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 version of the written questions. We highly recommend typesetting your solutions via LaTeX, though it is not required. If you choose to hand write your responses, please make sure they are well organized and legible when scanned. The source LaTeXfor all problem sets is available on GitHub.

Coding Submission: You should modify the code in `submission.py` between `START CODE HERE` and `END CODE HERE`, but you can add other helper functions outside this block if you want. Do not make changes to files other than `submission.py`. When finished, upload only `submission.py` to gradescope.

Your code will be evaluated on two types of test cases, **basic** and **hidden**, which you can see in `grader.py`. Basic tests, which are fully provided to you, do not stress your code with large inputs or tricky corner cases. Hidden tests are more complex and do stress your code. The inputs of hidden tests are provided in `grader.py`, but the correct outputs are not. To run the tests, you will need to have `graderUtil.py` in the same directory as your code and `grader.py`. Then, you can run all the tests by typing `python grader.py`. This will tell you only whether you passed the basic tests. On the hidden tests, the script will alert you if your code takes too long or crashes, but does not say whether you got the correct output. You can also run a single test (e.g., `3a-0-basic`) by typing `python grader.py 3a-0-basic`. We strongly encourage you to read and understand the test cases, create your own test cases, and not just blindly run `grader.py`. You should make sure to (1) restrict yourself to only using libraries included in the starter code, and (2) make sure your code runs without errors.

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.

Advice for this homework:

- Words are simply strings separated by whitespace. Note that words which only differ in capitalization are considered separate (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) **[2 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) **[3 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) **[2 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) **[0 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) **[2 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) **[1 point (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.*