MSc Natural Language Processing - 2020/2021
UE 705 - Supervised Project

Corpus correction

Bibliographic report

Sutdents :
Margot GUETTIER
Kahina MENZOU
Papa Amadou SY

Supervisor :
Bruno GUILLAUME

December 2020

Contents

1 Introduction

For this tutored project we decided to focus on dependency annotations in corpora and more particularly on the annotation errors present in these corpora. Annotated corpora are great resources, especially for research in linguistics but also for computer scientists working in the field of natural language processing. The quality and thus the reliability of these morphosyntaxic annotations are of great importance. Nevertheless there are errors in these corpora, and, as Boyd et al[3] assert in their article, the presence of errors in linguistic annotations has been shown to create problems for the computational and theoretical linguistic uses of these corpora. Even the smallest errors can have significant impact on the linguistic usage of the annotations. Therefore, improving annotations and annotation schemes is a key element to improve the quality of these annotated corpora and their reliability.

Our project fits perfectly in this perspective. Indeed our goal is to implement an error detection system for corpora with dependency annotations. For this purpose we will use existing methods and possibly improve them, taking into consideration the positive and negative points that have already been raised by researchers interested in this problem.

In the remainder of this report, we will first carry out a state of the art on error detection methods. In the following part we will analyze the variation detection methods followed by the detection methods using parsers and finally we will present how to evaluate an error detection system.

2 Methodology

In corpora, the dependency annotation can be performed either automatically, using parsers, or manually when a better quality of annotation is desired. In this report we will focus on manually annotated corpora. Despite of these manual annotations, the presence of errors is inevitable. In this section, we will introduce different approaches for errors detection that we found in articles. The first important thing to notice is that all these methods can be divide into two categories. On the one hand approaches using variation detection, on the other hand method which detect errors by using parsers. We will first present the paper by Boyd et al [3] which is the basis of most other research using variation detection. In this part several variants of Boyd et al's approach will also be presented. We will close the presentation of this method by presenting a tool that has been developed according to the concept of variation detection. Then in a second part, we will introduce the methods using parser for errors detection.And finally, in a last part, we will introduce the evaluation methods of error detection systems.

2.1 Variation Detection

Boyd and al [3] use the method of variation detection developed for detecting errors in Part of speech annotation and constituency annotations. The method is based on variation detection, which consist on find identical n-gram in a corpus and compare their annotation. If the two n-gram are not annotated the same way, then we must have detected a potential error. Example:

(1) a.

je	le	vois
PRON	DET	VERB

‘I see him‘

b.

je	le	vois
PRON	PRON	VERB

‘I see him‘

In the example (1) we have twice the same sequence of word but part of speech tagging is different for each case. The method assume that one of the annotation is potentially wrong. The changing element is called the variation nuclei, and the word on the right and the one on the left are called right and left context[1]. The errors we can detect with the method are potential because we have to distinguish if it's actually an error or only a genuine ambiguity of the language. In order to adapt the method to dependency annotation the variation nuclei will be a pair of words which represent exactly one relation between a governor and a dependant. The relation is represented by a directed arc, to precise the direction, it is added to the dependency label a L if the governor is to the left and a R if it is to the right. There are two types of comparison between pairs of words: either in each pair, the two words are linked by a dependency relationship, or in one of the pairs, the two words are not linked by a dependency relationship. Boyd et al assign the label NIL to this non-relation. To improve the detection of errors and not detect genuine ambiguities contextualisation is important. Indeed context allows us to have a bigger segment to find errors and reduce the number of matches that we have if we only search the variation nuclei. There is three types of context:

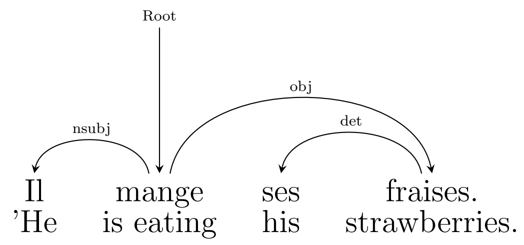
- The non-fringe heuristic context give one element of context around each word of the variation nuclei.
- The dependency context heuristic corresponds to the relation between the governor of the variation nuclei and one element of the rest of the sentence. Add a dependency context can reduce ambiguity when we compare two pairs of words.
- NIL[2] internal context heuristic, it is the context between the two words of the pair when they are not contiguous.

[1]A previous study conducted by Markus Dickinson [4] has shown that the immediate context is sufficient to detect errors with high precision.

[2]nil means 'zero, non-existent', so we assign the NIL label when two words in the same sentence are not related to each others. For example in the sentence 'I eat bread', *I* and *bread* are not related it would have a NIL label for this pair of words.

For the variation detection algorithm, on the one hand the system has to create a list with all the pairs of words with a dependency relation and assign them a label related to the kind of relation they have. The system also have to add to this list the words which are not in a dependency relation (limited to the boundaries of the sentence), in that case we assign the label NIL.

(2) a.



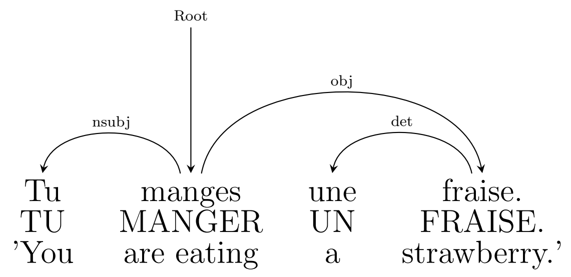
b. (Il, mange) ⇒ label nsubj-R
(mange, fraises) ⇒ label obj-L
(ses, fraises) ⇒ label det-R

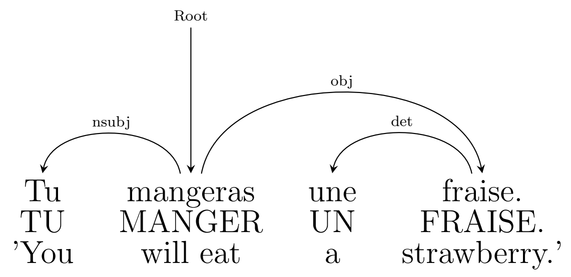
c. (il, ses)⇒ label NIL
(il, fraises) ⇒ label NIL
(mange, ses) ⇒ label NIL

If we follow the first step of the algorithm for the example (2a) we obtain (2b). Then we add the pair of words with a NIL relation (2c). Note that in a corpus the frequency of NIL relations is very high, but it is not necessary to build all of them. A NIL relation is useful only if the same pair of words appears with a non-NIL annotation elsewhere. This means that in (2c), the relation NIL between *il* and *ses* is useful only if the pair (*il*, *ses*) is linked by a relation somewhere else in the corpus.
Once we finished the list for the entire corpus, the second step of the algorithm is to determine if for one pair of words there is more than one label assign. If it is the case one of the annotations of this pair is a potential error.

2.1.1 Proposal of improvement

The research of Boyd et al [3] on the detection of annotation errors in corpus dependency has been repeated many times, either with the aim of developing tools as we will see later or to improve the method. It is this last point that we will discuss in this section. The main objective of the modifications we will present is to improve the recall and the precision of the variation detection method. In their article Marneffe et al [6] propose two modifications to limit the intrinsic variation of each word. The first proposal is to study dependency annotations by taking into account lemma (dictionary entry), instead of word-form (use of words in sentences, in context).

(3) a.


b.


If we analyze the two sentences (3a) and (3b) following Boyd's original method, never the relation *manges* → *fraise*, would be compared with *mangeras* → *fraise*, however it is just a change of tense, the context is identical so the dependent annotation should be identical. The use of lemmas will allow the comparison of these two pairs of words, since they are now identical *MANGER* → *FRAISE*. For the three languages (French, English, Finnish) they experimented on, the results for this first proposal are heterogeneous. Indeed, for French and English this method gives results close to the results obtained using word-form. However for Finnish which is a complex morphological language, this method generates too much loss of information and therefore a drop in recall and precision. In front of this impossibility to use lemmas for morphologically complex languages, they propose a second modification. This modification consists in a delezicalization, using morphological features instead of lemmas and word-form. For instance if we take the sentence from example (3a), instead of looking for pairs of words including the lemma *TU* we will look for pairs of words including a singular pronoun. For Finnish this method does indeed give better results than the use of word-form, however this method is still not as good as the original approach of Boyd et al.

2.1.2 ERRATOR: a tool for error detection

Based on the principle of variation detection, Guillaume Wisniewski [10] has developed a tool - Errator[3] - which allows the detection of errors in dependency annotated corpora. This tool uses the concept of pair of words but also the concept of surrounding context. For the continuation of the presentation of this method we will name sub-strings the combination of pair of word + surrounding context. This tool is based on the fact that if two identical sub-strings are annotated differently, then it is very likely that one of them is wrongly annotated.
In order to detect errors with the best possible accuracy, the sub-strings that will be compared must have as much context as possible.

(4) a. Every morning I go to school by bus with my sister and brother.
    b. When I get up I go to school by bus and I sing the whole way.

If we take as an example the two sentences (4a) (4b), we will have to take the biggest common part of these two sentences which is "I go to school by bus", this step is done thanks to the Generalized Suffix Tree tool. Then Errator extracts the

[3]https://perso.limsi.fr/wisniews/errator/

annotations corresponding to these sub-strings. If the annotations of the first sub-string is different from the annotations of the second sub-string then a potential error has been detected.

2.1.3 Variation detection weaknesses

As seen previously, variation detection is used to detect annotation errors in annotated corpora. In his article Markus Dickinson [5] presents the main problems encountered when one wishes to use the results of the variation detection for automatic error correction.
These problems can be split into two categories. Firstly the Uniform Non Majority cases and secondly the Non Uniform Non Majority cases. For the first case we distinguish three problems:

- If a variation which appears only twice with the same local context. Shown once incorrectly labeled and once correctly what we can do to distinguish is to look at the categories of the Part of speech of the nucleus.
- A second issue is that there is a lot of labels considered as a non-majority ones but in fact they are correct label.
- Sometimes there is an instance which varies in a context between two categories which are new, so here there is a problem of consistency of the conversion between those categories.

For the second case a problem arises when there is the set of occurrences where we have the same immediate context, the same POS tags, except that the nucleus has a set of different dependents so we must trust the rest of the context, i.e. broaden the context for better annotate.

2.1.4 Detecting and correcting annotation in the French TreeBank

As for the second part of this project we are going to work in particular on corpora in French. This study carried out on the French Treebank (FTB) will help us to target the errors we may encounter subsequently. Like the Penn TreeBank in English, in French the largest available resource of syntactically and Part of speech (POS) tagging annotated corpora is the French TreeBank. The article of Boudin et al [2] presents a series of experiments carried out on the detection and automatic correction of French Treebank errors. The FTB is the result of a supervised articles annotation project from ”Le Monde newspaper” and was built in a semi-automatic way. The corpus use for this experiment is made of 21.562 sentences $\Longrightarrow$ 628.767 words (tokens). In the selected corpus, 7.747 tokens which no label could be assigned either the label is missing, or the present label is not valid. In total, the generated corpus contains 2.090 sentences in which at least one unlabeled token is present. An automatic labeler is first applied to all texts. The outputs are then corrected manually for any errors made by the tool. Despite this, there are other human-made errors that may exist such as the lack of labels or the presence of empty XML elements. Regarding the choice of labels, the FTB was satisfied with the 13 labels of POS tags. However, a recurring problem arises: in French language, there are what are called compound words these words have always had a labeling problem. To remedy this problem a Tokenizer capable of detecting compound words is applied. Nevertheless, the existing methods have not yet reached a mature level. To reduce the losses of all compound words the FTB is limited to compound words whose lemma does not contain the space character. For instance in (5a), the word ”cerf-volant” is take into account because there is an hyphen between ”cerf” and ”volant”. While in (5b) the compound word ”pomme de terre” is not treated as a single word because there is no hyphen but spaces, even though semantically they work as a single entity.

(5) a. cerf-volant
    ‘kite‘

  b. pomme de terre
    ‘potato‘

Two complementary methods are used to correct annotation errors in the FTB:

1. The first method identifies unlabeled words and assigns them, the most frequent label for this token. This assignment can be ambiguous if for a given token several labels have similar frequencies. The solution is to assign only the most frequent tag if its frequency in the corpus is greater than the sum of the frequencies of the other candidate tags. Only tags with a frequency greater than 1 are used.

2. The second method uses n-gram variations in order to detect and correct annotation anomalies. The approach is based on the detection of labeling variations for the same n-gram of words. A n-gram variation means n-gram which contains a word annotated differently in another occurrence of the same n-gram. Thus constituting a variation nucleus. As a result, we can say that the more the contexts of a variation are similar, the greater the probability that it is an error.

For this second method a heuristic is proposed to correct some variations. This is as follows: we consider n-grams by decreasing size, then by decreasing number of occurrences. Candidates are selected for a correction according to two constraints:

- The presence of at least two lexical units.
- The presence of a variation, with no missing label, whose number of occurrences is strictly greater than the sum of the occurrences of the other variations. In fact, only the n-grams occurring at least three times are considered.

This last constraint also serves as a basis for proposing a correction. Indeed, the variation which validates the constraint is considered to be the correct sequence of labels. The number of corrected tokens increases when the minimum size of the processed n-grams is reduced.

(6) ,/PONCT l'/D **une/N** des/P plus/ADV $\Longrightarrow$,/PONCT l'/D **une/PRO** des/P plus/ADV
‘One of the most‘

The example (6), taken from the same article, illustrates the correction made with this heuristic. The word *Une* annotated as N is corrected to be PRO.

2.2 Detection of errors using parsers

In this part we will present the second kind of methods for errors detection which is detection by using parsers. Parsers have an advantage over human annotation, because even if errors are more frequent when annotating with parsers, the errors made are consistent, the same pair of words in the same context will not be annotated twice in different ways. Contrary to the human, which depending on its state of mind, its fatigue or even its understanding of the guidelines can annotate the same structure in several different ways. Three methods of using parsers to detect errors in corpora will be presented in this section.

2.2.1 Detecting errors

Error detection can be done using one or more parsers. In this part we will present you two detection errors method, and in the next part how these methods are used to correct those errors.
In their study, Volokh et al [9] used two differents parsers to proceed to the detection:

1. the graph-based MSTParser (Minimum-Spanning Tree Parser)
2. the transition-based MaltParser

The two parsers are trained on the data, then we parse the exact same data with these two parsers in order to replicate the Gold Standard. The accuracy will be very high (98% - 99%) because we train them on the data. The percentages which does not match the Gold Standard correspond either to structure not caught by the model, or a wrong annotation in the Gold Standard, is that case we are interested in. In that study they consider a relation in the Gold standard as potentially wrong if the predictions of both parsers have a prediction different from the Gold Standard.

This first method thus takes into account three elements, the predictions of the two parsers and the annotations of the Gold standard.

A second study also uses a parser system. Agrawal et al [1] uses a parser to detect potential errors in a corpus. It is a matter of comparing the parser predictions with the Gold Standard annotations, if they don't match then it's a potential error.

The final goal of these two studies is the correction of errors, in the next part we will detail the correction technique.

2.2.2 Correcting errors

Even if our project aims is the detection of errors, the two previous methods are used in order to correct errors. Thus in this part we will explicate how use parsers for errors correction.

After detecting errors with their the parsers, in order to correct them, Volokh et al [9] proposed to substitute the potential error in the Gold standard with the prediction of one of the parser (MSTParser), then they parse with a third parser (MDParser : Multilingual Dependency Parser). If this last parser predict the same as the new Gold standard, then we assume this error has been corrected.
How do they choose the parsers? The first two parsers are based on different approaches, this ways they tend to make different type of errors. To substitute the Gold Standard we use the MSTParser because the Maltparser and the MDParser are based on the same approach, it is to avoid bias.

This first correction method is fully automatic, so it may seem unreasonable to use it on a corpus that has been manually annotated. Indeed, this could reduce the quality of the corpus annotations. The second correction method proposed by Agrawal et al [1] is a semi-automatic alternative that can be use for manual annotated corpora. In this method the system does not completely replace the manual intervention of the linguist. Correcting the corpus of several thousand sentences manually is a very long and tedious task. The following correction approach aims to facilitate these manual corrections. As said before, parsers have an advantage, they are consistent in their decision, whether the decision is right or wrong, the same relationship will not be annotated twice differently. In the previous part we saw how Agrawal et al detects the errors, those errors will be validated or rejected by a manual annotator. In this way the work of the manual corrector is much less time consuming and simpler. In order to make this method feasible it is imperative that the parser used has a very good recall, so that the majority of errors will be part of the potential errors.

2.2.3 Parsers and active-learning

The proposed model by Rehbein et al [7] combines a generative and unsupervised method to estimate the reliability of annotators with active learning for the task of error detection in automatically annotated data. The model works as follows: during preprocessing, the model collects parse predictions from a committee of N dependency analyzers, then it creates two input matrices, one for dependency labels and one for attachments to form the two models of variational inference. After training, each model returns the posterior entropies for its respective decisions. Based on the entropy, it selects the next instances to annotate during active learning. The intuition behind this is that the labels or attachments with the greater entropy are probably incorrect. The annotator then enters the correct label. The information is used to update both the label and attachment matrices by randomly selecting one of the annotators and replacing the predictions of that annotator for the instance in question with the new prediction. Once the error correction is complete, we have to output the trees, either based on the predictions from the variational inference model (MACE-AL-TREE) or on the output from the Chu-Liu-Edmonds algorithm (MACE-AL-TREE-CLE). MACE-AL-TREE uses the final predictions of the variational inference model to select the most trustworthy labels and edges and combine them into a final tree. However, this does not necessarily result in a fully connected tree. The second approach uses the Chu-Liu-Edmonds algorithm, to select the highest scoring, well-formed tree from a weight matrix, where the weights are based on the votes from the parser committee, weighted by the competence estimates learned by the variational inference model (MACE-AL-TREE-CLE).

During the experiments, they used 5 parsers. The first experiment focuses on error detection in the German Universal Dependency tree bank which includes newswires, reviews and text from Wikipedia. The results were very low on the German Universal Dependency, indeed only 35,5% of errors have been detected. Then they used data from the English web treebank. This treebank includes data from five different web genres (reviews, weblogs, answers, emails, newsgroups). The table 1 below gives the results.

line no.		**answer LAS**	**ED prec**	**email LAS**	**ED prec**	**newsgroup LAS**	**ED prec**	**reviews LAS**	**ED prec**	**weblog LAS**	**ED prec**
	parser					*LAS for individual parsers*					
1	BIST_{Graph}	**85.03**		82.27		82.86		**86.73**		84.73	
2	Dense	84.57		**82.81**		82.43		86.41		84.41	
3	IMSTrans	84.99		82.21		81.94		86.19		84.75	
4	Malt	81.38		78.61		79.18		83.24		79.23	
5	RBG	85.01		82.23		**83.53**		86.41		**84.93**	
	# iterations					*MACE-AL-TREE*					
6	0	86.82	–	84.07	–	**85.72**	–	87.28	–	**86.98**	–
7	100	87.04	85.0	84.26	89.0	86.07	92.0	87.40	49.0	87.33	78.0
8	200	87.27	86.0	84.45	86.5	86.34	80.5	87.48	42.0	87.65	75.5
9	300	87.46	83.0	84.61	83.3	86.64	78.7	87.45	24.3	87.96	73.7
10	400	87.64	80.5	84.77	80.7	86.91	76.7	87.56	29.0	88.27	72.5
11	500	87.84	79.8	84.96	81.8	87.14	73.4	87.68	33.8	88.56	71.2
12	600	88.04	79.0	85.11	80.2	87.31	68.3	87.84	39.2	88.84	69.7
13	700	88.20	76.7	85.28	79.7	87.60	69.1	88.02	44.0	89.13	69.3
14	800	88.37	75.5	85.45	79.7	87.90	70.1	88.17	46.6	89.46	69.7
15	900	88.51	73.4	85.63	79.7	88.13	68.9	88.36	49.7	89.71	67.3
16	1000	88.69	73.1	85.79	79.2	88.41	69.2	88.54	52.7	89.97	67.3
	# iterations					*MACE-AL-TREE-CLE*					
17	0	**86.91**	–	**84.10**	–	85.67	–	87.47	–	86.87	–
18	500	87.86	74.2	84.93	76.8	87.08	72.4	88.19	59.6	88.49	72.8
19	1000	88.67	68.8	85.75	76.4	88.33	68.4	88.74	53.0	89.93	68.8
	# iterations					*Majority vote baseline*					
20	0	86.72	–	84.06	–	85.52	–	**88.05**	–	86.83	–
21	500	87.07	27.0	84.38	29.8	86.01	25.0	88.47	35.0	87.49	30.0
22	1000	87.35	24.4	84.59	24.2	86.44	23.7	88.91	35.8	88.01	26.7

Table 1: Performance for individual parsers and parse combinations based on the predictions of the VI model with AL (MACE-AL-TREE) and with CLE (MACE-AL-TREE-CLE) for generating the trees (simulation study; Labelled attachment score (LAS) and error detection (ED) precision)

The data was first separated into the five web genres. Each genre has been split into training data and test data. Then they ran an AL simulation for 1000 iterations, where in each iteration an instance is selected randomly and corrected according the gold[4]. Table 1 shows in its second section the parsing accuracies after increasing iterations of error correction (from 0 to 1000 where 0 is the output of MACE-AL-TREE with 0 iteration). In its third section, we have the details of the trees formed with the MACE-AL-TREE-CLE method. As we can see, there is not much difference between the two methods and it is difficult to say whether one method is better than the other. The differences in results could be explained by different initializations of the variational inference model.

2.3 Evaluation

The last question that arises concerning the methods presented above is the question of evaluation. How do we evaluate an error detection system when we do not know the number of errors present in the corpus. Several proposals have been made.

The evaluation of the automatic correction in the French TreeBank [2] is performed extrinsically by comparing the performance of different POS taggers in relation to the level of correction. In this study, two systems using maximum entropy models (MaxEnt) are chosen: version 3.0.4 of the Stanford POS Tagger [8] and the morpho-syntactic tagger of the ApacheOpenNLP suite.

1. The Stanford POS Tagger has been trained with a standard set of bidirectional lines on words and tags
2. ApacheOpenNLP suite has been trained with the default implementation which characterizes each word using features of the three preceding and following words. These traits are the prefixes and suffixes of four characters, the rudimentary information class of these characters (e.g. starts with a capital letter, is a number, is a symbol), the grammatical label and the surface form of the words.

[4]Either we have a human annotator who corrects annotations from parsers or we have a gold which does it.

To check the effectiveness of the methods via the two systems, those systems are trained and evaluated from the different correction levels of the FTB in this way:

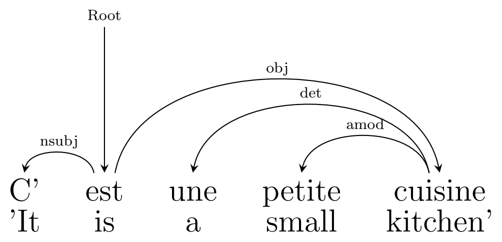
1. The *FTB + corr.1*: corresponds to the correction of the missing tags using method 1 (Identifying unlabeled words and assigns them, the most frequent tags for this token).

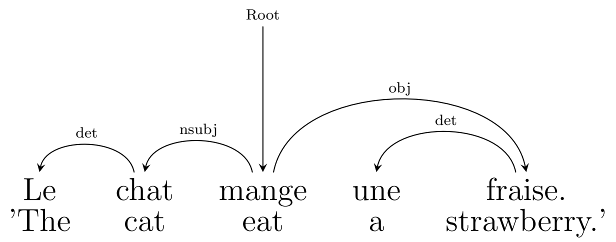
2. *FTB + corr.2* corresponds to the correction of annotation errors by the method of n-gram variations.

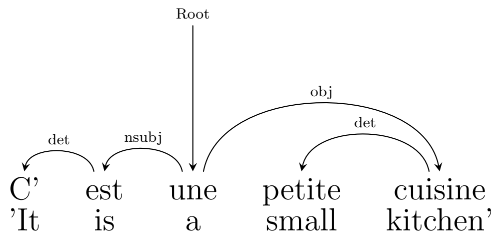
3. *FTB + corr.1/2* and *FTB + corr.2/1* correspond to the successive use of the two correction methods:*corr.1* then *corr.2* and vice versa.

The results that are presented in this study have all obtained by cross-validation in 10 strata. Three evaluation measures are considered relevant to our experiments: accuracy on tokens, accuracy on sentences, and accuracy on missing labels. The standard deviation (σ) of the scores over the 10 strata is also calculated. By applying the two systems (Stanford POS Tagger and ApacheOpenNLP suite) on tokens and sentences, we noticed that the precision score increases by following method 1 (Identifies unlabeled words and assigns them, the most frequent tags for this token) with *FTB+corr.1*. The score is reduced for the correction with n-gram (method 2) which means that this approach adds more errors. For the the missing labels the scores remains more or less stable.

To evaluate their system, that uses multiple parsers, Volokh et al [9] propose to replace the Gold Standard of their corpus by new annotations containing 100% errors. In order to be sure that all annotations are erroneous, for each sentence of the corpus they replace the annotations by the annotations of a sentence of the same length. They check that none of the new annotations are similar to the original annotations.

(7) a.


b.


c.


For instance we have the sentence (7a). We want to have 100% erroneous annotations. So we take a second sentence of same length (7b), and we replace the annotations of (7a) by the annotations of (7b) and we obtain (7c). As you can see all the annotations of (7c) are wrong. In this way they are aware of the number of errors and can see after application of their method what percentage of error is detected. Once they apply their method on their corpus, with annotations 100% erroneous, the percentage of errors detected is 45.9%. They insist on the fact that the errors were introduced randomly and may not correspond to the real errors that can be found in the annotated corpora.

3 Conclusion

For the first part of this project we researched which are the methods that are used today for error detection in dependency annotated corpora. Thus we could distinguish two main types of methods, the one that uses the concept of variation detection and the one that uses parsers. Among these methods, some of them supported fully automatic detection and correction of errors in the corpus. We note that for a manually annotated corpus, the use of a fully automatic correction tool is not recommended since it would degrade the results of a manual work of several months or even several years.

For the second part of this project, we will test in more detail the Errator tool which is already functional. Then we will implement one or more methods that we have discovered through these articles. Since we will experiment these methods on the French and English corpora of Universal Dependencies, which have been manually annotated, applying a fully automatic correction method, which as said before would degrade the data, would not be adequate. So we will only be interested in the error detection part. The final objective is to develop an error detection tool with the best accuracy possible for dependency annotated corpora. This will be followed by an implementation of an evaluation system for the proposed methods in order to see the limits, the strengths and the points to be improved in future work.

References

[1] Bhasha Agrawal et al. “An Automatic Approach to Treebank Error Detection Using a Dependency Parser”. In: vol. 7816. Mar. 2013, pp. 294–303. DOI: 10.1007/978-3-642-37247-6_24.

[2] Florian Boudin and Nicolas Hernandez. “Détection et correction automatique d’erreurs d’annotation morpho-syntaxique du French TreeBank (Detecting and Correcting POS Annotation in the French TreeBank) [in French]”. In: *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 2: TALN*. Grenoble, France: ATALA/AFCP, June 2012, pp. 281–291. URL: https://www.aclweb.org/anthology/F12-2021.

[3] Adriane Boyd, Markus Dickinson, and Detmar Meurers. “On Detecting Errors in Dependency Treebanks”. In: *Research on Language and Computation* 6 (Oct. 2008), pp. 113–137. DOI: 10.1007/s11168-008-9051-9.

[4] M. Dickinson. “Error detection and correction in annotated corpora”. In: 2005.

[5] Markus Dickinson. “Correcting Dependency Annotation Errors”. In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. Athens, Greece: Association for Computational Linguistics, Mar. 2009, pp. 193–201. URL: https://www.aclweb.org/anthology/E09-1023.

[6] Marie-Catherine de Marneffe et al. “Assessing the Annotation Consistency of the Universal Dependencies Corpora”. In: *Proceedings of the Fourth International Conference on Dependency Linguistics (Depling 2017)*. Pisa,Italy: Linköping University Electronic Press, Sept. 2017, pp. 108–115. URL: https://www.aclweb.org/anthology/W17-6514.

[7] Ines Rehbein and Josef Ruppenhofer. “Sprucing up the trees – Error detection in treebanks”. In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, Aug. 2018, pp. 107–118. URL: https://www.aclweb.org/anthology/C18-1010.

[8] Kristina Toutanova et al. "Feature-Rich Part-of-Speech Tagging with a Cyclic Dependency Network". In: *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*. 2003, pp. 252–259. URL: `https://www.aclweb.org/anthology/N03-1033`.

[9] Alexander Volokh and Günter Neumann. "Automatic Detection and Correction of Errors in Dependency Treebanks". In: Jan. 2011, pp. 346–350.

[10] Guillaume Wisniewski. "Errator: a Tool to Help Detect Annotation Errors in the Universal Dependencies Project". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018. URL: `https://www.aclweb.org/anthology/L18-1711`.

[11] Guillaume Wisniewski and François Yvon. "How Bad are PoS Tagger in Cross-Corpora Settings? Evaluating Annotation Divergence in the UD Project." In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 218–227. URL: `https://www.aclweb.org/anthology/N19-1019`.