CS134 (Fall 2012): Programming Assignment #4

Due: Dec. 14, 2012

Transition-based dependency parsing is a simple, but effective approach to developing robust data-driven dependency parsers. In this assignment you will develop a very basic transition-based parser using either an off-the-shelf Python-based classifier, or, preferably, your own Maximum Entropy or Naive Bayes classifier developed as part of earlier assignments.

The following steps will be required to complete this assignment:

Create classification instances You'll need to first take the training corpus and be able to derive a set of training instances for your classifier which will serve as a "guide" or "oracle" indicating which transition action to take for a given configuration. This will involve two key steps:

- You must first derive the correct set of transitions given a correct "gold standard" dependency parse (as found in the training data)
- You must then create a set of features for each instance making use of the current state of the stack, σ and buffer β.

Train the classifier You'll need to train up a classifier on the training data. You may also which to evaluate the classifier's accuracy on the evaluation data. This will give you an idea of how well the resulting parser may perform (a "perfect" classifier will provide a perfect parser), though it's worth emphasizing that certain errors will degrade the parsers accuracy more than others. In particular, errors made for transitions early in the sequence will degrade parser performance more than errors made towards the end.

There are then three possible options that build upon the work above. You need to complete at least one option for full credit, and significant extra credit is possible by completing two or all three options. The three options include:

Feature Development - develop a series of improved features that try to maximize the accuracy of your classifier

Labeled Dependencies - develop a classifier that acts as a guide/oracle for labeled dependency parsing

Parser Decoder - develop a decoder for full unlabeled dependency parsing using the trained classifier as a guide/oracle.

1 Data

The data you will use for this assignment consists a portion of the Penn Treebank Wall Street Journal corpus. The training data will consist of four sections of the corpus: `wsj.00.01.22.24.conll` while the test data consists of `wsj.23.conll`. Below is an example sentence from the training set file:

1	Mr.	_	NNP	NNP	_	2	NMOD	_	_
2	Vinken	_	NNP	NNP	_	3	SBJ	_	_
3	is	_	VBZ	VBZ	_	0	ROOT	_	_
4	chairman	_	NN	NN	_	3	PRD	_	_
5	of	_	IN	IN	_	4	NMOD	_	_
6	Elsevier	_	NNP	NNP	_	7	NMOD	_	_
7	N.V.	_	NNP	NNP	_	5	PMOD	_	_
8	,	_	,	,	_	7	P	_	_
9	the	_	DT	DT	_	12	NMOD	_	_
10	Dutch	_	NNP	NNP	_	12	NMOD	_	_
11	publishing	_	VBG	VBG	_	12	NMOD	_	_
12	group	_	NN	NN	_	7	NMOD	_	_
13	.	_	.	.	_	3	P	_	_

Below is a brief description of the relevant columns:

1 - Word index the index for the current word

2 - Token the string that is the current word

4 - Part-of-speech Part of speech for the current word (note column 5 contains the same information for historical reasons)

7 - The head index this is the word index for the head (i.e. parent) of the current word

8 - Label type The dependency label (grammatical function) for the incomming arc.

In this assignment, the focus is on unlabeled dependency parsing unless you pursue Option 2 (see below), so you won't necessarily need column 8.

A first part of your program will involve reading this into an appropriate data structure. You are free to do this in a manner you see fit, but at a minimum you'll need a data structure that captures the information present in the above column descriptions for each word, keeps the words in sequence for each sentence, and so forth.

2 Deriving Gold-standard Transition Sequences

Given the data input files described above, this step involves building up a transition sequence that would produce the gold standard dependency parse provided. In a sense, this involves running the parser "backwards". Below is a description of this algorithm.

Let A_g denote the set of arcs found in the gold-standard correct parse for a given sentence x. Let $h_g(i) = j \Leftrightarrow (j, i) \in A_g$.

function GOLD-PARSE($(x = (1, .., n), h_g)$)
  c $\leftarrow$ (η,(1,..,n), $A_0 = \emptyset$)
  $D_x \leftarrow \emptyset$
  while $c = (\sigma, \beta, A)$ is not terminal **do**
    if $\sigma = \eta$ **then**
      $c \leftarrow$ `Shift`(c)
    else
      $t \leftarrow$ `Oracle`(c, h_g)
      $D_x \leftarrow D_x \cup (c, t)$
      $c \leftarrow t(c)$
    end if
  end while
  return D_x
end function

The algorithm for the function `Oracle` is then defined as follows:

function ORACLE($(c = (\sigma|i, j|\beta, A), h_g)$)
  if $h_g(i) = j$ **then**
    return `Left-Arc`
  else if $h_g(j) = i$ **then**
    return `Right-Arc`
  else
    return `Shift`
  end if
end function

The result of this process is a set of configuration and transition pairs $D_x = \{(c_0, t_0), (c_1, t_1), .., (c_n, t_n)$. From this set of pairs, you should derive a simple set of features to train your transition guide/oracle. At a minimum you should include the following:

Front of queue For a buffer/queue $i|\beta$ include a feature that encodes `topOfBuffer`$= i$

Top of stack For a stack $\sigma|i$ include a feature that encodes `topOfStack`$= i$

Stack/Buffer bi-gram For a configuration with $\sigma|i$ and $j|\beta$ include a feature encoding `bufferStackPair`$=(i, j)$

POS variants Include versions of the above three features where the lexical values are replaced with their corresponding parts-of-speech

3 Training a Model to Approximate the Oracle

Given the instances generated above, the next step is to train a classifier to act as a guide/oracle. Ideally, you should use your maximum entropy classifier from PA3 or your Naive Bayes classifier

from PA1. However, if you had problems with your implementations, we suggest using the Maximum Entropy (or Naive Bayes) classifiers from the Natural Language Toolkit (NLTK).

At this point there are three different options for this assignment; only one option is required, however, extra-credit is possible by completing/attempting two or more options.

4 Classifier Optimization [Option 1]

The first option involves developing a more advanced set of features than the small set described in Section **??**. You should aim to add a fairly extensive number of additional features, leveraging some of the feature sets reported in the literature. The accuracy of your classifier should be measured on the evaluation data,

`wsj.23.conll`.

Of course, you'll need to convert the test file to transition sequences using the `Gold-Parse` algorithm you've developed.

Perform a detailed analysis and breakdown of the errors, such as: precision/recall for the three different actions (leftArc, rightArc, shift), accuracy as a function of the position in the transition sequence (do more errors occur early on or later in the sequence?), accuracy broken down by part-of-speech (for the top-of-stack and/or front-of-queue terms).

Present these results for your final, complete set of features as well as for various interesting sub-sets of features. Present an analysis discussing what features worked, any other interesting observations and discuss future directions.

5 Arc-Labeling [Option 2]

This option involves altering your `Gold-Parse` algorithm to use the gold-standard labels to derive $LeftArc_k$ and $RightArc_k$, $1 \leq k \leq N$, for N different possible labels as they appear in the dataset.

Using the basic set of features in Section **??**, determine the performance of your classifier on the expanded set of transitions along with a break-down of errors by label type. Introduce at least two new types of features that improve performance on the labeled transition classification task and present the results.

6 Decoding [Option 3]

The third option involves "closing the loop" by building a working parser. The decoding step involves building up a parse structure incrementally by following a particular transition sequence for an input sentence. Given the classifier trained in Section **??**, you can construct the parse for an input sentence via the algorithm:

function GOLD-PARSE($(x = (1, .., n))$)
  c $\leftarrow$ (η,(1,..,n), $A_0 = \emptyset$)
  $D_x \leftarrow \emptyset$
  while $c = (\sigma, \beta, A)$ is not terminal **do**
    if $\sigma = \eta$ **then**
      $c \leftarrow$ `Shift`(c)

else
$c \leftarrow [g(c)](c)$
end if
end while
return $c.A$
end function

Evaluate your parser on the testing data in the file

`wsj.23.conll`

You can do this by either writing your own program to compare the output of your parser against the gold-standard dependency arcs. Or - you can use an existing evaluation program such `MaltEval`, available at http://w3.msi.vxu.se/ jni/malteval/

Provide at least one interesting breakdown of the errors - e.g. in terms of dependency arc-length or part-of-speech of the dependent, etc.

7 Grading

Your grade will not be determined by the accuracy of your parser, however extra-credit is possible for very good accuracies (i.e. those that approach scores reported in the literature). Along these lines, you are free to utilize the full training portion of the WSJ corpus (also provided); the large amount of additional training data should improve performance.

In contrast to earlier assignments, for this assignment, you are not provided any starter code. You should make an effort, however, to document and structure your code in an intelligible manner to help both in the grading of the assignment and so that we are able to help you with your program if necessary. Part of your grade for this assignment will be determined by the quality of your code.

Finally, note that for the core part of the assignment and all three "Options", there is a fair degree of freedom with regard to how you approach the problem. If you are especially interested in working with another language, for example, that is certainly a possibility - either instead of or in addition to the English data provided. Incorporating other sources of features (e.g. lexical information) is another idea. In short, this assignment can be viewed as a mini-Final Project if you wish. However, be sure to discuss any ideas with us before you get started.