EDAN20 Assignment 6
Dependency parsing

Matilda Andersson, ma4748an-s

October 2020

1 Introduction

The purpose of this laboratory report is to describe and comment on the transition-based dependency parser constructed using machine learning techniques. The focus of this assignment was to write a program that parsed a sentence with a known dependency graph, then to extract features from the parsing actions, and training a classifier to apply on a test corpus. The data used for the assignment was Swedish Talbanken corpus from the Universal Dependencies, which also was the corpus used for the previous assignment.

2 Process description

The following subsections contain a more in depth description of the process for creating the transition-based dependency parser.

2.1 Transition parser

After the process of reading and cleaning the data we started working on the transition parser. The goal here was to create the gold standard graph for each sentence with a projective dependency graph, where gold standard parsing consists of the sequence of parsing actions, left-arc, right-arc, shift, and reduce that produces the manually-obtained graph. In order to derive the action sequences that produces the manually-parsed sentences from an annotated corpus, we used the oracle function which takes a stack, queue, graph as parameters and performs the actions needed for deriving the required transition sequences. If the graph is projective and well structured, the oracle function will produce the transition sequence that applied to the sentence will create the gold standard annotation. On the other hand, a non-projective sentence will result in a parsed graph and manually-annotated sentence which are not equal. An example of a non projective sentence taken from the corpus is: ***"Vad beror ökningen på?"***. A sentence is projective if, from the head to the dependent, there exists a path from the head to every word that lies between. Which is a requirement the aforementioned sentence does not satisfy and hence, it is non-projective.

Figure 1 depicts a manual parsing of the sentence *"Individuell beskattning av arbetsinkomster"* with the stack and the queue after each step of the transition sequence. Each step in the figure has been given corresponding number that, if the action resulted in an arc, can be seen over an arc in

the resulting sentence graph drawn at the left bottom corner. The actions performed for parsing this sentence followed the following rules:

- **sh: Shift** - pushes the input word onto the stack. Does not produce an edge in the graph.
- **ra: Right-arc** - connects the token at the top of the stack with the input word by applying a right arc and pushes input word to the top of the stack.
- **la: Left-arc** - connects the input token with the token on top of the stack by applying a left arc.

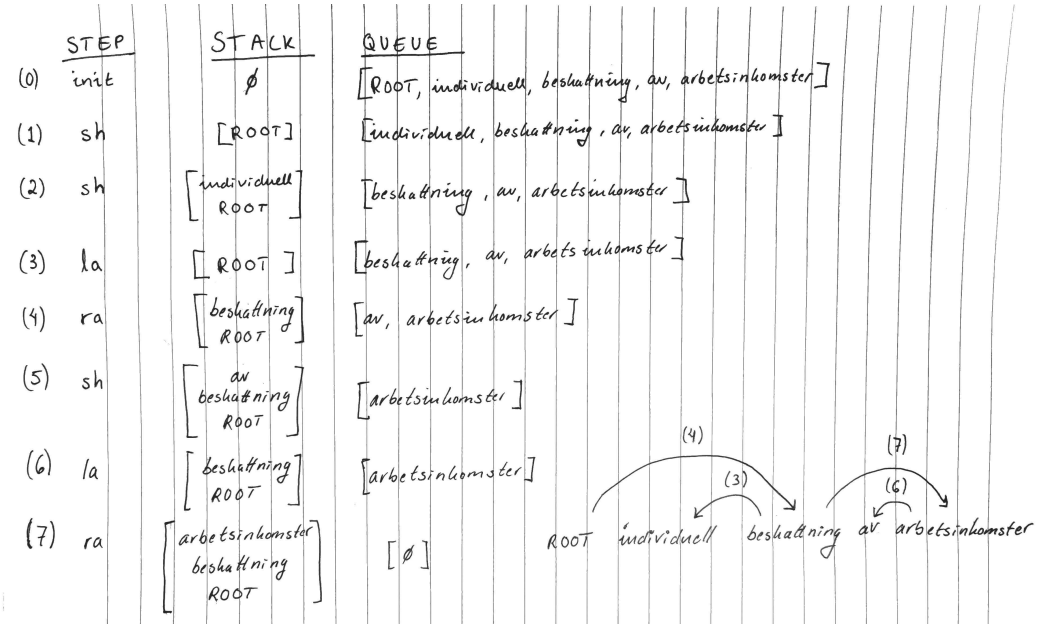

Figure 1: Manually applied transition sequence to the sentence "Individuell beskattning av arbetsinkomster".

2.2 Training a classifier and evaluating

Moving on to the step of training our classifier we first focused on producing the correct feature vectors. In order to be able to predict the next action from a given parsing state we also needed to extract feature vectors from each step of the parsing process. This was done with the code seen in figure 2 and we obtained feature vectors consisting of the first and second word from the stack based on both the part of speech and the word, the first and second word from the queue also based on both the part of speech and the word, and lastly two boolean values describing if a left, as well as, a right arc can be produces. The training procedure produced a classifier that we then embedded in Nivre's parser in order to predict the next action to perform, and during parsing, Nivre's parser could then call the classifier and using the prediction and the current context it chose the next action to perform.

In order to evaluate and measure the accuracy of the parse we used the CoNLL evaluation script referred to in the assignment. This gave us a score of 0.6911223438190116 when using depth=2 and the non improved extraction function, and when excecuting the evaluation script after running the prediction on a model trained by an improved feature vector (that being a feature vector composed with the improved version of the extract function) we could see a significant improvement in the score that landed on 0.7542327133532905.

```
print("Extracting the features...")
for sent_cnt, sentence in enumerate(formatted_corpus_train_clean):

    if sent_cnt % 1000 == 0:
        print(sent_cnt, 'sentences on', len(formatted_corpus_train_clean), flush=True)

    stack, queue, graph = init_config(sentence)
    while queue:
        X_dict.append(extract(depth, stack, queue, graph, sentence))
        stack, queue, graph, trans = oracle(stack, queue, graph)
        y_symbols.append(trans)
    stack, graph = empty_stack(stack, graph)

print("Extracting the features: Done")
```

Figure 2: Code for extracting feature vectors from each step of the parsing process.

3 Additional reading

The authors of the referred study propose a transition-based neural network model based on a feed forward neural network approach and manages to obtain state-of-the-art results in NLP problems such as part-of-speech tagging, dependency parsing and sentence compression. With dependency parsing being the main topic of this weeks assignment. The essence of their proposed methods lie in applying a feed forward neural network on a transition-based system, more specifically, Nivre's system. However, there is a difference in machine learning approaches used in study and the simple regressions used in this assignment, but non the less they both use a transition system. Furthermore, the use of Beam Search attempts to globally maximize the probability of a given prediction by deciding the optimal decision path, unlike our model which maximized on a local basis and exhibit a very local behaviour.