Sprucing up the trees – Error detection in treebanks

Ines Rehbein
Leibniz ScienceCampus
Heidelberg/Mannheim
rehbein@cl.uni-heidelberg.de

Josef Ruppenhofer
Leibniz ScienceCampus
Heidelberg/Mannheim
ruppenhofer@ids.mannheim.de

Abstract

We present a method for detecting annotation errors in manually *and* automatically annotated dependency parse trees, based on ensemble parsing in combination with Bayesian inference, guided by active learning. We evaluate our method in different scenarios: (i) for error detection in dependency treebanks and (ii) for improving parsing accuracy on in- and out-of-domain data.

1 Introduction

Structural syntactic information is an important ingredient for many NLP applications. In recent years, the Universal Dependencies (UD) framework (Nivre et al., 2016) has become increasingly popular as a source for syntactically annotated training data, mainly for two reasons. First, the UD project provides a unified framework for multilingual applications and second, its annotation scheme encodes semantic information in a more transparent way, yielding better support for semantic applications.

The project already provides treebanks for over 60 languages, some of them, however, rather small. To obtain high parsing accuracies, a sufficient amount of training data is needed, preferably from different genres and domains. Since treebanking is a notoriously time-consuming task, most of the UD treebanks have been automatically converted from other frameworks and thus include some noise introduced in the conversion, in addition to the usual annotation noise that stems from inconsistencies in the human annotations. Due to limited funding, a full manual quality check is often not feasible. Therefore, methods that are able to detect errors in automatically predicted parse trees are of great value for data clean-up and might also be of use for domain adaptation settings and when dealing with low-resource languages.

We present such a method, aimed at the detection of parser errors in manually *and* automatically predicted parse trees. We evaluate our approach in two different scenarios, i) in an active learning (AL) setup where we try to detect and manually correct errors in existing treebanks, and ii) in a domain adaptation setting where we automatically improve parsing performance without manual correction. We make an implementation of our approach publically available.[1]

2 Model

Our error detection model is an adaptation and substantial extension of MACE-AL (Rehbein and Ruppenhofer, 2017) which combines a generative, unsupervised method for estimating annotator reliability (MACE) (Hovy et al., 2013) with active learning for the task of error detection in *automatically* annotated data. MACE-AL can be applied to any classification task and has been tested in two applications, POS tagging and Named Entity Recognition (NER) (Rehbein and Ruppenhofer, 2017). The model is, however, not applicable to tasks with structured output such as trees or graphs. In contrast to POS tagging or NER where we try to detect annotation errors in the predicted labels for individual tokens, when looking for errors in parse trees we have to deal with directed, labelled relations between nodes, and changing the relation or label between two nodes in the tree usually requires the adjustment of other attachment and labelling decisions in the same tree.

[1] MACE-AL-TREE can be downloaded from http://www.cl.uni-heidelberg.de/research/downloads/

Proceedings of the 27th International Conference on Computational Linguistics, pages 107–118
Santa Fe, New Mexico, USA, August 20-26, 2018.

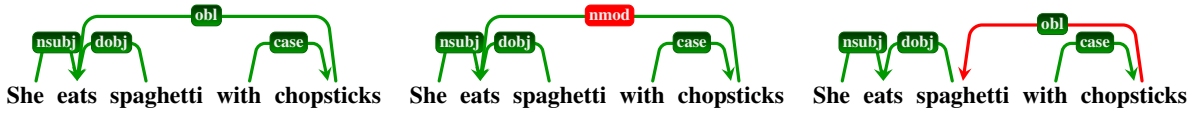

Figure 1: Correct tree (left), tree with label error (center) and tree with attachment error (right).

A high-precision method for error detection in automatically predicted trees, however, would be of tremendous use for many treebanking projects. Below, we first describe the approach of Hovy et al. (2013) and Rehbein and Ruppenhofer (2017) for unstructured data (§2.1) and then our extension of the model for the purpose of error detection in tree structures (§2.2).

2.1 MACE-AL: Error detection in unstructured data

MACE-AL combines variational inference (VI) with active learning (AL) to model the reliability of *automatically* predicted annotations. As input, it takes the output of a committee of classifiers (e.g. the output of N POS taggers) and uses *Bayesian inference* to learn which taggers' predictions are more trustworthy than others. The method is unsupervised, meaning that the distribution of the true labels $Y = \{y_1, y_2, ..., y_n\}$ is unknown. Instead, the model tries to approximate the posterior distribution over the set of unobserved random variables Y by a variational distribution $Q(Y)$, based on the observed data X (the predictions made by the classifier committee).

MACE-AL is based on MACE (Hovy et al., 2013), a tool for estimating annotator reliability in a crowdsourcing setting. MACE implements a simple graphical model where the observed annotations A are generated as follows. The (unobserved) "true" label T_i is sampled from a uniform prior, based on the assumption that the annotators always try to predict the correct label and thus the majority of the annotations should, more often than not, be correct. To model each annotator's behaviour, a binary variable S_{ij} (also unobserved) is drawn from a Bernoulli distribution that describes whether annotator j is trying to predict the correct label for instance i or whether s/he is just spamming (a behaviour not uncommon in crowdsourcing settings). If S_{ij} is 0, the "true" label T_i is used to generate the annotation A_{ij}. If S_{ij} is 1, the predicted label A_{ij} for instance i comes from a multinomial distribution with parameter vector ξ_j.

Rehbein and Ruppenhofer (2017) showed that MACE, while highly successful when applied to human annotations from crowdsourcing, is not able to outperform the majority baseline when the predictions have been generated *automatically* by a classifier committee. MACE-AL, however, addresses this shortcoming by combining variational inference with human feedback from active learning. Active learning is an approach to minimise human annotation effort by only selecting instances for labelling that provide useful information to the classifier. It has been shown that annotating only a small set of carefully selected instances not only reduces manual annotation effort but can yield the same accuracy as when training on a larger set of randomly selected instances (Settles, 2009).

The extended model, guided by human feedback, is now able to pick up on the signal and significantly outperforms the majority baseline as well as a query by committee strategy for error detection where the next instance for manual inspection is selected based on the entropy in the classifiers' predictions.

2.2 MACE-AL-TREE – An extension to structured data

To be able to apply the model to tree structures, we need to solve the following two problems. First, we need to track two different types of errors, namely *labelling* errors and *attachment* errors, and encode them in a meaningful way so that the variational model is able to learn from the data. Second, we need to collect the local predictions from the variational model and translate them back into a tree structure.

Error encoding We need to track *labelling* errors where a directed arc has been assigned the wrong label (Figure 1 (center)), and *attachment* errors where the relation between two nodes has been predicted incorrectly (Figure 1 (right)). We do this by training two separate variational models, one for the labelling decisions and the other one for the attachment relations between nodes.[2]

[2]Combining both into one model would blow up the number of possible labels and thus make the model unnecessarily hard to learn, and would also take too much time for training to be used in an active learning scenario.

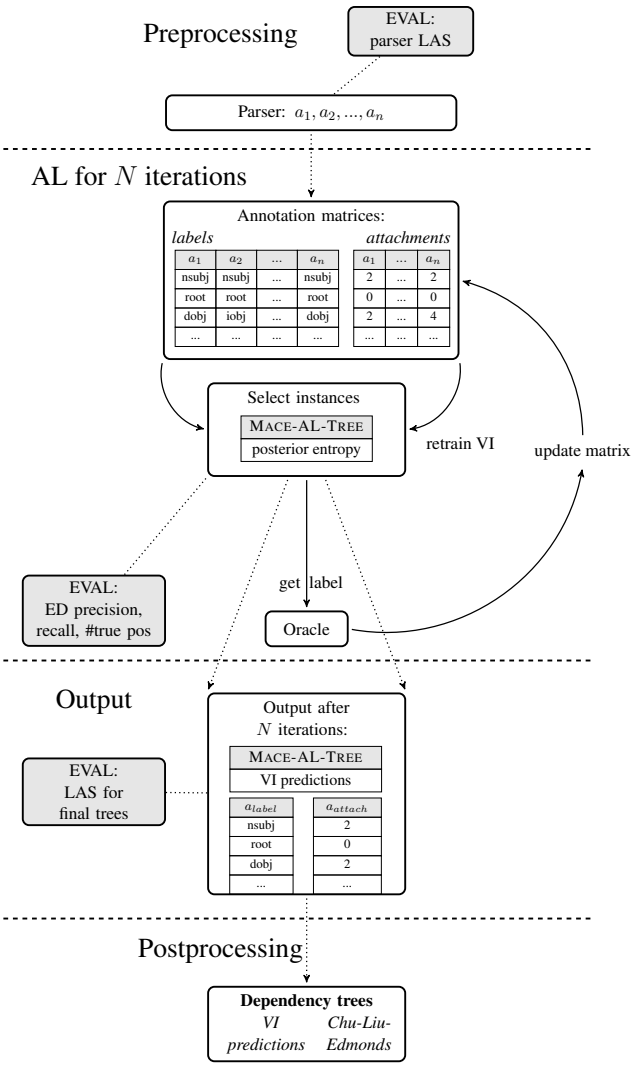

Figure 2: Error detection model with VI and AL (Error Detection (ED) precision: no. of True Positives / total no. of instances selected for error correction; True Positives: no. of instances selected for correction that are true errors; Recall: correctly identified errors / total no. of errors in the corpus).

Let's assume we have a dependency tree forest where each tree consists of a set of nodes V, one for each word in the sentence, and of a set of edge labels R that hold between the nodes. Let $L = l_1, l_2, ..., l_m$ be the set of all possible edge labels. $A = a_1, ..., a_n$ is the set of annotators, in our case the committee of dependency parsers used for obtaining the predicted parse trees. We then train two variational models where the first model aims at learning the reliability of each annotator with regard to the edge label predictions, and the second model learns the reliability of the attachment decisions for each of the parsers. The input to the *labelling* model for MACE-AL-TREE is a matrix with all labels predicted by the annotators (parsers) A_{1-n}, and the MACE-AL-TREE *attachment* model takes a matrix with the head information for each token, extracted from the parser output of the different annotators A_{1-n}.

After preprocessing the data to obtain the parser output trees, we transform the predicted trees as described above and train the variational inference model of MACE-AL-TREE separately on the label and attachment encodings (Figure 2). After training, each model outputs the posterior entropies for its respective decisions about the preprocessed input. We then alternate in extracting the label and attachment decisions with the highest posterior entropy, according to the variational model. The intuition behind this is simple: The labels or attachments with the highest entropy are most likely incorrect.

We either present the selected instances to the human annotator and request the correct annotation or, in a simulation, we extract the correct annotation from the gold tree. In the first setting, the human annotator is shown the sentence with the token in question highlighted in blue while and all the nodes that are predicted as potential heads of the token by any of the parsers are highlighted in red. The annotator

then enters the correct label *and* head for this token. 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.[3] Note that the update does not necessarily result in an increase in accuracy (or decrease in the number of errors) because the prediction may already have been correct to start with. After updating the annotation matrices, we start over and retrain the variational model on the updated annotations. To save time, we alternate in retraining the models so that each model is retrained after every other iteration. We can repeat this process as many times as we like.

Generating trees Once we are done with error correction, we want to output the trees. However, what we get from the variational model are local decisions for individual labels and edges, and it is not straight-forward to generate connected trees based on these decisions. We test two different methods for generating the output trees. In the first setting, MACE-AL-TREE, we simply use the final predictions of the variational inference model to select the most trustworthy labels and edges and combine those in a final tree. Please note that this does not necessarily result in a fully connected tree, a problem that also affects the output of many local, greedy parsers. Our second approach uses the Chu-Liu-Edmonds (CLE) algorithm (Chu and Liu, 1965; Edmonds, 1967) 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).

Overall workflow Figure 2 illustrates the workflow of our error detection model. During preprocessing, we collect parse predictions from a committee of N dependency parsers. Based on these predictions, we create the two input matrices, one for dependency labels and one for attachments, that we use to train the two variational inference models. Based on the posterior entropy from the variational inference model, we select the next instances to be annotated during active learning. After the instances have been corrected, we alternately retrain the variational inference model: in even iterations we retrain the labeling model and in odd runs the attachment model. This saves time over retraining both models in each iteration. We do, however, update both matrices in each iteration, based on the attachment and label annotations we get from the human annotator (the oracle).

Once we are done with active learning, we collect the annotations for the labels and the attachments and generate the output 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).

A key advantage of our model is that, while making use of the feedback from active learning, we do not have to retrain the parsers after each iteration, which would be infeasible due to the time requirements. Instead, we only train the parsers once, for offline preprocessing, before active learning starts.

3 Experiments

In our experiments, we use five different dependency parsers for preprocessing to predict the parse trees. We employ the neural BiLSTM parser (BIST$_{Graph}$) of Kiperwasser and Goldberg (2016) and the RBG parser (Lei et al., 2014), both graph-based, as well as the transition-based IMSTrans parser (Björkelund and Nivre, 2015) and the slightly outdated Malt parser (Nivre et al., 2006).[4] As the fifth model, we use our in-house implementation of the head-selection parser of Zhang et al. (2017) which also employs bidirectional LSTMs (Hochreiter and Schmidhuber, 1997) for learning the feature representations. We chose the parsers so that they cover a range of different parsing algorithms and approaches, as it has been shown before that the parsers' predictions, due to the inductive biases of the algorithms involved, can complement each other (McDonald and Nivre, 2007). All parsers use default settings as reported in the literature. For the BIST$_{Graph}$ parser, we selected the model with the highest LAS on the development set after 10 training iterations.[5] We run the Malt parser with the settings *arc-eager, liblinear* without any further optimisation. For all experiments, we remove the language-specific extended labels and only train the parsers on the universal dependency relations.

[3] Other update strategies such as always updating the predictions of the annotator with the lowest performance, or updating the predictions of all annotators did not yield the same increase in results.

[4] It has been shown that the most important success criterion for ensemble parsing is the diversity of the base parsers rather than model complexity or a high accuracy for the individual parsers (Surdeanu and Manning, 2010).

[5] We started with 30 iterations but noticed no further increase in LAS after the first 10 iterations.

genre	# trees	dev	test	train	# unlabelled sent.
answers	3,488	1,000	2,488	13,134	27,274
email	4,900	1,000	3,900	11,722	1,194,173
newsgroups	2,391	1,000	1,391	14,231	1,000,000
rewiews	3,813	1,000	2,813	12,809	1,965,350
weblogs	2,030	1,000	1,030	14,592	524,834

Table 1: Distribution in the UD-En web treebank (training set does not include data from the respective genre but contains all the remaining treebank sentences *excluding* the ones for this particular genre).

3.1 Error detection with active learning for manually annotated trees

The first experiment focusses on error detection in the German UD treebank which includes newswire, reviews and text from Wikipedia. Our goal is to test our error detection method in a realistic setup where we want to improve the quality of an existing dataset with minimal manual effort.

We run the task as an active learning experiment with a real human annotator in the loop who provides the correct labels for the selected error candidates. The annotator is a trained linguist with experience in linguistic annotation. The annotation task, however, is not easy due to the non-canonicity of the web data which includes many ungrammatical structures and does not conform to standard spelling conventions. Therefore, we asked a second trained linguist to adjudicate the disagreements between the original treebank annotations and the modifications carried out by the first annotator, using MaltEval (Nilsson and Nivre, 2008) to visualise the trees.

For preprocessing, we split the training part of the treebank into two sections and trained the five parsers on each section. Then we used the parsing models to predict the trees for the test set. Splitting the training data has the advantage that we now have 10 different versions of the test set (5 parsers x 2 training sets) that we can use to extract the input matrices for the active learning approach. A systematic and principled evaluation of the impact of data size and the number of annotators for the error detection approach is still outstanding.[6] Here we start with a set of 7 annotators (parsing models) for preprocessing as this setup yielded good results for POS tagging and NER (Rehbein and Ruppenhofer, 2017).[7]

Then we run 200 iterations of active learning where in each iteration the human judge had to assign the correct dependency label and head for the next token selected for annotation. After 200 iterations, we generated the parse trees for the corrected instances using the CLE algorithm weighted by the competence scores from the VI model (see §2.2) and compared them to the original treebank annotations.

The evaluation shows that out of the 200 instances from the test set inspected by the first annotator, the annotator did not agree with the original treebank annotation in 96 cases. Our second annotator agreed with the first judge on 71 of the 96 instances. For 15 instances, the annotator prefered the original annotations and for 8 instances, the second judge would have chosen a label or attachment different from the original treebank and from the first judge. This corresponds to an error detection precision of 35.5% (71/200) which should be considered as a *lower bound* as the instances where the second judge did neither agree with the first annotator nor with the original annotations might also be errors.

3.2 Error detection with AL on out-of-domain data

After validating our approach on the task of identifying errors in manually annotated trees, we now present an AL simulation study where we try to detect errors in *automatically* predicted parse trees. For this, we use the data from the English web treebank (Linguistic Data Consortium release LDC2012T13), originally annotated with constituency trees (Bies, 2012) and automatically converted to universal dependencies. The treebank includes data from five different web genres (reviews, weblogs, answers, emails, newsgroups) and provides us with a good test bed for error detection in a domain adaptation scenario. The simulation study also allows us to systematically evaluate different parameters in a controlled setting, without the need to employ human annotators.

[6] Preliminary experiments on English showed the same results with a smaller number of annotators for some datasets while for others a higher number of annotators was crucial to obtain a high error detection precision.

[7] To obtain a diverse parser committee, we selected the BIST and Dense models trained on the first section of the treebank, the Malt parser (arc-eager, lib-linear) trained on the second section, and both models for the RBG and IMSTrans parsers.

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 2: 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)

We preprocess the data as follows. First, we separate the data according to the five web genres and put all the trees for each genre into a genre-specific test set, pretending that no annotated trees for this genre are available for training. Instead, we use all remaining genres as training data. We further split the available test data for each genre into a development set, using the first 1,000 trees of each genre, and a test set including all the remaining trees (Table 1).

Table 2 (lines 1-5) shows the results for the individual baseline parsers on the different web genres. Please note that we did not optimise the parsers on the data and thus the comparison should be taken with a grain of salt. However, it is interesting to see that there is not one best parser but that the different parsers (excluding the 'vintage' Malt system) all yield results in the same range and the best performing parser varies depending on the dataset.[8]

Then we run an AL simulation for 1,000 iterations, where in each iteration one instance is selected according to our error detection model and the correct label and attachment decision is retrieved from the gold standard. Table 2 (lines 6-16) shows the parsing accuracies after increasing iterations of error correction (from 0-1000 where 0 is the output of MACE-AL-TREE with 0 iterations of active learning).

We first compare the results for MACE-AL-TREE with the ones for the best individual parser and observe an increase in LAS between 0.6% (for reviews) and more than 2% (for newsgroups and weblogs). This increase was obtained *without* any manual error correction. Using MACE-AL-TREE in this particular setup can be seen as a form of ensemble parsing where we use the predictions of the variational inference model to select the edges and labels of the parser output trees to be included in the final tree.

Now we want to evaluate the precision of the error detection method after increasing iterations of error correction via active learning. We calculate *error detection (ED) precision* as the number of error candidates that are real errors, divided by the number of instances selected for error detection.

Table 2 (lines 6-16) shows the LAS and ED precision for the different web genres. In the beginning, ED precision is quite high (between 78-92%) for nearly all genres, with the exception of the reviews.

[8]The results are not directly comparable to the ones from the SANCL shared task where the parsers were trained on the PTB while we use the data from the other web genres as training data.

line no.	*# iterations*	**ED prec**	**LAS**	*line no.*	*# iterations*	**ED prec**	**LAS**
1	0	–	88.8				
2	100	74.0	89.7	7	600	57.8	92.8
3	200	66.0	90.3	8	700	55.3	93.3
4	300	64.3	91.1	9	800	54.4	93.9
5	400	62.7	91.7	10	900	52.7	94.3
6	500	60.0	92.3	11	1000	50.1	94.6

Table 3: Results for error detection on data from the German TiGer treebank for increasing iterations of AL; trees generated with MACE-AL-TREE-CLE (simulation study; ED prec: error detection precision).

Here we start with an ED precision of 49% after 100 iterations. While for the other genres it becomes slightly harder with increasing numbers of iterations to detect new errors and ED precision slowly decreases, for the reviews the picture is rather mixed. After inspecting 1,000 instances we have a higher precision than in the beginning. We can only suspect that this might be an artefact of either idiosyncracies in the data or inconsistencies in the gold annotations.

While precision for error detection is high for all genres, the increase in LAS seems to vary across individual genres. This is because the number of tokens in the test sets also varies. The inspected 1,000 tokens correspond to 2,6% of the answers data, 2.2% for emails, 3,9% for newsgroups, 2,4% for reviews and 4.4% for the weblogs data. This means that we examined between 2.2 to 4.4% of the tokens in the data which resulted in an increase in labelled accuracy (LAS) in the range of 2 to 5%, which seems like a very good trade-off between time investment and reward in terms of data quality.[9]

Next, we compare the two different methods for generating the final parse trees. The first methods simply uses the predictions of the variational inference model (MACE-AL-TREE) and combines the highest-ranked labels and attachments into a tree. Our second method uses the Chu-Liu-Edmonds algorithm to generate well-formed trees (MACE-AL-TREE-CLE) that also allow for non-projectivity. Interestingly, as shown in Table 2 (lines 17-19), we notice only very small differences between the two methods and it is hard to say which method is superior as the differences in results might also be an effect of different initialisations of the variational inference model.[10] This observation is in line with the results from Zhang et al. (2017) who compared the output of their greedy local bi-LSTM parser with the same trees that had undergone additional postprocessing with the CLE algorithm. The authors also report only insignificant improvements for postprocessing German and Czech parser output.

Finally, we want to evaluate the importance of the generative unsupervised model as a component in our error detection model. Surdeanu and Manning (2010) emphasize that the most important success criterion for ensemble parsing is not the complexity of the model but the diversity of the baseline parsers. Therefore it is conceivable that we could get similar results using a simpler model, based merely on the predictions of the parsers *without* the variational inference model. To test this, we run 1,000 iterations of AL but select the next instance based on the entropy in the predictions of the baseline parsers instead of using the posterior entropy from the VI model (Table 2, lines 20-22) and generate the final trees running the CLE algorithm on the unweighted votes of the parser ensemble.

Table 2 shows that for 4 out of 5 genres we get substantially higher results using MACE-AL-TREE. After 1,000 iterations our model shows an increase in LAS that is between 1-2% higher than the one for the simpler model (lines 16, 19 and 22). The only outlier is the reviews subcorpus which also seemed to behave differently from the other genres with regard to error detection precision.[11]

[9] Improvements in LAS over the best individual parser after correcting 1,000 instances: answers 3.6%, emails 2.9%, newsgroup 4.8%, reviews 2.0%, weblogs 5.0%.

[10] As we use the model in an active learning setup where we need to keep the time requirements for training as small as possible, we only initialise the model once. The original model of Hovy et al. (2013) (MACE) allows for a higher number of restarts with random initialisation and then selects the best model.

[11] The lower error detection precision for the reviews in combination with higher LAS (line 22: ED: 35.8%, LAS: 88.91% after 1,000 iterations) is due to the fact that the initial predictions for the reviews based on the majority vote of the baseline parsers yielded a higher accuracy, which means that we start with a smaller amount of errors in the data. As ED precision is based on the overall number of errors in the set, we obtain a lower precision even though the model was able to detect more errors than MACE-AL-TREE.

core args		*coord & modifiers*		*clause structure*		*other*	
SB	42	CJ	28	RC	12	CM, CC,	
OA	23	CD	2	PAR	12	DM,	
PD	11	APP	8	OC	10	NK,	28
DA	8	MO	58	RE	6	NMC	
AG	2	MNR	23	RS	2	PNC	
PG	2	OP	11	ROOT	11	SVP	

head type	#	*head type*	#
modifiers	128	clauses	43
punctuation	77	core args	43
coordination	39	apposition	10
noun kernel	19	other	26

Table 4: Distribution of labelling errors (left table) and attachment errors (right table) in the German TiGer data found in 1000 iterations of AL error correction.

3.3 Error detection with AL on in-domain data

We showed that our approach yields high error detection precision in out-of-domain scenarios. We now want to test whether our method is also suited for in-domain settings where we start with a higher parser output quality, which means that errors in the data might be harder to find.

Again, we run a simulation study, but this time on German newspaper text from the TiGer treebank (Brants et al., 2002). We use the version from the SPRML 2014 shared task (Seddah et al., 2014) with 40,472 sentences in the training set and 5000 sentences for testing. We train five parsers[12] on the training portion of the data and predict parse trees for the automatically tagged test file provided by the shared task organisers. Parsing accuracies for the individual parsers on our testset with automatically predicted POS and morphological tags are in the range of 84.2-90.3% UAS and of 81.1-87.6% LAS.

We then run an AL simulation on the first 500 sentences in the testset, with 1000 iterations of active learning. In each iteration we select one instance and replace its head and label with the gold information. Again, it is not guaranteed that each modification will improve results as we might select instances which, according to the VI model, are already predicted correctly. Table 3 shows results for increasing iterations of error correction with AL, and error detection (ED) precision and LAS.

We can observe the same trend on the German in-domain data as we saw for the English out-of-domain data. LAS for the generated output trees based on the predictions of the parser ensemble with CLE (Table 3, line 1), weighted by the competence score from the VI model, is significantly higher with 88.9% than the score for the best individual parser (87.6% LAS). Looking at increasing numbers of AL iterations (lines 2-11), we also see a high error detection (ED) precision, starting with 70% and slowly decreasing as it gets harder to find new errors. After 1000 iterations, we still have an error detection precision of $> 50\%$, meaning that every second instance that we inspect is a real error. In terms of accuracy, running 1000 iterations of error correction on the TiGer subcorpus translates into an increase in LAS from 88.8% to 94.6% while keeping the number of instances for manual correction reasonably low.

3.4 Error analysis

Having established that our model is able to detect errors in manually and automatically annotated parse trees on in-domain and out-of-domain data, we would now like to learn more about the types of errors our model is able to detect. We thus evaluate the output trees generated from the German newspaper corpus (Section 3.3) *before* and *after* running 1000 iterations of AL for error correction.

Overall, our error correction resulted in 385 changed attachment decisions and in 298 modified dependency labels. In 167 cases, both label and attachment have been changed. Looking only at the label errors, we can see that the errors are distributed over 25 different dependency labels. The most frequent ones concern labels for core arguments (subject (SB), direct and indirect object (OA/DA), predicate (PD) and genitive attributes (AG/PG)) that are, due to the semi-free German word order and ambiguity arising through case syncretism, one of the major sources for parsing errors.

Other frequent labelling errors include the distinction between verb and noun attachment for PPs (modifier (MO)/noun modifier to the right (MNR)). The distinction between modifying (MO) and obligatory PPs (OP) is another frequent error that the model was able to detect. Our model also finds errors concerning coordination (CJ, CD) and clause structure, such as relative clauses (RC), parenthesis (PAR), clausal objects (OC), repeated element (RE) and reported speech (RS).

[12]For German, we used the IMSTrans parser, the head-selection parser and the BIST$_{Graph}$ parser that we also used in the previous set of experiments. In addition, we also use the transition-based biLSTM parser of (Kiperwasser and Goldberg, 2016) (BIST$_{Trans}$) and the RBG parser (Lei et al., 2014).

# unlabelled sentences	**LAS answer**	**LAS email**	**LAS newsgroup**	**LAS reviews**	**LAS weblog**
Results without self-training (best individual parser)					
	85.03	82.81	83.53	86.73	84.93
Baseline 1: self-training (best individual parser)					
5,000	85.11	83.40	83.23	87.19	85.14
10,000	85.33	83.25	84.06	87.06	85.32
15,000	85.43	83.37	83.62	87.38	84.82
20,000	85.26	83.25	82.52	87.01	85.27
25,000	–	83.46	82.56	87.50	85.79
Baseline 2: self-training (agreement betw. 2 parsers)					
5,000	85.36	82.58	83.38	87.33	85.80
10,000	–	82.65	82.56	87.28	86.22
15,000	–	82.53	83.34	87.31	85.57
20,000	–	83.16	83.64	87.62	85.26
25,000	–	83.09	82.82	87.74	85.04
MACE-AL-TREE *(with self-training)*					
5,000	85.77	83.07	83.93	87.39	86.27
10,000	85.89	83.71	84.25	87.82	86.28
15,000	85.74	83.85	83.83	87.54	86.08
20,000	86.09	83.51	84.41	87.93	86.26
25,000	–	**84.11**	85.09	**88.04**	86.16
MACE-AL-TREE *(without self-training)*					
	86.82	84.07	**85.72**	87.28	**86.98**

Table 5: Baselines and results for self-training on the different web genres.

Looking at the attachment errors that have been detected by our model (Table 4, right), we find that the most frequent head types that have been re-attached are modifiers (MO, MNR). This not only includes PPs but also adverbal and adjectival modification. The next frequent error type are incorrectly attached punctuations. This, however, is not an error type we are concerned with and it might make sense to exclude punctuation from the error correction.[13] More interesting attachment errors include coordination (CJ, CD), parenthesis (PAR), apposition (APP) and again the core arguments (OA, AG, DA, SB, PD).

This shows that our model is not biased toward a small set of specific error types but is able to detect frequent parsing errors that are well-known from the literature (Kummerfeld et al., 2012).

3.5 Domain adaptation with self-training

In the final set of experiments we want to test whether we can use our method to automatically correct parser output for self-training, to improve parsing accuracy for out-of-domain data. As before, we use the English web treebank, split into different genres as described above. Again, we pretend that no annotated training data for the different genres are available. This time, however, we make use of the supplementary raw text data for the five web genres (Table 1) that were provided for the SANCL shared task (Petrov and McDonald, 2012). The data is segmented into sentences and pre-tokenised. For preprocessing, we use UDPipe (Straka and Straková, 2017) with the pretrained models from the CoNLL-2017 shared task.[14] We also remove unlabelled sentences with a length > 60 tokens.[15]

We parse the additional data using the following settings. As our first baseline, we use the parser that achieved the best LAS on this particular genre for preprocessing. For our second baseline, we apply the best two parsers for each genre to parse the raw text and then select only those trees for self-training where the two parsers agree (excluding punctuation). In the last setting, we use MACE-AL-TREE (but without active learning) to generate the parse trees for each bin of automatically parsed text, based on the predictions of the parser ensemble.

We then use the output of each setting and combine the bins with the original training data for each genre, resulting in new training sets of increasing sizes. Then we train the BIST$_{Graph}$ parser on the new data and select the best model, based on the results on the development set after 10 iterations of training.

[13] In the original constituency-based TiGer treebank, all punctuation marks are attached to a virtual root node and have been reattached during the conversion to dependencies. This results in many arbitrary attachment decisions which makes it hard for the parser to learn these attachments.

[14] The models are available from `https://lindat.mff.cuni.cz/repository/xmlui/handle/11234/1-1990`.

[15] For the answers where we only have 27,274 raw sentences, this results in only 4 samples after removing long sentences, and only one sample for the agreement-based self-training (baseline 2).

Table 5 reports results for the three different settings. Looking at the self-training results where we retrain the BIST$_{Graph}$ parser on the predictions of the best individual parser, we can see mostly small improvements but sometimes also a decrease in LAS. The results for baseline 2 are also a bit mixed. When adding only 5,000 additional sentences, we obtain slightly higher results as compared to baseline 1. When adding more data, however, it shows that this improvement is not stable and sometimes results are lower than for the first baseline. A possible explanation is that the agreement-based selection strategy favors shorter sentences (as those are the ones where the two parsers tend to agree more often), and that we thus only add less complex structures that are not very informative for the parser. In addition, we introduce a bias in the training set that also might have a negative effect on results.

The third setting, however, clearly outperforms our baselines and shows a significant increase in parsing accuracy. Nonetheless, when comparing the results for MACE-AL-TREE *with* self-training to the results *without* self-training we see that our self-training approach fails to beat the results obtained on the original data on three web genres. We only observe improved results on the emails and on the reviews. These two genres are also the ones that have the largest amount of unknown words which might explain why self-training is more promising for emails and reviews than for the other three genres, as Reichart and Rappoport (2007) have shown that the number of unknown words can be a good indicator for the potential benefit from self-training.

4 Related work

Most studies on error detection in treebanks have focussed on finding errors in manual annotations (Dickinson and Meurers, 2003; Ule and Simov, 2004; van Noord, 2004; Dickinson and Meurers, 2005; Agrawal et al., 2013). Dickinson and Meurers (2003; 2005) proposed the use of *variation n-grams* to detect inconsistencies in manually annotated constituency treebanks and Boyd et al. (2008) extend this line of work to dependency trees. This approach, however, only works for manually annotated treebanks while for automatic parses where the errors are consistent and systematic, looking for variation in the predictions will not be very helpful. Volokh and Neumann (2011) address the latter problem and use two different parsing models to reproduce the annotations in the *training data*. They consider an instance to be erroneous if the two parsers agree in their predictions for the attachment of the token but disagree with the gold standard. The authors report a high precision for automatic error correction. However, their method only addresses attachment errors but does not handle label errors.

Work on predicting *automatic* parser errors includes Dickinson and Smith (2011) who develop a grammar-based method for error detection. They use ngrams extracted from automatic parses and weigh them by comparing them to the rules in a small gold grammar. Follow-up work (Dickinson and Smith, 2017) builds on this approach and uses parse simulations to extend the training grammar.

Relevant for our work are also studies on parse accuracy prediction. Ravi et al. (2008) predict parser performance for data from new domains. Our error detection method can be applied to automatic parses in a domain adaptation setting and can not only predict the accuracy of the parses but also improve the accuracy of the parser output. Our work can be seen as a combination of semi-supervised and ensemble-based methods and is thus similar in spirit to Søgaard and Rishøj (2010) who use tri-training in combination with self-training for dependency parsing and report an increase in labelled attachment score (LAS) for different languages in the range of 1.4 to 2.6%. Their self-training experiments, however, were based on a much larger amount of additional data (100,000 sentences vs. up to 25,000 sentences in our self-training experiments). It would be interesting to see whether more unlabelled data can further improve results, as suggested by the results for MACE-AL-TREE with self-training.

5 Conclusions

We presented a method for error detection in treebanks, based on the combination of (i) ensemble parsing, (ii) an unsupervised generative inference model, and (iii) human guidance from active learning. We validated our model on the task of error detection in manual and automatic annotations using active learning and showed that we can identify parse errors with precisions in the range of 35% to over 90%. Our approach can also be used to improve the quality of automatic annotations on out-of-domain data, without any human feedback, where we obtained gains in accuracy of up to 2% and more.

References

Bhasha Agrawal, Rahul Agarwal, Samar Husain, and Dipti M. Sharma. 2013. An automatic approach to treebank error detection using a dependency parser. In *Proceedings of the 14th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, CICLing'13, pages 294–303.

Ann et al. Bies. 2012. English Web Treebank LDC2012T13.

Anders Björkelund and Joakim Nivre. 2015. Non-deterministic oracles for unrestricted non-projective transition-based dependency parsing. In *Proceedings of the 14th International Conference on Parsing Technologies*, pages 76–86, Bilbao, Spain, July. Association for Computational Linguistics.

Adriane Boyd, Markus Dickinson, and Detmar Meurers. 2008. On detecting errors in dependency treebanks. *Research on Language and Computation*, 6:113–137.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. 2002. The TIGER treebank. In *Proceedings of the Workshop on Treebanks and Linguistic Theories*, TLT'02, pages 24–41.

Yoeng-Jin Chu and Tseng-Hong Liu. 1965. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400.

Markus Dickinson and Detmar Meurers. 2003. Detecting inconsistencies in treebanks. In *The Second Workshop on Treebanks and Linguistic Theories*, TLT.

Markus Dickinson and W. Detmar Meurers. 2005. Prune Diseased Branches to Get Healthy Trees! How to Find Erroneous Local Trees in a Treebank and Why It Matters. In *Proceedings of the Fourth Workshop on Treebanks and Linguistic Theories*, TLT 2005, pages 41–52, Barcelona, Spain.

Markus Dickinson and Amber Smith. 2011. Detecting dependency parse errors with minimal resources. In *Proceedings of the 12th International Conference on Parsing Technologies*, IWPT, pages 241–252.

Markus Dickinson and Amber Smith. 2017. Simulating dependencies to improve parse error detection. In *Proceedings of the 15th International Workshop on Treebanks and Linguistic Theories*, TLT15, pages 76–88.

Jack Edmonds. 1967. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computing*, 9(8):1735–1780, November.

Dirk Hovy, Taylor Berg-Kirkpatrick, Ashish Vaswani, and Eduard Hovy. 2013. Learning whom to trust with MACE. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT'13, pages 1120–1130, Atlanta, Georgia, USA.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics (TACL)*, 4:313–327.

Jonathan K. Kummerfeld, David Hall, James R. Curran, and Dan Klein. 2012. Parser showdown at the Wall Street Corral: An empirical investigation of error types in parser output. In *The 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL '12, pages 1048–1059.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi S. Jaakkola. 2014. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 1381–1391.

Ryan McDonald and Joakim Nivre. 2007. Characterizing the errors of data-driven dependency parsing models. In *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, EMNLP-CoNLL'2007, pages 122–131.

Jens Nilsson and Joakim Nivre. 2008. MaltEval: An evaluation and visualization tool for dependency parsing. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation*, LREC'08, Marrakech, Morocco, may. European Language Resources Association (ELRA).

Joakim Nivre, Johan Hall, and Jens Nilsson. 2006. MaltParser: A data-driven parser-generator for dependency parsing. In *Linguistic Resources and Evaluation*, LREC'2006, pages 2216–2219.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal dependencies v1: A multilingual treebank collection. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France, may. European Language Resources Association (ELRA).

Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web. Notes of the First Workshop on Syntactic Analysis of Non-Canonical Language (SANCL).

Sujith Ravi, Kevin Knight, and Radu Soricut. 2008. Automatic prediction of parser accuracy. In *Empirical Methods in Natural Language Processing*, EMNLP'08, pages 887–1575.

Ines Rehbein and Josef Ruppenhofer. 2017. Detecting annotation noise in automatically labelled data. In *Proceedings of the 55th annual meeting of the Association for Computational Linguistics*, ACL'17, Vancouver, Canada.

Roi Reichart and Ari Rappoport. 2007. Self-training for enhancement and domain adaptation of statistical parsers trained on small datasets. In *The 45th Annual Meeting of the Association of Computational Linguistics*, pages 616–623.

Djamé Seddah, Sandra Kübler, and Reut Tsarfaty. 2014. Introducing the SPMRL 2014 shared task on parsing morphologically-rich languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland, August. Dublin City University.

Burr Settles. 2009. Active learning literature survey. Computer sciences technical report, University of Wisconsin–Madison.

Anders Søgaard and Christian Rishøj. 2010. Semi-supervised dependency parsing using generalized tri-training. In *Proceedings of the 23rd International Conference on Computational Linguistics*, COLING '10, pages 1065–1073.

Milan Straka and Jana Straková. 2017. Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 88–99, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Proceedings of the North American Chapter of the Association for Computational Linguistics Conference (NAACL-2010)*, Los Angeles, CA, June.

Tylman Ule and Kiril Simov. 2004. Unexpected productions may well be errors. In *The Fourth International Conference on Language Resources and Evaluation*, LREC'04.

Gertjan van Noord. 2004. Error mining for wide-coverage grammar engineering. In *The 42nd Annual Meeting of the Association for Computational Linguistics*, pages 446–453.

Alexander Volokh and Günter Neumann. 2011. Automatic detection and correction of errors in dependency treebanks. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 346–350.

Xingxing Zhang, Jianpeng Cheng, and Mirella Lapata. 2017. Dependency parsing as head selection. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, EACL'17, pages 665–676.