A generalized method for iterative error mining in parsing results

Daniël de Kok
University of Groningen
d.j.a.de.kok@rug.nl

Jianqiang Ma
University of Groningen
j.ma@student.rug.nl

Gertjan van Noord
University of Groningen
g.j.m.van.noord@rug.nl

Abstract

Error mining is a useful technique for identifying forms that cause incomplete parses of sentences. We extend the iterative method of Sagot and de la Clergerie (2006) to treat n-grams of an arbitrary length. An inherent problem of incorporating longer n-grams is data sparseness. Our new method takes sparseness into account, producing n-grams that are as long as necessary to identify problematic forms, but not longer.

Not every cause for parsing errors can be captured effectively by looking at word n-grams. We report on an algorithm for building more general patterns for mining, consisting of words and part of speech tags.

It is not easy to evaluate the various error mining techniques. We propose a new evaluation metric which will enable us to compare different error miners.

1 Introduction

In the past decade wide-coverage grammars and parsers have been developed for various languages, such as the Alpino parser and grammar (Bouma et al., 2001) for Dutch and the English Resource Grammar (Copestake and Flickinger, 2000). Such grammars account for a large number of grammatical and lexical phenomena, and achieve high accuracies. Still, they are usually tailored to general domain texts and fail to reach the same accuracy for domain-specific texts, due to missing lexicon entries, fixed expressions, and grammatical constructs. When parsing new texts there are usually two types of parsing errors:

- The parser returns an incorrect parse. While the parser may have constructed the correct parse, the disambiguation model chose an incorrect parse.
- The parser can not find an analysis that spans the full sentence. If that sentence is allowed in the language, the grammar or lexicon is incomplete.

While the first type of errors can be alleviated by improving the disambiguation model, the second type of problems requires extension of the grammar or lexicon. Finding incomplete descriptions by hand can become a tedious task once a grammar has wide coverage. Error mining techniques aim to find problematic words or n-grams automatically, allowing the grammar developer to focus on frequent and highly suspicious forms first.

2 Previous work

In the past, two major error mining techniques have been developed by Van Noord (2004) and Sagot and de la Clergerie (2006). In this paper we propose a generalized error miner that combines the strengths of these methods. Both methods follow the same basic principle: first, a large (unannotated) corpus is parsed. After parsing, the sentences can be split up in a list of parsable and a list of unparsable sentences. Words or n-grams that occur in the list of unparsable sentences, but that do not occur in the list of parsable sentences have a high suspicion of being the cause of the parsing error.

2.1 Suspicion as a ratio

Van Noord (2004) defines the suspicion of a word as a ratio:

$$S(w) = \frac{C(w|error)}{C(w)} \tag{1}$$

where $C(w)$ is the number of occurrences of word w in all sentences, and $C(w|error)$ is the

Proceedings of the 2009 Workshop on Grammar Engineering Across Frameworks, ACL-IJCNLP 2009, pages 71–79,
Suntec, Singapore, 6 August 2009. ©2009 ACL and AFNLP

number of occurrences of w in unparsable sentences. Of course, it is often useful to look at n-grams as well. For instance, Van Noord (2004) gives an example where the word *via* had a low suspicion after parsing a corpus with the Dutch Alpino parser, while the Dutch expression *via via* (via a complex route) was unparsable.

To account for such phenomena, the notion of suspicion is extended to n-grams:

$$S(w_i..w_j) = \frac{C(w_i..w_j|error)}{C(w_i..w_j)} \tag{2}$$

Where a longer sequence $w_h...w_i...w_j...w_k$ is only considered if its suspicion is higher than each of its substrings:

$$S(w_h...w_i...w_j...w_k) > S(w_i...w_j) \tag{3}$$

While this method works well for forms that are unambiguously suspicious, it also gives forms that just happened to occur often in unparsable sentences by 'bad luck' a high suspicion. If the occurrences in unparsable sentences were accompanied by unambiguously suspicious forms, there is even more reason to believe that the form is not problematic. However, in such cases this error mining method will still assign a high suspicion to such forms.

2.2 Iterative error mining

The error mining method described by Sagot and de la Clergerie (2006) alleviates the problem of 'accidentally suspicious' forms. It does so by taking the following characteristics of suspicious forms into account:

- If a form occurs within parsable sentences, it becomes less likely that the form is the cause of a parsing error.
- The suspicion of a form should depend on the suspicions of other forms in the unparsable sentences in which it occurs.
- A form observed in a shorter sentence is initially more suspicious than a form observed in a longer sentence.

To be able to handle the suspicion of a form within its context, this method introduces the notion of *observation suspicion*, which is the suspicion of a form within a given sentence. The suspicion of a form, outside the context of a sentence, is then defined to be the average of all observation suspicions:

$$S_f = \frac{1}{|O_f|} \sum_{o_{i,j} \in O_f} S_{i,j} \tag{4}$$

Here O_f is the set of all observations of the form f, $o_{i,j}$ is the j^{th} form of the i^{th} sentence, and $S_{i,j}$ is the observation suspicion of $o_{i,j}$. The observation suspicions themselves are dependent on the form suspicions, making the method an iterative process. The suspicion of an observation is the suspicion of its form, normalized by suspicions of other forms occurring within the same sentence:

$$S_{i,j}^{(n+1)} = error(s_i) \frac{S_{F(o_{i,j})}^{(n+1)}}{\sum_{1 \leq j \leq |S_i|} S_{F(o_{i,j})}^{(n+1)}} \tag{5}$$

Here $error(s_i)$ is the sentence error rate, which is normally set to 0 for parsable sentences and 1 for unparsable sentences. $S_{F(o_{i,j})}$ is the suspicion of the form of observation $o_{i,j}$.

To accommodate the iterative process, we will have to redefine the form suspicion to be dependent on the observation suspicions of the previous cycle:

$$S_f^{(n+1)} = \frac{1}{|O_f|} \sum_{o_{i,j} \in O_f} S_{i,j}^{(n)} \tag{6}$$

Since there is a recursive dependence between the suspicions and the observation suspicions, starting and stopping conditions need to be defined for this cyclic process. The observation suspicions are initialized by uniformly distributing suspicion over observed forms within a sentence:

$$S_{i,j}^{(0)} = \frac{error(s_i)}{|S_i|} \tag{7}$$

The mining is stopped when the process reaches a fixed point where suspicions have stabilized.

This method solves the 'suspicion by accident' problem of ratio-based error mining. However, the authors of the paper have only used this method to mine on unigrams and bigrams. They note that they have tried mining with longer n-grams, but encountered data sparseness problems. Their paper does not describe criteria to determine when to use unigrams and when to use bigrams to represent forms within a sentence.

3 N-gram expansion

3.1 Inclusion of n-grams

While the iterative miner described by Sagot and de la Clergerie (2006) only mines on unigrams and bigrams, our prior experience with the miner described by Van Noord (2004) has shown that including longer n-grams in the mining process can capture many additional phenomena. To give one example: the words *de* (*the*), *eerste* (*first*), and *beste* (*best*) had very low suspicions during error mining, while the trigram *eerste de beste* had a very high suspicion. This trigram occurred in the expression *de eerste de beste* (*the first you can find*). While the individual words within this expression were described by the lexicon, this multiword expression was not.

3.2 Suspicion sharing

It may seem to be attractive to include all n-grams within a sentence in the mining process. However, this is problematic due to *suspicion sharing*. For instance, consider the trigram w_1, w_2, w_3 in which w_2 is the cause of a parsing error. In this case, the bigrams w_1, w_2 and w_2, w_3 will become suspicious, as well as the trigram w_1, w_2, w_3. Since there will be multiple very suspicious forms within the same sentence the unigram w_2 will have no opportunity to manifest itself.

A more practical consideration is that the number of forms within a sentence grows at such a rate ($n + (n - 1) ... + 1$) that error mining becomes unfeasible for large corpora, both in time and in space.

3.3 Expansion method

To avoid suspicion sharing we have devised a method for adding and expanding n-grams when it is deemed useful. This method iterates through a sentence of unigrams, and expands unigrams to longer n-grams when there is evidence that it is useful. This expansion step is a preprocessor to the iterative miner, that uses the same iterative algorithm as described by Sagot and De la Clergerie. Within this preprocessor, suspicion is defined in the same manner as in Van Noord (2004), as a ratio of occurrences in unparsable sentences and the total number of occurrences.

The motivation behind this method is that there can be two expansion scenarios. When we have the bigram w_1, w_2, either one of the unigrams can be problematic or the bigram w_1, w_2. In the former case, the bigram w_1, w_2 will also inherit the high suspicion of the problematic unigram. In the latter case, the bigram will have a higher suspicion than both of its unigrams. Consequently, we want to expand the unigram w_1 to the bigram w_1, w_2 if the bigram is more suspicious than both of its unigrams. If w_1, w_2 is equally suspicious as one of its unigrams, it is not useful to expand to a bigram since we want to isolate the cause of the parsing error as much as possible.

The same methodology is followed when we expand to longer n-grams. Expansion of w_1, w_2 to the trigram w_1, w_2, w_3 will only be permitted if w_1, w_2, w_3 is more suspicious than its bigrams. Since the suspicion of w_3 aggregates to w_2, w_3, we account for both w_3 and w_2, w_3 in this comparison.

The general algorithm is that the expansion to an n-gram $i..j$ is allowed when $S(i..j) > S(i..j-1)$ and $S(i..j) > S(i+1..j)$. This gives us a sentence that is represented by the n-grams $n_0..n_x$, $n_1..n_y$, ... $n_{|s_i|-1}..n_{|s_i|-1}$.

3.4 Data sparseness

While initial experiments with the expansion algorithm provided promising results, the expansion algorithm was too eager. This eagerness is caused by data sparseness. Since longer n-grams occur less frequently, the suspicion of an n-gram occurring in unparsable sentences goes up with the length of the n-gram until it reaches its maximum value. The expansion conditions do not take this effect into account.

To counter this problem, we have introduced an expansion factor. This factor depends on the frequency of an n-gram within unparsable sentences and asymptotically approaches one for higher frequencies. As a result more burden of proof is inflicted upon the expansion: the longer n-gram either needs to be relatively frequent, or it needs to be much more suspicious than its (n-1)-grams. The expansion conditions are changed to $S(i..j) > S(i..j-1) \cdot extFactor$ and $S(i..j) > S(i+1..j) \cdot extFactor$, where

$$extFactor = 1 + e^{-\alpha|O_{f,unparsable}|} \tag{8}$$

In our experiments $\alpha = 1.0$ proved to be a good setting.

3.5 Pattern expansion

Previous work on error mining was primarily focused on the extraction of interesting word n-grams. However, it could also prove useful to allow for patterns consisting of other information than words, such as part of speech tags or lemmas. We have done preliminary work on the integration of part of speech tags during the n-gram expansion. We use the same methodology as word-based n-gram expansion, however we also consider expansion with a part of speech tag.

Since we are interested in building patterns that are as general as possible, we expand the pattern with a part of speech tag if that creates a more suspicious pattern. Expansion with a word is attempted if expansion with a part of speech tag is unsuccessful. E.g., if we attempt to expand the word bigram $w_1 w_2$, we first try the tag expansion $w_1 w_2 t_3$. This expansion is allowed when $S(w_1, w_2, t_3) > S(w_1, w_2) \cdot extFactor$ and $S(w_1, w_2, t_3) > S(w_2, t_3) \cdot extFactor$. If the expansion is not allowed, then expansion to $S(w1, w2, w3)$ is attempted. As a result, mixed patterns emerge that are as general as possible.

4 Implementation

4.1 Compact representation of data

To be able to mine large corpora some precautions need to be made. During the n-gram expansion stage, we need quick access to the frequencies of arbitrary length n-grams. Additionally, all unparsable sentences have to be kept in memory, since we have to traverse them for n-gram expansion. Ordinary methods for storing n-gram frequencies (such as hash tables) and data will not suffice for large corpora.

As Van Noord (2004) we used perfect hashing to restrict memory use, since hash codes are generally shorter than the average token length. Additionally, comparisons of numbers are much faster than comparisons of strings, which speeds up the n-gram expansion step considerably.

During the n-gram expansion step the miner calculates ratio-based suspicions of n-grams using frequencies of an n-gram in parsable and unparsable sentences. The n-gram can potentially have the length of a whole sentence, so it is not practical to store n-gram ratios in a hash table. Instead, we compute a suffix array (Manber and Myers, 1990) for the parsable and unparsable sentences[1]. A suffix array is an array that contains indices pointing to sequences in the data array, that are ordered by suffix.

We use suffix arrays differently than Van Noord (2004), because our expansion algorithm requires the parsable and unparsable frequencies of the (n-1)-grams, and the second (n-1)-gram is not (necessarily) adjacent to the n-gram in the suffix array. As such, we require random access to frequencies of n-grams occurring in the corpus. We can compute the frequency of any n-gram by looking up its upper and lower bounds in the suffix array[2], where the difference is the frequency.

4.2 Determining ratios for pattern expansion

While suffix arrays provide a compact and relatively fast data structure for looking up n-gram frequencies, they are not usable for pattern expansion (see section 3.5). Since we need to look up frequencies of every possible combination of representations that are used, we would have to create d^l suffix arrays to be (theoretically) able to look up pattern frequencies with the same time complexity, where d is the number of dimensions and l is the corpus length.

For this reason, we use a different method for calculating pattern frequencies. First, we build a hash table for each type of information that can be used in patterns. A hash table contains an instance of such information as a key (e.g. a specific word or part of speech tag) and a set of corpus indices where the instance occurred in the corpus as the value associated with that key. Now we can look up the frequency of a sequence $i..j$ by calculating the set intersection of the indices of j and the indices found for the sequence $i..j-1$, after incrementing the indices of $i..j-1$ by one.

The complexity of calculating frequencies following this method is linear, since the set of indices for a given instance can be retrieved with a $O(1)$ time complexity, while both incrementing the set indices and set intersection can be performed in $O(n)$ time. However, n can be very large: for instance, the start of sentence marker forms a substantial part of the corpus and is looked up once for every sentence. In our implementation we limit the time spent on such patterns by caching very frequent bigrams in a hash table.

4.3 Removing low-suspicion forms

Since normally only one form within a sentence will be responsible for a parsing error, many forms will have almost no suspicion at all. However, during the mining process, their suspicions will be recalculated during every cycle. Mining can be sped up considerably by removing forms that have a negligible suspicion.

[1] We use the suffix sorting algorithm by Peter M. McIlroy and M. Douglas McIlroy.

[2] Since the suffix array is sorted, finding the upper and lower bounds is a binary search in $O(log\ n)$ time.

If we do not drop forms, mining of the Dutch Wikipedia corpus described in section 5.3, with n-gram expansion and the extension factor enabled, resulted in 4.8 million forms with 13.4 million form observations in unparsable sentences. If we mine the same material and drop forms with a suspicion below 0.001 there were 3.5 million forms and 4.0 million form observations within unparsable sentences left at the end of the iterative mining process.

5 Evaluation

5.1 Methodology

In previous articles, error mining methods have primarily been evaluated manually. Both Van Noord (2004) and Sagot and de la Clergerie (2006) make a qualitative analysis of highly suspicious forms. But once one starts experimenting with various extensions, such as n-gram expansion and expansion factor functions, it is difficult to qualify changes through small-scale qualitative analysis.

To be able to evaluate changes to the error miner, we have supplemented qualitative analysis with a automatic quantitative evaluation method. Since error miners are used by grammar engineers to correct a grammar or lexicon by hand, the evaluation metric should model this use case:

- We are interested in seeing problematic forms that account for errors in a large number of unparsable sentences first.

- We are only interested in forms that actually caused the parsing errors. Analysis of forms that do not, or do not accurately pinpoint origin of the parsing errors costs a lot of time.

These requirements map respectively to the recall and precision metrics from information retrieval:

$$P = \frac{|\{S_{unparsable}\} \cap \{S_{retrieved}\}|}{|\{S_{retrieved}\}|} \quad (9)$$

$$R = \frac{|\{S_{unparsable}\} \cap \{S_{retrieved}\}|}{|\{S_{unparsable}\}|} \quad (10)$$

Consequently, we can also calculate the f-score (van Rijsbergen, 1979):

$$F - score = \frac{(1+\beta^2) \cdot (P \cdot R)}{(\beta^2 \cdot P + R)} \quad (11)$$

The f-score is often used with $\beta = 1.0$ to give as much weight to precision as recall. In evaluating error mining, this can permit cheating. For instance, consider an error mining that recalls the start of sentence marker as the first problematic form. Such a strategy would instantly give a recall of 1.0, and if the coverage of a parser for a corpus is relatively low, a relatively good initial f-score will be obtained. Since error mining is often used in situations where coverage is still low, we give more bias to precision by using $\beta = 0.5$.

We hope to provide more evidence in the future that this evaluation method indeed correlates with human evaluation. But in our experience it has the required characteristics for the evaluation of error mining. For instance, it is resistant to recalling of different or overlapping n-grams from the same sentences, or recalling n-grams that occur often in both parsable and unparsable sentences.

5.2 Scoring methods

After error mining, we can extract a list of forms and suspicions, and order the forms by their suspicion. But normally we are not only interested in forms that are the most suspicious, but forms that are suspicious and frequent. Sagot and de la Clergerie (2006) have proposed three scoring methods that can be used to rank forms:

- Concentrating on suspicions: $M_f = S_f$

- Concentrating on most frequent potential errors: $M_f = S_f|O_f|$

- Balancing between these possibilities: $M_f = S_f \cdot ln|O_f|$

For our experiments, we have replaced the observation frequencies of the form ($|O_f|$) by the frequency of observations within unparsable sentences ($|\{O_{f,unparsable}\}|$). This avoids assigning a high score to very frequent unsuspicious forms.

5.3 Material

In our experiments we have used two corpora that were parsed with the wide-coverage Alpino parser and grammar for Dutch:

- Quantitative evaluation was performed on the Dutch Wikipedia of August 2008[3]. This corpus consists of 7 million sentences (109 million words). For 8.4% of the sentences no full analysis could be found.

[3] http://ilps.science.uva.nl/WikiXML/

- A qualitative evaluation of the extensions was performed on the Flemish Mediargus newspaper corpus (up to May 31, 2007)[4]. This corpus consists of 67 million sentences (1.1 billion words). For 9.2% of the sentences no full analysis could be found.

Flemish is a variation of Dutch written and spoken in Belgium, with a grammar and lexicon that deviates slightly from standard Dutch. Previously, the Alpino grammar and lexicon was never specifically modified for parsing Flemish.

6 Results

6.1 Iterative error mining

We have evaluated the different mining methods with the three scoring functions discussed in section 5.2. In the results presented in this section we only list the results with the scoring function that performed best for a given error mining method (section 6.3 provides an overview of the best scoring functions for different mining methods).

Our first interest was if, and how much iterative error mining outperforms error mining with suspicion as a ratio. To test this, we compared the method described by Van Noord (2004) and the iterative error miner of Sagot and de la Clergerie (2006). For the iterative error miner we evaluated both on unigrams, and on unigrams and bigrams where all unigrams and bigrams are used (without further selection). Figure 6.1 shows the f-scores for these miners after *N* retrieved forms.

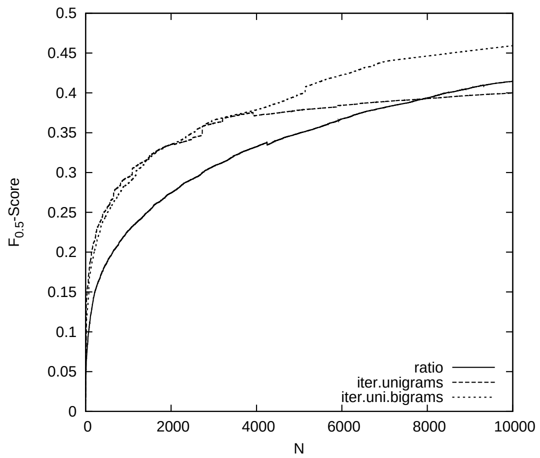

Figure 1: F-scores after retrieving N forms for ratio-based mining, iterative mining on unigrams and iterative mining on uni- and bigrams.

The unigram iterative miner outperforms the ratio-based miner during the retrieval of the first 8000 forms. The f-score graph of the iterative miner on unigrams flattens after retrieving about 4000 forms. At that point unigrams are not specific enough anymore to pinpoint more sophisticated problems. The iterative miner on uni- and bigrams performs better than the ratio-based miner, even beyond 8000 forms. More importantly, the curves of the iterative miners are steeper. This is relevant if we consider that a grammar engineer will only look at a few thousands of forms. For instance, the ratio-based miner achieves an f-score of 0.4 after retrieving 8448 forms, while the iterative miner on uni- and bigrams attains the same f-score after retrieving 5134 forms.

6.2 N-gram expansion

In our second experiment we have compared the performance of iterative mining on uni- and bigrams with an iterative miner using the n-gram expansion algorithm described in section 3. Figure 6.2 shows the result of n-gram expansion compared to mining just uni- and bigrams. Both the results for expansion with and without use of the expansion factor are shown.

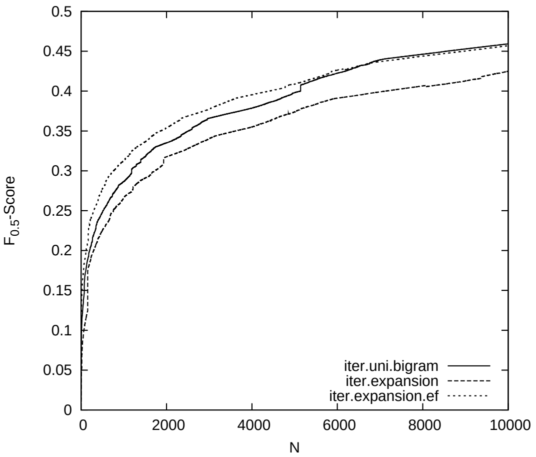

Figure 2: F-scores after retrieving N forms for iterative mining on uni- and bigrams, and iterative mining using n-gram expansion with and without using an expansion factor.

We can see that the expansion to longer n-grams gives worse results than mining on uni- and bigrams when data sparseness is not accounted for. The expansion stage will select forms that may be accurate, but that are more specific than needed. As such, the recall per retrieved form is lower on

[4] http://www.mediargus.be/

average, as can be seen in figure 6.2. But if sparseness is taken into account through the use of the expansion factor, we achieve higher f-scores than mining on uni- and bigrams up to the retrieval of circa five thousand forms. Since a user of an error mining tool will probably only look at the first few thousands of forms, this is a welcome improvement.

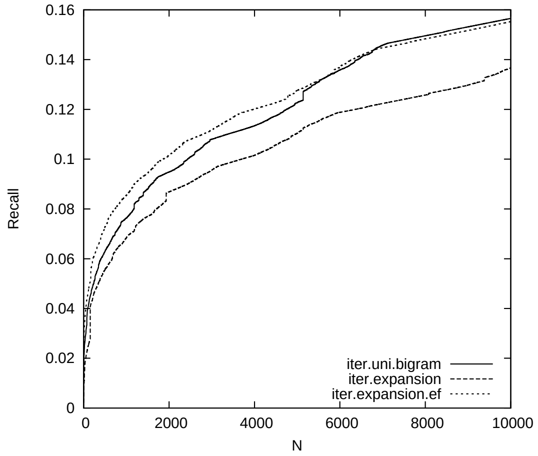

Figure 3: Recall after retrieving N forms for iterative mining on uni- and bigrams, and iterative mining using n-gram expansion with and without using an expansion factor.

Among the longer n-grams in the mining results for the Mediargus corpus, we found many Flemish idiomatic expressions that were not described in the Alpino lexicon. For example:

- had er (AMOUNT) voor veil [had (AMOUNT) for sale]
- (om de muren) van op te lopen [to get terribly annoyed by]
- Ik durf zeggen dat [I dare to say that]
- op punt stellen [to fix/correct something]
- de daver op het lijf [shocked]
- (op) de tippen (van zijn tenen) [being very careful]
- ben fier dat [am proud of]
- Nog voor halfweg [still before halfway]
- (om duimen en vingers) van af te likken [delicious]

Since these expressions are longer than bigrams, they cannot be captured properly without using n-gram expansion. We also found longer n-grams describing valid Dutch phrases that were not described by the grammar or lexicon.

- Het stond in de sterren geschreven dat [It was written in the stars that]
- zowat de helft van de [about half of the]
- er zo goed als zeker van dat [almost sure of]
- laat ons hopen dat het/dit lukt [let us hope that it/this works]

6.3 Scoring methods

The miners that use n-gram expansion perform best with the $M_f = S_f|O_f|$ function, while the other miners perform best with the $M_f = S_f \cdot ln|O_f|$ function. This is not surprising – the iterative miners that do not use n-gram expansion can not make very specific forms and give relatively high scores to forms that happen to occur in unparsable sentences (since some forms in a sentence will have to take blame, if no specific suspicious form is found). If such forms also happen to be frequent, they may be ranked higher than some more suspicious infrequent forms. In the case of the ratio-based miner, there are many forms that are ‘suspicious by accident’ which may become highly ranked when they are more frequent than very suspicious, but infrequent forms. Since the miners with n-gram expansion can find specific suspicious forms and shift blame to them, there is less chance of accidentally ranking a form to highly by directly including the frequency of observations of that form within unparsable sentences in the scoring function.

6.4 Pattern expansion

We have done some preliminary experiments with pattern expansion, allowing for patterns consisting of words and part of speech tags. For this experiment we trained a Hidden Markov Model part of speech tagger on 90% of the Dutch Eindhoven corpus using a small tag set. We then extracted 50000 unparsable and about 495000 parsable sentences from the Flemish Mediargus corpus. The pattern expansion preprocessor was then used to find interesting patterns.

We give two patterns that were extracted to give an impression how patterns can be useful. A frequent pattern was *doorheen N* (*through* followed by a (proper) noun). In Flemish a sentence such as *We reden met de auto doorheen Frankrijk* (literal: *We drove with the car through France*) is allowed, while in standard Dutch the particle *heen* is separated from the preposition *door*. Consequently, the same sentence in standard Dutch is *We reden met de auto door Frankrijk heen*. Mining on word n-grams provided hints for this difference in Flemish through forms such as *doorheen Krottegem*, *doorheen Engeland*, *doorheen Hawai*, and *doorheen Middelkerke*, but the pattern provides a more general description with a higher frequency.

Another pattern that was found is *wegens Prep Adj* (*because of* followed by a preposition and an adjective). This pattern captures prepositional modifiers where *wegens* is the head, and the following words within the constituent form an argument, such as in the sentence *Dat idee werd snel opgeborgen wegens te duur* (literal: *That idea became soon archived because of too expensive*). This pattern provided a more general description of forms such as *wegens te breed* (*because it is too wide*), *wegens te deprimerend* (*because it is too depressing*), *wegens niet rendabel* (*because it is not profitable*), and *wegens te ondraaglijk* (*because it is too unbearable*).

While instances of both patterns were found using the word n-gram based miner, patterns consolidate different instances. For example, there were 120 forms with a high suspicion containing the word *wegens*. If such a form is corrected, the other examples may still need to be checked to see if a solution to the parsing problem is comprehensive. The pattern gives a more general description of the problem, and as such, most of these 120 forms can be represented by the pattern *wegens Prep Adj*.

Since we are still optimizing the pattern expander to scale to large corpora, we have not performed an automatic evaluation using the Dutch Wikipedia yet.

7 Conclusions

We combined iterative error mining with expansion of forms to n-grams of an arbitrary length, that are long enough to capture interesting phenomena, but not longer. We dealt with the problem of data sparseness by introducing an expansion factor that softens when the expanded form is very frequent.

In addition to the generalization of iterative error mining, we introduced a method for automatic evaluation. This allows us to test modifications to the error miner without going through the tedious task of ranking and judging the results manually.

Using this automatic evaluation method, we have shown that iterative error mining improves upon ratio-based error mining. As expected, adding bigrams improves performance. Allowing expansion beyond bigrams can lead to data sparseness problems, but if we correct for data sparseness the performance of the miner improves over mining on just unigrams and bigrams.

We have also described preliminary work on a preprocessor that allows for more general patterns that incorporate additional information, such as part of speech tags and lemmas. We hope to optimize and improve pattern-based mining in the future and evaluate it automatically on larger corpora.

The error mining methods described in this paper are generic, and can be used for any grammar or parser, as long as the sentences within the corpus can be divided in a list of parsable and unparsable sentences. The error miner is freely available[5], and is optimized to work on large corpora. The source distribution includes a graphical user interface for browsing mining results, showing the associated sentences, and removing forms when they have been corrected in the grammar or lexicon.

References

Gosse Bouma, Gertjan van Noord, and Robert Malouf. 2001. Alpino: Wide-coverage Computational Analysis of Dutch. In *Computational Linguistics in The Netherlands 2000*.

Ann Copestake and Dan Flickinger. 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of LREC 2000*, pages 591–600.

Udi Manber and Gene Myers. 1990. Suffix arrays: a new method for on-line string searches. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 319–327. Society for Industrial and Applied Mathematics.

Benoît Sagot and Éric de la Clergerie. 2006. Error mining in parsing results. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 329–336, Morristown, NJ, USA. Association for Computational Linguistics.

[5] http://www.let.rug.nl/~dekok/errormining/

Gertjan Van Noord. 2004. Error mining for wide-coverage grammar engineering. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 446, Morristown, NJ, USA. Association for Computational Linguistics.

C. J. van Rijsbergen. 1979. *Information retrieval*. Butterworths, London, 2 edition.