Assignment 4. Machine Learning

Gary Geunbae Lee
CSED442 - Artificial Intelligence

Contact: TA Seonghyeon Lee (sh0416@postech.ac.kr)

General Instructions

This (and every) assignment has a written part and a programming part.

✎ This icon means a written answer is expected in `sentiment.pdf`.

⌨ This icon means you should write code in `submission.py`.

You should modify the code in `submission.py` between

```
# BEGIN_YOUR_CODE
```

and

```
# END_YOUR_CODE
```

but you can add other helper functions outside this block if you want. Do not make changes to files other than `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`.

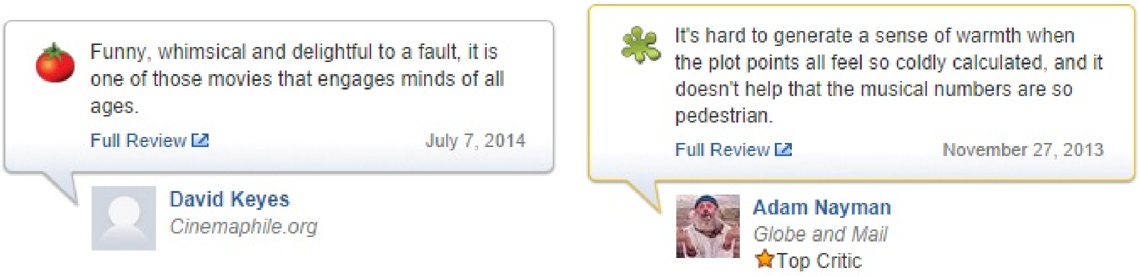
Advice for this homework:

- 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. Warmup

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



Funny, whimsical and delightful to a fault, it is one of those movies that engages minds of all ages.

Full Review July 7, 2014

David Keyes
Cinemaphile.org

It's hard to generate a sense of warmth when the plot points all feel so coldly calculated, and it doesn't help that the musical numbers are so pedestrian.

Full Review November 27, 2013

Adam Nayman
Globe and Mail
Top Critic

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.

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

- (+1) pretty good
- (−1) bad plot
- (−1) not good
- (+1) pretty scenery

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 $\phi(x) = \{\text{pretty} : 1, \text{good} : 1\}$. 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.

Problem 1a [2 points] ✎

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, what are the weights of the six words ('pretty', 'good', 'bad', 'plot', 'not', 'scenery') 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 1b [2 points] ✎

Create a small labeled dataset of four mini-reviews using the words 'not', 'good', and 'bad', where the labels make intuitive sense. Each review should contain one or two words, and no repeated words. Prove that no linear classifier using word features can get zero error on your dataset. Remember that this is a question about classifiers, not optimization algorithms: your proof should be true for any linear classifier, regardless of how the weights are learnt. After providing such a dataset, propose a single additional feature that we could augment the feature vector with that would fix this problem.

Hint: think about the linear effect that each feature has on the classification score.

Problem 2: Predicting Movie Ratings

Suppose that we are now interested in predicting a numeric rating for each movie review. We will use a non-linear predictor that takes a movie review x and returns $\sigma(\mathbf{w} \cdot \phi(x))$, where $\sigma(z) = (1 + e^{-z})^{-1}$ is the logistic function that squashes a real number to the range $[0, 1]$. Suppose that we wish to use the squared loss.

Problem 2a [2 points] ✎

Write out the expression for $\text{Loss}(x, y, \mathbf{w})$.

Problem 2b [3 points] ✎

Compute the gradient of the loss.
Hint: you can write the answer in terms of the predicted value $p = \sigma(\mathbf{w} \cdot \phi(x))$.

Problem 2c [3 points] ✎

Assuming $y = 0$, what is the smallest magnitude that the gradient can take? That is, find a way to set $\mathbf{w}$ to make $\|\nabla \text{Loss}(x, y, \mathbf{w})\|$ as small as possible. You are allowed to let the magnitude of $\mathbf{w}$ go to infinity.

Hint: try to understand intuitively what is going on and the contribution of each part of the expression. If you find doing too much algebra, you're probably doing something suboptimal.

Motivation: the reason that we're interested in the magnitude of the gradients is because it governs how far gradient descent will step. For example, if the gradient is close to zero when $\mathbf{w}$ is very far from the origin, then it could take a long time for gradient descent to reach the optimum (if at all); this is known as the vanishing gradient problem in training neural networks.

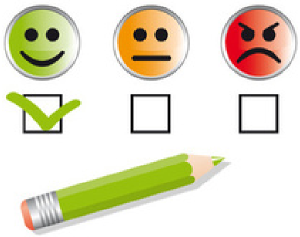
Problem 2d [3 points] ✎

Assuming $y = 0$, what is the largest magnitude that the gradient can take? Leave your answer in terms of $\|\phi(x)\|$.

Problem 2e [3 points] ✎

The problem with the loss function we have defined so far is that is it is non-convex, which means that gradient descent is not guaranteed to find the global minimum, and in general these types of problems can be difficult to solve. So let us try to reformulate the problem as plain old linear regression. Suppose you have a dataset $\mathbf{D}$ consisting of (x, y) pairs, and that there exists a weight vector $\mathbf{w}$ that yields zero loss on this dataset. Show that there is an easy transformation to a modified dataset $\mathbf{D}'$ of (x, y') pairs such that performing least squares regression (using a linear predictor and the squared loss) on $\mathbf{D}'$ converges to a vector $\mathbf{w}^*$ that yields zero loss on $\mathbf{D}'$. Concretely, write an expression for y' in terms of y and justify this choice. This expression should not be a function of $\mathbf{w}$. For this part of the problem, assume that y is a real valued variable in the range (0, 1).

Problem 3: 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 3a [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 3b [4 points] ⌨

Implement the function *learnPredictor* using stochastic gradient descent, minimizing the hinge loss. Print the training error and test error after each iteration through the data, so it's easy to see if 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.

Problem 3c [2 points] ⌨

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!

Problem 3d [2 points] ✏

When you run the `grader.py` on test case *3b-2*, it should output a *weights* file and a *error-analysis* file. Look through 10 example incorrect predictions and for each one, 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 get you to get intuition about the problem.

Problem 3e [2 points] ⌨

Now we will try a crazier feature extractor. Some languages are written without spaces between words. But is this step 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).

Problem 3f [3 points] ✏

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.

Problem 4: K-means Clustering

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. $\phi(x_1) = [0, 0]$
2. $\phi(x_2) = [0, 1]$
3. $\phi(x_3) = [2, 0]$
4. $\phi(x_4) = [2, 2]$

Problem 4a [2 points] ✎

Run 2-means on this dataset. Please show your work. What are the final cluster assignments z and cluster centers μ? Run this algorithm twice, with initial centers:

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

Problem 4b [5 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 will need to precompute certain quantities. As a reference, our code runs in under a second on Myth, on all test cases. You might find *generateClusteringExamples* in `util.py` useful for testing your code.

Problem 4c [5 points] ✎

Sometimes, we have prior knowledge about which points should belong in the same cluster. Suppose we are given a set S of example pairs (i, j) which must be assigned to the same cluster. For example, suppose we have 5 examples; then $S = \{(1, 2), (1, 4), (3, 5)\}$ says that examples 1, 2, 4 must be in the same cluster and that examples 3 and 5 must be in the same cluster. Provide the modified k-means algorithm that performs alternating minimization on the reconstruction loss.