Unexpected Productions May Well be Errors

Tylman Ule* **and Kiril Simov†**

*Seminar für Sprachwissenschaft
Universität Tübingen
ule@sfs.uni-tuebingen.de

†Linguistic Modelling Laboratory
Bulgarian Academy of Sciences
kivs@bultreebank.org

Abstract

We present a method for detecting annotation errors in treebanks. It assumes that errors are unexpected small tree fragments. We generate statistics over configurations of these fragments using a standard statistical test. We use the test result and the characteristics of their distributions as features to classify unseen configurations as likely errors via machine learning. Evaluation shows that the resulting list of error candidates is reliable, independent of corpus size, annotation quality, and target language.

Setting up language resources involves considerable effort, because human intervention is inevitable and costly. Human annotators are essential, because they usually outperform automatic methods in terms of annotation accuracy, but they still make their own kind of errors. In addition to genuine mistakes, they do not always behave identically each time when presented with the same infrequent problem. Thus one can expect a number of errors to be present in any hand-built language resource.

We divide these errors into the following categories: *violations of the annotation guidelines* and *violations of language principles not covered by the annotation guidelines*. Additionally, following Blaheta (2002), errors can also be: *detectable* – errors that are easy to spot and fix by using queries over the annotation that define impossible configurations and transformations for correction; *fixable* – errors which can be found automatically, but that require human intervention for correction; *systematic inconsistencies* – errors which are not covered by the annotation guidelines, or errors not described precisely enough in the guidelines. These two classifications of errors in annotated corpora are orthogonal, but not independent: we can expect errors that are violations of the annotation guidelines to be usually detectable and fixable, and those that are a violation of language principles, but not covered by the annotation guidelines, to be more frequently systematic inconsistencies.

Each class of errors requires a specific way for detection and correction. Detectable errors covered by the guidelines are the easiest in this respect. They can be addressed by encoding the guidelines in a formal way and by testing the corpus for consistency. Detecting the other types of errors requires additional linguistic knowledge. Such knowledge is not always available or easy to acquire, so that other mechanisms are desirable for error detection. We divide those methods into *symbolic* and *non-symbolic* approaches. The *symbolic* approaches are based on (linguistically motivated) pattern matching selecting possible deviations from linguistically correct occurrences. Patterns can be devised by human annotators, or they can be extracted (semi-)automatically from the corpus itself. The *non-symbolic* approaches use statistical methods to find rare events in the annotated corpus, where an event is a certain fragment of the annotation. In general, these methods can find errors in each of the above categories, but they are especially useful when pattern-based approaches are not easily applicable, because patterns are difficult to find.

We present such a non-symbolic method that attacks errors and inconsistencies in structural annotation, and that shows good performance across languages and annotation schemes. We detect errors and inconsistencies that appear as unexpected events in a corpus using a variant of Directed Treebank Refinement (DTR; Ule, 2003) on artificially introduced errors and apply machine learning (ML) to produce fully automatically a list of likely error candidates.

1. Methods and Data

1.1. Unexpected Productions

DTR aims at spotting productions of nonterminal nodes in treebanks that behave not as expected when they appear in certain contexts. DTR looks at all types of nonterminal nodes f_i in a treebank and compares the distribution of each of f_i's productions over the whole treebank with its productions when appearing under a certain parent node (the *context* type k: c_{ik}). DTR is applied iteratively, and in each iteration it delivers a single type of *focus* node f_i that has the most unexpected distribution of productions in a certain context c_{ik}. We compare the distributions of the m different production types p_{im} of node f_i, where production means sequence of direct children. In order to compare these distributions, we employ the χ^2 metric, which computes the sum of squared differences between expected and observed frequencies of node type f_i having production type p_{im} in context type c_{ik}, normalised by the expected frequency:

$$\chi^2_{ik} = \sum_m \frac{\left(expfreq(c_{ik}, p_{im}) - obsfreq(c_{ik}, p_{im})\right)^2}{expfreq(c_{ik}, p_{im})}$$

The χ^2 statistical test prescribes minimal values for expected and observed frequencies. With lower values, the test yields higher significance than it should, making it statistically unsound. We use this as a feature, and (mis-)employ the χ^2 test for spotting errors: very unexpected events ($expfreq(c_{ik}, p_{im}) \ll 1$) are rated high even when occurring few times (e.g. when $obsfreq(c_{ik}, p_{im}) = 1$ and $expfreq(c_{ik}, p_{im}) = 1/1000$, then $\chi^2_{ik} \approx 1000$). An event thus is a (context, focus, production) triple: (c_{ik}, f_i, p_{im}). We argue that very unexpected events, that moreover occur rarely in a corpus, may well be errors. We call these events *error candidates*.

1.2. Ranking Error Candidates

DTR typically involves several hundred iterations, and each iteration covers a focus node in context with many different productions, yielding a high number of candidate errors. In order to focus on the most likely candidates, we choose to employ a supervised ML regime using memory-based learning implemented in Timbl (Daelemans et al., 2003).[1] We introduce artificial errors into the corpus and generate the following features that characterise error candidates from the (c_{ik}, f_i, p_{im}) triples, resulting in positive training data:

$obsfreq(c_{ik}, p_{im})$ occurrences observed in the corpus

$expfreq(c_{ik}, p_{im})$ the expected number of occurrences

$\frac{\left(expfreq(c_{ik},p_{im}) - obsfreq(c_{ik},p_{im})\right)^2}{expfreq(c_{ik},p_{im})}$ its contribution to χ^2_{ik}

χ^2_{ik} the overall χ^2_{ik} over all m

$termratio$ fraction of overall χ^2_{ik} contributed by this triple

$rank$ triple is $rank$ highest contributor to χ^2_{ik}

$rankratio$ the relative position as contributor: $rank/m$

alt the number of other contributors to χ^2_{ik}, i.e. $m-1$

$obsfreq(c_{ik} > f_i)$ the number of times c_{ik} dominates f_i

$iter$ the iteration of DTR detecting (c_{ik}, f_i, p_{im})

$iterratio$ fraction of $iter$ from all DTR iterations

The motivation for the above list of features is to present all non-symbolic information to Timbl that could be relevant for identifying errors. Output of the ML stage is four classes: error in f, error in p, error in c, or no error. Combinations are not represented as separate classes: f also is used when an additional error occurs in p or c, and p is the class also for errors in p and c. Having more classes or just binary classes did not improve precision but harmed recall on the most reliable focus node.

Our method of error detection (ED) is based on the ranking of the error candidates with respect to the parameters provided by DTR. We apply ML techniques to support ranking the error candidates because the actual ranking is hard to define explicitly, as there are many dependencies among the parameters. However, recall of ML is rather low overall, so that we rejoin the ML output with all other error candidates. We sort the resulting list of triples so that generally triples marked as errors by ML and occurring less frequently are given first. The sort keys we use are (with matching list items sorted first):

1	$obsfreq(c_{ik}, p_{im}) \leq 3$	6	smaller $expfreq(c_{ik}, p_{im})$
2	ML says focus node error	7	higher $rankratio$
3	ML predicts some error	8	higher $termratio$
4	$rankratio = termratio$	9	lower $iterratio$
5	smaller $obsfreq(c_{ik}, p_{im})$		

The sort keys 2 and 3 account for the ML's reliable classification. Key 4 prefers focus nodes that have few but equally unexpected productions. The other keys generally rank those events higher that are less expected, and that occur infrequently. This sorting combines all information acquired by ED in a single ordered list. The resulting list of error candidates is presented to a human annotator, who has to judge whether the errors are true positives.

[1] We use Timbl version 4.3.1 for our experiments.

1.3. The BTB and TB Treebanks

We apply ED to two manually annotated treebanks: BulTreeBank and Tübinger Baumbank des Deutschen / Zeitung. BulTreeBank (BTB) is an HPSG-based treebank for Bulgarian annotated with detailed syntactic information (Simov et al., 2001). It contains more than 10000 sentences that have been extracted from grammars of the Bulgarian language and from electronic texts. Its annotation scheme is constituency-based. However, each constituent is additionally classified with respect to head-dependant relations like: head-complement, head-subject, etc. Keeping the original word order unchanged, we have introduced discontinuous constituents. The reference interaction among the constituents is expressed by coreferential relations. In the experiments reported below we use the most elaborated part of the treebank, which consists of 580 sentences.

The second treebank under consideration is the Tübinger Baumbank des Deutschen / Zeitung (TB; Telljohann et al., 2003). It consists of texts from the newspaper *taz* which are annotated according to the topological field model of German, and also including constituent structure, where the constituents are marked with their grammatical function in the clause. We use four data sets for our experiments, consisting of less revised data (*early*), almost finished data (*late*), and release data (*release*), which includes the sentences of the *early* and *late* data sets, but which has undergone more extensive revisions (see table 1 for the sizes of the data sets).[2]

ED is implemented to operate on a data model that we call the *export* model (Brants, 1997). It represents linguistic annotation as directed acyclic graphs with labeled nodes and edges. ED operates on this data model, only ignoring secondary edges. It is significant how the structure of the annotation in a treebank is represented in the export model, because this representation determines the distribution of the relevant events. When generating the export representation of the data sets, we decided to ignore information about grammatical functions in order to overcome sparse data problems caused by infrequent lexical information.

1.4. Evaluation via Artificial Errors

Evaluation of methods similar to ours is a challenge, because the original training material is meant to be error free, and the results can only be evaluated indirectly by manually checking whether the method discovers some errors in the treebank, which only yields precision, but not recall of the method. Thus we need a corpus of errors for training and testing. In order for the resulting corpus to be objective, we decided to introduce artificial errors automatically and randomly by permuting node labels in a given percentage of all nodes. This procedure has the advantage of introducing a set of errors with given properties, such as the number

[2] The *release* data set is available at `http://www.sfs.uni-tuebingen.de/de_tuebadz.shtml`.

of the introduced errors, their nature (via changing the list of the categories involved), or the places to introduce them (via patterns for selecting a subset of nodes in the treebank).

We are aware that randomly changing node labels does not resemble all kinds of errors equally well, but will be more similar to typos (where a wrong label is accidentally selected) than to misinterpreted larger structures. Randomly changed node labels may even be correct according to the annotation guidelines when the guidelines do not prescribe a single solution. It will be useful, though, when those parts of the guidelines become apparent.

2. Experiments and Results

We perform two sets of experiments that concentrate on the ability of ED to spot artificial errors, and on its ability to spot errors in the original data. For evaluating ED's ability to detect artificial errors we inject errors into 0.01%, 0.1% and 1% of the nodes in the BTB, and TB *early*/*late*/*release* datasets. DTR is applied to these twelve data sets with the stopping condition of $\alpha < 1$.[3] Each resulting list of error candidates is classified by ML using ten-fold cross-validation and then sorted to produce a ranked list as explained above. Table 1 shows the number of artificial errors introduced into the datasets and the overall number of these errors covered by the full list produced by ED (i.e. all errors present in some part of the (c, f, p) triples in the list). Table 2 shows precision and recall of the ML stage for the *focus* error class.

	BTB	TB *late*	TB *early*	TB *release*
sent.	580	3074	7398	15260
nodes	15013	56601	132640	318596
1.00%	155/44	579/362	1279/856	3168/2641
0.10%	12/5	57/38	125/90	306/246
0.01%	2/0	10/6	10/8	35/26

Table 1: Artificial Errors Introduced/Detected

1.00%	.42/.35	.72/.72	.60/.65	.68/.69
0.10%	0.0/0.0	.49/.59	.44/.61	.73/.69
0.01%	0.0/0.0	0.0/0.0	0.0/0.0	.30/.30

Table 2: ML Precision/Recall for *focus* Errors

In addition to the number of errors present in the ED list it is most relevant how much human labour is needed to decide whether an error candidate is an actual error.[4] Given that human labour involved in finding an error is proportional to the number of corpus positions that have to be checked to find a true error, figures 1 (a) to (c) show the amount of labour necessary to find a given proportion of the artificial errors. They plot the number of wrongly proposed error candidates per correctly identified error, i.e. the number of corpus positions without error you have to check manually until you find an error. Going through the list top-down, the X axis shows the percentage of all artificial errors covered by the top ranks of the list so far. Figure 1 shows that for all combinations of size and language of treebanks, and for all relative numbers of artificial errors, ED points to many true artificial errors first. It shows good performance in that for spotting up to 25% of all errors, you have to check at most ten corpus positions per error. In most cases you find more than 50% of the errors by looking at two corpus positions per error. The method seems to be applicable already to relatively small corpora (BTB), and it performs well for unfinished (TB *early*) as well as for highly edited data (BTB and TB *release*). As expected, it seems to be easier to spot errors in cleaner and larger data sets.

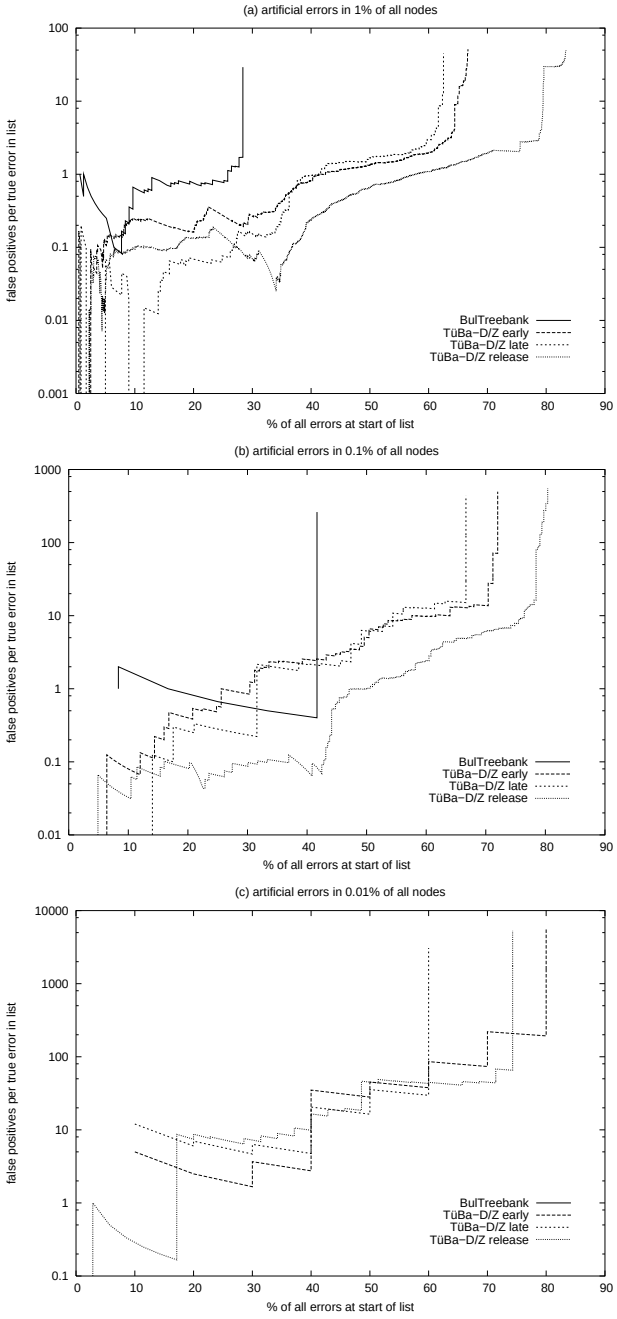

Figure 1: False Positives in Top Ranks of Error List

The second set of experiments tries to evaluate the relevance of ED for detecting errors in the original data sets. For one of the above experiments (BTB, 0.1% art. errors)

[3] In a statistically valid experiment α is the certainty that the null hypothesis of two distributions being equal can be rejected.

[4] Note that a single (c_{ik}, f_i, p_{im}) can occur often in a corpus.

we checked whether there are errors in the original data among the first candidates of artificial errors. We found that among the first 27 error candidates, there were 11 errors. The remaining 16 candidates include five examples where the guidelines are unclear, and eleven productions that are correct, but rare. We performed a similar experiment for TB, checking the highest ranked candidate errors of the *release* data set that are found by training Timbl on data with 0.01% artificial errors and applying it to the original, clean data. The resulting ED list shows 12 errors and 3 unclear cases among the first 20 candidates. Errors included e.g. a finite verb within an infinite verbal phrase, and a missing field node between sentential and phrasal nodes. Rerunning the same experiment with Timbl trained on data with 0.1% artificial errors results in 2 errors and 9 unclear cases among the first 30 candidates, indicating that the training data should resemble the rather clean target data.

It is likely that randomly changing node labels does not resemble well the distribution of naturally occurring errors. We are optimistic, though, that only few kinds of natural errors cannot be detected at all, because figure 1 shows that more than 75% of all errors can be found for large and clean datasets. We plan to inspect the remaining error types closely in the future in order to reveal which errors generally cannot be detected.

3. Discussion and Related Work

ED focuses on errors that distort the probability distribution of context-free productions. While these errors may only be a subset of all errors, we believe that they are very relevant for improving the usefulness of a corpus as a training resource for parsers, because probabilistic parsers usually condition the probability of a node's production fully or partly on the node label. ED can thus be seen as a means to clean a corpus from errors particularly harming parser performance. The abstract units, considered in this work, are defined as a context-free grammar, i.e. productions in the context of a parent node. But the method is not restricted to this definition of (c, f, p) triples. Hence, it can also be defined in terms of e.g. dependency relations.

There are several lines of related research. Dickinson and Meurers (2003) use the notion of *variation n-grams* — a sequence of word form tokens with different annotations in different occurrences in the corpus. The variations between n-grams are likely errors in the corpus. Their method is similar to ours in that potential errors need to be inspected by humans. However, in our case there is a measure which helps us to rank candidate errors. In their method, the context of potential errors is defined by (lexical) word form tokens, whereas we use syntactic categories rather than word forms. Kveton and Oliva (2002) show how errors can be detected in POS-tagged corpora. Their approach is based on searching for *impossible n-grams* in a corpus. They directly point to the occurrences of errors, but at the same time their method depends on hand-crafted definitions of relevant n-grams. An advantage of this method over ours is that it is in principle able to detect errors that occur systematically in certain contexts; however, it requires more linguistic knowledge.

Each corpus contains two types of linguistic information: *explicit* and *implicit*. The former is usually given in the documentation of a corpus, and the latter is based on the annotators' intuition encoded in the particular annotation; both can be erroneous. As pointed out before, usually explicit errors are easy to spot via clear rules. Spotting implicit errors requires at least the following: a description of the places where these errors may occur, a description of the context on that the errors depend, and a method for recognising potential errors in a context. Defining errors relative to implicit linguistic information to a great extent requires linguistic intuition and also experiments for verification. The advantage of our method is that it is not limited to certain definitions of errors and contexts. Moreover, the model generated in the ML stage of ED abstracts from language-specific details and thus allows training on a larger and better developed treebank of one language and applying the resulting model to a a treebank of a different language for which less training data is available.

Similar to the other methods for detecting errors, ED will be most useful in an interactive environment. We therefore plan to incorporate it into the CLaRK interactive annotation tool which will also allow changing error and context definitions easily.[5]

4. Conclusion

We have presented a method for detecting errors and inconsistencies in the structural annotation of treebanks. The method is based on the observation that the productions of nonterminals should behave consistently across all contexts in a corpus. We generalise from the output of a statistical test by applying machine-learning to features extracted from its output. The method performs well across different languages and sizes of corpora, and it seems to be equally applicable also to corpora that still undergo annotation.

5. References

Blaheta, D., 2002. Handling noisy training and testing data. In *Proceedings of EMNLP 2002*. Philadelphia, USA.

Brants, T., 1997. *The NeGra Export Format for Annotated Corpora*. Computerlinguistik, Univ. des Saarlandes.

Daelemans, W., J. Zavrel, K. van der Sloot, and A. van den Bosch, 2003. TiMBL: Tilburg Memory Based Learner, version 5.0, Ref. Guide. Technical Report 03-10, ILK.

Dickinson, M. and W. D. Meurers, 2003. Detecting inconsistencies in treebanks. In *Proceedings of TLT 2003*. Växjö, Sweden.

Kveton, P. and K. Oliva, 2002. (Semi-)automatic detection of errors in PoS-tagged corpora. In *Proceedings of COLING 2002*. Taipei, Taiwan.

Simov, K., G. Popova, and P. Osenova, 2001. *A Rainbow of Corpora: Corpus Linguistics and the Languages of the World*, chapter: HPSG-based syntactic treebank of Bulgarian (BulTreeBank). Munich: Lincom-Europa.

Telljohann, H., E. W. Hinrichs, and S. Kübler, 2003. *Stylebook for the German Treebank of Written German (TüBa-D/Z)*. Sem. für Sprachwiss., Univ. Tübingen.

Ule, T., 2003. Directed Treebank Refinement for PCFG parsing. In *Proceedings of TLT 2003*. Vaxjö, Sweden.

[5] Available at `http://www.bultreebank.org/clark`.