A Graphical Interface for Automatic Error Mining in Corpora

Gregor Thiele Wolfgang Seeker Markus Gärtner Anders Björkelund Jonas Kuhn
Institute for Natural Language Processing
University of Stuttgart
{thielegr,seeker,gaertnms,anders,kuhn}@ims.uni-stuttgart.de

Abstract

We present an error mining tool that is designed to help human annotators to find errors and inconsistencies in their annotation. The output of the underlying algorithm is accessible via a graphical user interface, which provides two aggregate views: a list of potential errors in context and a distribution over labels. The user can always directly access the actual sentence containing the potential error, thus enabling annotators to quickly judge whether the found candidate is indeed incorrectly labeled.

1 Introduction

Manually annotated corpora and treebanks are the primary tools that we have for developing and evaluating models and theories for natural language processing. Given their importance for testing our hypotheses, it is imperative that they are of the best quality possible. However, manual annotation is tedious and error-prone, especially if many annotators are involved. It is therefore desirable to have automatic means for detecting errors and inconsistencies in the annotation.

Automatic methods for error detection in treebanks have been developed in the DECCA project[1] for several different annotation types, for example part-of-speech (Dickinson and Meurers, 2003a), constituency syntax (Dickinson and Meurers, 2003b), and dependency syntax (Boyd et al., 2008). These algorithms work on the assumption that two data points that appear in identical contexts should be labeled in the same way. While the data points in question, or *nuclei*, can be single tokens, spans of tokens, or edges between two tokens, context is usually modeled as n-grams over the surrounding tokens. A nucleus that occurs multiple times in identical contexts but is labeled differently shows variation and is considered a potential error.

Natural language is ambiguous and variation found by an algorithm may be a genuine ambiguity rather than an annotation error. Although we can support an annotator in finding inconsistencies in a treebank, these inconsistencies still need to be judged by humans. In this paper, we present a tool that allows a user to run automatic error detection on a corpus annotated with part-of-speech or dependency syntax.[2] The tool provides the user with a graphical interface to browse the variation nuclei found by the algorithm and inspect their label distribution. The user can always switch between high-level aggregate views and the actual sentences containing the potential error in order to decide if that particular annotation is incorrect or not. The interface thus brings together the output of the error detection algorithm with a direct access to the corpus data. This speeds up the process of tracking down inconsistencies and errors in the annotation considerably compared to working with the raw output of the original DECCA tools. Several options allow the user to fine-tune the behavior of the algorithm. The tool is part of ICARUS (Gärtner et al., 2013), a general search and exploration tool.[3]

2 The Error Detection Algorithm

The algorithm, described in Dickinson and Meurers (2003a) for POS tags, works by starting from individual tokens (the nuclei) by recording their assigned part-of-speech over an entire treebank. From there, it iteratively increases the context for each instance by extending the string to both sides to include adjacent tokens. It thus successively builds larger n-grams by adding tokens to the left

[1] http://www.decca.osu.edu

[2] Generalizing the tool to support any kind of positional annotation is planned.

[3] http://www.ims.uni-stuttgart.de/data/icarus.html

Proceedings of the Demonstrations at the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 57–60,
Gothenburg, Sweden, April 26-30 2014.

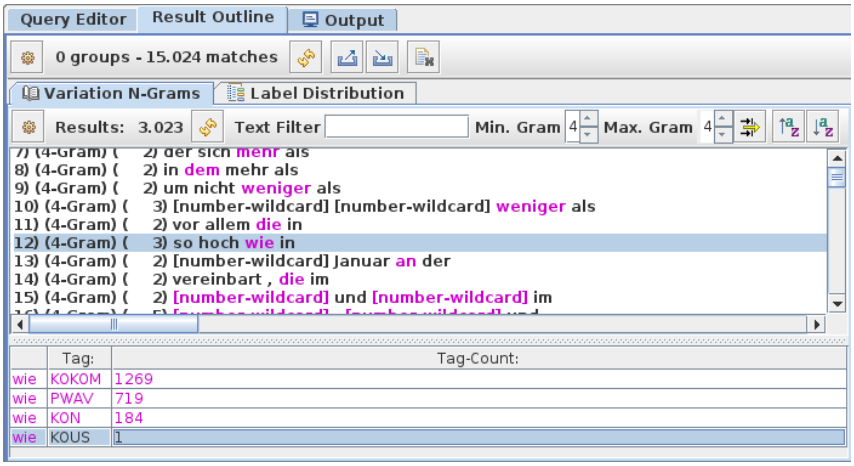

Figure 1: The variation n-gram view.

or to the right. Instances are grouped together if their context is identical, i. e. if their token n-grams match. Groups where all instances have the same label do not show variation and are discarded. The algorithm stops when either no variation nuclei are left or when none of them can be further extended. All remaining groups that show variation are considered potential errors. Erroneous annotations that do not show variation in the data cannot be found by the algorithm. This limits the usefulness of the method for very small data sets. Also, given the inherent ambiguity of natural language, the algorithm is not guaranteed to exclusively output errors, but it achieves very high precision in experiments on several languages.

The algorithm has been extended to find errors in constituency and dependency structures (Dickinson and Meurers, 2003b; Boyd et al., 2008), where the definition of a nucleus is changed to capture phrases and dependency edges. Context is always modeled using n-grams over surrounding tokens, but see, e. g., Boyd et al. (2007) for extensions.

3 Graphical Error Mining

To start the error mining, a treebank and an error mining algorithm (part-of-speech or dependency) must be selected. The algorithm is then executed on the data to create the variation n-grams. The user can choose between two views for browsing the potential errors in the treebank: (1) a view showing the list of variation n-grams found by the error detection algorithm and (2) a view showing label distributions over word forms.

3.1 The Variation N-Gram View

Figure 1 shows a screenshot of the view where the user is presented with the list of variation n-grams output by the error detection algorithm. The main window shows the list of n-grams. When the user selects one of the n-grams, information about the nucleus is displayed below the main window. The user can inspect the distribution over labels (here part-of-speech tags) with their absolute frequencies. Above the main window, the user can adjust the length of the presented n-grams, sort them, or search for specific strings.

For example, Figure 1 shows a part of the variation n-grams found in the German TiGer corpus (Brants et al., 2002). The minimum and maximum length was restricted to four, thus the list contains only 4-grams. The 4-gram *so hoch wie in* was selected, which contains *wie* as its nucleus. In the lower part, the user can see that *wie* occurs with four different part-of-speech tags in the treebank, namely *KOKOM*, *PWAV*, *KON*, and *KOUS*. Note that the combination with *KOUS* occurs only once in the entire treebank.

Double clicking on the selected 4-gram in the list will open up a new tab that displays all sentences that contain this n-gram, with the nucleus being highlighted. The user can then go through each of the sentences and decide whether the annotated part-of-speech tag is correct. Each time the user clicks on an n-gram, a new tab will be created, so that the user can jump back to previous results without having to recreate them.

A double click on one of the lines in the lower part of the window will bring up all sentences that contain that particular combination of word form

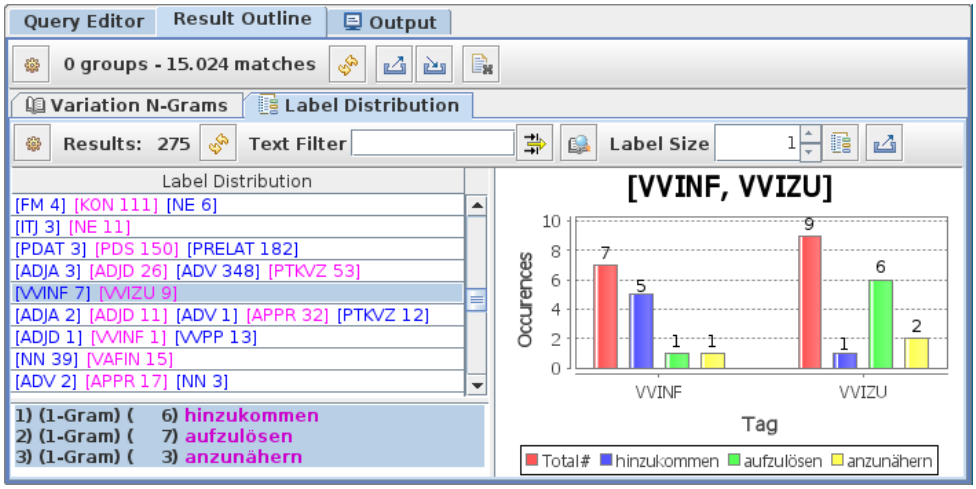

Figure 2: The label distribution view.

and part-of-speech tag. The fourth line will, for example, show the one sentence where *wie* has been tagged as *KOUS*, making it easy to quickly judge whether the tag is correct. In this case, the annotation is incorrect (it should have been *PWAV*) and should thus be marked for correction.

3.2 The Label Distribution View

In addition to the output of the algorithm by Dickinson and Meurers (2003a), the tool also provides a second view, which displays tag distributions of word forms to the user (see Figure 2). To the left, a list of unique label combinations is shown. Selecting one of them displays a list of word forms that occur with exactly these tags in the corpus. This list is shown below the list of label combinations. To the right, the frequencies of the different labels are shown in a bar chart. The leftmost bar for each label always shows the total frequency summed over all word forms in the set. Selecting one or more in the list of word forms adds additional bars to the chart that show the frequencies for each selected word form.

As an example, Figure 2 shows the tag combination *[VVINF][VVIZU]*, which are used to tag infinitives with and without incorporated *zu* in German. There are three word forms in the corpus that occur with these two part-of-speech tags: *hinzukommen*, *aufzulösen*, and *anzunähern*. The chart on the right shows the frequencies for each word form and part-of-speech tag, revealing that *hinzukommen* is mostly tagged as *VVINF* but once as *VVIZU*, whereas for the other two word forms it is the other way around. This example is interesting if one is looking for annotation errors in the TiGer treebank, because the two part-of-speech tags should have a complementary distribution (a German verb either incorporates *zu* or it does not).

Double clicking on the word forms in the list in the lower left corner will again open up a tab that shows all sentences containing this word form, regardless of their part-of-speech tag. The user may then inspect the sentences and decide whether the annotations are erroneous or not. If the user wants to see a specific combination, which is more useful if the total number of sentences is large, she can also click on one of the bars in the chart to get all sentences matching that combination. In the example, the one instance of *hinzukommen* being tagged as *VVIZU* is incorrect,[4] and the instances of the two other verbs tagged as *VVINF* are as well.

3.3 Dependency Annotation Errors

As mentioned before, the tool also allows the user to search for errors in dependency structures. The error mining algorithm for dependency structures (Boyd et al., 2008) is very similar to the one for part-of-speech tags, and so is the interface to the n-gram list or the distribution view. Dependency edges are therein displayed as triples: the head, the dependent, and the edge label with the edge's direction. As with the part-of-speech tags, the user can always jump directly to the sentences that contain a particular n-gram or dependency relation.

[4] Actually, the word form *hinzukommen* can belong to two different verbs, *hinzu-kommen* and *hin-kommen*. However, the latter, which incorporates *zu*, does not occur in TiGer.

4 Error Detection on TiGer

We ran the error mining algorithm for part-of-speech on the German TiGer Treebank (the dependency version by Seeker and Kuhn (2012)) and manually evaluated a small sample of n-grams in order to get an idea of how useful the output is.

We manually checked 115 out of the 207 variation 6-grams found by the tool, which amounts to 119 different nuclei. For 99.16% of these nuclei, we found erroneous annotations in the associated sentences. 95.6% of these are errors where we are able to decide what the right tag should be, the remaining ones are more difficult to disambiguate because the annotation guidelines do not cover them.

These results are in line with findings by Dickinson and Meurers (2003a) for the Penn Treebank. They show that even manually annotated corpora contain errors and an automatic error mining tool can be a big help in finding them. Furthermore, it can help annotators to improve their annotation guidelines by pointing out phenomena that are not covered by the guidelines, because these phenomena will be more likely to show variation.

5 Related Work

We are aware of only one other graphical tool that was developed to help with error detection in treebanks: Ambati et al. (2010) and Agarwal et al. (2012) describe a graphical tool that was used in the annotation of the Hindi Dependency Treebank. To find errors, it uses a statistical and a rule-based component. The statistical component is recall-oriented and learns a MaxEnt model, which is used to flag dependency edges as errors if their probability falls below a predefined threshold. In order to increase the precision, the output is post-processed by the rule-based component, which is tailored to the treebank's annotation guidelines. Errors are presented to the annotators in tables, also with the option to go to the sentences directly from there. Unlike the algorithm we implemented, this approach needs annotated training data for training the classifier and tuning the respective thresholds.

6 Conclusion

High-quality annotations for linguistic corpora are important for testing hypotheses in NLP and linguistic research. Automatically marking potential annotation errors and inconsistencies are one way of supporting annotators in their work. We presented a tool that provides a graphical interface for annotators to find and evaluate annotation errors in treebanks. It implements the error detection algorithms by Dickinson and Meurers (2003a) and Boyd et al. (2008). The user can view errors from two perspectives that aggregate error information found by the algorithm, and it is always easy to go directly to the actual sentences for manual inspection. The tool is currently extended such that annotators can make changes to the data directly in the interface when they find an error.

Acknowledgements

We thank Markus Dickinson for his comments. Funded by BMBF via project No. 01UG1120F, CLARIN-D, and by DFG via SFB 732, project D8.

References

Rahul Agarwal, Bharat Ram Ambati, and Anil Kumar Singh. 2012. A GUI to Detect and Correct Errors in Hindi Dependency Treebank. In *LREC 2012*, pages 1907–1911.

Bharat Ram Ambati, Mridul Gupta, Samar Husain, and Dipti Misra Sharma. 2010. A High Recall Error Identification Tool for Hindi Treebank Validation. In *LREC 2010*.

Adriane Boyd, Markus Dickinson, and Detmar Meurers. 2007. Increasing the Recall of Corpus Annotation Error Detection. In *TLT 2007*, pages 19–30.

Adriane Boyd, Markus Dickinson, and Detmar Meurers. 2008. On Detecting Errors in Dependency Treebanks. *Research on Language and Computation*, 6(2):113–137.

Sabine Brants, Stefanie Dipper, Silvia Hansen-Shirra, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *TLT 2002*, pages 24–41.

Markus Dickinson and W. Detmar Meurers. 2003a. Detecting Errors in Part-of-Speech Annotation. In *EACL 2003*, pages 107–114.

Markus Dickinson and W. Detmar Meurers. 2003b. Detecting Inconsistencies in Treebanks. In *TLT 2003*, pages 45–56.

Markus Gärtner, Gregor Thiele, Wolfgang Seeker, Anders Björkelund, and Jonas Kuhn. 2013. ICARUS – An Extensible Graphical Search Tool for Dependency Treebanks. In *ACL: System Demonstrations*, pages 55–60.

Wolfgang Seeker and Jonas Kuhn. 2012. Making Ellipses Explicit in Dependency Conversion for a German Treebank. In *LREC 2012*, pages 3132–3139.