CS 378, Spring 2021: Final Project

Due date for report & code submission: Friday May 15, 11:59pm CST
Due date for output leaderboard submission: Thursday May 14, 11:59pm CST

This is a group assignment, where each group consists of two students. Individual project is allowed but not recommended.

Academic Honesty: Please see the course syllabus for information about collaboration in this course. While you may discuss the assignment with other students, **all code you write and your writeup must be your own!**

Your goal in this assignment is to develop a name-entity recognizer for social media text. The assignment approximates, as much as possible, the challenges of building a solution to a *real problem in the real world*, including all the mess and noise.

Background Increasingly, politicians on Twitter have been making impulsive and ill-advised comments about people, groups, countries, organizations, businesses, and other entities that they eventually come to regret. REGRET™ is an up-and-coming startup that provides a tweet-checking service marketed to these politicians. Right before a politician presses send, the service conducts a sanity check to the tweet. The first step in the process is identifying named entities. Each named entity then goes through a sanity check process with regard to the content of the tweet.

Needless to say, our named-entity recognition system must operate with both high accuracy *and* speed. False negatives may cause an embarrassing tweet to slip through the system without proper vetting, and false positives (i.e., *alternative* entities) may confuse or irritate the politician. At the same time, we must scan each tweet as quickly as possible: REGRET™ is the last line of defense preventing politicians' problematic other-directed tweets from being released to the world!

Named Entity Recognition on Social Media Your task is to build a named-entity recognizer for Twitter text, a key component of our tweet-scanning system. Given a tweet, you are to identify sub-spans of words that represent named entities. You are not required to distinguish between entity type. For example, given the tweet:

Input:	RT	@ForestLogic	Vaughan	Fox	Say	what	you	want	,	but	nice	one	@fawaz_alha
Output:			ENTITY	ENTITY									

Your system is expected the generate the output above, where two entities are detected. Entities can include multiple tokens and can have noisy capitalization. For example:

Input:	Highlight	of	my	day	:	eating	a	Carlos	bakery	lobster	tail
Output:								ENTITY			

Technical Guidelines You should use Python to implement your system. The following packages are allowed: NumPy, Pandas, SciPy, PyTorch, word2vec, GloVe, and Hugging Face Transformers, which includes various models and tools. Feel free to use TACC cluster that you've set up during HW4. You may use stemmers, POS tagger, and constituency and dependency parsers. For any other frameworks and tools, please ask on the piazza forum for approval. If you are not sure, ask! You may **not** use an off-the-shelf system for entity recognition, even if you plan to re-train it.

If you use external libraries, including these listed above, they must be listed in a PIP-formatted `requirements.txt` file located in the root of your submitted file. Document precisely how to install your dependencies in your `README.md` file.

Please assume a vanilla Ubuntu installation. You may use any algorithm you see fit. This is not limited to machine learning techniques. Remember, your approach must be scaleable and fast. In your writeup, you should report the run times required to generate the results on the evaluation set. Please also report your hardware configuration (e.g., how long does it take to train with what hardware) so we can better understand your performance.

Data, Formatting, and Evaluation You are free to use any data that you consider suitable and useful for this task. We are providing data for training and development, but you may use any external data you see fit, including gazetteers, lexical ontologies, etc. The training data `train.txt` includes a token on every line. Sentences are separated by empty lines. Each token is annotated with B for beginning of the named entity, I for continuation of a named entity, or O for a token that is not part of a named entity. Each named entity must start with B and all other tokens of the named entity, except the first one, are annotated with I.

To evaluate your output use:

```
python tageval.py LABELED_DATA OUTPUT
```

Where `LABELED_DATA` is the gold standard annotated data and `OUTPUT` is the output of your model. The `OUTPUT` file is formatted slightly differently and omits the words. Each line includes only the BIO label. See the file `train.out`, which is aligned to `train.txt`, for an example. The evaluation is order sensitive, so make sure to maintain order in your output.

Getting Started The starter code is provided, which contains a simple script for evaluating your model output.

Leaderboard We will use a leaderboard on GradeScope to manage the evaluation. You will submit your output file formatted like `train.out`, named `test.out`. Your group should not submit **more than once in 24 hour period**, to avoid test set leakage. REMEMBER, you are not supposed to run multiple times on the test set. At best, you should run 2-3 times, we might deduct points from groups that submitted more than 5 times total. Also, all analysis and ablation studies should be done in the development set.

Grading The final grade will be graded based on the performance (30% of grade) and the report (70% of grade). The grading of your test performance on the leaderboard will be computed as $(\frac{\max(0, f-0.35)}{0.58-0.35} \cdot 30)$, where f is your test F1 score from the leaderboard.[1]

Submission Guideline As a part of this assignment, you will submit three elements: your code (zipped file), your test output (test.out), and your report (pdf).

An important part of this assignment is writing the report. You have to use the provided report template[2], and the maximum page limit is 3 pages. You will submit the written final report to Gradescope as usual.

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 EIDs of both students, and the team name. 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.

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:

[1] For a reference point, the highest performance in previous semesters on this assignment was 0.6814, from a masters class at Cornell in spring semester of 2019.

[2] `https://www.overleaf.com/read/shvvtszkncry`

- 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.