Homework 5

ESE 4020/5420

Due on Thursday 11/21/2024 at 11:59pm

<u>Important note</u>: **To get full mark on this problem set, you only need to solve one of the questions 2 and 5. You can solve both for extra credit that will count towards your total homework grade.**

For Problem 1, no other package except numpy and matplotlib should be used (using functions from any other package such as sklearn will result in 0 points). **For problem 1 please use the coding template in Canvas.** For other problems, you can use the packages of your choice.

Problem 1. (Numerically Solving Logistic Regression) We have learned that in logistic regression, the coefficients β_0 and $\boldsymbol{\beta}$ can be found using MLE estimates. It is easy to see that the Log Likelihood function for m data points can be written as

$$\ln \mathcal{L}(\beta_0, \boldsymbol{\beta}) = -\sum_{i=1}^{m} \ln\left(1 + e^{-y_i(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i)}\right).$$

In general, there is no closed form expression for the optimal $\beta_0, \boldsymbol{\beta}$ that maximize the above log-likelihood. Hence, we will use an iterative algorithm to solve for the coefficients.

Observe that in general, $\max_\alpha f(\alpha) = \min_\alpha -f(\alpha)$. Hence, we will minimize the negative-log-likelihood,

$$L(\beta_0, \boldsymbol{\beta}) = -\frac{1}{m} \ln \mathcal{L}(\beta_0, \boldsymbol{\beta}) = \frac{1}{m} \sum_{i=1}^{m} \ln\left(1 + e^{-y_i\left(\beta_0 + \boldsymbol{\beta}^\top \mathbf{x}_i\right)}\right)$$

which serves as our loss function. Note that our input data $\boldsymbol{x}_i \in \mathbb{R}^d$ are now d-dimensional vectors, and $\boldsymbol{\beta} \in \mathbb{R}^d$ is also a d-dimensional vector. Hence, our model has a total of $d + 1$ parameters (including β_0).

Our objective is to iteratively find $\beta_0, \boldsymbol{\beta}$ that decrease this loss at each step, i.e. we want to find a sequence of iterates $\{\beta_0^{(k)}, \boldsymbol{\beta}^{(k)}\}_{k=1}^{K}$ such that $L(\beta_0^{(k+1)}, \boldsymbol{\beta}^{(k+1)}) \leq L(\beta_0^{(k)}, \boldsymbol{\beta}^{(k)})$. We will do this using gradient descent.

In this problem we will be working with real image data where the goal is to classify if the image is 0 or 1 using logistic regression.
The input $X \in \mathbb{R}^{m \times d}$, is a matrix, where a single data point $\mathbf{x}_i \in \mathbb{R}^d$ with $d = 784$. The labels are contained in a vector $y \in \mathbb{R}^m$, where each label $y_i \in \{0, 1\}$.

(a) Coding Question:

- Load the data and visualize one input data point as an image for each of label 0 and label 1. The data should be reshaped back to $[28 \times 28]$ to be able to visualize it.

  Hint: use plt.imshow

- The data is in between 0 to 255. Normalize the data to [0, 1].

- Set y_i = 1 for images labeled 0 and y_i = -1 for images labeled 1. Split the data randomly into train and test with a ratio of 80:20.

- Initialize the coefficients (iterates) $\beta_0^{(1)}, \boldsymbol{\beta}^{(1)}$ using a normal distribution of mean 0 and variance 1. (Hint: for $\boldsymbol{\beta}^{(1)}$, you can initialize all d entries to be $\mathcal{N}(0, 1)$).

- Write a function that computes the loss. Note:

$$L(\beta_0, \boldsymbol{\beta}) = \frac{1}{m} \sum_{i=1}^{m} \ln \left(1 + e^{-y_i\left(\beta_0 + \sum_{j=1}^{d} \boldsymbol{\beta}_j \mathbf{x}_{i,j}\right)}\right)$$

  where $\boldsymbol{x}_{i,j}$ is the j-th entry of data point $\boldsymbol{x}_i$.

- To minimize the loss function, we will use gradient descent. We can write the gradients of loss function as

$$\frac{\partial L}{\partial \beta_0} = -\frac{1}{m} \sum_{i=1}^{m} \frac{e^{-y_i \cdot (\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i)}}{1 + e^{-(y_i \cdot (\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i))}} y_i = d\beta_0$$
$$\nabla_{\boldsymbol{\beta}} L = -\frac{1}{m} \sum_{i=1}^{m} \frac{e^{-y_i \cdot (\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i)}}{1 + e^{-(y_i \cdot (\beta_0 + \boldsymbol{\beta}^\top \boldsymbol{x}_i))}} y_i \boldsymbol{x}_i = d\boldsymbol{\beta}$$

  Write a function to compute the gradients.

- Update the parameters as

$$\boldsymbol{\beta} = \boldsymbol{\beta} - 0.05 * d\boldsymbol{\beta}$$
$$\beta_0 = \beta_0 - 0.05 * d\beta_0$$

  (Gradient updates should be computed based on the train set.)

- Repeat the process for 50 iterations and report the loss after the 50th iterate.

- Plot the loss for each iteration for the train and test sets.

- For the classification rule derived compute the accuracy on the test set for each iteration and plot the accuracy.

This format may help you write your code:

```
import numpy as np
from matplotlib import pyplot as plt

def compute_loss(data, labels, B, B_0):

    return logloss

def compute_gradients(data, labels, B, B_0):

    return dB, dB_0

if __name__ == '__main__':
    x = np.load(data)
    y = np.load(label)

    ## Split the data to train and test
    x_train, y_train, x_test, y_test = #split_data

    B = np.random.randn(1, x.shape[1])
    B_0 = np.random.randn(1)

    lr = 0.05

    for _ in range(50):

        ## Compute Loss
        loss = compute_loss(x_train, y_train, B, B_0)

        ## Compute Gradients
        dB, dB_0 = compute_gradients(x_train, y_train, B, B_0)

        ## Update Parameters
        B = B - lr*dB
        B_0 = B_0 - lr*dB_0

        ##Compute Accuracy and Loss on Test set (x_test, y_test)
        accuracy_test =
        loss_test =

   ##Plot Loss and Accuracy
```

It may help if you vectorize your code. Fifty iterations should run in 10 seconds or less.

Congratulations on finishing the coding questions for this part! There are two more written questions.

(b) Written Questions:

- Why is random splitting better than sequential splitting in our case?
- Logistic regression is a classification problem. We classify as +1 if $P(Y = 1|X) \geq 0.5$. Derive the classification rule for the threshold 0.5.

Problem 2. Recall that in classification we assume that each data point is an i.i.d. sample from a (unknown) distribution $P(X = x, Y = y)$. In this question, we are going to design the data distribution P and evaluate the performance of logistic regression on data generated using P. Keep in mind that we would like to make P as simple as we could. In the following, we assume $x \in \mathbb{R}$ and $y \in \{0, 1\}$, i.e. the data is one-dimensional and the label is binary. Write $P(X = x, Y = y) = P(X = x)P(Y = y|X = x)$. We will generate $X = x$ according to the uniform distribution on the interval $[0, 1]$ (thus $P(X = x)$ is just the pdf of the uniform distribution).

(a) (Written) Design $P(Y = y|X = x)$ such that (i) $P(y = 0) = P(y = 1) = 0.5$; and (ii) the classification accuracy of any classifier is at most 0.9; and (iii) the accuracy of the Bayes optimal possible classifier is at least 0.8.

(b) (Coding) Using Python, generate $n = 100$ training data points according to the distribution you designed above and train a binary classifier using logistic regression on training data.

(c) (Coding) Generate and $n = 100$ test data points according to the distribution you designed in part 1 and compute the prediction accuracy (on the test data) of the classifier that you designed in part 2. Also, compute the accuracy of the Bayes optimal classifier on the test data. Why do you think Bayes optimal classifier is performing better?

(d) (Coding) Redo parts (b), (c) with $n = 1000$. Are the results any different? Why?

Problem 3. *Cross Validation.* Load the dataset poly_data.csv. The first column is a vector of inputs x and the second column is a vector of responses y. Suppose we believe it was generated by some polynomial of the inputs with Gaussian error, i.e. for some (unknown) p,

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_p x_i^p + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$. We would like to recover the true coefficients of the underlying process. A polynomial regression can be estimated by including all powers of x as predictors in the model. However, the problem is that we don't know what the true value of p is. We will use k-fold cross validation to solve this problem.

(a) Pick a set of polynomial models, i.e. all polynomials of degree 1 to degree 40. Compute the k-fold cross validation error (mean squared error) for each of these models. Report the value of k that you use and plot the cross-validation error as a function of polynomial degree. Which polynomial degree fit the data the best?

(b) Choose the best polynomial model obtained from the previous part, and use to it regress the entire dataset. Report the polynomial coefficients and make a scatter plot of the x_i's and y_i's with your fitted polynomial.

Problem 4. *Explore-Then-Commit.* In this problem, we will try to theoretically analyze the regret of the Explore-Then-Commit algorithm for multi-armed bandits.

Consider a two-armed bandit. The reward given by pulling each arm is as follow. Assume that pulling the first arm will give a reward drawn i.i.d. from Bern(μ_1) and pulling the second arm will give a reward drawn i.i.d. from Bern(μ_2), with $\Delta = \mu_1 - \mu_2 > 0$. We will denote the reward recieved at time t by r_t.

The Explore-Then-Commit algorithm goes as follows. Let m be a positive integer. The parameter m characterizes the length of the *exploration phase* of the algorithm:

Exploration Phase The algorithm first pulls the first arm for m rounds and then the second arm for m rounds. Then, estimates the unknown values μ_1 and μ_2 as follows:

$$\hat{\mu}_1 = \frac{1}{m}\sum_{t=1}^{m} r_t, \quad \hat{\mu}_2 = \frac{1}{m}\sum_{t=m+1}^{2m} r_t.$$

The algorithm will then use these estimates to determine the best arm. Let

$$a^* = \arg\max_{i\in\{1,2\}} \hat{\mu}_i,$$

be the arm with the larger estimated mean in the exploration phase.

Exploitation Phase In this phase, the algorithm will only pull the arm a^* for the rest of the $T - 2m$ rounds, where T is the total number of rounds.

(a) Let T_1 be the number of times that arm 1 is pulled and T_2 be the number of times that arm 2 is pulled. Write down the expected regret $\mathbb{E}[R_T]$ of the algorithm in terms of $\mathbb{E}[T_1], \mathbb{E}[T_2]$, and Δ.

(b) Use the Hoeffding inequality to show that $\mathbb{P}[\,\hat{\mu}_2 > \hat{\mu}_1\,] \le 2e^{-m\Delta^2/2}$.

(c) Use part (b) to show that $\mathbb{E}[T_2] \le m + 2(T-2m)e^{-m\Delta^2/2}$.

(d) Using the bounds derived in the previous sections, show that

$$\mathbb{E}[R_T] \le m\Delta + 2T\Delta e^{-m\Delta^2/2}.$$

(e) Find the m that minimizes this regret bound.

(Extra Points) Show that for the optimal m, we have

$$\mathbb{E}[R_T] \leq \frac{C_1}{\Delta} + \frac{C_2 \log(T\Delta^2)}{\Delta},$$

where C_1 and C_2 are two constants. Assume that $1 > \Delta > \frac{1}{\sqrt{T}}$. By finding the Δ that maximizes the second term, show that

$$\mathbb{E}[R_T] \leq 1 + C\sqrt{T},$$

where C is some constant (some number).

Problem 5. *Comparing Bandit Algorithms.* In this problem, we will implement the UCB and $\epsilon-$greedy algorithm in Python and compare their regret using numerical simulations. Consider a 4-armed bandit problem. The reward after pulling each arm is drawn i.i.d. from the following distributions: Arm 1: Bern(0.02), Arm 2: Bern(0.1), Arm 3: Bern(0.25), and Arm 4: Bern(0.4). Implement the UCB, and the ϵ-greedy algorithm.

Run these algorithms on the given 4-armed bandit problem for $T = 5000$ rounds, and compute the regret of each algorithm at each round. Repeat this process 1000 times and compute the average regret (over this 1000 trials) of each algorithm at each round. Plot the average regret for each algorithm as a function of rounds.