INFO 159/259 Homework 6
Transition-Based Dependency Parser

October 31, 2017; due November 13, 2017 (11:59pm)

In this homework, you will be implementing parts of a transition-based dependency parser. You'll find the following files on bCourses in `Files/HW6`:

- `hw6.py`: main file for the transition-based dependency parser
- `train.projective.conll`: training dataset
- `train.projective.short.conll`: a smaller training dataset
- `dev.projective.conll`: dataset to evaluate your parser's performance

If training is taking a long time with the regular training file `train.projective.conll`, feel free to use the smaller dataset `train.projective.short.conll`.

All data files contains the following 8 columns:

index	**word**	lemmatized word	pos_1	**pos_2**	_	**head index**	**dependency label**
1	forces	force	NOUN	NNS	_	7	nsubj
2	This	this	DET	DT	_	3	det

For this homework, we will only use 5 of the 8 columns: index (col1), word (col2), pos_2 (col5), head index (col7) and dependency label (col8). In code, their corresponding variable names are `(idd, tok, pos, head, lab)`.

Other than the extra credit question, you should only be making changes to the code in the `TO BE IMPLEMENTED` section. For the extra credit question, you will be making changes to the `featurize_configuration` function.

1 Checking for Projectivity (0.5 pt)

Function to be implemented for this part:

- `is_projective`

The transition-based dependency parser we've been considering requires projective trees for training; before training the parser on a tree, we need to make sure the tree is projective.

Recall from class and the textbook, "An arc from a head to a dependent is said to be projective if there is a path from the head to every word that lies between the head and the dependent in the sentence. A dependency tree is then said to be projective if all the arcs that make it up are projective."

Implement `is_projective` that takes in a sentence with its dependency tree and returns `True` if and only if the tree is projective.

2 Creating an Oracle (1.5 pt)

Functions to be implemented for this part:

- `perform_shift`
- `perform_arc`
- `tree_to_actions`

The parser relies on a classifier to decide the best action (`SHIFT`, `LEFTARC` or `RIGHTARC`) to take at each step, given the current state of the configuration. To train a classifier, we want to have training data in the form of (configuration, action) pairs. The raw inputs from the data files, however, are in the form of dependency trees (lists of `(idd, tok, pos, head, lab)` tuples).

Your task is to implement the training oracle, which transforms the input trees to (configuration, action) pairs in `tree_to_actions`. As part of the implementation, fill out helper functions `perform_shift` and `perform_arc`. The helper functions generate the proper configuration and action for the corresponding `SHIFT`, `LEFTARC` and `RIGHTARC` operations, and update the states of the stack, input buffer and arcs as needed. They should be called in `tree_to_actions`.

On a high level, we are trying to create:

- X: configuration of the parser, which is determined by the states of the input buffer, stack and arcs. The configuration is used for feature generation
- y: action for the parser (e.g. `SHIFT`, `LEFTARC_nsubj`)

Details for the logic of the algorithm can be found under the lecture slide and the textbook SLP3 Ch 14.4.1.

Implemented for you
Once we have X (with featurization) and y generated by the training oracle, we can train a classifier using logistic regression to predict the distribution over actions to take given a configuration on previously unseen data. This part of the code has been implemented for you in `train`.

3 Tree Parsing with Predictions (1 pt)

Function to be implemented for this part:

- `action_to_tree`

Now that we have a classifier, it's time to evaluate the performance of the parser. Essentially we are trying to carry out the reverse steps of part 2–based on recommended actions (predictions) from the classifier, we want to generate a dependency tree for a sentence.

Implement `action_to_tree`, which will update the dependency tree based on the action predictions. Note that the classifier's recommendation at each step includes predicted probabilities for all possible actions (higher probability indicates more preferable action), and not all actions will be valid. Be sure to make use of the helper function `isvalid` to check the validity of an action.

4 Extra Credit (1 pt)

Function to be implemented for this part:

- `featurize_configuration`

Congratulations on completing the core part of the assignment! If you feel extra adventurous, try improving the parser's performance by adding new features / updating existing features in `featurize_configuration`. An extra 0.5 pt will be awarded for a 10% or more improvement in correctness, e.g. from 65% (baseline) to 75%, and an extra 1pt will be awarded for a 15% or more improvement.

5 Deliverables

Submit your `hw6.py` on bCourses.