

Assignment 2. Sentiment analysis

Prof. Hwanjo Yu
CSED342 - Artificial Intelligence

Contact: Kawon Jeong (tree468@postech.ac.kr)

Deadline: March 17, 2025, at 23:59 (50% penalty per day of delay)

General Instructions

Assignment 2 consists of four problems where Problems 1-2 are written and Problem 3-4 are programming tasks, with each problem's score indicated alongside it. For the written problems, please submit your answers and solutions in PDF format. You may create the PDF by converting from Overleaf or by taking clear, legible photos of your handwritten solutions and combining them into a single PDF; any issues arising from illegible handwriting are solely your responsibility.

For the programming problem, please note that the assignment has been developed in Python 3.12, so please use **Python 3.12** to implement your code. You must submit the programming file as **submission.py** without changing the file name.

Ultimately, your final submission must be a {YOUR_STUDENT_ID}.zip file that contains both {YOUR_STUDENT_ID}.pdf (your written solution) and **submission.py**.

When working on your code, please make your modifications only within the designated section in **submission.py**—that is, between the markers

```
# BEGIN_YOUR_ANSWER
```

and

```
# END_YOUR_ANSWER
```

You can add other helper functions outside the answer block if you want, but do not import other libraries and do not make changes to files other than **submission.py**. You can solve all the problems without any libraries other than *random* and *numpy*.

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

Problem 1 [10 points]

Here are two reviews of *Perfect Blue*, from Rotten Tomatoes:

	Panos Kotzathanasis <i>Asian Movie Pulse</i>	 "Perfect Blue" is an artistic and technical masterpiece; however, what is of utmost importance is the fact that Satoshi Kon never deteriorated from the high standards he set here, in the first project that was entirely his own.	January 26, 2020
Full Review			

	Derek Smith <i>Cinematic Reflections</i>	 [An] nime thriller [that] often plays as an examination of identity and celebrity, but ultimately gets so lost in its own complex structure that it doesn't end up saying much at all.	August 19, 2006
Full Review Original Score: 2/4			

Rotten Tomatoes has classified these reviews as “positive” and “negative,” respectively, as indicated by the intact tomato on the top and the splatter on the bottom. 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. (−1) disappointing story
2. (−1) very bored
3. (+1) interesting story
4. (+1) very fresh

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 second review maps to the (sparse) feature vector $\phi(x) = \{\text{very} : 1, \text{bored} : 1\}$. 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 x is the review text, y is the correct label, \mathbf{w} is the weight vector.

Suppose we run stochastic gradient descent once for each of the 4 samples in the order given above, updating the weights according to

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

After the updates, what are the weights of the six words (“very”, “story”, “disappointing”, “interesting”, “bored”, “fresh”) that appear in the above reviews?

- Use $\eta = 0.1$ as the step size.
- Initialize $\mathbf{w} = [0, 0, 0, 0, 0, 0]$.
- The gradient $\nabla_{\mathbf{w}} \text{Loss}_{\text{hinge}}(x, y, \mathbf{w}) = 0$ when margin is exactly 1.

The answer should be a weight vector that contains a numerical value for each of the word in the reviews (“very”, “story”, “disappointing”, “interesting”, “bored”, “fresh”), in this order.

Problem 2 [10 points]

Suppose that we are now interested in predicting a numeric rating for movie reviews. 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 $(0, 1)$. For this problem, assume that the movie rating y is a real-valued variable in the range $[0, 1]$. **Do not** use math software such as Wolfram Alpha to solve this problem.

Problem 2a [3 points]

Suppose that we wish to use **squared loss**. Write out the expression for $\text{Loss}(x, y, \mathbf{w})$ for a single datapoint (x, y) . Feel free to use σ in the expression.

Problem 2b [7 points]

Given $\text{Loss}(x, y, \mathbf{w})$ from the previous part, compute the gradient of the loss with respect to \mathbf{w} , $\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$. Write the answer in terms of the predicted value $p = \sigma(\mathbf{w} \cdot \phi(x))$.

Problem 3 [10 points]

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

Do not import *numpy* and any additional libraries other than *random* for this section. Hint: Look at the provided ‘util.py’ for some helpful utility functions that you are able to use when implementing your code. Throughout this problem, avoid modifying Python dictionaries directly; instead, use the provided utility methods ‘dotProduct’ and ‘increment’.

Problem 3a [3 points]

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

Problem 3b [7 points]

Implement the function `learnPredictor` using stochastic gradient descent and minimize the **hinge loss**. Print the training error and validation error after each epoch 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 validation set through `test3bc` (in `grader.py`) to get full credit.

Problem 4 [10 points]

In this problem, we are now interested in predicting a numeric rating for movie reviews using a non-linear predictor with Multi-Layer Perceptron (MLP). The predictor consists of two layers (a hidden layer and an output layer), and each layer has a sigmoid activation function. You should train the predictor with the **squared loss** using stochastic gradient descent. Note that you are not allowed to use Python libraries that support automatic differentiation (e.g., PyTorch, TensorFlow). Use Numpy to implement your codes.

Suppose we have a training dataset $\mathcal{D}_{\text{train}} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)\}$ and a feature extractor ϕ with

1. $\phi(x_1) = [0, 1, 0, 0, 1, 0]$, $y_1 = 0$
2. $\phi(x_2) = [1, 0, 1, 0, 0, 0]$, $y_2 = 0$
3. $\phi(x_3) = [0, 1, 0, 1, 0, 0]$, $y_3 = 1$
4. $\phi(x_4) = [1, 0, 0, 0, 0, 1]$, $y_4 = 1$.

Problem 4a [3 points]

Implement the `forward` function in `MLPPredictor` to predict the probability of $y = 1$ using 6-dimensional features with sigmoid activation. Do not modify the `init` function of `MLPPredictor` provided. Note that you need to save intermediate activations within the `forward` function for backpropagation (Problem 4b).

Problem 4b [3 points]

Implement the `loss` and `backward` functions in `MLPPredictor` to backpropagate the squared loss to network parameters: $W1$, $b1$, $W2$, and $b2$. Utilize the saved activations from the forward process to compute the gradient for each parameter.

Hint: Chain rules can be used to derive the gradients of each network parameter.

Problem 4c [4 points]

Implement the `update` and `train` functions in `MLPPredictor` to train the network using stochastic gradient descent. Within the `train` function, you may need to call the `forward`, `loss`, `backward`, and `update` functions you implemented in the previous problems. Use the batch size of 1 for stochastic gradient descent and do not shuffle the training data within the `train` function.