WS2021 Statistical Dependency Parsing Report: Transition-based Dependency Parser

Ching-Yi,Chen

01.Mar.2021

1 Introduction

The report illustrates the implementation of **ArcEager** Transition-based Dependency Parsing. It is the transition system for projective dependency parsing, which capable of processing English and German data. The feature engineer in this parser is based on two research papers[2][3]. There are **single-word features, word pair feature, three-word feature**, and **distance feature**. Besides, **morph features** specific to German is also used inn the parser.

The parser is modeled by **averaged-perceptron**. The detailed implementation is followed by the instruction of SDP course in University Stuttgart[1]. The goal of the experiment is to examine the effect of each feature respectively. The experiments are evaluated by Unlabeled Attachment Score(UAS). Overall, in this report, you will see how the extended feature helps the parser to make progress and also the detailed implementation of the transition-based parser.

2 Methodology

2.1 Pipeline

To train a parser, I firstly build an oracle parser. The next step is to feed the correct tree to the oracle parser. Oracle parser will do feature extraction. Then, feeding features to the model. While the model is trained successfully(in my case, the success threshold indicated that training accuracy is higher than 95%), the decoder will operate based on the trained model. In other words, the parser will search for the best structure in the space of all possible structures.

As for the validation and testing process, when the parser gets the new data, it will extract features and feed them to the trained-model. Then the model will give the parser score matrix. Eventually, during the decoding process, ArcEager transition-based parser will choose a reasonable transition according to the score matrix.

2.2 Decoding: Arc-eager

I used ArcEager as the decoder. Compared with ArcStandard, ArcEager is not strictly bottom-up, it can find arcs easier than arc-standard. Thus, ArcEager is able to reach the terminal state even under the situation that a certain token misses its head. From the perspective of transitions type, ArcEager has an additional transition `RIGHTARC` that removes items (with head) from the stack. The `RIGHTARC` transition is also different, in ArcEager, `RIGHTARC` shifts the item onto a stack instead of removing it from the buffer.

However, both ArcStand and ArcEager might cause the **ambiguity** that more than one transition sequence leads to the same tree.[1] There are four transitions in the ArcEager: `LEFTARC`, `RIGHTARC`, `REDUCE`, `SHIFT`.

2.3 Feature Engineer

To guide the decoder(ArcEager) and find the optimal transition, the parser needs to extract features from the trees and train a model to assign scores for them. Representing as the information for the model to make a prediction, feature templates play a significant role in determining the performance of the parser.

Table 1 illustrates the **feature template** in the parser. I followed the template from[2], there are single-word, word pairs, and three-word features. Additionally, based on the paper[3], which improves the accuracy of parser through rich non-local features. The **distance** feature in the parser is computed by combining the distance between the top of the item in the stack and the first item in the buffer to certain feautre(e.g. pos-tag). Moreover, to tackle the morph information in German, the **morph** features are also added into the feature template.

2.4 Machine learning model: Average-Perceptron

After extracting the feature template, the next step is to train a model and then compute scores for further decoding. Average-perceptron is the model behind the parser. The input of the model is configuration. There are four transitions in ArcEager, which means four classes. Thus, this is a multi-class classification task. After the model finishes training, the model will calculate the weight. Weight is a matrix of class vectors for every input. Compared with perceptron, the average perceptron will average all weight vectors during training. Therefore, the average perceptron will have generalization.

Type	Template
Single-word (f1)	S[0]-form,pos; S[0]-form; S[0]-pos; B[0]-form,pos; B[0]-form, B[0]-pos; B[1]-form,pos; B[1]-form, B[1]-pos; B[2]-form,pos; B[2]-form, B[2]-pos;
Word pair (f2)	B[0]-pos+B[1]-pos+B[2]-pos; S[0]-pos+B[0]-pos+B[1]-pos; h(S[0])-pos+S[0]-pos+B[0]-pos; S[0]-pos+ld(S[0])-pos+B[0]-pos; S[0]-pos+rd(S[0])-pos+B[0]-pos; S[0]-pos+B[0]-pos+ld(B[0])-pos;
Three-word (f3)	B[0]-pos+B[1]-pos+B[2]-pos; S[0]-pos+B[0]-pos+B[1]-pos; h(S[0])-pos+S[0]-pos+B[0]-pos; S[0]-pos+ld(S[0])-pos+B[0]-pos; S[0]-pos+rd(S[0])-pos+B[0]-pos; S[0]-pos+B[0]-pos+ld(B[0])-pos;
Distance	S[0]-form+dist; S[0]-pos+dist; B[0]-form+dist; B[0]-pos+dist; S[0]-lemma+B[0]-lemma+dist; S[0]-pos+B[0]-pos+dist;
Morph (German)	S[0]-morph; B[0]-morpp; B[0]-pos,morph; S[0]-morph+B[0]-morph; S[0]-morph+B[0]-pos,morph;

Table 1: Feature Template

3 Experiment

3.1 Dataset Statistics

The experiments conducted using English and German Treebank data. Data split into training, development, and testing set. The statistical distribution of data shows in Table 2.

Number of sentences			
Treebank	train	dev	test
English	30060	1083	1382
German	40472	5000	5000

Table 2: Data Statistics

3.2 Experiment Setup

The baseline model is **average perceptron with the single-word feature**. Each model is trained for 15 epochs. There are two experiments in the report. Firstly, to investigate the **effect of each type of feature**, I incrementally added **one** type of feature. Eventually, the model will be trained on all extended features. Secondly, for German Treebank, to see the **effect of morphological features**, I made an experiment on the **presence** and **absence** of a morphological feature.

4 Result

The result evaluated on the development data using **Unlabeled Attachment Score(UAS)**. In Table 3, the performance of all extended features expectantly is the best. What is worth noticing is that after we added **f2 (word-pair feature)** to the feature template, both English and German parser make significant progress(+3%). Although the English parser steadily making progress while adding more types of features. German parser did not improve a lot even after adding the feature template with all extended features.

However, Table 4 tells us if we do not add the **morphological feature** for the German parser, it will perform **badly** even it has all extended features (f1+f2+f3+distance).

Dev(UAS)	English	German (+ morph feature)
f1	79.83	80.9
f1+f2	84.54	83.47
f1+f2+f3	86.9	84.79
f1+f2+f3+distance	89.01	84.80

Table 3: **Feature Experiment**: f1 means single word feature, f1+f2 means single word and word pairs feature, f1+f2+f3 means single word, word pairs, and three words feature, f1+f2+f3+distance means all extended feature plus distance feature

5 Discussion and Error Analysis

Based on the observation of the result, I learned that my parser performs worse in German TreeBank. However, morphological features help the parser to make a lot progress(+20% accuracy). Compared with English, **German has more morphological grammars** that need to take into account, for instance, the

Dev (UAS)	German
all extended feature + distance +morph feature	84.80
all extended feature + distance	64.46

Table 4: **Morph Feature Experiment**: complete extended features with or without morph feature

oblique Cases (dative and genitive).

Thus, instead of focus on pos-tag and form features, making use of extended morph features might be an option to improve the German parser. The experiment demonstrated the importance of the morph feature to train the German parser. As for the English parser, although the performance is getting better while adding additional features, it still has room to be improved compared with the parser in other research papers. One option would be to add more **non-local features** such as valency feature, unigram information, or third-order feature.[3]

6 Conclusion and Future

In conclusion, this report illustrated the implementation of ArcEager Transition-based Dependency Parsing, and the effectiveness of feature extraction. Especially for the morphological feature for German Treebank. For future work, if I continuously focus on **feature engineer**, I will try to consider more non-local features. It would also be interesting to investigate the effect of morphological features for the German parser. If focusing on the **parser architecture**, I could try to do the non-projective parser by adding a swap transition. Additionally, for **neural network parser**, no more manual feature engineer is needed, and I will try to use stack-pointer networks as advanced implementation.

References

[1] Agnieszka Falenska and Xiang Yu. "Teaching Material on Statistical Dependency Parsing Course". In: *Institute for Natural Language Processing* University Stuttgart (2020).

[2] Joakim Nivre. "Algorithms for Deterministic Incremental Dependency Parsing". In: *Computational Linguistics* 34.4 (2008), pp. 513–553. DOI: 10.1162/coli.07-056-R1-07-027. URL: https://www.aclweb.org/anthology/J08-4003.

[3] Yue Zhang and Joakim Nivre. "Transition-based Dependency Parsing with Rich Non-local Features". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 188–193. URL: https://www.aclweb.org/anthology/P11-2033.