Coding Assignment 4

Due: October 25, 2022, at 14:00 Pacific Time (2:00 PM).

This assignment counts for 10% of the course grade.

Assignments turned in after the deadline but on or before October 26 are subject to a 10% grade penalty.

Overview

In this assignment you will write perceptron classifiers (vanilla and averaged) to identify hotel reviews as either truthful or deceptive, and either positive or negative. You may use the word tokens as features, or any other features you can devise from the text. The assignment will be graded based on the performance of your classifiers, that is how well they perform on unseen test data compared to the performance of a reference classifier.

Data

A set of training and development data are available as a compressed ZIP archive on Blackboard. The uncompressed archive contains the following files:

- One file `train-labeled.txt` containing labeled training data with a single training instance (hotel review) per line (total 960 lines). The first 3 tokens in each line are:
  1. a unique 7-character alphanumeric identifier
  2. a label `True` or `Fake`
  3. a label `Pos` or `Neg`

  These are followed by the text of the review.

- One file `dev-text.txt` with unlabeled development data, containing just the unique identifier followed by the text of the review (total 320 lines).
- One file `dev-key.txt` with the corresponding labels for the development data, to serve as an answer key.

- Readme and license files (which you won't need for the exercise).

The submission script will train your models on the labeled training data and test them on the development data. The grading script will train your model on the combined labeled training and development data, and test the models on unseen data in a similar format.

Programs

The perceptron algorithms appear in Hal Daumé III, A Course in Machine Learning (v. 0.99 draft), Chapter 4: The Perceptron.

You will write two programs in **Python 3** (Python 2 has been deprecated): `perceplearn.py` will learn perceptron models (vanilla and averaged) from the training data, and `percepclassify.py` will use the models to classify new data.

The learning program will be invoked in the following way:

```
> python perceplearn.py /path/to/input
```

The argument is a single file contatining the training data; the program will learn perceptron models, and write the model parameters to two files: `vanillamodel.txt` for the vanilla perceptron (Algorithm 5 in the above reference), and `averagedmodel.txt` for the averaged perceptron (Algorithm 7). The format of the model files is up to you, but they should follow the following guidelines:

1. The model files should contain sufficient information for percepclassify.py to successfully label new data.
2. The model files should be human-readable, so that model parameters can be easily understood by visual inspection of the files.

The classification program will be invoked in the following way:

```
> python percepclassify.py /path/to/model /path/to/input
```

The first argument is the path to the model file (vanillamodel.txt or averagedmodel.txt), and the second argument is the path to a file containing the test data; the program will read the parameters of a perceptron model from the model file, classify each entry in the test data, and write the results to a text file called `percepoutput.txt` in the same format as the answer key.

Submission

All submissions will be completed through Vocareum.

Multiple submissions are allowed; only the final submission will be graded. Each time you submit, a submission script is invoked. The submission script trains your models on the training data, runs your classifiers on the development data, and reports the results (the classifier is run twice: once with the vanilla model, and again with the averaged model). Do not include the data in your submission: the submission script reads the data from a central directory, not from your personal directory. You should only upload your program files to Vocareum, that is `percepclassify.py` and `perceplearn.py` (plus any required auxiliary files, such as code shared between the programs or a word list *that you wrote yourself*).

You are encouraged to **submit early and often** in order to iron out any problems, especially issues with the format of the final output.

The performance of you classifier will be measured automatically; failure to format your output correctly may result in very low scores, which will not be changed.

For full credit, make sure to submit your assignment well before the deadline. The time of submission recorded by the system is the time used for determining late penalties. If your submission is received late, whatever the reason (including equipment failure and network latencies or outages), it will incur a late penalty.

If you have any issues with Vocareum with regards to logging in, submission, code not executing properly, etc., please make a post on Piazza so the instructional team can look into the issue.

Grading

After the due date, we will train your model on the combined training and development data, run your classifier on unseen test data twice (once with the vanilla model, and once with the averaged model), and compute the F1 score of your output compared to a reference annotation for each of the four classes (true, fake, positive, and negative). Your grade will be based on the performance of your classifier. We will calculate the mean of the four F1 scores for each model and scale it to the performance of a perceptron classifier developed by the instructional staff (so if that classifier has F1=0.8, then a score of 0.8 will receive a full credit, and a score of 0.72 will receive 90% credit;

your vanilla perceptron will be compared to a reference vanilla perceptron, and your averaged perceptron will be compared to a reference averaged perceptron). The overall grade will be the mean of the grades for the vanilla and averaged perceptrons.

Note that the measure for grading is the *macro-average* over classes; macro- and micro-averaging are explained in Manning, Raghavan and Schutze, Introduction to information retrieval, Chapter 13: Text classification and Naive Bayes. For more information on F1, see Manning, Raghavan and Schutze, Introduction to information retrieval, Chapter 8: Evaluation in information retrieval.

Notes

- **Problem formulation.** Since a perceptron is a binary classifier, you need to treat the problem as two separate binary classification problems (true/fake and positive/negative); each of the model files (vanilla and averaged) needs to have the model parameters for both classifiers.
- **Features and tokenization.** You'd need to develop some reasonable method of identifying features in the text; typical features are word tokens, but you may implement additional features such as word bigrams. Some common options for tokenization are removing certain punctuation, or lowercasing all the letters. To achieve reasonable runtime you will need to limit the number of features, so you may find it useful to ignore certain high-frequency or low-frequency tokens. You may use any tokenization method *which you implement yourselves*. Experiment, and choose whichever works best. **Important:** experimenting with features and tokenization should only take place after you have the basic program running correctly.
- **Stop words.** Some students like to have a list of stop words, that is tokens that the classifier should ignore. A list of words could help performance, but often doesn't; the only way to find out is through experimentation (this is why we only allow word lists that you develop yourselves). If you choose to develop a word list, do it after you get the basic logic of the program working to your satisfaction. Otherwise, you can't test if the word list helps or hurts, and the list just adds complexity to the code and makes debugging more difficult.
- **Runtime efficiency.** Vocareum imposes a limit on running times, and if a program takes too long, Vocareum will kill the process. Your program therefore needs to run efficiently. You need an efficient way to store the training instances, in order to avoid reading them over and over again (reading and parsing text is slow). Also, feature vectors for individual training instances are typically fairly sparse: for a reference solution with about 1000 features, the mean number of non-zero features per training instance is about 77; it would be highly inefficient to multiply and add all the 900+ zeros at every step. The reference solution stores the training data as a python dict indexed by the unique identifiers of the reviews, and the feature vector for each training instance as a dict of the form feature:count. With about 1000 features and

100 iterations (which is probably more than needed, due to overfitting), run times for the reference solution are under 5 seconds for running `perceplearn.py` on the training data, running on a MacBook Pro from 2016.

- **Overfitting.** The perceptron has a tendency to overfit the training data, so you should experiment in order to find out a good number of iterations to stop at.
- **Swapped labels.** If the results appear random, check the program to verify that it is printing out the intended labels. In the past we have seen student submissions where the classifier runs correctly, but then writes out "True" when it means positive, and "Pos" when it means true. It's a silly bug, but we've seen it happen more than once. This bug tends to happen in programs which store the labels is structures like `label[0]` or `label[1]`; we haven't encountered this bug in programs whose coding style uses `label[truthfulness]` or `label[positivity]`.

Collaboration and external resources

- This is an individual assignment. You may not work in teams or collaborate with other students. You must be the sole author of 100% of the code you turn in.
- You may not look for solutions on the web, or use code you find online or anywhere else.
- You may not download the data from any source other than the files provided on Blackboard, and you may not attempt to locate the test data on the web or anywhere else.
- You may use packages in the Python Standard Library, as well as numpy. You may not use any other packages.
- You may use external resources to learn basic functions of Python (such as reading and writing files, handling text strings, and basic math), but the extraction and computation of model parameters, as well as the use of these parameters for classification, must be your own work.
- Failure to follow the above rules is considered a violation of academic integrity, and is grounds for failure of the assignment, or in serious cases failure of the course.
- We use plagiarism detection software to identify similarities between student assignments, and between student assignments and known solutions on the web. **Any attempt to fool plagiarism detection, for example the modification of code to reduce its similarity to the source, will result in an automatic failing grade for the course.**
- Please discuss any issues you have on the Piazza discussion boards. Do not ask questions about the assignment by email; if we receive questions by email where the response could be helpful for the class, we will ask you to repost the question on the discussion boards.