EMPIRICAL METHODS FOR NATURAL LANGUAGE PROCESSING

Python implementations of a Semantic Dependency Parser

Alejandro Jiménez Gutiérrez

1902010207

alejandro.jimenezg01@estudiante.uam.es

Abstract:

The task of this project was to develop a semantic dependency parser. The task is the same one as Semeval 2014 task 8. The approach taken in this project is based on a transition method using arc-eager algorithm. Multiple literature have been used in order to acquire the needed knowledge to develop the project. The classifier used for the system is an averaged Perceptron that is trained with the data created by the Oracle. It must be noted that the transition based approach has a limitation, this is that it is not able to create non-projective DAG structures. It can only create DAG's with non crossing arcs. This will therefore affect the trained model and in consequence will not be able to predict arcs in non-projective DAG's.

The dm.sdp file provided has been separated with 80% of the sentences into train.sdp and 20% into test.sdp. The training model has been trained using the train.sdp file and then validated with test.sdp. The results have been annotated and will be stated later on.

Parser:

To develop the parser the main ideas have been taken from *Speech and Language Processing. Daniel Jurafsky & James H. Martin. Chapter 15*. So the parser coded basically has an Oracle which following arc-eager algorithm will create instances associating the action to take with the features present in that moment (state). These actions obviously are the composition of the instruction plus the relationship given in the dependency. An example would be “LA_Arg1”.

After all the sentences have gone through the Oracle and all the features have been associated with instructions in instances which will serve as the training data, an averaged Perceptron trains a model iterating 15 time in total updating the weight of each feature for every action. The model had 64 possible labels or actions and more than 3 million resulting features.

After over 15 hours of training the model is ready to use and test with the test.sdp data. The results were the following.

```
/home/alex/PycharmProjects/hw2/venv/bin/python  /home/alex/PycharmProjects/hw2/
parse.py
Reading sdp file...
Done reading sdp file!
Reading sdp file...
Done reading sdp file!
Model loaded!

Parsing sentences...
6801it [14:31,  7.81it/s]
Done parsing.

Labelled Results:
     LP:  0.7794366224706146
     LR:  0.6802971124490458
     LF:  0.7265002800431677

Unlabelled Results:
     UP:  0.8101458762836233
     UR:  0.7071003394108919
     UF:  0.7551238792581362

Process finished with exit code 0
```

As it can be appreciated the results are not bad. However, they could be further improved changing and testing with different features.

The features used are described in the source code of the project and use information from the state of the sentence in the parser, paying attention to the stack, buffer, relations, distances and left most and right most dependencies.

Source Code:

I inspired the main core of my project on a dependency parser from GitHub that processes different input data and has different requirements because it is meant for trees and not for DAGs. It was very helpful to orient me in the right direction because I must admit that I was struggling to start off the project.

There are four files:

- model.py: Contains the all the necessary functions to create and save a model, this include of course all the features.

- reader.py: Processes the input sdp file and decodes its information into a DAG structure represented by the class Sentence.

- parser.py: Contains both the Oracle and the Parsing functions plus all the other needed functions in the Parser class. It also contains the main program and can run in three modes changing the variable *model_type*. These modes are train | test | dev. The **train** mode is the one in charge of generating the model file. **Test** is in charge of validating the model with the file test.sdp and printing the precision, recall and F_score both labeled and unlabeled. And **dev** is the one in charge of completing the esl.input and cesl.input files in esl.output and cesl.output

- reader.py: Has the class FileEncoder which will take the sentences, the input file and the arcs parsed by the system and complete the input file.

- split_sdp.dm: Contains the code to do the split of the dm.sdp int train.sdp and test.sdp.

Output Files

In the delivery there are two files esl.output and cesl.output which are the completed versions of esl.input and cesl.input. I expect the F score to be similar to the on achieved in my tests, but it depends on how different the source of this sentence is from the original WSJ one.

1902010207 Jimenez Gutierrez, Alejandro

Difficulties Encountered:

This is my first course on both NLP or machine learning, therefore there was a lot of new content in the slides and a lot of previous knowledge that I did not have and was trying to acquire during the course from courses on Youtube about Machine Learning and language processing. Mainly those published by Standford university.

I am not a Python expert neither because I have mainly worked in C and Java in my university back home so getting used to Python and to NumPy was a bit hard.

I struggled a lot for the first weeks of the project in understanding what the task exactly was and it took me a long time to get to know exactly what I needed to code and what exactly its internal structure should be. After reading a lot of resources, both from the slides and some online papers I finally found out what the project was exactly.

I hope the results of the task meet the expectations of the teacher and TA.