11-711: Algorithms for NLP

Homework Assignment #4: Transition-based Dependency Parsing

Out: November 10, 2015
Due: December 1, 2015

In this assignment you will implement an arc-standard transition-based parser by applying a multiclass classifier that will learn to parse sentences with dependency trees. You will also explore how to parse non-projective trees, although no implementation will be required in the latter case.

Data

- Training set (train.conll)
- Development set (development.conll).
- The test set will only be available two days before the deadline.

Important note: you are not allowed to use any training data annotated with dependencies other than the given training and development files. [1]

Data format Explanations about the conll data format can be found at
`http://ilk.uvt.nl/conll/#dataformat`
In a few words, column 1 is the token identifier, column 2 is the token itself, column 7 is the id of the head, and column 8 is the dependency relation. Although any parser may use any of the other token information, including:

- ID: token counter, starting at 1 for each new sentence
- FORM: word form or punctuation symbol
- LEMMA: lemma or stem of word form
- POSTAG: fine-grained part-of-speech tag
- FEATS: extra features (_ indicates no extra feature)
- HEAD: head ID of the current token
- DEPREL: dependency relation to the HEAD

Each field is delimited by a tab character. Sentences are delimited by a blank line. The last two columns (PHEAD, PDEPREL) are optional for decoding. We won't use them.

[1] Note that we should be able to replicate your number, and in order to get your results, your training set should be train.conll

1 Oracle.

The first step is to implement an Oracle. We are going to implement the Nivre's arc-standard oracle (explained in class) which is well known to be easy to implement and to provide consistent results across languages. The oracle code will look as slide #18 of the lecture about transition-based parsing. (tbparsing.pdf).

2 Training

For training, your code must learn a model following the set of transitions produced by the parser oracle. Given the parser state (topmost words of the buffer and the stack) the classifier will make a prediction, choosing which of the possible transitions is best. The left-arcs and the right-arcs may be enriched with dependency labels, this means that the number of classes you are predicting is 2 x number-labels (left and right) + 1 (shift). The default implementation provided uses a simple perceptron classifier for this task, but we encourage you to play with other possibilities, such as SVMs[2].

3 Decoding

Given a sentence annotated with POS tags, your code should be able to produce the dependency tree of the sentence and output a conll format file. This implies running your classifier several times to make the predictions during decoding.

4 Evaluation

For evaluation, you may use eval.pl evaluation script (`http://ilk.uvt.nl/conll/software.html`) (in this website there are some scripts that might be useful). Recent parsers provide results higher than 90% Labelled Attachment Score, although do not expect to get those numbers since you will be training your parser with a reduced training set to make it runnable on a personal computer. A reasonable result should be around 80% Labelled Attachment Score.

As a baseline, we run a couple of parsers out of the box (shown below). The results are shown in Table 1:

- Baseline. This is the expected score of the parser after one iteration of training with the baseline features, assuming you have implemented the oracle correctly.
- MaltParser (Nivre et al.2007). This parser is the most famous transition-based parser and it uses SVMs for learning. We run it out of the box following `http://maltparser.org` by using POS tags.
- LSTM-Parser. (Dyer et al.2015), which is a recurrent neural net transition-based parser. We stopped the training process after 14 epochs, and we run it out of the box as described in the ACL paper by using POS tags.

[2] A good SVM library, available in many programming languages is libsvm. See https://www.csie.ntu.edu.tw/ cjlin/libsvm/

	Development		Test	
	UAS	LAS	UAS	LAS
Baseline	52.81	1.93	54.67	2.03
MALT	82.73	79.86	83.79	80.95
LSTM	87.06	84.43	87.91	85.21

Table 1: Parsing results

5 Starter code

Starter code in Python is available on class website. You can either choose to use it or ignore it.

6 Non-projective parsing

Give your own ideas, or cite related research, about how to parse non-projective trees by modifying the arc-standard parsing algorithm.

Requirements

A written report (3 pages) with the experimental results on the tasks should be submitted no later than December 1. 2015. The report should look like a research paper. In addition, you should submit your running code with explanations about how to replicate your results. The code can be written in any programming language (we strongly recommend you follow the Python starter code). It must, however, run properly under Linux. To receive a B on this assignment, your parser should achieve at least 80% UAS. Tying MaltParser's score will get you an A.

Extra Credit

There are several ways of getting extra credit,[3]

1. If you manage to beat the score of MaltParser, shown in Table 1, you will get 1 extra point per each UAS point (in the test set).

2. If you beat the LSTM parser, shown in Table 1, you will get 3 extra points per each UAS point (in the test set).

3. If you look at the literature of transition-based parsing, people have come up with a lot of ideas to improve parsers. Extra credit will be given for (correct!) implementation of ambitious ideas.

4. As mentioned above, you are not allowed to use more training data annotated with dependencies than what you are getting for training and development. However, you may incorporate unlabeled data into your model.

[3]Yes, this means that your final credit could be over 100.

References

Dyer, Chris, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*.

Nivre, Joakim, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kbler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.