the three highest-ranked tags are in this context, we experimented with filtering out the problematic tags from the automatic correction output. Specifically, we remove the corpus occurrence from evaluation for a tagger if the tagger has assigned the problematic tag in question. We ignore the original tag here because we are interested in tags which are hard for a tagger to distinguish.

Filtering out the tags for the TnT automatic correction output (see section 5.3.1), we see a gradual increase in precision when removing the problematic tags from the evaluation, as shown in figure 6.3.[98]

In chapter 5 we improved tagger performance by using complex ambiguity tags (see section 5.4), and we here evaluate the effect of filtering out problematic tags in figure 6.4. We can see that removing the two worst tags (IN and DT) gives us a

[98] As in chapter 5, we here report precision figures for the corpus positions in the benchmark which the tagger changed, the positions it did not change, and its performance overall on the variation-flagged samples.

Removed tag(s)	Changed	Unchanged	Total
(none)	68.47% (76/111)	87.30% (165/189)	80.33% (241/300)
IN	74.49% (73/98)	90.45% (161/178)	84.78% (234/276)
IN,DT	74.23% (72/97)	91.28% (157/172)	85.13% (229/269)
IN,DT,RB	76.74% (66/86)	91.56% (141/154)	86.25% (207/240)

Figure 6.3: TnT results when filtering out the three most problematic tags

total of 89.89% precision and in so doing only sets aside 23 (out of 300) tags to be hand-examined. The slight drop in performance in filtering out RB indicates that the tagger's performance on RB is actually better than the overall TnT performance and illustrates the point that we do not want to filter out too many tags.

Removed tag(s)	Changed	Unchanged	Total
(none)	86.36% (76/88)	86.79% (184/212)	86.67% (260/300)
IN	88.09% (74/84)	89.60% (181/202)	89.16% (255/286)
IN,DT	87.95% (73/83)	90.72% (175/194)	89.89% (249/277)
IN,DT,RB	87.32% (62/71)	89.87% (151/166)	87.32% (213/237)

Figure 6.4: TnT results with complex ambiguity tags when filtering out the three most problematic tags

Assuming that human effort will result in 100% correction for the 23 samples of IN and DT-tagged positions, this would mean that we will get 272 out of 300 correct, or 90.67%. But how much human effort will this require? Of the 21,575 variation

positions, 755 are IN and 227 are DT, adding up to 982 total positions to check by hand. And if we only wish to check the changed positions, there are only 225 (170 IN and 55 DT).

We can thus see that sorting tags based on an automatic metric of how problematic they are can be quite useful. Perhaps a better metric can be derived, by basing it on token counts, but using something as simple as the metric we calculated, we have obtained good results.

We have discussed removing the tags based on the resolved tag which the tagger assigned, but for the model with complex ambiguity tags we have more information than that in the ambiguity class. Instead of removing the worst tag (IN), we can instead try removing all tags where the worst tag was a part of the ambiguity class. This, however, is not as effective. When removing the tags for any spot which has IN in its ambiguity class, we obtain a precision of 87.55% (211/241) (85.71% (60/70) for changed positions, 88.30% (151/171) for unchanged), and in the process we remove 59 tags, out of the 300. The performance is worse because some of the tags that IN is in variation with are much less problematic.

6.2.2 Filtering problematic tags for the Decision Tree Tagger

We find comparable results for the Decision Tree Tagger (see section 5.3.2) on the 300 samples as we did with the TnT tagger, as shown in figures 6.5 and 6.6. As described earlier for figure 5.2, we used the defaults for the system in one case (Default) and assigned every non-variation position its original tag in the other (Using tags). We make no further comment here.

	IN?	Changed	Unchanged	Total
Default	YES	73.83% (79/107)	90.15% (174/193)	84.33% (253/300)
Default	NO	77.78% (77/99)	91.85% (169/184)	86.93% (246/283)
Using tags	YES	72.82% (75/103)	89.34% (176/197)	83.67% (251/300)
Using tags	NO	76.84% (73/95)	91.44% (171/187)	86.52% (244/282)

Figure 6.5: Decision Tree Tagger results when filtering out IN

	IN?	Changed	Unchanged	Total
Default	YES	86.21% (75/87)	85.92% (183/213)	86.00% (258/300)
Default	NO	89.02% (73/82)	89.00% (178/200)	89.01% (251/282)
Using tags	YES	85.88% (73/85)	86.05% (185/215)	86.00% (258/300)
Using tags	NO	88.75% (71/80)	89.11% (180/202)	89.01% (251/282)

Figure 6.6: Decision Tree Tagger results with ambiguity tags when filtering out IN

6.2.3 Filtering problematic tags for the Brill Tagger

Removing IN tags from the output of the Brill tagger (using all 26 templates, described in section 5.3.3) also results in higher precision values, as shown in figure 6.7. Note that the Brill tagger performs equally well for the threshold values of 7, 10, and 15 after removing IN from evaluation. This indicates that their (already small) differences in precision were mostly due to this single difficult tag.

Threshold	IN?	Changed	Unchanged	Total
7	YES	75.79% (72/95)	83.41% (171/205)	81.00% (243/300)
7	NO	80.95% (68/84)	88.24% (165/187)	85.98% (233/271)
10	YES	75.00% (69/92)	84.13% (175/208)	81.33% (244/300)
10	NO	80.25% (65/81)	88.48% (169/191)	86.03% (234/272)
15	YES	74.49% (73/98)	85.64% (173/202)	82.00% (246/300)
15	NO	79.31% (69/87)	88.83% (167/188)	85.82% (236/275)
20	YES	70.59% (72/102)	84.85% (168/198)	80.00% (240/300)
20	NO	76.40% (68/89)	88.04% (162/184)	84.25% (230/273)

Figure 6.7: Brill Tagger results when filtering out IN

6.2.4 Filtering problematic tags for TiMBL

Identifying and removing problematic tags shows much promise as an aid in error correction on all three taggers we have looked at. It seems that this is even more the case with an MBL system: removing the two most problematic tags (IN and DT) from the automatically-corrected set drastically improves the results, especially on changed positions.

Using the default options (see section 5.3.4), we can compare the results of a regular run of TiMBL with the results after removing IN, as shown in figure 6.8. Recall, too, that the best results were obtained for $k = 3$ and $k = 4$, with the other options remaining as the defaults. After removing IN, we find even better results, with $k = 3$ being the best.

The results improve even more when we remove both IN and DT, the two most problematic tags, as shown in figure 6.9, so much so that we begin to approach the level of accuracy we obtained with an n-gram tagger. With $k = 3$, for example, we went from 79.33% (238/300) total accuracy to 88.08% (229/260) total accuracy.

k	IN?	Changed	Unchanged	Total
1	YES	74.42% (32/43)	78.99% (203/257)	78.33% (235/300)
1	NO	81.58% (31/38)	82.08% (197/240)	82.01% (228/278)
3	YES	76.19% (64/84)	83.80% (181/216)	81.67% (245/300)
3	NO	82.19% (60/73)	88.38% (175/198)	86.72% (235/271)
4	YES	73.33% (66/90)	85.24% (179/210)	81.67% (245/300)
4	NO	77.50% (62/80)	89.17% (173/194)	85.77% (235/274)

Figure 6.8: TiMBL results when filtering out IN

k	IN,DT?	Changed	Unchanged	Total
3	YES	76.19% (64/84)	83.80% (181/216)	81.67% (245/300)
3	NO	84.29% (59/70)	89.47% (170/190)	88.08% (229/260)
4	YES	73.33% (66/90)	85.24% (179/210)	81.67% (245/300)
4	NO	79.22% (61/77)	89.84% (168/187)	86.74% (229/264)

Figure 6.9: TiMBL results when filtering out IN and DT

The question, then, is why do we see so much change? Or, to phrase it differently, why is TiMBL even more inaccurate at tagging the most problematic tags than TnT or DTT? To answer that, we first must remember that TiMBL is matching the data as well as it can, and that we have shown these tags to be problematic in various contexts. So, when one tries to match the data, one will just as often as not guess the wrong variation tag.

Another difference is that a tagger like TnT can be swayed by large probabilities in a given context. TiMBL, on the other hand, has a single way of weighting features which applies over the whole corpus. For example, in the context preceding *$*, we see that the chance of *about* being tagged RB (vs. IN) increases significantly for TnT

(see section 5.4 under *The rationale for complex ambiguity tags*). When we look at the same corpus position (1346) for TiMBL, we are looking at static features (not influenced by the surrounding choices of tags by TiMBL) and we do not know which feature is locally the most important tag or how important that feature is. That is, we find the following for *about*, where IN was selected by the tagger for this position.

(72)

w	t_{-2}	t_{-1}	t_{+1}	t_{+2}	tag
about	NN	IN	$	CD	IN

Over the whole corpus, the feature for t_{+1} is the most important (using Gain Ratio weighting), followed by t_{-1}, but we do not know how important each is for this position. All we know is the *k-nn* neighbors which match this situation. We do not have a way to say that RB is more likely than normal for *about* when preceded by IN and followed by $. If a lot of IN-tagged cases in memory also have NN and CD as t_{-2} and t_{+2}, respectively, then those suddenly have a much higher importance than with the HMM tagger. And indeed the tag of RB is never given for *about* when the first tag in the sequence is NN, even though the first tag is almost irrelevant for these purposes.

6.2.5 BNC-sampler

We also derived the most problematic tag in the BNC-sampler, again by comparing the tag variations in the trigrams with those in the unigrams. Interestingly, the tag II (general preposition) was deemed the most problematic, parallel to the most problematic tag in the WSJ (IN).

TnT By filtering out this tag, we obtain only very slightly better results for TnT, as shown in figure 6.10. With some human assistance, however, we can identify more

	II?	changed	unchanged	total
Regular	YES	46.00% (23/50)	93.20% (233/250)	85.33% (256/300)
Regular	NO	45.45% (20/44)	93.36% (225/241)	85.96% (245/285)
Ambclass	YES	43.75% (14/32)	91.79% (246/268)	86.67% (260/300)
Ambclass	NO	44.83% (13/29)	92.22% (237/257)	87.41% (250/286)

Figure 6.10: TnT results on the BNC-sampler when filtering out II

problematic tags in the error detection phase. As discussed in chapter 2, variations involving verbal tags were particularly damaging to the variation n-gram error detection method. Thus, we can experiment with the results we get by removing verbal tags from consideration. Although this may seem like a good idea in principle, in practice it does little to change the outcome with respect to precision, as shown in figure 6.11.

	Verbs?	changed	unchanged	total
Regular	YES	46.00% (23/50)	93.20% (233/250)	85.33% (256/300)
Regular	NO	50.00% (17/34)	90.74% (98/108)	80.99% (115/142)
Ambclass	YES	43.75% (14/32)	91.79% (246/268)	86.67% (260/300)
Ambclass	NO	47.62% (10/21)	87.80% (108/123)	81.94% (118/144)

Figure 6.11: TnT results on the BNC-sampler when filtering out verbal tags

Note, though, that most of the tags which we have removed did not need to be changed, and so we have grouped the tokens into two classes: those more likely to need fixing and those less likely. In the former case, we have non-verb tags, comprising 142 (144) examples. Of those, about 25 need repair. In the latter case, we have the

problematic verbal tags, comprising 158 (156) examples. Of these only about 10 need to be repaired. Thus, we see that the first class is about three times more likely to need repair, promising results for token error detection.

Decision Tree Tagger Results for the decision tree tagger are virtually identical to those for TnT in their slight improvements, as shown in figure 6.12, and so we make no comment on them.

	II?	Changed	Unchanged	Total
Regular	YES	47.73% (21/44)	92.19% (236/256)	85.67% (257/300)
Regular	NO	48.65% (18/37)	92.28% (227/246)	86.57% (245/283)
Ambclass	YES	50.0% (17/34)	92.11% (245/266)	87.33% (262/300)
Ambclass	NO	55.17% (16/29)	91.80% (235/256)	88.07% (251/285)

Figure 6.12: Decision Tree Tagger results on the BNC-sampler when filtering out II

TiMBL As with the TnT and DTT work above, filtering out the most problematic tag (II) for TiMBL does very little to improve accuracy, as shown in figure 6.13.

6.2.6 Summary for identifying problematic tags

Although the BNC-sampler results are still not as encouraging as with the WSJ, the best results for the WSJ, as given in figure 6.14 for the four different tagging approaches, are getting close to 90% correct. By identifying and isolating the most problematic tag, the taggers are much more reliable.

k	II?	Changed	Unchanged	Total
1	YES	53.33% (8/15)	90.88% (259/285)	89.00% (267/300)
1	NO	53.85% (7/13)	90.84% (248/273)	89.16% (255/286)
3	YES	31.48% (17/54)	93.50% (230/246)	82.33% (247/300)
3	NO	30.43% (14/46)	93.62% (220/235)	83.27% (234/281)
4	YES	29.69% (19/64)	94.07% (222/236)	80.33% (241/300)
4	NO	27.27% (15/55)	94.22% (212/225)	81.07% (227/280)

Figure 6.13: TiMBL results on the BNC-sampler when filtering out II

Tagger	Changed	Unchanged	Total
TnT	88.09% (74/84)	89.60% (181/202)	89.16% (255/286)
DTT	89.02% (73/82)	89.00% (178/200)	89.01% (251/282)
Brill	80.25% (65/81)	88.48% (169/191)	86.03% (234/272)
TiMBL	82.19% (60/73)	88.38% (175/198)	86.72% (235/271)

Figure 6.14: The best results after removing IN

6.3 Majority label classification

In the previous section, we showed how information from the variation n-gram output can assist in identifying problematic tags. In this section, we will show that information about the distribution of tags within a trigram can further provide us with an indication of the correct tag. We will use the majority tag from a non-fringe variation trigram nucleus in two ways, first as a classifier in its own right and then as part of a voting system of taggers.

To see how informative the majority tag is for selecting the correct tag in a variation, we first selected the majority tag of a trigram variation nucleus as the correct tag, dubbing this method "majority rules."[99] We use the non-fringe trigrams as we have done before, with the added reason that the frequencies of tags in longer n-grams are too small to be very informative. Taking the trigrams gives a better picture as to how the tags generally pattern in a context.

With majority by itself we obtain moderate results, as shown in figure 6.15, where we obtain a total precision of 82.5%. It uses a very informative context, that of the surrounding lexical items, as opposed to many taggers which use the surrounding tags. Such taggers look at tags in order to avoid data sparseness, but that is not as much of a problem when training and testing on the same data. Even so, 20% of our sample set (60 examples) have no majority tag—i.e., the tags occur an equal number of times—and so are unable to provide a clear tag.

Changed	Unchanged	Total
78.43% (40/51)	83.60% (158/189)	82.5% (198/240)

Figure 6.15: The accuracy of the majority tag

Because a method using only the majority tag for correction considers no more than the surrounding words and only the tag variations for that position, it needs some assistance in sorting out the general corpus properties, i.e., what patterns the

[99]More properly, *majority* should be called *plurality*: there are some cases where the most frequent tag occurs less than 50% of the time because there are more than two tags involved. It is possible that these cases should be trusted less, but for now we treat them on par with the other cases, and we use the term *majority* as a more intuitive mnemonic.

tags normally follow in the corpus. Knowing that, for example, RB is more popular than IN in a given variation takes no account of how RB and IN normally pattern.

More sophisticated classifiers sometimes do better than majority rules at correction because they consider more general properties. For the following, we will consider the properties of an HMM tagger (TnT) for comparison to majority rules; most taggers share these properties in one way or another. First, they have a more general context of POS tags, which informs the tagger about how types of words generally pattern. Secondly, they often use a wider window of context; in the case of TnT, each tag is directly affected by the surrounding two items on each side, as opposed to just the word on the left and the word on the right, as with majority rules. Thirdly, by using lexical probabilities, TnT accounts for the more general patterns of tags for a given word. Majority rules only knows that within a particular context, tag A occurs more often than tag B; an HMM tagger is able to further figure out which of tag A or tag B is more popular overall for the focus word.

But most classifiers are not attuned to the fact that these positions are special—as majority rules is—and that annotators disagreed about how to annotate them. Furthermore, as we saw with TiMBL, sometimes the probabilities of items further away from the focus word actually hurt the probability of the correct tag. That is, the tagger can use too much context, giving too much credence to context which is irrelevant for the current tag. For example, the choice between a past tense verb and a past participle is dependent upon the preceding words, and the probabilities following the verb may sway the tagger to select the wrong tag. Majority rules based on non-fringe trigrams is only influenced by the immediately-surrounding context.

Taking these complementary properties into account, we turn to using other classifiers in conjunction with majority rules. If both a classifier and the majority tag agree, then we have two different, somewhat complementary sources of evidence for the assigned tag, and so we should feel more confident about using that label. This is essentially the same thing as using a voting system of taggers to identify which tags to keep (cf. van Halteren et al., 2001), except that one of our "taggers" is the simple majority rules method.

Turning to the results, as shown in figures 6.16 and 6.17, we get upwards of 90% precision for the class of labels which are the majority and which were also selected by the tagger, for both the altered and unaltered TnT models.

	Changed	Unchanged	Total
Same as majority	84.85% (28/33)	89.86% (133/148)	88.95% (161/181)
Different from majority	62.79% (27/43)	81.25% (13/16)	67.80% (40/59)
No majority	60.0% (21/35)	76.0% (19/25)	66.67% (40/60)

Figure 6.16: TnT results, based on whether the tag agrees with the majority tag

	Changed	Unchanged	Total
Same as majority	97.06% (33/34)	89.51% (145/162)	90.82% (178/196)
Different from majority	85.19% (23/27)	88.24% (15/17)	86.36% (38/44)
No majority	74.07% (20/27)	72.73% (24/33)	73.33% (44/60)

Figure 6.17: TnT results with complex ambiguity tags, based on whether the tag agrees with the majority tag

1. Tagger agrees with the majority tag.
2. Tagger disagrees with the majority tag.
3. There is no majority tag.

Figure 6.18: Ranking of the reliability of tags

The number of elements which fall into the set obtained by taking the union of the majority and tagger cases are almost two-thirds of the whole set: 181 out of 300 in the original TnT case and 196 out of 300 in the case of TnT with complex ambiguity tags. The ratio for changed tags is slightly lower: 33 out of 111 (regular TnT) and 34 out of 88 (complex TnT), indicating that these changed areas are less obvious cases to tag for either method.

We can observe that the worst results are not found where the majority label is different from the classifier, but where there is no majority label available. This should not be too surprising, given that no majority usually means that there is not much data, causing the taggers to be less reliable. With these results, we can establish a hierarchy for how trustworthy a tag. This is presented in figure 6.18.

As mentioned, what we are doing is basically voting, but by focusing on whether the original tags were changed or not, we are actually voting with three pieces of evidence: 1) tagger information, 2) majority label, and 3) original corpus data. Unlike a straightforward voting algorithm, however, it is not clear how to score the original data. It should not always receive a positive vote; we see that in the complex ambiguity case where the tagger and majority labels agree, they are more often correct when they have changed the data, and thus are voting against the original data.

Viewing this as a three-way voting task might be useful, though: note that when the tagger and the corpus agree, but the majority label does not (the *unchanged* category of the *different* rows), we find a fairly high percentage. Thus, two pieces of evidence outweigh the other. But we do not always find higher percentages when two of the three pieces of data agree. The *changed* category of the *different* rows shows that when the tagger disagrees with both the corpus and the majority, the tagger is still more often correct.

Thus, we see that the most reliable piece of information is the tagger, followed by the majority label, and finally by the corpus label (we are, after all, trying to correct spots we have deemed likely in need of correction). With this in mind, we can move towards a more fine-grained ranking of the reliability of the information we have. We give such a ranking in figure 6.19, where 1 is the most reliable source of information (T = tagger, C = corpus, M = majority). If we are only looking at the changed tags, this can be simplified to figure 6.18.

1. Tagger has changed the data and agrees with the majority tag (TM).
2. Tagger has not changed the data and agrees with the majority tag (TCM).
3. Tagger has not changed the data and disagrees with the majority tag (TC).
4. Tagger has changed the data and disagrees with the majority tag (T).
5. There is no majority tag.

Figure 6.19: Ranking of the reliability of tags (positive votes in parentheses)

6.4 Finer-grained distinctions

Although the large groupings provided by the majority tag information represent a significant gain in terms of being able to prioritize our ability to automatically correct, we are still in pursuit of corpus positions which can be corrected with 100% accuracy. We have fairly broad groupings, but perhaps if we can rank the spots even further, we can identify a point above which automatic correction is allowable.

To that end, we establish numerical scores for each corpus position, scores which loosely correspond to how confident we can be in changing the tag at that position. We selected two main schemes for ranking corpus positions: 1) using the percentage of time in the variation trigram where the majority tag was used in the original corpus (what we will call *proportion*), and 2) using the amount of variance in a variation (what we will call *variance*). Other schemes could be used, but these seem to capture the relevant properties.

6.4.1 The methods of ranking

The idea of using the *proportion* of a tag is rather simple: we take all the tags for the nucleus of a trigram variation and find the percentage of tags which correspond to the majority. That is, we find how much of a majority it is. For example, in *the American depositary*, *American* is once annotated NNP (proper noun) and twice annotated JJ (adjective). The proportion score is 2/3, or 0.67 (i.e., 67%). The formula is given in (73), where t_{maj} is the majority tag, t_i is a tag in the variation, and $C(X)$ is a frequency count.

(73) $\frac{C(t_{maj})}{\sum_i C(t_i)}$

What we have loosely called *variance*, on the other hand, represents how skewed, or varied, the distribution of tags is in a trigram. We sum the squares of differences between the frequency of each tag and the mean, and then we average this "skew value" (Kaljurand, 2004) by dividing by the number of tags which are in variation. The formula is given in (74), where t_i is a tag in the variation, $\bar{t}$ is the mean of the tags in the variation, and $|T|$ is the total number of tags in the variation.

(74) $$\frac{\sum_i (C(t_i) - \bar{t})^2}{|T|}$$

For example, with the trigram *the American depositary*, the mean of the variations is 1.5 (=(1+2)/2)). The sum of the squares of the differences is: $(1-1.5)^2+(2-1.5)^2 = (-0.5)^2 + 0.5^2 = 0.5$. Dividing this by two (the number of unique tags) results in a score of 0.25.

There are significant differences in the aims and outcomes of these two other methods. The proportion method normalizes the data to some extent. For instance, if the majority tag outranks the other tag 3 out of 4 times, this is the same thing as being the selected tag 30 out of 40 times. In both cases, the proportion score is 0.75.

Because the variance measure, however, takes into account absolute counts in its calculations, the same proportions with higher frequency counts will have higher variance values. For example, the variance of the 3 out of 4 case is 1. For the case of 30 out of 40, the variance is 100. The scores here are considerably different, whereas they were equal with the proportion method. Note that when there is no majority tag, neither score is informative; the proportion will always be 50% (if there are only two tags) and the variance will be zero.[100]

[100]When there are three tags with the same counts, the variance is still zero, matching the situation of two tags with the same counts. The proportion score, however, drops to 0.33, and so is more swayed by the number of tags.

The question we want to answer is: which score is the better indicator that something needs to be changed? We are using these scores to rank the output of a tagger, and we want to see which corpus positions are most in need of change. Note, then, that which score works best might differ depending on which group of tags we are looking at. For instance, the proportion score—derived from the majority—might be best when the tagger agrees with the majority tag, but maybe less so when it does not.

In some ways the proportion score seems to be a better indicator of how reliable a change is: tags which are changed and are a high majority intuitively should be changed. But in other ways it seems that the variance might be a better indicator of trustworthiness of a tag. The variance score says that outscoring another tag by 20 occurrences is much stronger evidence for the majority tag than outscoring another tag by 2.

6.4.2 Results of ranking

We will compare these two methods of ranking in two different ways, emphasizing different properties of the results. The first way is to automatically group scores into tiers and see how many of each tier are correct or incorrect. This tier-based evaluation will show in broad terms how effective the more fine-grained rankings are. The second way of evaluation is to find the rank of the first incorrect example. This will show how many examples can be automatically corrected before manual correction should begin.

Tier-based evaluation

Using tiers of scores to evaluate the proportion and variance rankings is useful to see how well the rankings distribute the correct and incorrect tags. This method of evaluation places a priority on sorting good tags from bad tags but without necessarily demanding 100% correction (which will be useful when discussing token error detection in section 6.5).

Tier-based evaluation for proportion ranking We first look at how the proportion score does in ranking the output of the majority rules correction method (excluding cases of no majority), as shown in figure 6.20.

Tier	All (240) Precision	Changed (51) Precision
0.4	100% (2/2)	100% (2/2)
0.5	70.83% (17/24)	71.43% (5/7)
0.6	71.01% (49/69)	76.47% (13/17)
0.7	83.64% (46/55)	84.62% (11/13)
0.8	95.23% (40/42)	88.89% (8/9)
0.9	95.83% (46/48)	100% (3/3)

Figure 6.20: The accuracy of different tiers for majority rules using the proportion ranking

We obtain the tiers by truncating all decimal values to the tenth decimal place and taking all those values together; for example, 0.56, 0.53, and 0.5 all go into the 0.5 tier. Aside from the small number of examples in the 0.4 tier, we see a steady upward trend for all positions and for the changed positions. We move from a precision of

70% in the 0.5 bracket to precisions in the 90% range in the 0.9 bracket. Even without using a more sophisticated classification technique, we see that we can obtain high precision within the higher tiers for majority rules correction. In other words, when restricted to strong majorities (0.7 and above), majority rules correction is better than we first thought (as in figure 6.15).

The same upwards trends are observed using the proportion ranking on the output of the regular TnT correction method and the method using TnT with complex ambiguity classes. These results are shown in figures 6.21 and 6.22, respectively. Although we do not show the results here, the same trends (with higher numbers) are observed when isolating those spots where the tagger agrees with the majority.

Tier	All (300) Precision	Changed (111) Precision
0.4	100% (2/2)	n/a
0.5	61.18% (52/85)	53.33% (24/45)
0.6	81.16% (56/69)	78.13% (25/32)
0.7	83.64% (46/55)	72.22% (13/18)
0.8	88.10% (37/42)	81.82% (9/11)
0.9	100% (48/48)	100% (5/5)

Figure 6.21: The accuracy of different tiers for TnT using the proportion ranking

Tier	All (300) Precision	Changed (88) Precision
0.4	100%	n/a
0.5	75% (63/84)	76.47% (26/34)
0.6	91.30% (63/69)	96% (24/25)
0.7	87.27% (48/55)	88.24% (15/17)
0.8	90.48% (38/42)	88.89% (8/9)
0.9	95.83% (46/48)	100% (3/3)

Figure 6.22: The accuracy of different tiers for TnT with complex ambiguity tags using the proportion ranking

Here we see a much more drastic leap from the 0.5 tier to the 0.6 tier, indicating that proportion ranking is even more useful for sorting classifier output. This makes sense: for majority rules, the proportion ranking is simply a more fine-grained analysis of the same (majority) information. For classifiers, this is truly a new piece of information, indicating where the evidence is weak or strong.[101]

Tier-based evaluation for variance ranking We want to compare the proportion rankings to the variance rankings, but there is no exact one-to-one comparison. The variance rankings have different values and thus form different tiers. We can, however, observe if the general trends are the same or not. We assigned the tiers as follows: after truncating the decimal point, we put the values into groups which were basically logarithmic, ending up with classes of: 0, 1-100, 101-1000, and greater than 1000.

[101]One could also use the ranked output of TnT and group the tags by the probabilities TnT assigned. We expect similar results (and preliminary results indicate as much), but do not try that here, given that not all classification output has the option of probabilistic rankings of choices.

Again, we first looked at the majority rules method of correction, and here we see both an advantage and a disadvantage of the variance ranking over the proportion ranking. For the top two tiers, we obtain 100% precision. At a high enough level of variance, we can be confident in majority rules without using a classifier. The reason this is true is that the variance ranking takes into account not only how strong a majority is (the relative difference between the frequencies of two tags), but also that there is a wide absolute difference between the most popular tag and the next most popular tag. But this is true for a small number of examples—only for 12 out of 51 samples, in the case of the changed positions. The precision is much worse for a greater number of samples (the 0 and 1-100 tiers) than with the proportion ranking.

Tier	All (240) Precision	Changed (51) Precision
0	74.22% (95/128)	72.41% (21/29)
1-100	79.55% (35/44)	70% (7/10)
101-1000	100% (38/38)	100% (9/9)
1001+	100% (20/20)	100% (3/3)

Figure 6.23: The accuracy of different tiers for majority rules using the variance ranking

Turning to the TnT output, in figures 6.24 and 6.25, we see the same general upwards trend in precision as was observed for the proportion method in figures 6.21 and 6.22. And, as with the majority rules method, we see very high accuracies in the upper two tiers. But again a downside to the variance measure is the sheer number of examples which are grouped into the lowest tier. A full 63% (188/300) of cases are

in the lowest tier, while only 28% (85/300 and 84/300) of the cases for the proportion method are in the lowest (0.5) tier.[102] And with the proportion method, we see a sharp rise between the 0.5 bracket and the 0.6, from roughly 60% to 80% in the case of the regular TnT method and from 75% to 90% in the case of the TnT method with complex ambiguity classes. Thus, the proportion method does a better job of distributing the corpus positions.

Tier	All (300) Precision	Changed (111) Precision
0	72.34% (136/188)	62.79% (54/86)
1-100	88.89% (48/54)	85.71% (12/14)
101-1000	97.37% (37/38)	87.5% (7/8)
1001+	100% (20/20)	100% (3/3)

Figure 6.24: The accuracy of different tiers for TnT using the variance ranking

Tier	All (300) Precision	Changed (88) Precision
0	82.98% (156/188)	83.58% (56/67)
1-100	85.19% (46/54)	88.89% (8/9)
101-1000	100% (38/38)	100% (9/9)
1001+	100% (20/20)	100% (3/3)

Figure 6.25: The accuracy of different tiers for TnT with complex ambiguity tags using the variance ranking

[102]Technically, 0.4 is the lowest tier, but since there are so few cases there, it would probably be best to simply group them with the 0.5 group.

Winner	Group	Method	First for All	First for Changed
Variance	All	Proportion	31 (240)	6 (51)
		Variance	63 (240)	13 (51)

Figure 6.26: The ranking of the first item which is a wrong tag assignment by majority rules

Ranking evaluation

In order to place a priority on doing as much automatic correction as possible, we count how many examples can be automatically corrected before a miscorrection is encountered. This is simply another way to look at the same results and was alluded to when we mentioned how the variance method gives 100% precision in the higher tiers. We are testing how many examples it takes before the precision is no longer 100%.

We present the results here, for the majority rules method in figure 6.26 and for the TnT methods in figures 6.27 and 6.28. Since we want as many examples as possible to be correct before the first incorrect answer is encountered, higher values are preferable.

In general, we see variance as the better method, but it is important to note where this happens. First of all, with the majority rules method of correction by itself, it works better than the proportion ranking. And for the regular TnT and the TnT with complex ambiguity tags methods, it works better when the tagger agrees with the majority. This is likely because variance gives information not already present in the majority label.

Winner	Group	Method	First for All	First for Changed
Proportion	All	Proportion	58 (300)	6 (111)
		Variance	23 (300)	4 (111)
Variance	Agrees w/ maj.	Proportion	64 (181)	5 (33)
		Variance	79 (181)	14 (33)
Proportion	Disagrees w/ maj.	Proportion	5 (59)	4 (43)
		Variance	1 (59)	1 (43)

Figure 6.27: The ranking of the first item which is a wrong tag assignment by TnT

Winner	Group	Method	First for All	First for Changed
Variance	All	Proportion	31 (300)	4 (88)
		Variance	67 (300)	19 (88)
Variance	Agrees w/ maj.	Proportion	31 (196)	16 (34)
		Variance	64 (196)	16 (34)
Variance	Disagrees w/ maj.	Proportion	3 (44)	2 (27)
		Variance	7 (44)	4 (27)

Figure 6.28: The ranking of the first item which is a wrong tag assignment by TnT with complex ambiguity tags

The regular TnT results, as given in figure 6.27, again highlight the advantage gained when prioritizing positions where the tagger agrees with the majority. For the variance ranking, we find that the twenty-third value of all 300 is wrong, but when we narrow it down to the 181 positions where the tagger agrees with the majority we find that it is not until the seventy-ninth value that we find an incorrect tag, a huge improvement. Recall that the variance ranking has high values for high absolute differences; this generally means that TnT has plenty of data to make a decision, and so it makes an informed decision.

The variance method clearly has its strengths, but because the proportion method does well for the regular TnT method, we cannot clearly say which ranking system is better. It is clear, however, from this section and the previous one that using some ranking system (or combination thereof) is beneficial for dealing with the most likely corrections first. Indeed, with the variance method where the tagger agrees with the majority, we find 30-40% of the cases which are correct before an incorrect case is encountered. Being able to automatically correct 30% of the errors would represent a huge gain in efficiency in corpus correction.

6.5 Token error detection

We have shown in the previous section that in the case of POS annotation of the WSJ corpus, we can designate a set of errors which can be automatically corrected. This is a significant achievement, but we have not shown the method to be generally applicable, and so it does not guarantee accurate corrections for every corpus. To err on the side of safety, thus, we propose taking the automatic correction results and using them for token error detection.

Because the variation n-gram method detects error types, part of the task we have to deal with in correction is not only to correct tags, but to figure out which tag tokens to correct. With the variation n-gram error detection method, a type is identified as an error, but which tokens are errors is left undecided. For correction, we have to identify tokens which are wrong; the task of token error detection has thus far been integrated into the error correction stage.

If we look back over the results in this chapter and the previous one, we can see that the methods we have devised for correction are highly effective for the task of token error detection. For the WSJ corpus, we have obtained high percentages of correction, which means that: a) an error was successfully detected, and b) the suggested replacement tag was correct. In other words, using these correction methods as token error detection methods is actually better than simple token error detection; it is error detection with a fairly accurate guess for the correct label.

For example, we reported in section 5.4 that TnT with complex ambiguity tags obtains 86.36% precision on changed positions, corresponding to 4640 successful corrections out of 5373 changes. If we examine only the 5373 changes, then we have a token error detection rate of approximately 86.36%. Compare the reported accuracy of 20.5% in van Halteren (2000) for the similar task of comparing the benchmark to a tagger without having identified areas of inconsistency first.

Furthermore, for those detected instances, we have an accurate assignment of a correct tag. So, even if one were to correct these 5373 spots by hand in order to guarantee 100% error correction, one would not have to sort through much in the way of false corrections. Similar to Brants and Skut (1998) and Brants and Plaehn (2000), we can enhance annotation efforts by giving annotators a choice of two tags, the original tag and the suggested correction. Brants and Plaehn (2000) shown that this method of annotation speeds up the annotation process, and so going through 5373 examples by hand is very feasible.

The error correction results for the BNC-sampler were less good than those for the WSJ in terms of precision, but positive in terms of token error detection. Although we did not repair the tag appropriately in a majority of cases, we did succeed in

grouping the token occurrences into two groups: those more likely to need repair and those less likely to need it. This is consistent with the results from the experiments on the WSJ.

The way to sort groups most effectively is to maximize the precision of the unchanged tokens—the closer we are to putting 100% of the tokens into the unchanged group, the more we have appropriately sorted the tokens into the right classes. For example, as shown in figure 6.29 (taken from figure 5.2) for the decision tree tagger, we obtained a precision of 90.15% on the unchanged positions.

	Changed	Unchanged	Total
Default	73.83% (79/107)	90.15% (174/193)	84.33% (253/300)

Figure 6.29: Results of running the Decision Tree Tagger on the 300 samples

This means that the tagger did a good job of changing tags only when necessary, and so nearly every tag that it should have changed is in the changed portion of the sample. If we obtain 100% on the unchanged positions, we know that we have 100% recall within the changed positions.

6.6 Summary for automatic sorting

Building on the work of automatically correcting a corpus and using information present in the variation n-grams, in this chapter we have identified several different methods for sorting the output of an error correction method, based on how reliable the output is. We sorted the output in a few different ways.

First, we automatically identified problematic tagset distinctions, and we did this by using information found in the variation unigrams and trigrams. The effect of this was shown in an improvement in correction after removing the most problematic tag.

We then turned to sorting methods which relied on using the majority label information from a word's variation trigram. Using the majority and the distribution of tags in a trigram, we were able to further sub-rank the correction output, thereby prioritizing corpus positions for correction. In fact, we were able to identify a set of errors which can be automatically corrected. This is significant, in that automatically detecting and correcting errors has never been done before in the same process—i.e., people have found ways to automatically detect errors (see section 1.3.3) or have written rules to correct erroneous patterns (e.g., Oliva, 2001), but never has the whole process been automated from start to finish, for even a subset of the data.

Finally, we discussed how automatic correction methods could be re-interpreted as token error detection methods. In addition to merely identifying the incorrect tag for a corpus position, though, these methods provide a suggested tag, which allows one to quickly detect and correct errors in a corpus.

The general correction procedures we have described for POS annotation can in principle be applied to our syntactic error detection work. Since we also have variation n-grams for syntactic annotation, it is clear how to derive a majority, but as for adapting classification techniques, we do not have a strict classification for each corpus position since the annotation is non-positional. Thus, we cannot simply expand the correction methods from this and the previous chapter to syntactic annotation.

To sketch this out a bit more, however: instead of using simple classifiers, one can use a PCFG induced from a treebank to assign structure to the same treebank—i.e., train and evaluate on the same corpus. From the resulting corpus, one can extract the relevant category (or NIL) labels for a given string, to see if the analysis agrees with benchmark or not and what it suggests instead. Given that parsers are generally less accurate than taggers, it is not clear whether this methodology will be effective or not, and it will require non-trivial extensions to deal with the NIL tags.

CHAPTER 7

Summary and Outlook

In this thesis we have outlined an error detection and correction process for annotated corpora. After showing the need for error detection and correction in annotated corpora (chapter 1), we addressed that need by defining/developing and demonstrating the effectiveness of the variation n-gram method for finding errors in a range of increasingly complex annotation types (chapters 2, 3, and 4). We then showed that the output of the variation n-gram method can be combined with classification techniques in order to provide semi-automatic correction of annotation errors (chapters 5 and 6).

This is the first time that an automatic error detection method—independent of corpus, language, or annotation scheme—has been explored so thoroughly and for such a range of annotation types. To the best of our knowledge, this also is the first time that anyone has demonstrated how to add semi-automatic correction on top of automatic detection of annotation errors.

Despite the success of the method so far, future work can and needs to examine and extend the variation n-gram method for detecting annotation errors to a) make it applicable for a wider range of annotation types, b) increase recall of the method through the refinement of what constitutes comparable contexts and by a combination

with other error detection methods, c) improve and extend the error correction stage for more annotation types, and d) research and evaluate the effect of annotation errors and their correction on the use of corpus annotation for human language technology and on theoretical linguistics.

7.1 Error detection for more annotation types

We have seen that the method for detecting variation n-grams is applicable as long as it is possible to establish a one-to-one mapping between recurring stretches of data and the annotation for which variation is to be detected. Future work can examine the conceptual and technical issues involved in applying variation detection to a broad range of different types of annotation, focusing on two main groups of linguistic annotation—semantic/pragmatic, and bi-texts (as used for statistical machine translation)—as well as detecting errors in the insertion of material such as null elements into a corpus.

For semantic and pragmatic annotation, one can focus on lexical semantic, coreference, and Rhetorical Structure Theory annotation. PropBank (Kingsbury et al., 2002) is a good place to start, given the correspondence of the semantic propositions in PropBank to the syntactic annotation in the Penn Treebank. We envisage that the variation nucleus approach developed in chapter 4 for discontinuous syntactic annotation will be applicable for multi-word expressions (Rayson et al., 2004), so that the central focus of this work will be on establishing a relevant notion of context to disambiguate between ambiguity and error. Semantic and pragmatic distinctions are

not necessarily dependent on the immediately surrounding words, in the way that part-of-speech and syntactic distinctions are, so a more elaborate notion of context is needed.

As a brief sketch of how to redefine the context, we start with the fact that our work thus far required identical words for nuclei and contexts. One can redefine the notion of similarity for both nuclei and contexts using techniques from machine learning methods such as memory-based learning (MBL) (Daelemans et al., 1996, 1999). As we discussed in chapter 5, MBL is a form of *k-nn* (*k* nearest neighbor) classifying, where the case to be labeled is matched with the most similar case(s) in memory. One can use different semantic-specific features to define the similarity metric, which will indicate when two items are the same.

In the future, we envisage exploring the consistency of translations in aligned bi-texts, corpora which contain two languages where translation pairs have been identified. Correspondences can be identified on a word or on a phrase level, and the approach we developed for the syntactic annotation of discontinuous constituency is likely to be of use in the bi-text context to detect inconsistencies in the bi-text alignment.

Finally, we saw in section 3.2.1 how null elements had been inserted into a treebank for theoretical reasons and in section 4.5.2 how inserted punctuation in a spoken corpus had an effect on the method. One question we did not address at the time was how to tell if the inserted material had been consistently inserted into the corpus. That is, were the same insertion decisions made in the same context? In principle, the variation *n*-grams method can address such a question. We sketch here one way to adapt the method for finding inconsistencies in inserted material. One can first

strip the corpus of all inserted material and then define a variation nucleus as being of length two. For each nucleus, one can then go back to the original corpus and see if an element had been inserted between the two nucleus tokens. If so, the nucleus receives that inserted item as its label; otherwise, it receives the label NIL. In such a way, one can find out if corpus annotators consistently inserted material.

7.2 Increased recall

We saw that the variation n-gram method is relatively precise in detecting errors and that it detects a significant number of errors. However, it is extremely unlikely that we have detected all the errors in a corpus, simply because our method relies on repeated text, and not every string with erroneous annotation appears multiple times. Thus, it should be possible to increase the recall of errors by further generalizing the notion of a recurring variation nucleus or the notion of comparable contexts within the variation n-gram method and by developing other error detection methods.

In chapters 3 and 4, we defined contexts for syntactic annotation in terms of part-of-speech tags—instead of words—and we saw that it greatly increased the recall. There is a tradeoff between precision and recall that needs to be more fully explored, but future work can continue this line of research by basing the contexts on any of the levels of annotation in a corpus, or abstractions thereof. Thus, for a wide range of annotations, one can increase recall, i.e., the number of errors detected. Other context generalizations also seem to be available if one is willing to include language or corpus specific information in computing the contexts. In the WSJ corpus, for example, different numerical amounts, which frequently appear in the same context, could be treated identically.

Future work can also delve into increasing recall by developing other error detection methods, some of which we have mentioned in this thesis. For part-of-speech annotation, we used closed class analysis and implementation of tagging guideline rules (see section 2.4) to find more errors, but did not fully evaluate these methods. Both of these techniques can be developed further and can be applied to annotations other than part-of-speech in one way or another, thereby increasing the number of errors found. And in section 3.3, we developed a context-independent method for detecting errors in treebanks based on the notion that grammar rules in a treebank should be applied consistently. As this method demonstrates, by revisiting the notion of a variation nucleus, one can continue to seek out other kinds of inconsistencies to increase recall.

7.3 Extending error correction

In this thesis, we demonstrated the usefulness of error correction for POS annotation, based on the output of the variation n-gram method. There are several directions in which to take these results: future work can go into making the methods more robust, extending the methods to the general process of part-of-speech tagging, and extending them to other annotation types.

We showed how useful our methods of automatic correction were for the WSJ corpus, but they were not as effective for the BNC-sampler. In an effort to make error correction more applicable to a wide range of POS-annotated corpora with different kinds of annotation schemes, future work can place a priority on developing a taxonomy of the errors provided by the error detection phase, akin to the work in Blaheta (2002). Different kinds of errors require different kinds of correction: some

can be corrected irrespective of the surrounding words, while others are dependent on the context, and still others may need human intervention (cf. Oliva, 2001). As we demonstrated in chapter 6, we have made some progress into automatically classifying the errors into an appropriate category, but it remains a high priority to more precisely categorize the errors.

The error correction work we have described can also be extended to the general task of part-of-speech tagging. We saw in chapter 5 that one way to correct inconsistencies was to train the classifier on ambiguity classes that caused the inconsistencies. These inconsistencies stem from ambiguities in the data, which is what makes classification difficult in the first place, and thus there is reason to believe that a tagger trained on complex ambiguity classes can be a useful part-of-speech tagger, provided that we can sufficiently establish the appropriate ambiguity classes for each word. Future work will determine to what extent the method works.

Finally, we want to be able to semi-automatically correct different layers of annotation, just as we detected errors in different kinds of annotation. Although classifiers are not generally used for assigning syntactic structure to a sentence (since the annotation is non-positional), some of the general correction procedures we have described can in principle also be applied to syntactic error correction work. Following with the idea that a piece of NLP technology can be used to enforce consistent behavior, one can use a PCFG induced from a treebank to assign structure to that same treebank and again focus on spots flagged by the variation n-gram method. Corrections, however, can percolate up the tree, and so one has to carefully determine where to stop correcting. Likewise, as different errors can potentially interact within a given sentence, one has to work out a way to use information from the majority label in

an n-gram—which we showed to be successful for POS error correction. Because NIL strings mean that there are different bracketings, a string of one size may have a majority bracketing which conflicts with the majority bracketing of the string when it is expanded further. Work will have to go into resolving such conflicts.

7.4 Consequences of better corpus annotation

Although we have done some testing on the effect of different types of annotation errors (e.g., the work on removing erroneous grammar rules in section 3.3.4), in the future one can further the work of determining the impact of errors and their correction on the use of corpus annotation in human language technology. Our experiment with LoPar (section 3.3.4) suffered from the fact that the testing data had not been cleaned. Thus, by automatically cleaning (and hand-checking) a part of a corpus used for evaluation, one can determine the differences between the originally reported error rate of some technology and its error rate on the cleaned testing data.

Turning to the training data, we have only tried seeing the effects of erroneous training data with syntactic annotation, but we have gone to great lengths to correct POS annotation. Future work will further test the impact of errors on training data by performing the tenfold cross-validation experiment of van Halteren (2000) with automatically corrected POS training data (manually-checked) against the original data. We know from our experiments that the automatically corrected portions of the WSJ are significantly better than the original data, and we also can determine which distinctions were difficult to maintain over the corpus. Thus, one can be sure of first improving the training data in a semi-automatic way; then training a technology on that data; and finally testing on the cleaned evaluation set, as mentioned above.

One final note is in order about better corpus annotation. The process of error detection has shown us that corpus annotation can only be as good as the annotation scheme allows. Future work can further explore what kinds of corpus distinctions are difficult to make and if they tell us anything about interesting linguistic phenomena.

BIBLIOGRAPHY

Abeillé, Anne (ed.) (2003). *Treebanks: Building and using syntactically annoted corpora*. Dordrecht: Kluwer Academic Publishers.

Abney, Steven, Robert E. Schapire and Yoram Singer (1999). Boosting Applied to Tagging and PP Attachment. In Pascale Fung and Joe Zhou (eds.), *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing (EMNLP) and Very Large Corpora (VLC)*. pp. 38–45. http://www.cs.huji.ac.il/~singer/papers/bnlp.ps.gz.

Agrawal, Rakesh and Ramakrishnan Srikant (1994). Fast Algorithms for Mining Association Rules in Large Databases. In Jorge B. Bocca, Matthias Jarke and Carlo Zaniolo (eds.), *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*. Morgan Kaufmann, pp. 487–499.

Ali, K.M. and M.J. Pazzani (1996). Error Reducation through Learning Multiple Descriptions. *Machine Learning* 24(3), 173–202.

Allen, James and Mark Core (1996). *Draft of DAMSL: Dialog Act Markup in Several Layers*. Tech. rep., University of Rochester, Department of Computer Science.

Atwell, Eric (1993). Corpus-Based Statistical Modelling of English Grammar. In Souter and Atwell (1993), pp. 195–214.

Baker, John Paul (1997). Consistency and Accuracy in Correcting Automatically Tagged Data. In Roger Garside, Geoffrey Leech and Tony McEnery (eds.), *Corpus annotation: linguistic information from computer text corpora*, Harlow, England: Addison Wesley Longman Limited, chap. 17, pp. 243–250.

Berger, A., P.F. Brown et al. (1994). The Candide System for Machine Translation. In *Proceedings ARPA Human Language Technology Workshop*. Plainsboro, NJ: Morgan-Kaufmann Publishers Inc., pp. 152–157.

Bies, Ann, Mark Ferguson, Karen Katz and Robert MacIntyre (1995). *Bracketing Guidelines for Treebank II Style Penn Treebank Project*. University of Pennsylvania. www.ircs.upenn.edu/arabic/manuals/root.pdf.

Bikel, Daniel M., Richard Schwartz and Ralph M. Weischedel (1999). An algorithm that learns what's in a name. *Machine Learning* 34, 211–231.

Bird, Steven and Mark Liberman (2000). A Formal Framework for Linguistic Annotation. *Speech Communication* 33(1-2), 23–60.

Blache, Philippe and Daniel Hirst (2000). Multi-level annotation for spoken-language corpora. In *Proceedings of the Sixth International Conference on Spoken Language Processing (ICSLP-2000)*. pp. 481–484. http://www.lpl.univ-aix.fr/~blache/papers/ICSLP-final.pdf.

Black, E., S. Abney et al. (1991). A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*. Defense Advanced Research Projects Agency, Pacific Grove, CA: Morgan Kaufmann.

Blaheta, Don (2002). Handling noisy training and testing data. In *Proceedings of the 7th conference on Empirical Methods in Natural Language Processing (EMNLP)*. pp. 111–116. http://www.cs.brown.edu/~dpb/papers/dpb-emnlp02.html.

Blevins, James (1990). Syntactic Complexity: Evidence for Discontinuity and Multidomination. Ph.D. thesis, University of Massachusetts.

Bod, Rens (2003). Extracting stochastic grammars from treebanks. In Abeillé (2003), pp. 333–349.

Böhmová, Alena, Jan Hajič, Eva Hajičová and Barbora Hladká (2003). The Prague Dependency Treebank. In Abeillé (2003), pp. 103–127.

Bonami, Olivier, Danièle Godard and Jean-Marie Marandin (1999). Constituency and word order in French subject inversion. In Gosse Bouma, Erhard W. Hinrichs, Geert-Jan M. Kruijff and Richard T. Oehrle (eds.), *Constraints and Resources in Natural Language Syntax and Semantics*, Stanford: CSLI Publications, Studies in Constraint-Based Lexicalism, pp. 21–40.

Bond, Francis, Sanae Fujita et al. (2004). The Hinoki Treebank: Toward Text Understanding. In *Proceedings of the 5th International Workshop on Linguistically Interpreted Corpora (LINC-04)*. Geneva, pp. 7–10. http://www.coli.uni-saarland.de/conf/linc-04/bond.pdf.

Brants, Sabine, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius and George Smith (2002). The TIGER Treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories (TLT)*. Sozopol, Bulgaria, pp. 24–41. http://www.ims.uni-stuttgart.de/projekte/TIGER/paper/treeling2002.pdf.

Brants, Thorsten (1996). Estimating Markov Model Structures. In *Proceedings of the Fourth Conference on Spoken Language Processing (ICSLP-96)*. Philadelphia, PA, pp. 893–896. http://www.coli.uni-saarland.de/~thorsten/publications/Brants-ICSLP96.pdf.

Brants, Thorsten (2000a). Inter-Annotator Agreement for a German Newspaper Corpus. In *Proceedings of Second International Conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece. http://www.coli.uni-saarland.de/~thorsten/publications/Brants-LREC00.pdf.

Brants, Thorsten (2000b). TnT – A Statistical Part-of-Speech Tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference (ANLP 2000)*. Seattle, WA, pp. 224–231. http://www.coli.uni-saarland.de/~thorsten/publications/Brants-ANLP00.pdf.

Brants, Thorsten and Oliver Plaehn (2000). Interactive Corpus Annotation. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece. http://www.coli.uni-saarland.de/~thorsten/publications/Brants-Plaehn-LREC00.pdf.

Brants, Thorsten and Wojciech Skut (1998). Automation of Treebank Annotation. In *Proceedings of New Methods in Language Processing (NeMLP-98)*. Syndey, Australia, pp. 49–57. http://www.coli.uni-sb.de/~thorsten/publications/Brants-Skut-NeMLaP98.pdf.

Brants, Thorsten, Wojciech Skut and Hans Uszkoreit (1999). Syntactic Annotation of a German Newspaper Corpus. In *Proceedings of the ATALA Treebank Workshop*. Paris, France, pp. 69–76. http://www.coli.uni-sb.de/~thorsten/publications/Brants-ea-ATALA99.ps.gz.

Brill, Eric (1994). Some Advances in Transformation-Based Part of Speech Tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*. Menlo Park, CA: AAAI Press, pp. 722–727. http://www.cs.jhu.edu/~brill/TAGGING_ADVANCES.ps.

Brill, Eric (1995a). Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging. *Computational Linguistics* 21(4), 543–565. http://acl.ldc.upenn.edu/J/J95/J95-4004.pdf.

Brill, Eric (1995b). Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In *Proceedings of the Third Workshop on Very Large Corpora (VLC)*. pp. 1–13. http://acl.ldc.upenn.edu/W/W95/W95-0101.pdf.

Brill, Eric and Mihai Pop (1999). Unsupervised Learning of Disambiguation Rules for Part of Speech Tagging. In Kenneth W. Church (ed.), *Natural Language Processing Using Very Large Corpora*, Dordrecht: Kluwer Academic Press, pp. 27–42.

Briscoe, E. (1994). Prospects for practical parsing: robust statistical techniques. In P. de Haan and N. Oostdijk (eds.), *Corpus-based Research into Language: A Feschrift for Jan Aarts*, Amsterdam: Rodopi, pp. 67–95.

Bröker, Norbert (1998). Separating Surface Order and Syntactic Relations in a Dependency Grammar. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual meeting of the Association for Computational Linguistics (ACL)*. Montreal, pp. 174–180. http://acl.ldc.upenn.edu/P/P98/P98-1026.pdf.

Brown, P, S Della Pietra, V Della Pietra and R Mercer (1991). Word sense disambiguation using statistical methods. In *Proceedings of the 29th Meeting of the Association for Computational Linguistics (ACL-91)*. Berkley CA, pp. 264–270. http://acl.ldc.upenn.edu/P/P91/P91-1034.pdf.

Calder, Jo (1997). On aligning trees. In *Proceedings of the Second Conference of Empirical Methods in Natural Language Processing (EMNLP-97)*. Brown University, pp. 75–80. http://xxx.lanl.gov/abs/cmp-lg/9707016.

Cardie, Claire and David Pierce (1998). Error-driven pruning of Treebank grammars for base noun phrase identification. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual meeting of the Association for Computational Linguistics (ACL)*. Montreal, pp. 218–224. http://acl.ldc.upenn.edu/P/P98/P98-1034.pdf.

Carroll, John, Anette Frank, Dekang Lin, Detlef Prescher and Hans Uszkoreit (eds.) (2002). *Proceedings of the Workshop "Beyond PARSEVAL. Towards Improved Evaluation Measures for Parsing Systems" at the 3rd International Conference on Language Resources and Evaluation (LREC 2002)*, Las Palmas, Gran Canaria. http://www.cogs.susx.ac.uk/lab/nlp/carroll/papers/beyond-proceedings.pdf.

Charniak, Eugene (1996). *Tree-Bank Grammars*. Tech. Rep. CS-96-02, Department of Computer Science, Brown University, Providence, RI. ftp://ftp.cs.brown.edu/pub/techreports/96/cs96-02.ps.Z.

Charniak, Eugene, Curtis Hendrickson, Neil Jacobson and Mike Perkowitz (1993). Equations for Part-of-Speech Tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*. Menlo Park, CA, pp. 784–789. http://www.cs.brown.edu/people/ec/papers/equfortag.ps.

Church, Kenneth W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In *Second Conference on Applied Natural Language Processing (ANLP)*. Austin, TX, pp. 136–143. http://acl.ldc.upenn.edu/A/A88/A88-1019.pdf.

Church, Kenneth W. (1992). Current practice in part of speech tagging and suggestions for the future. In Simmons (ed.), *Sbornik praci: In Honour of Henry Kučera*, Michigan Slavic studies, pp. 13–48.

Clark, James (1999). *XSL Transformations (XSLT) Version 1.0*. Tech. rep., W3C. http://www.w3.org/TR/xslt.

Cohn, David A., Les Atlas and Richard E. Ladner (1994). Improving Generalization with Active Learning. *Machine Learning* 15(2), 201–221.

Collins, Michael John (1996). A New Statistical Parser Based on Bigram Lexical Dependencies. In Arivind Joshi and Martha Palmer (eds.), *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics (ACL-96)*. San Francisco, pp. 184–191. http://acl.ldc.upenn.edu/P/P96/P96-1025.pdf.

Cutting, Doug, Julian Kupiec, Jan Pedersen and Penelope Sibun (1992). A Practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*. Trento, Italy, pp. 133–140. http://acl.ldc.upenn.edu/A/A92/A92-1018.pdf.

Daelemans, Walter, Antal van den Bosch and Jakub Zavrel (1999). Forgetting Exceptions is Harmful in Language Learning. *Machine Learning* 34, 11–41.

Daelemans, Walter, Jakub Zavrel, Peter Berck and Steven Gillis (1996). MBT: A Memory-Based Part of Speech Tagger-Generator. In *Proceedings of the Fourth Workshop on Very Large Corpora (VLC)*. Copenhagen, pp. 14–27. http://acl.ldc.upenn.edu/W/W96/W96-0102.pdf.

Daelemans, Walter, Jakub Zavrel, Ko van der Sloot and Antal van den Bosch (2003). TiMBL: Tilburg Memory-Based Learner - version 5.0 Reference Guide. http://www.cnts.ua.ac.be/~walter/papers/2003/dzsb03.pdf.

Dagan, Ido and Sean P. Engelson (1995). Committee-Based Sampling For Training Probabilistic Classifiers. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*. Tahoe City, CA, pp. 150–157. citeseer.ist.psu.edu/17150.html.

Daniels, Michael W. (2005). Generalized ID/LP Grammar: A Formalism for Parsing Linearization-Based HPSG Grammars. Ph.D. thesis, The Ohio State University. http://ling.osu.edu/~daniels/thesis.html.

Daniels, Mike and W. Detmar Meurers (2002). Improving the Efficiency of Parsing with Discontinuous Constituents. In Shuly Wintner (ed.), *Proceedings of NLULP'02: The 7th International Workshop on Natural Language Understanding and Logic Programming*. Copenhagen: Roskilde Universitetscenter, no. 92 in Datalogiske Skrifter, pp. 49–68. http://www.ling.ohio-state.edu/~dm/papers/daniels-meurers-02.html.

Daniels, Mike and W. Detmar Meurers (2004). A grammar formalism and parser for linearization-based HPSG. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING-04)*. Geneva, Switzerland. http://www.ling.ohio-state.edu/~daniels/gidlpparser.pdf.

DeRose, Steven J. (1988). Grammatical Category Disambiguation by Statistical Optimization. *Computational Linguistics* 14(1), 31–39. http://acl.ldc.upenn.edu/J/J88/J88-1003.pdf.

Dickinson, Markus and W. Detmar Meurers (2003a). Detecting Errors in Part-of-Speech Annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*. Budapest, Hungary, pp. 107–114. http://ling.osu.edu/~dickinso/papers/dickinson-meurers-03.html.

Dickinson, Markus and W. Detmar Meurers (2003b). Detecting Inconsistencies in Treebanks. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003)*. Växjö, Sweden, pp. 45–56. http://ling.osu.edu/~dickinso/papers/dickinson-meurers-tlt03.html.

Dickinson, Markus and W. Detmar Meurers (2005a). Detecting Annotation Errors in Spoken Language Corpora. In *The Special Session on treebanks for spoken language and discourse at the 15th Nordic Conference of Computational Linguistics (NODALIDA-05)*. Joensuu, Finland. http://ling.osu.edu/~dickinso/papers/dickinson-meurers-nodalida05.html.

Dickinson, Markus and W. Detmar Meurers (2005b). Detecting Errors in Discontinuous Structural Annotation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*. Ann Arbor, MI, USA, pp. 322–329. http://ling.osu.edu/~dickinso/papers/dickinson-meurers-05.html.

Dimitrova, Ludmila, Tomaž Erjavec, Nancy Ide, Heiki-Jan Kaalep, Vladimir Petkevič, and Dan Tufis (1998). Multext-East: Parallel and Comparable Corpora and Lexicons for Six Central and Eastern European Languages. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics (ACL-98) and the 17th International Conference on Computational Linguistics (COLING-98)*. Montreal, pp. 315–319. http://acl.ldc.upenn.edu/P/P98/P98-1050.pdf.

Donohue, Cathryn and Ivan A. Sag (1999). Domains in Warlpiri. In *Abstracts of the Sixth Int. Conference on HPSG*. Edinburgh, Scotland: University of Edinburgh, pp. 101–106. http://www-csli.stanford.edu/~sag/papers/warlpiri.ps.

Dowty, David R. (1996). Towards a Minimalist Theory of Syntactic Structure. In Harry Bunt and Arthur van Horck (eds.), *Discontinuous Constituency*, Berlin and New York, NY: Mouton de Gruyter, vol. 6 of *Natural Language Processing*.

Elworthy, David (1994). Automatic Error Detection in Part of Speech Tagging. http://arxiv.org/abs/cmp-lg/9410013.

Erjavec, Tomaž (2001). *Specifications and Notation for MULTEXT-East Lexicon Encoding*. Tech. rep. http://nl.ijs.si/ME/V2/msd/html/.

Erjavec, Tomaž, Cvetana Krstev, Vladimír Petkevič, Kiril Simov, Marko Tadić and Duško Vitas (2003). The MULTEXT-East Morphosyntactic Specifications for Slavic Languages. In *Proceedings of the Workshop on Morphological Processing of Slavic Languages*. Budapest, Hungary, pp. 25–32.

http://acl.ldc.upenn.edu/eacl2003/papers/workshop/w12.pdf.

Erk, Katrin, Andrea Kowalski, Sebastian Pado and Manfred Pinkal (2003). Towards a Resource for Lexical Semantics: A Large German Corpus with Extensive Semantic Annotation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL-03)*. Sapporo, Japan, pp. 537–544. http://acl.ldc.upenn.edu/P/P03/P03-1068.pdf.

Eskin, Eleazar (2000). Automatic Corpus Correction with Anomaly Detection. In *Proceedings of the First Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-00)*. Seattle, Washington, pp. 148–153. http://www.cs.columbia.edu/~eeskin/papers/treebank-anomaly-naacl00.ps.

Eyes, Elizabeth and Geoffrey Leech (1992). Progress in UCREL research: Improving corpus annotation practices. Amsterdam: Rodopi, pp. 123–143.

Fredkin, Edward (1960). Trie Memory. In *Communications of the Association for Computing Machinery (CACM)*, vol. 3, pp. 490–499.

Gaizauskas, R. (1995). *Investigations into the grammar underlying the Penn Treebank II*. Tech. Rep. Research Memorandum CS-95-25, University of Sheffield. ftp://ftp.dcs.shef.ac.uk/home/robertg/papers/dcs-tr-95-25.pdf.

Gale, W., K. Church and D. Yarowsky (1992). A Method for Disambiguating Word Senses in a Corpus. *Computers and the Humanities* 26, 215–439.

Garside, Roger, Geoffrey Leech and Geoffrey Sampson (eds.) (1987). *The Computational Analysis of English: A Corpus-Based Approach*. London and New York: Longman.

Hajič, Jan (1998). Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. In Eva Hajičová (ed.), *Issues of Valency and Meaning. Studies in Honor of Jarmila Panevová*, Prague Karolinum, Charles University Press, pp. 12–19.

Hajičová, Eva (1998). Prague Dependency Treebank: From Analytic to Tectogrammatical Annotation. In *Proceedings of the First Workshop on Text, Speech, Dialogue*. Brno, Czech Republic, pp. 45–50. http://ufal.ms.mff.cuni.cz/pdt/Corpora/PDT_1.0/References/index.html.

Hajičová, Eva, Jarmila Panevova and Petr Sgall (1998). Language Resources Need Annotations to Make Them Really Reusable: The Prague Dependency Treebank. In *Proceedings of the First Conference on Linguistic Resources and Evaluation (LREC)*. Granada, Spain, pp. 713–718.

Hemphill, Charles T., John J. Godfrey and George R. Doddington (1990). The ATIS spoken language systems pilot corpus. In *Proceedings of a Workshop on Speech and Natural Language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 96–101.

Hepple, Mark (1994). Discontinuity and the Lambek Calculus. In *Proceedings of the 15th Conference on Computational Linguistics (COLING-94)*. Kyoto, Japan. ftp://ftp.dcs.shef.ac.uk/home/hepple/papers/coling94.ps.

Hinrichs, Erhard, Julia Bartels, Yasuhiro Kawata, Valia Kordoni and Heike Telljohann (2000). The Tübingen Treebanks for Spoken German, English, and Japanese. In Wahlster (2000), pp. 552–576.

Hirakawa, Hideki, Kenji Ono and Yumiko Yoshimura (2000). Automatic Refinement of a POS Tagger Using a Reliable Parser and Plain Text Corpora. In *Proceedings of the 18th International Conference on Computational Linguistics (COLING-00)*. Saarbrücken, Germany, pp. 313–319. http://acl.ldc.upenn.edu/C/C00/C00-1046.pdf.

Huck, Geoffrey (1985). Exclusivity and discontinuity in phrase structure grammar. In *West Coast Conference on Formal Linguistics (WCCFL)*. Stanford: CSLI Publications, vol. 4, pp. 92–98.

Huck, Geoffrey and Almerindo Ojeda (eds.) (1987). *Discontinuous Constituency*. No. 20 in Syntax and Semantics. Orlando, FL: Academic Press.

Jackendoff, Ray (1977). *X' Syntax: A Study of Phrase Structure*. Cambridge, MA: MIT Press.

Järvinen, Timo (2003). Bank of English and Beyond: Hand-crafted parses for functional annotation. In Abeillé (2003), pp. 43–59.

Johansson, Stig (1986). *The Tagged LOB Corpus: Users' Manual*. Norwegian Computing Centre for the Humanities, Bergen. http://helmer.aksis.uib.no/icame/lobman/lob-cont.html.

Johnson, Mark (1985). Parsing with discontinuous constituents. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics (ACL-85)*. pp. 127–132. http://acl.ldc.upenn.edu/P/P85/P85-1015.pdf.

Kaljurand, Kaarel (2004). Checking treebank consistency to find annotation errors. http://math.ut.ee/~kaarel/NLP/Programs/Treebank/ConsistencyChecking/.

Kasper, Robert T., Mike Calcagno and Paul C. Davis (1998). Know When to Hold 'Em: Shuffling Deterministically in a Parser for Nonconcatenative Grammars. In *Proceedings of the Thirty-Sixth Annual Meeting of the Association for Computational Linguistics (ACL-98)*. pp. 663–669. http://acl.ldc.upenn.edu/P/P98/P98-1109.pdf.

Kathol, Andreas (1995). Linearization-Based German Syntax. Ph.D. thesis, Ohio State University. Revised version published by Oxford University Press, 2000.

Kilgarriff, A. (1998). Gold standard datasets for evaluating word sense disambiguation programs. In *Computer Speech and Language*, vol. 12. ftp://ftp.itri.bton.ac.uk/reports/ITRI-98-08.ps.

Kingsbury, Paul, Martha Palmer and Mitch Marcus (2002). Adding Semantic Annotation to the Penn Treebank. In *Proceedings of the Human Language Technology Conference (HLT 2002)*. San Diego, California. http://www.cis.upenn.edu/~ace/HLT2002-propbank.pdf.

Klein, Dan and Christopher D. Manning (2001). Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-01)*. pp. 330–337. http://acl.ldc.upenn.edu/P/P01/P01-1044.pdf.

König, Esther, Wolfgang Lezius and Holger Voormann (2003). *TIGERSearch User's Manual*. IMS, University of Stuttgart. http://www.tigersearch.de.

Kordoni, Valia (2003). Strategies for annotation of large corpora of multilingual spontaneous speech data. In *Proceedings of the Workshop on Multilingual Corpora: Linguistic Requirements and Technical Perspectives*. Lancaster, pp. 53–57. http://www.coli.uni-sb.de/muco03/Proceedings.htm.

Kroch, Anthony S. and Aravind K. Joshi (1987). Analyzing Extraposition in a Tree Adjoining Grammar. In Huck and Ojeda (1987), pp. 107–149.

Krotov, Alexander, Mark Hepple, Robert J. Gaizauskas and Yorick Wilks (1998). Compacting the Penn Treebank Grammar. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual meeting of the Association for Computational Linguistics (ACL)*. pp. 699–703. http://acl.ldc.upenn.edu/P/P98/P98-1115.pdf.

Kübler, Sandra (2003). Parsing Without Grammar – Using Complete Trees Instead. In *Proceedings of Recent Advances in Natural Language Processing (RANLP 2003)*. Borovets, Bulgaria, pp. 193–204. http://www.sfs.uni-tuebingen.de/~kuebler/papers/ranlp03.ps.

Kübler, Sandra and Andreas Wagner (2000). Evaluating POS Tagging under Sub-optimal Conditions. Or: Does Meticulousness Pay? In *Proceedings of the Proceedings of International Conference on Artificial and Computational Intelligence for Decision, Control and Automation in Engineering and Industrial Applications (ACIDCA'2000)*. pp. 77–82. http://www.sfs.uni-tuebingen.de/~kuebler/papers/acidca.ps.

Květoň, Pavel and Karel Oliva (2002). Achieving an Almost Correct PoS-Tagged Corpus. In Petr Sojka, Ivan Kopeček and Karel Pala (eds.), *Text, Speech and Dialogue (TSD)*. Heidelberg: Springer, no. 2448 in Lecture Notes in Artificial Intelligence (LNAI), pp. 19–26.

Leech, Geoffrey (1997). *A Brief Users' Guide to the Grammatical Tagging of the British National Corpus*. UCREL, Lancaster University. http://www.hcu.ox.ac.uk/BNC/what/gramtag.html.

Leech, Geoffrey, Roger Garside and Michael Bryant (1994a). CLAWS4: The tagging of the British National Corpus. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*. Kyoto, Japan, pp. 622–628. http://www.comp.lancs.ac.uk/computing/users/paul/ucrel/papers/coling.html.

Leech, Geoffrey, Roger Garside and Michael Bryant (1994b). The large-scale grammatical tagging of text: Experience with the British National Corpus. Amsterdam: Rodopi, pp. 47–63.

Lenerz, Jürgen (2001). Word Order Variation: Competition or Co-Operation. In Gereon Müller and Wolfgang Sternefeld (eds.), *Competition in Syntax*, Berlin and New York, NY: Mouton de Gruyter, pp. 249–281.

Lönneker, Birte and Primož Jakopin (2004). Checking *POSBeseda*, a Part-of-Speech tagged Slovenian corpus. In *Proceedings of the Fourth Language Technologies Conference of the Slovenian Language Technologies Society*. Ljubljana, Slovenia, pp. 48–55. http://nl.ijs.si/isjt04/zbornik/sdjt04-09lonneker.pdf.

Ma, Qing, Bao-Liang Lu, Masaki Murata, Michnori Ichikawa and Hitoshi Isahara (2001). On-line Error Detection of Annotated Corpus Using Modular Neural Networks. In *International Conference on Artificial Neural Networks (ICANN2001)*. Vienna, Austria, pp. 1185–1192. http://www.math.ryukoku.ac.jp/~qma/papers/ICANN2001_err_detec.ps.gz.

Magerman, David (1995). Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95)*. pp. 276–283. http://acl.ldc.upenn.edu/P/P95/P95-1037.pdf.

Manning, Christopher D. and Hinrich Schütze (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: The MIT Press.

Marcus, M., Beatrice Santorini and M. A. Marcinkiewicz (1993). Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics* 19(2), 313–330. http://acl.ldc.upenn.edu/J/J93/J93-2004.pdf.

Marcus, Mitchell, Beatrice Santorini, Mary Ann Marcinkiewicz and Ann Taylor (1999). Treebank-3 Corpus. Linguistic Data Consortium. Philadelphia. http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC99T42.

Marquez, Lluis and Lluis Padro (1997). A Flexible POS Tagger Using an Automatically Acquired Language Model. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*. Madrid, Spain, pp. 238–245. http://acl.ldc.upenn.edu/P/P97/P97-1031.pdf.

McCawley, James D. (1982). Parentheticals and discontinuous constituent structure. *Linguistic Inquiry* 13(1), 91–106.

Meurers, Walt Detmar (2005). On the use of electronic corpora for theoretical linguistics. Case studies from the syntax of German. *Lingua* 115(1), 1619–1639. http://ling.osu.edu/~dm/papers/meurers-03.html.

Mitchell, Thomas M. (1997). *Machine Learning*. New York: McGraw-Hill Higher Education.

Mitkov, Ruslan, Constantin Orasan and Richard Evans (1997). The importance of annotated corpora for NLP: the cases of anaphora resolution and clause splitting. In *Proceedings of the Workshop on Corpora and NLP: Reflecting on Methodology at TALN'99*. Cargese, Corse, pp. 60 – 69. http://clg.wlv.ac.uk/papers/mitkov-99b.pdf.

Morrill, Glynn V. (1995). Discontinuity in categorial grammar. *Linguistics and Philosophy* 18, 175–219.

Müller, Frank H. and Tylman Ule (2002). Annotating topological fields and chunks – and revising POS tags at the same time. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING-02)*. Taipei, Taiwan, pp. 695–701. http://acl.ldc.upenn.edu/C/C02/C02-1131.pdf.

Müller, Stefan (1999). *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. No. 394 in Linguistische Arbeiten. Tuebingen: Max Niemeyer Verlag.

Müller, Stefan (2004). Continuous or Discontinuous Constituents? A Comparison between Syntactic Analyses for Constituent Order and Their Processing Systems. *Research on Language and Computation, Special Issue on Linguistic Theory and Grammar Implementation* 2(2), 209–257. http://www.cl.uni-bremen.de/~stefan/Pub/discont.html.

Nakagawa, Tetsuji and Yuji Matsumoto (2002). Detecting Errors in Corpora Using Support Vector Machines. In *Proceedings of the 17th International Conference on Computational Lingusitics (COLING 2002)*. Taipei, Taiwan, pp. 709–715. http://acl.ldc.upenn.edu/C/C02/C02-1101.pdf.

Nelson, Gerald, Sean Wallis and Bas Aarts (2002). *Exploring Natural Language: Working with the British Component of the International Corpus of English*. Amsterdam: John Benjamins Publishing Company.

Oepen, Stephan, Dan Flickinger and Francis Bond (2004). Towards holistic grammar engineering and testing—grafting treebank maintenance into the grammar revision cycle. In *Beyond Shallow Analyses—Formalisms and Statistical Modelling for Deep Analysis (Workshop at The First International Joint Conference on Natural Language Processing (IJCNLP-04))*. Hainan, China. http://www-tsujii.is.s.u-tokyo.ac.jp/bsa/papers/oepen.pdf.

Ojeda, Almerindo (1987). Discontinuity, multidominances and unbounded dependency in Generalized Phrase Structure Grammar. In Huck and Ojeda (1987), pp. 257–282.

Oliva, Karel (2001). The Possibilities of Automatic Detection/Correction of Errors in Tagged Corpora: a Pilot Study on a German Corpus. In Václav Matoušek, Pavel Mautner, Roman Mouček and Karel Taušer (eds.), *Text, Speech and Dialogue. 4th International Conference, TSD 2001, Zelezna Ruda, Czech Republic, September 11-13, 2001, Proceedings*. Springer, vol. 2166 of *Lecture Notes in Computer Science*, pp. 39–46.

Padro, Lluis and Lluis Marquez (1998). On the Evaluation and Comparison of Taggers: the Effect of Noise in Testing Corpora. In *Proceedings of the 17th International Conference on Computational Linguistics (COLING) and the 36th Annual meeting of the Association for Computational Linguistics (ACL)*. pp. 997–1002. http://acl.ldc.upenn.edu/P/P98/P98-2164.pdf.

Penn, Gerald (1999). Linearization and WH-extraction in HPSG: Evidence from Serbo-Croatian. In Robert D. Borsley and Adam Przepiórkowski (eds.), *Slavic in HPSG*, Stanford: CSLI Publications, pp. 149–182.

Pla, Ferran and Antonio Molina (2004). Improving part-of-speech tagging using lexicalized HMMs. *Natural Language Engineering* 10(2), 167–189.

Plátek, Martin, Tomáš Holan, Vladimir Kuboň and Karel Oliva (2001). Word-Order Relaxations and Restrictions within a Dependency Grammar. In G. Satta (ed.), *Proceedings of the Seventh International Workshop on Parsing Technologies (IWPT)*. Beijing: Tsinghua University Press, pp. 237–240.

Przepiórkowski, Adam and Marcin Woliński (2003). The Unbearable Lightness of Tagging: A Case Study in Morphosyntactic Tagging of Polish. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*. pp. 109–116. http://acl.ldc.upenn.edu/eacl2003/papers/workshop/w03.pdf.

Quinlan, J. Ross (1986). Induction of Decision Trees. *Machine Learning* 1(1), 81–106.

Ragas, H. and C.H.A. Koster (1998). Four classification algorithms compared on a Dutch corpus. In *Proceedings of the Association for Computing Machinery Special Interest Group on Information Retrieval Conference (ACM SIGIR 98)*. pp. 369–370. http://portal.acm.org/citation.cfm?id=291059.

Rambow, Owen and Aravind Joshi (1994). A Formal Look at Dependency Grammars and Phrase-Structure Grammars, with Special Consideration of Word-Order Phenomena. In L. Wanner (ed.), *Current Issues in Meaning-Text-Theory*, London: Pinter, pp. 167–190. http://arxiv.org/abs/cmp-lg/9410007.

Ramshaw, Lance and Mitch Marcus (1995). Text Chunking Using Transformation-Based Learning. In David Yarovsky and Kenneth Church (eds.), *Proceedings of the Third Workshop on Very Large Corpora (VLC)*. Somerset, New Jersey: Association for Computational Linguistics, pp. 82–94. http://acl.ldc.upenn.edu/W/W95/W95-0107.pdf.

Ratnaparkhi, Adwait (1996). A maximum entropy model part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-96)*. Philadelphia, PA, pp. 133–141. http://acl.ldc.upenn.edu/W/W96/W96-0213.pdf.

Rayson, Paul, Dawn Archer, Scott Piao and Tony McEnery (2004). The UCREL Semantic Analysis System. In *Proceedings of the Workshop on Beyond Named Entity Recognition: Semantic Labelling for NLP tasks*. Lisbon, Portugal, pp. 7–12.

Reape, Mike (1993). A Formal Theory of Word Order: A Case Study in West Germanic. Ph.D. thesis, University of Edinburgh, Edinburgh, Scotland.

Richter, Frank and Manfred Sailer (2001). On the Left Periphery of German Finite Sentences. In W. Detmar Meurers and Tibor Kiss (eds.), *Constraint-Based Approaches to Germanic Syntax*, Stanford, CA: CSLI Publications, Studies in Constraint-Based Lexicalism, pp. 257–300.

Riloff, Ellen (1993). Automatically Constructing a Dictionary for Information Extraction Tasks. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*. Menlo Park, CA: AAAI Press, pp. 811–816. http://www.cs.utah.edu/~riloff/psfiles/aaai93.pdf.

Rocha, Marco (1997). Supporting Anaphor Resolution in Dialogues With a Corpus-Based Probabilistic Model. In *Proceedings of the ACL'97/EACL'97 workshop on Operational Factors in Practical, Robust Anaphora Resolution*. Madrid, Spain, pp. 54–61. http://acl.ldc.upenn.edu/W/W97/W97-1308.pdf.

Sampson, Geoffrey (1995). *English for the Computer: The SUSANNE Corpus and Analytic Scheme*. Oxford: Clarendon Press.

Sampson, Geoffrey (1996). From central embedding to corpus linguistics. In Jenny Thomas and Mick Short (eds.), *Using Corpora for Language Research: Studies in the Honour of Geoffrey Leech*, London: Longman Group Limited, chap. 2, pp. 14–26.

Sampson, Geoffrey (2003). *SUSANNE Corpus and Analytic Scheme*. Tech. rep. http://www.grsampson.net/RSue.html.

Sampson, Geoffrey and Anna Babarczy (2003). Limits to annotation precision. In *Proceedings of the 4th International Workshop on Linguistically Interpreted Corpora (LINC-03)*. Budapest, Hungary, pp. 61–68. http://acl.ldc.upenn.edu/eacl2003/papers/workshop/w03.pdf.

Santorini, Beatrice (1990). *Part-Of-Speech Tagging Guidelines for the Penn Treebank Project (3rd Revision, 2nd printing)*. Tech. Rep. MS-CIS-90-47, The University of Pennsylvania, Philadelphia, PA. ftp://ftp.cis.upenn.edu/pub/treebank/doc/tagguide.ps.gz.

Schiller, Anne, Simone Teufel, Christiane Stöckert and Christine Thielen (1995). *Vorläufige Guidlines für das Taggen deutscher Textcorpora mit STTS*. Tech. rep., IMS, Univ. Stuttgart and SfS, Univ. Tübingen. http://www.sfs.nphil.uni-tuebingen.de/Elwis/stts/stts-guide.ps.gz.

Schmid, Helmut (1997). Probabilistic part-of-speech tagging using decision trees. In D.H. Jones and H.L. Somers (eds.), *New Methods in Language Processing*, London: UCL Press, pp. 154–164.

Schmid, Helmut (2000). LoPar: Design and Implementation. In *Arbeitspapiere des Sonderforschungsbereiches 340*, IMS Stuttgart, no. 149. http://www.ims.uni-stuttgart.de/~schmid/lopar.pdf.

Segond, F., A. Schiller, G. Grefenstette and J. Chanod (1997). An experiment in semantic tagging using hidden Markov model tagging. In P. Vossen, G. Adriaens, N. Calzolair, A. Sanfilippo and Y. Wilks (eds.), *Proceedings of the ACL/EACL'97 Workshop on Automatic Information Extraction and Building of Lexical Semantic Resources*. Madrid, Spain, pp. 78–81. http://acl.ldc.upenn.edu/W/W97/W97-0811.pdf.

Sinclair, John M. (1992). The automatic analysis of corpora. In Jan Svartvik (ed.), *Directions in Corpus Linguistics*, Berlin and New York, NY: Mouton de Gruyter, vol. 65 of *Trends in Linguistics: Studies and monographs*, pp. 379–397.

Skut, Wojciech, Brigitte Krenn, Thorsten Brants and Hans Uszkoreit (1997). An Annotation Scheme for Free Word Order Languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)*. Washington, D.C., pp. 88–95. http://www.coli.uni-sb.de/~thorsten/publications/Skut-ea-ANLP97.ps.gz.

Soderland, S., D. Fisher, J. Aseltine and W. Lehnert (1995). CRYSTAL: Inducing a Conceptual Dictionary. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*. Montreal, Quebec, Canada, pp. 1314–1321. http://www.cis.udel.edu/~vijay/cis889/ie/papers/crystal.pdf.

Souter, Clive and Eric Atwell (eds.) (1993). *Corpus-Based Computational Linguistics*. No. 9 in Language and Computers: Studies in Practical Linguistics. Amsterdam: Rodopi.

Stegmann, Rosmary, Heike Telljohann and Erhard W. Hinrichs (2000). *Stylebook for the German Treebank in VERBMOBIL*. Verbmobil-Report 239, Universität Tübingen, Tübingen, Germany.

Taylor, Ann, Mitchell Marcus and Beatrice Santorini (2003). The Penn Treebank: An Overview. In Abeillé (2003), pp. 5–22.

Thielen, Christine and Anne Schiller (1996). Ein kleines und erweitertes Tagset fürs Deutsche. In Helmut Feldweg and Erhard W. Hinrichs (eds.), *Lexikon und Text: Wiederverwendbare Methoden und Ressourcen zur linguistischen Erschließung des Deutschen*, vol. 73 of *Lexicographica: Series maior*, pp. 215–226.

Thompson, Cynthia A., Mary Elaine Califf and Raymond J. Mooney (1999). Active learning for natural language parsing and information extraction. In *Proceedings of the 16th International Conference on Machine Learning*. San Francisco, CA: Morgan Kaufmann, pp. 406–414.

Ule, Tylman (2003). Directed Treebank Refinement for PCFG Parsing. In *Proceedings of the Second Workshop on Treebanks and Linguistic Theories (TLT 2003)*. Växjö, Sweden, pp. 177–188.

Ule, Tylman and Kiril Simov (2004). Unexpected Productions May Well be Errors. In *Proceedings of Fourth International Conference on Language Resources and Evaluation (LREC 2004)*. Lisbon, Portugal, pp. 1795–1798. http://www.sfs.nphil.uni-tuebingen.de/~ule/Paper/us04lrec.pdf.

University Centre for Computer Corpus Research on Language (UCREL) (1997). *BNC Sampler Corpus: Guidelines to Wordclass Tagging*. Tech. rep., Lancaster University. http://www.comp.lancs.ac.uk/ucrel/bnc2sampler/guide_c7.htm.

van Halteren, Hans (2000). The Detection of Inconsistency in Manually Tagged Text. In Anne Abeillé, Thosten Brants and Hans Uszkoreit (eds.), *Proceedings of the Second Workshop on Linguistically Interpreted Corpora (LINC-00)*. Luxembourg.

van Halteren, Hans, Walter Daelemans and Jakub Zavrel (2001). Improving Accuracy in Word Class Tagging through the Combination of Machine Learning Systems. *Computational Linguistics* 27(2), 199–229.

Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In *IEEE Transactions on Information Theory*. Princeton, NJ, vol. 13, pp. 260–269.

Volk, M. and G. Schneider (1998). Comparing a statistical and a rule-based tagger for German. http://www.ling.su.se/CL/CLpublications/Konvens98_Tagging.pdf.

Voutilainen, Atro and Timo Järvinen (1995). Specifying a shallow grammatical representation for parsing purposes. In *Proceedings of the Seventh Conference of the European Chapter of the Association for Computational Linguistics (EACL-95)*. Dublin, Ireland, pp. 210–214. http://acl.ldc.upenn.edu/E/E95/E95-1029.pdf.

Wahlster, Wolfgang (ed.) (2000). *Verbmobil: Foundations of Speech-to-Speech Translation*. Artificial Intelligence. Springer.

Wallis, Sean (2003). Completing Parsed Corpora. In Abeillé (2003), pp. 61–71.

Weischedel, Ralph, Marie Meteer, Richard Schwartz, Lance Ramshaw and Jeff Palmucci (1993). Coping with Ambiguity and Unknown Words through Probabilistic Models. *Computational Linguistics* 19(2), 359–382. http://acl.ldc.upenn.edu/J/J93/J93-2006.pdf.

Wichmann, Anne (1993). Gradients and categories in intonation: a study of the perception and production of falling tones. In Souter and Atwell (1993), pp. 71–84.

Wynne, Martin (1996). *A Post-Editor's Guide to CLAWS7 Tagging*. UCREL, Lancaster University, Lancaster. http://www.hcu.ox.ac.uk/BNC/what/claws7.html.

Yamamoto, Mikio and Kenneth W. Church (2001). Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus. *Computational Linguistics* 27(1), 1–30. http://portal.acm.org/citation.cfm?id=972779.

Yao, Tianfang, Wei Ding and Gregor Erbach (2002). Correcting word segmentation and part-of-speech tagging errors for Chinese named entity recognition. In Simmons (ed.), *The Internet Challenge: Technologies and Applications*, Dordrecht: Kluwer, pp. 29–36.

Zavrel, Jakub, Peter Berck and Willem Lavrijssen (2000). Information Extraction by Text Classification: Corpus Mining for Features. citeseer.nj.nec.com/zavrel00information.html.

Zhu, Xingquan and Xindong Wu (2004). Class Noise vs. Attribute Noise: A Quantitative Study. *Artificial Intelligence Review* 22(3), 177–210.

APPENDIX A

Annotation schemes

A.1 The Penn Treebank (Wall Street Journal)

A.1.1 POS annotation scheme

48 tags (36 word tags + 12 punctuation tags) (Santorini, 1990)

CC	Coordinating conjunction	TO	to
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund or present participle
IN	Preposition or subordinating conjunction	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd person singular pres
JJR	Adjective, comparative	VBZ	Verb, 3rd person singular present
JJS	Adjective, superlative	WDT	*Wh*-determiner
LS	List item marker	WP	*Wh*-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	*Wh*-adverb
NNS	Noun, plural	#	Pound sign
NP	Proper noun, singular	$	Dollar sign
NPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	”	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

A.1.2 Syntactic annotation scheme

Clause-level tags 5 Clause-level tags (Bies et al., 1995)

S	Simple declarative clause
SBAR	Clause introduced by a (possibly empty) subordinating conjunction
SBARQ	Direct question introduced by a *wh*-word or *wh*-phrase
SINV	Inverted declarative sentence
SQ	Inverted yes/no question, or main clause of a *wh*-question

Phrase-level tags 21 Phrase-level tags (Bies et al., 1995)

ADJP	Adjective Phrase
ADVP	Adverb Phrase
CONJP	Conjunction Phrase
FRAG	Fragment
INTJ	Interjection
LST	List marker
NAC	Not A Constituent
NP	Noun Phrase
NX	Used within certain complex noun phrases to mark the head of the noun phrase
PP	Prepositional Phrase
PRN	Parenthetical
PRT	Particle
QP	Quantifier Phrase
RRC	Reduced Relative Clause
UCP	Unlike Coordinated Phrase
VP	Verb Phrase
WHADJP	*Wh*-adjective Phrase
WHADVP	*Wh*-adverb Phrase
WHNP	*Wh*-noun Phrase
WHPP	*Wh*-prepositional Phrase
X	Unknown, uncertain, or unbracketable

A.2 The BNC-sampler

A.2.1 CLAWS7 (C7) Tagset, or Enriched Tagset

148 POS tags (University Centre for Computer Corpus Research on Language (UCREL), 1997; Wynne, 1996)

APPGE	possessive pronoun, pre-nominal
AT	article
AT1	singular article
BCL	before-clause marker
CC	coordinating conjunction
CCB	adversative coordinating conjunction
CS	subordinating conjunction
CSA	*as* (as conjunction)
CSN	*than* (as conjunction)
CST	*that* (as conjunction)
CSW	*whether* (as conjunction)
DA	"after-determiner", or post-determiner capable of pronominal function
DA1	singular post-determiner
DA2	plural post-determiner
DAR	comparative post-determiner
DAT	superlative post-determiner
DB	"before-determiner", or pre-determiner capable of pronominal function
DB2	plural before-determiner
DD	central determiner (capable of pronominal function)
DD1	singular determiner
DD2	plural determiner
DDQ	*wh*-determiner
DDQGE	*wh*-determiner, genitive
DDQV	*wh*-ever determiner
EX	existential *there*
FO	formula
FU	unclassified word
FW	foreign word
GE	germanic genitive marker
IF	*for* (as preposition)
II	general preposition
IO	*of* (as preposition)
IW	*with*, *without* (as prepositions)
JJ	general adjective
JJR	general comparative adjective
JJT	general superlative adjective
JK	catenative adjective
MC	cardinal number,neutral for number
MC1	singular cardinal number
MC2	plural cardinal number
MCGE	genitive cardinal number, neutral for number
MCMC	hyphenated number

MD	ordinal number
MF	fraction,neutral for number
ND1	singular noun of direction
NN	common noun, neutral for number
NN1	singular common noun
NN2	plural common noun
NNA	following noun of title
NNB	preceding noun of title
NNL1	singular locative noun
NNL2	plural locative noun
NNO	numeral noun, neutral for number
NNO2	numeral noun, plural
NNT1	temporal noun, singular
NNT2	temporal noun, plural
NNU	unit of measurement, neutral for number
NNU1	singular unit of measurement
NNU2	plural unit of measurement
NP	proper noun, neutral for number
NP1	singular proper noun
NP2	plural proper noun
NPD1	singular weekday noun
NPD2	plural weekday noun
NPM1	singular month noun
NPM2	plural month noun
PN	indefinite pronoun, neutral for number
PN1	indefinite pronoun, singular
PNQO	objective *wh*-pronoun
PNQS	subjective *wh*-pronoun
PNQV	*wh-ever* pronoun
PNX1	reflexive indefinite pronoun
PPGE	nominal possessive personal pronoun
PPH1	3rd person sing. neuter personal pronoun
PPHO1	3rd person sing. objective personal pronoun
PPHO2	3rd person plural objective personal pronoun
PPHS1	3rd person sing. subjective personal pronoun
PPHS2	3rd person plural subjective personal pronoun
PPIO1	1st person sing. objective personal pronoun
PPIO2	1st person plural objective personal pronoun
PPIS1	1st person sing. subjective personal pronoun
PPIS2	1st person plural subjective personal pronoun
PPX1	singular reflexive personal pronoun

PPX2	plural reflexive personal pronoun
PPY	2nd person personal pronoun
RA	adverb, after nominal head
REX	adverb introducing appositional constructions
RG	degree adverb
RGQ	*wh*-degree adverb
RGQV	*wh-ever* degree adverb
RGR	comparative degree adverb
RGT	superlative degree adverb
RL	locative adverb
RP	prep. adverb, particle
RPK	prep. adv., catenative
RR	general adverb
RRQ	*wh*-general adverb
RRQV	*wh-ever* general adverb
RRR	comparative general adverb
RRT	superlative general adverb
RT	quasi-nominal adverb of time
TO	infinitive marker
UH	interjection
VB0	*be*, base form
VBDR	*were*
VBDZ	*was*
VBG	*being*
VBI	*be*, infinitive
VBM	*am*
VBN	*been*
VBR	*are*
VBZ	*is*
VD0	*do*, base form
VDD	*did*
VDG	*doing*
VDI	*do*, infinitive
VDN	*done*
VDZ	*does*
VH0	*have*, base form
VHD	*had* (past tense)
VHG	*having*
VHI	*have*, infinitive
VHN	*had* (past participle)
VHZ	*has*

VM	modal auxiliary
VMK	modal catenative
VV0	base form of lexical verb
VVD	past tense of lexical verb
VVG	*-ing* participle of lexical verb
VVGK	*-ing* participle catenative
VVI	infinitive
VVN	past participle of lexical verb
VVNK	past participle catenative
VVZ	*-s* form of lexical verb
XX	*not*, *n't*
ZZ1	singular letter of the alphabet
ZZ2	plural letter of the alphabet
YBL	punctuation tag - left bracket
YBR	punctuation tag - right bracket
YCOL	punctuation tag - colon
YCOM	punctuation tag - comma
YDSH	punctuation tag - dash
YEX	punctuation tag - exclamation mark
YLIP	punctuation tag - ellipsis
YQUE	punctuation tag - question mark
YQUO	punctuation tag - quotes
YSCOL	punctuation tag - semicolon
YSTP	punctuation tag - full-stop

A.3 TIGER

A.3.1 Syntactic category labels

25 labels (Brants et al., 2002)

AA	Superlative Phrase with *am*
AP	Adjective Phrase
AVP	Adverbial Phrase
CAC	Coordinated Adposition
CAP	Coordinated Adjektive phrase
CAVP	Coordinated Adverbial phrase
CCP	Coordinated Complementiser
CH	Chunk
CNP	Coordinated Noun Phrase
CO	Coordination
CPP	Coordinated Adpositional Phrase
CS	Coordinated Sentence
CVP	Coordinated Verb Phrase (non-finite)
CVZ	Coordinated *zu*-marked Infinitive
DL	Discourse level Constituent
ISU	Idiosyncratis Unit
MTA	Multi-Token Adjective
NM	Multi-Token Number
NP	Noun Phrase
PN	Proper Noun
PP	Adpositional Phrase
QL	Quasi-Language
S	Sentence
VP	Verb Phrase (non-finite)
VZ	*Zu*-marked Infinitive

Index of Citations

Erjavec et al. (2003), 58
Erjavec (2001), 58
Erk et al. (2003), 3
Eskin (2000), 23, 24, 52
Eyes and Leech (1992), 14, 17
Fredkin (1960), 122
Gaizauskas (1995), 7, 96, 105
Gale et al. (1992), 2
Garside et al. (1987), 4
Hajičová et al. (1998), 116
Hajičová (1998), 2
Hajič (1998), 2, 116
Hemphill et al. (1990), 139
Hepple (1994), 111
Hinrichs et al. (2000), 18, 22, 110, 112, 113, 115, 130
Hirakawa et al. (2000), 25
Huck (1985), 111
Järvinen (2003), 5, 17
Jackendoff (1977), 87
Johansson (1986), 4, 10
Johnson (1985), 119
König et al. (2003), 73
Kübler and Wagner (2000), 9, 15, 16
Kübler (2003), 164
Kaljurand (2004), 19, 47, 78, 219
Kasper et al. (1998), 124
Kathol (1995), 111
Kilgarriff (1998), 15, 17, 19, 20
Kingsbury et al. (2002), 2, 140, 234
Klein and Manning (2001), 103
Kordoni (2003), 18
Kroch and Joshi (1987), 111
Krotov et al. (1998), 7, 95, 103
Květŏn and Oliva (2002), 9, 25
Lönneker and Jakopin (2004), 59
Leech et al. (1994a), 3
Leech et al. (1994b), 13, 14
Leech (1997), 2, 4, 10, 17, 50, 54, 55
Lenerz (2001), 111
Müller (1999), 111
Müller and Ule (2002), 25
Ma et al. (2001), 26
Magerman (1995), 156
Manning and Schütze (1999), 7, 24, 88, 107, 150
Marcus et al. (1993), 2, 3, 5–7, 10, 20, 32, 51, 88
Marcus et al. (1999), 72
Marquez and Padro (1997), 11, 13, 14, 190
McCawley (1982), 111
Meurers (2005), 2, 3
Mitchell (1997), 156
Mitkov et al. (1997), 2
Morrill (1995), 111
Müller (2004), 111, 118
Nakagawa and Matsumoto (2002), 23, 24
Nelson et al. (2002), 22
Oepen et al. (2004), 19
Ojeda (1987), 111
Oliva (2001), 13, 18, 28, 42, 62, 231, 238
Padro and Marquez (1998), 11, 12, 184
Penn (1999), 111
Plátek et al. (2001), 111
Pla and Molina (2004), 170, 171
Przepiórkowski and Woliński (2003), 49
Quinlan (1986), 6
Ragas and Koster (1998), 2
Rambow and Joshi (1994), 111
Ramshaw and Marcus (1995), 2
Ratnaparkhi (1996), 4, 8, 12, 16, 49
Rayson et al. (2004), 140, 234
Reape (1993), 111
Richter and Sailer (2001), 111
Riloff (1993), 2
Rocha (1997), 2
Sampson and Babarczy (2003), 20, 49
Sampson (1995), 2, 4, 55, 56
Sampson (1996), 2, 3

Sampson (2003), 56, 57
Santorini (1990), 2, 40, 48, 62, 63, 91, 94, 146, 173, 180, 182, 191, 201, 202
Schiller et al. (1995), 2, 131
Schmid (1997), 7, 154, 155, 175
Schmid (2000), 103
Segond et al. (1997), 2
Sinclair (1992), 3
Skut et al. (1997), 2, 3, 17, 110, 112, 114, 116
Soderland et al. (1995), 2
Stegmann et al. (2000), 130, 131, 137
Taylor et al. (2003), 114
Thielen and Schiller (1996), 131
Thompson et al. (1999), 18
Ule and Simov (2004), 13, 24, 78
Ule (2003), 24, 173
Viterbi (1967), 152
Volk and Schneider (1998), 14, 159
Voutilainen and Järvinen (1995), 20, 21, 48
Wallis (2003), 18
Weischedel et al. (1993), 149, 150
Wichmann (1993), 2, 3
Wynne (1996), 17, 59
Yamamoto and Church (2001), 35
Yao et al. (2002), 25
Zavrel et al. (2000), 2
Zhu and Wu (2004), 6
van Halteren et al. (2001), 10–12, 14, 16, 27, 215
van Halteren (2000), 10–12, 14, 16, 26–29, 52, 144, 147, 168, 229, 239