XCS221 Assignment 1 — Sentiment Analysis

Due Sunday, November 22 at 11:59pm PT.

Guidelines

1. These questions require thought, but do not require long answers. Please be as concise as possible.
2. If you have a question about this homework, we encourage you to post your question on our Slack channel, at http://xcs221-scpd.slack.com/
3. Familiarize yourself with the collaboration and honor code policy before starting work.
4. For the coding problems, you may not use any libraries except those defined in the provided started code. In particular, ML-specific libraries such as `scikit-learn` are not permitted.

Submission Instructions

Written Submission: All students must submit an electronic PDF containing solutions to the written questions. As long as the submission is legible and well-organized, the course staff has no preference between a handwritten and a typeset LaTeXsubmission. Students wishing to typeset their documents should follow these recommendations:

- Type responses only in `submission.tex`.
- If you choose to submit a typeset document, please submit the compiled PDF, **not** `submission.tex`.
- Use the commented recommendations within the Makefile and README.md to get started.

Coding Submission: All assignment code is in the `src/` subirectory. You will submit only the `src/submission.py` file. Please only make changes between the lines containing `### START_CODE_HERE ###` and `### END_CODE_HERE ###`. Do not make changes to files other than `src/submission.py`.

The unit tests in `src/grader.py` will be used to autograde your submission. Run the autograder locally using the following terminal command within the `src/` subdirectory:

```
python grader.py
```

There are two types of unit tests used by our autograders:

- `basic`: These unit tests will verify only that your code runs without errors on obvious test cases.
- `hidden`: These unit tests will verify that your code produces correct results on complex inputs and tricky corner cases. In the student version of `src/grader.py`, only the setup and inputs to these unit tests are provided. When you run the autograder locally, these test cases will run, but the results will not be verified by the autograder.

For debugging purposes, a single unit test can be run locally. For example, you can run the test case `3a-0-basic` using the following terminal command within the `src/` subdirectory:

```
python grader.py 3a-0-basic
```

Before beginning this course, we highly recommend you walk through our Anaconda tutorial to familiarize yourself with our coding environment. Please use the env defined in `src/environment.yml` to run your code. This is the same environment used by our autograder.

Honor code

We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions independently, and without referring to written notes from the joint session. In other words, each student must understand the solution well enough in order to reconstruct it by him/herself. In addition, each student should write on the problem set the set of people with whom s/he collaborated. Further, because we occasionally reuse problem set questions from previous years, we expect students not to copy, refer to, or look at the solutions in preparing their answers. It is an honor code violation to intentionally refer to a previous year's solutions.

Introduction

Advice for this homework:

- In this context of this assignment, words are simply strings separated by whitespace. This includes case and punctuation (e.g. "great" and "Great" are considered different words).
- You might find some useful functions in `util.py`. Have a look around in there before you start coding.

1. **Sentiment Classification**

In this problem, we will build a binary linear classifier that reads movie reviews and guesses whether they are "positive" or "negative." In this problem, you must implement the functions without using libraries like Scikit-learn.

(a) **[5 points (Coding)]**

Implement the function `extractWordFeatures`, which takes a review (string) as input and returns a feature vector $\phi(x)$ (you should represent the vector $\phi(x)$ as a `dict` in Python).

(b) **[6 points (Coding)]**

Implement the function `learnPredictor` using stochastic gradient descent and minimize the hinge loss. Consider printing the training error and test error after each iteration to make sure your code is working. You must get less than 4% error rate on the training set and less than 30% error rate on the dev set to get full credit.

(c) **[4 points (Coding)]**

Create an artificial dataset for your `learnPredictor` function by writing the `generateExample` function (nested in the `generateDataset` function). Use this to double check that your `learnPredictor` works!

(d) **[3 points (Written)]**

When you run the grader.py on test case `3b-2`, it should output a `weights` file and a `error-analysis` file. Look through some example incorrect predictions and for five of them, give a one-sentence explanation of why the classification was incorrect. What information would the classifier need to get these correct? In some sense, there's not one correct answer, so don't overthink this problem. The main point is to convey intuition about the problem.

(e) **[4 points (Coding)]**

Now we will try a crazier feature extractor. Some languages are written without spaces between words. So is splitting the words really necessary or can we just naively consider strings of characters that stretch across words? Implement the function `extractCharacterFeatures` (by filling in the `extract` function), which maps each string of n characters to the number of times it occurs, ignoring whitespace (spaces and tabs).

(f) **[3 points (Written)]**

Run your linear predictor with feature extractor `extractCharacterFeatures`. Experiment with different values of n to see which one produces the smallest test error. You should observe that this error is nearly as small as that produced by word features. How do you explain this?

Construct a review (one sentence max) in which character n-grams probably outperform word features, and briefly explain why this is so.

Note: You should replace the `featureExtractor` in `test3b2()` in `grader.py`, i.e., let `featureExtractor = submission.extractCharacterFeatures(\_\_)` and report your results. Don't forget to recover `test3b2()` after finishing this question.