Assignment 2. Sentiment Analysis

Hwanjo Yu
CSED342 - Artificial Intelligence

Contact: TA Seungha Hong (shhong97@postech.ac.kr)
Deadline: 21 Mar 2023 at 2:00 pm. (50% penalty for every 1 day)

General Instructions

You should write answers in `submission.py` between

    # BEGIN_YOUR_ANSWER

and

    # END_YOUR_ANSWER

In **Problem 1** and **Problem 3a** you just need to provide your answers.

In **Problem 2** and **Problem 3b** you have to implement some functions. You can add other helper functions outside the answer block if you want. Do not make changes to files other than `submission.py`.

Please use **Python 3.9** to develop your code. You have to write answers in the same format as provided in `submission.py`.

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 all the tests, type

    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`.

Advice for this homework:

- Words are simply strings separated by whitespace. Don't normalize the capitalization of words (treat *great* and *Great* as different words).

- You might find some useful functions in `util.py`. Have a look around in there before you start coding.

Problems

Problem 1. Hinge Loss

Here are two reviews of "Frozen," courtesy of Rotten Tomatoes (no spoilers!):

Rotten Tomatoes has classified these reviews as "positive" and "negative," respectively, as indicated by the in-tact tomato on the left and the splattered tomato on the right. In this assignment, you will create a simple text classification system that can perform this task automatically.

Problem 1a [2 points]

We'll warm up with the following set of four mini-reviews, each labeled positive (+1) or negative (−1):

- (+1) so touching

- (+1) quite impressive

- (−1) not impressive

- (−1) quite boring

Each review x is mapped onto a feature vector $\phi(x)$, which maps each word to the number of occurrences of that word in the review. For example, the first review maps to the (sparse) feature vector $\phi(x) = \{\text{so} : 1, \text{touching} : 1\}$. Recall the definition of the hinge loss:

$$\text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = \max\{0, 1 - \mathbf{w} \cdot \phi(x)y\},$$

where y is the correct label.

Suppose we run stochastic gradient descent, updating the weights according to

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}),$$

once for each of the four examples in order. After the classifier is trained on the given four data points, <u>what are the weights of the six words</u> ('so', 'touching', 'quite', 'impressive', 'not', 'boring') that appear in the above reviews? Use $\eta = 1$ as the step size and initialize $\mathbf{w} = [0, ..., 0]$. Assume that $\nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = 0$ when the margin is exactly 1.

Problem 2: Sentiment Classification

In this problem, we will build a binary linear classifier that reads movie reviews and guesses whether they are "positive" or "negative."

Problem 2a [2 points]

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).

Problem 2b [8 points]

We're going to train a linear predictor, which can be represented by a logistic regression model. Here is the definition of linear predict:

$$f_w(x) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \phi(x) > 0 \\ -1 & \text{if } \mathbf{w} \cdot \phi(x) < 0 \end{cases}$$
$$= \begin{cases} +1 & \text{if } \sigma(\mathbf{w} \cdot \phi(x)) > 0.5 \\ -1 & \text{if } \sigma(\mathbf{w} \cdot \phi(x)) < 0.5 \end{cases}$$

where σ is a logistic(or sigmoid) function.

Your task is to implement the function *learnPredictor* using stochastic gradient descent, minimizing the negative log-likelihood loss (NLL) defined as:

$$\text{Loss}_{\text{NLL}}(x, y, \mathbf{w}) = -\log(p_{\mathbf{w}}(y \mid x))$$

$$p_{\mathbf{w}}(y \mid x) = \begin{cases} \sigma(\mathbf{w} \cdot \phi(x)) & \text{if } y = 1 \\ 1 - \sigma(\mathbf{w} \cdot \phi(x)) & \text{if } y = -1 \end{cases}$$

You should first derive $\nabla_{\mathbf{w}} \text{Loss}_{\text{NLL}}(x, y, \mathbf{w})$, then exploit the formula to update weights for each example. Also, you can print the training error and test error after each iteration through the data, so it's easy to see if your code is working.

Problem 2c [3 points]

The previous features include unigram(single) words only, which cannot consider the context of a word in an utterance. In this task, we'll incorporate n-gram words into features. In other words, feature vector will counts number of occurrences of consecutive words of length n. Implement *extractNgramFeatures* which extract n-gram word features.

Problem 3: K-means Clustering

Problem 3a [2 points]

Suppose we have a feature extractor ϕ that produces 2-dimensional feature vectors, and a toy dataset $\mathcal{D}_{\text{train}} = \{x_1, x_2, x_3, x_4\}$ with

1. $\phi(x_1) = [-1, 0]$

2. $\phi(x_2) = [2, 0]$

3. $\phi(x_3) = [0, 3]$

4. $\phi(x_4) = [4, 3]$

Run 2-means on this dataset. What are the final cluster centers μ? Run this algorithm until it converges, with initial centers:

1. $\mu_1 = [-2, 0]$ and $\mu_2 = [3, 0]$

2. $\mu_1 = [-1, -1]$ and $\mu_2 = [2, 3]$

Problem 3b [6 points]

Implement the *kmeans* function. You should initialize your k cluster centers to random elements of *examples*.

After a few iterations of k-means, your centers will be very dense vectors. In order for your code to run efficiently and to obtain full credit, you may need to precompute certain quantities. As a reference, our code runs in under a second on all test cases. You might find *generateClusteringExamples* in `util.py` useful for testing your code.