NYUAD Course CS-AD 220 – Spring 2016
Natural Language Processing
Assignment #3 : POS Tagging and Parsing
Assigned Mar 31, 2016 / Due Apr 17, 2016 (11:59pm)

I. Grading & Submission

This assignment is about the development of a dependency parser and a part-of-speech (POS) tagger for English. The assignment accounts for 15% of the full grade. It consists of five exercises. **There is also a bonus exercise that can count for up to 5% of the full grade**. The additional exercise consists of a parsing competition on an unseen test set. Participation earns 2%. The first, second and third ranked systems earn additional 3%, 2% and 1%, respectively.

The assignment is due on April 17 (11:59pm). The student should upload a zipped directory with the name "Assignment-3-<student-name>.zip" to NYU Classes under Assignment-3.

What to include in the submission zip file:

(1) Include a report **(in PDF)** addressing the five (plus one) exercises. You can explain what you did by providing the commands you used as well as a narrative explaining what you did and why you did it. Provide neat and clear answers: e.g., use tables and graphs to organize the results you obtained. Provide clear conclusions you derive from the exercise in addition to insights and observations.
(2) The outputs on development set with each of the different systems you create in the exercises. Name the files clearly: e.g. **dev.out.ex.1.a.conll**.
(3) The output on the test set. Only one result allowed. Name the file **test.out.conll**.

What NOT to include in the zip file:

(1) The training data
(2) The development data
(3) Any models created

II. Before Starting

You need to install the following tools: MaltParser and HunPOS tagger. You will also need data for training, development and blind testing, as well as evaluation scripts.

The data sets and evaluation scripts are available on the class website:

https://sites.google.com/a/nyu.edu/nyuad-cs-ad-220-natural-language-processing-spring-2016/files/CS-AD-220-S2016-Assignment-3.zip

A. Data Sets

This data is copyrighted and should not be distributed or shared outside of this class.

The data sets are based on the English Penn Treebank[1] in the basic Stanford-style dependency[2] used in the Universal Dependency Treebank Project.[3] The data sets come in the CONLL data format.[4] (See footnotes for more details).

Let's consider this example from the training data:

```
1       The     _       DET     DT      _       4       det     _       _
2       luxury  _       NOUN    NN      _       4       compmod _       _
3       auto    _       NOUN    NN      _       4       compmod _       _
4       maker   _       NOUN    NN      _       7       nsubj   _       _
5       last    _       ADJ     JJ      _       6       amod    _       _
6       year    _       NOUN    NN      _       7       nmod    _       _
7       sold    _       VERB    VBD     _       0       ROOT    _       _
8       1,214   _       NUM     CD      _       9       num     _       _
9       cars    _       NOUN    NNS     _       7       dobj    _       _
10      in      _       ADP     IN      _       7       adpmod  _       _
11      the     _       DET     DT      _       12      det     _       _
12      U.S.    _       NOUN    NNP     _       10      adpobj  _       _
```

The columns with "_"s can be ignored for this exercise. The columns in order are: word id, word form, lemma (ignore), coarse-grained POS, fine-grained POS tag, features (ignore), id of head/parent word, dependency relation, projective parent (ignore), projective dependency relation (ignore).

How to read the tree: *The root of the dependency tree is word #7 "sold", which is a verb. The word "maker" (#4) is the subject (nsubj) of the verb "sold"; and the word "cars" (#9) is the direct object (dobj) of the verb "sold"... and so on.*

B. Evaluation Scripts

In the assignment package, there are two evaluation scripts: one for POS tagging accuracy (pos-eval.pl) and one for parsing accuracy (conll-eval.pl).

POS accuracy The POS tagging accuracy script computes the percentage of words with correct POS tags. It needs a gold file and a predicted file consisting of tab-separated word and POS (one word-POS per line).

Parsing accuracy There are multiple metrics that can be used to evaluate dependency parsing accuracy. We will only use Labeled Attachment Score (LAS). LAS is the percentage of words whose parent/head and relation are determined correctly against a gold parsed file. We will ignore punctuation words in this assignment (this is done automatically and by default in the evaluation script). The evaluation script has a quiet (-q) mode that prints overall performance; alternatively, it prints out a lot of details, which may be helpful in debugging and analyzing the systems you put together. Run the script (`perl conll-eval.pl`) without parameters to see the usage description.

[1] https://www.cis.upenn.edu/~treebank/
[2] http://www-nlp.stanford.edu/software/dependencies_manual.pdf
[3] https://code.google.com/p/uni-dep-tb/
[4] http://ilk.uvt.nl/conll/#dataformat.

C. MaltParser

MaltParser is a system for data-driven dependency parsing, which can be used to induce a parsing model from treebank data and to parse new data using an induced model. MaltParser is developed by Johan Hall, Jens Nilsson and Joakim Nivre at Växjö University and Uppsala University, Sweden.

(1) Download the latest version of the parser from: http://www.maltparser.org/download.html
(2) Install the parser following the instructions in: http://www.maltparser.org/install.html
(3) Follow the instructions on how to build a parser and test it following the instructions in: http://www.maltparser.org/userguide.html

MaltParse trains and parses data in the CONLL format.

Note: You may have trouble with learning a model using all of the data because of memory allocation limitations. Try expanding the memory using the –Xmx/-Xms parameters:

```
java –jar -Xmx5000m -Xms5000m maltparser-1.8.1/maltparser-1.8.1.jar
-c parser-all -i en-universal-train.conll -m learn
```

To see all options to the MaltParser:

```
java –jar -Xmx5000m -Xms5000m maltparser-1.8.1/maltparser-1.8.1.jar --help
```

D. HunPOS

HunPOS is a state of the art open-source HMM tagger.

(1) Download and install the latest version of the HunPOS tagger https://code.google.com/p/hunpos/downloads/list
(2) Learn how to use the HunPOS tagger using the HunPOS manual https://code.google.com/p/hunpos/wiki/UserManualI

To get the usage of HunPOS: run `./hunpos-train –h` and `./hunpos-tag –h`

The data format for training and testing HunPOS consists of two tab-separated columns: word and tag. For example:

```
The	DET
luxury	NOUN
auto	NOUN
maker	NOUN
last	ADJ
year	NOUN
sold	VERB
1,214	NUM
cars	NOUN
in	ADP
the	DET
U.S.	NOUN
```

You will have to extract the data for training the POS tagger from the Treebank data provided to you. You can write code to do this or simply use standard Unix commands such as cut and paste.

III. Exercises

EX 1. Training the Parser: Learning Curve Analysis (20 points / 3% of full grade)

Train the parser on the training data set provided to you and evaluate on the development data set for different training data sizes. Build five versions of the parser using five training data sizes: 4, 40, 400, 4000, All (39,832) sentences. Take the first 4, 40, 400 and 4000 sentences from the corpus for the partial training cases. You can do the extraction of sentences using Unix commands only. Use the default training configuration for the parser.

What to deliver in the report:

a. A *learning curve* **table** that shows the performance on the development data set for each training size.
b. Do you see a pattern? What do you predict the accuracy will be for 400,000 sentences? What about 4,000,000 sentences?

EX 2. Training the Parser: Parsing Algorithms (20 points / 3% of full grade)

MaltParser supports a number of parsing algorithms, which are described in the User Guide. In this exercise, you will compare the performance of different algorithms when training on the full training data (no learning curve needed). Your goal is to find the best algorithm to use for this training and development set. You need to minimally try three configurations in addition to the default configuration you tried above. Restrict your search to the algorithms in the *Nivre* and *Stack* families.

What to deliver in the report:

a. A **table** showing the performance on the development data set for each parsing algorithm explored.
b. Identify the best performing algorithm.

EX 3. Training the POS Tagger (20 points / 3% of full grade)

Train the HunPOS tagger using all the training data provided and evaluate on the development set. Vary the following parameters to find the optimal training and tagging settings that maximize the accuracy of the POS tagging : tag order, emission order, and the unknown-word handling parameters. Run "`./hunpos-train -h`" and "`./hunpos-tag -h`" to see how to set these values. You need to **minimally** explore six settings combinations.

Since the training data provides two types of POS tags, combine the two tags into a joint tag, e.g.: `ADP-IN, DET-DT, and NOUN-NNP.`

What to deliver in the report:

a. A **table** showing the POS tagging accuracy on the development data set for all explored tagging parameter combinations. Make clear what the parameters are.

b. Identify the best performing parameters.
c. Why do you think the best parameter combination works best? Are you surprised/not surprised? Why?

EX 4. Evaluating the Parser with Predicted Tags (20 points / 3% of full grade)

The parsing experiments done in EX 1 and 2 used gold POS tags in the development. This is clearly unrealistic. In this exercise, you will use the predicted POS tags using the best settings determined in EX 1, 2 and 3.

What to deliver in the report:

a. A **table** showing the parsing accuracy using gold POS tags (best result in EX 2) and the parsing accuracy using predicted POS tags (produced by the best POS tagger parameters in EX 3). All other parameters must be the same.
b. Identify the best preforming setting. Are you surprise or not surprised? Why?

EX 5. Parsing Error Analysis (20 points / 3% of full grade)

In this section, you will conduct an error analysis on the result of EX 4 (with predicted POS tags). Compare the **first 20** sentences in the development set (predicted tags and dependencies) against their gold version. You can use the parsing evaluation script in the non-quiet mode (without –q) to help you in this task.

What to deliver in the report:

a. A written analysis of the different kinds of errors in terms of classes or patterns that you will determine. Provide your own statistics on the errors types based on the 20-sentence sample. Do not simply copy and paste the output of the automatic evaluation script or paraphrase it.
b. What can be done to address these errors? Provide at least two ideas. Explain why you think these ideas can help.

EX 6. Bonus Parsing Competition (2%-5% of full grade)

This exercise is optional. You are provided a test set in the assignment package. You will need to POS tag the test and parse it. The instructor will do the evaluation. Participating earns you 2% of the full grade automatically (given that the file is well formatted and there are no evaluation problems). Not participating at all earns you 0%. The best three performing systems (ranked 1st,2nd,3rd) will get an additional bonus of (3%, 2% and 1% of full grade, respectively). You can minimally use the best settings you determined in the previous exercises. Alternatively, you can improve on these settings using additional ideas not discussed in the exercise. The only restriction is this: you cannot use additional gold data not provided in this assignment.

What to deliver in the report:

a. A detailed description of your system’s settings.
b. The parsed test data.

==

Have fun and start working on the assignment early!

==