CS5740, Spring 2020: Assignment 5

Submission Due: May 12, 11:59pm

Report page limit: 3 pages

In this assignment, you will build a dependency parser. You will generate a dataset of parser configurations and transitions using the arc-standard approach with the provided CoNLL data. You will use this data to train a classifier (e.g., SVM or maximum entropy) that generates dependency parses. There is no leaderboard for this assignment.

Forming Groups Please form your groups within 72 hours of the assignment release. First, create your group on CMS, and then form it on Github Classroom. We will not be able to grade assignments where CMS and Github are not in sync. As long as you are not part of a group on CMS, we will consider as not part of a group. **Following the 72 hours deadline, we will do random assignments. We will deduct 3 points from the grade of each group that requires random assignment.** Please use the forum to form groups. If we identify cases of students that do not work together, we will be forced to penalize the assignment heavily.

Please submit your report PDF on both CMS and Gradescope by the deadline.

Submission **Please see the submission guidelines at the end of this document.**

Starter Repository:
`https://classroom.github.com/g/T2T8yZRA` (GitHub Classroom invitation link)

Data Download:
`http://www.cs.cornell.edu/courses/cs5740/2020sp/r/a5-data.zip`

Report Template:
`https://www.overleaf.com/read/ddckqmvxgzhv`

Task. You will implement an arc-standard dependency parser that predicts a dependency parse (arcs only) for a sentence. You can use whatever classifier you like for classifying each transition (e.g., SVM, maximum entropy).

Data. Follow the instructions in Assignment 1 for setting up teams on GitHub Classroom. For this assignment, there is no starter code. Remember that all the team code should be pushed to the repository by the deadline. Only code present in the master branch of your repository will be graded! You must implement your solution in Python, and may use SciPy, NumPy, and libsvm. Other tools must be approved on Piazza.

Download the data listed at the top of the page. The CoNLL data is divided into training, development, and test datasets. Because there is no leaderboard for this assignment, you should evaluate your models on the provided test data. Use your test data only for the final evaluation. Ablations and other analysis should be done on the development data.

Evaluation. You will evaluate your model on unlabeled attachment score (UAS). Use the provided Perl script `dependency_eval.pl` to compare your predictions with the gold labels. This script also generates a labeled attachment score (LAS) – however, you can ignore this number. Make sure to "predict" a dummy label for every predicted arc so that the script will run properly.

Extra credit: We will give a bonus of +10 points for implementing labeled arc prediction and achieving competitive results (at least 85% LAS).

Because there is no leaderboard for this assignment, you will run this script on both the development and test datasets (using the `-g` flag) and your predictions (using the `-s` flag). If you want to print UAS and LAS accuracy at the sentence-level for error analysis, add the `-b` flag.

```
perl dependency_eval.pl -g dev.conll -s [YOUR PREDICTION] -q 2> /dev/null
perl dependency_eval.pl -g dev.conll -s [YOUR PREDICTION] -qb 2> /dev/null
```

Note that this Perl script will produce error messages even when your input is fine. For this reason, we're redirecting error messages from your screen. If your output looks wrong, you can turn off redirecting. You could also try running the script on the development set twice, which should result in 100% UAS and 100% LAS. This is for debugging only. Please do not report these numbers!

Expected Results. If you are scoring 75% or below (unlabeled attachment score), then something is wrong. A score of 85%+ is OK. It will start to get really interesting as you get closer to 92%.

Performance Grading The grading of your test performance on the leaderboard will be computed as $\frac{\max(0,\text{UAS}-0.7)}{0.88-0.7} \times 12$, where UAS is the unlabeled attachment test score.

Submission, Grading, and Writeup Guidelines Your submission on CMS and Gradescope is a writeup in PDF format. **The writeup must follow the template provided. Please replace the TODOs with your content. Do not modify, add, or remove section, subsection, and paragraph headers.** The writeup must include at the top of the first page: the names of the students, the NetIDs of both students, the team name, and the URL of the Github repository. The writeup page limit is **3 pages**. We will ignore any page beyond the page limit in your PDF (do not add a cover page). We have access to your repository, and will look at it. Your repository must contain the code in a form that allows to run it form the command line (i.e., Jupyter notebooks are not accepted).

The following factors will be considered: your technical solution, your development and learning methodology, and the quality of your code. If this assignment includes a leaderboard, we will also consider your performance on the leaderboard. Our main focus in grading is the quality of your empirical work and implementation. Not fractional difference on the leaderboard. We value solid empirical work, well written reports, and well documented implementations. Of course, we do consider your performance as well. The assignment details how a portion of your grade is calculated based on your empirical performance.

In your write-up, be sure to describe your approach and choices you made. Back all your analysis and claims with empirical development results, and use the test results only for the final evaluation numbers. It is sometimes useful to mention code choices or even snippets in write-ups — feel free to do so if appropriate, but this is not necessary.

Some general guidelines to consider when writing your report and submission:

- Your code must be in a runnable form. We must be able to run your code from vanilla Python command line interpreter. You may assume the allowed libraries are installed. Make sure to document your code properly.
- Your submitted code must include a `README.md` file with execution instructions. Make sure to document your code properly.
- Please use tables and plots to report results. If you embed plots, make sure they are high resolution so we can zoom in and see the details. However, they must be readable to the naked eye. Specify exactly what the numbers and axes mean (e.g., F1, precisions, etc).
- It should be made clear what data is used for each result computed.
- Please support all your claims with empirical results.
- All the analysis must be performed on the development data. It is OK to use tuning data. Only the final results of your best models should be reported on the test data.
- All features and key decisions must be ablated and analyzed.

- All analysis must be accompanied with examples and error analysis.
- Major parameters (e.g., embedding size, amount of data) analysis must include sensitivity analysis. Plots are a great way to present sensitivity analysis for numerical hyper parameters, but tables sometimes work better. Think of the best way to present your data.
- If you are asked to experiment with multiple models and multiple tasks, you must experiment and report on all combinations. It should be clear what results come from which model and task.
- Clearly specify what are the conclusions from your experiments. This includes what can be learned about the tasks, models, data, and algorithms.
- Make figures clear in isolation using informative captions.