Homework 5

ESE 402/542

Due on 12/01/2021 at 11:59pm

(For Problems 1 and 2, no other package except numpy and matplotlib should be used for the programming questions. For problem 3 you can use the packages of your choice.)

Problem 1.

(a) In this problem we will analyze logistic regression learned in class.
Sigmoid function can be written as $S(x) = \frac{1}{1+e^{-x}}$

- For a given variable X assume $P(Y = +1|X)$ is modeled as $P(Y = +1|X) = S(\beta_0 + \beta_1 X)$. Plot a 3d figure showing the relation between output and variable β_0 and β_1 when X = 1. Take values between [-2, 2] for both β_0 and β_1 with a step size of 0.1 to plot the 3d plot.

(b) In class, we have done binary classification with labels $Y = \{0, 1\}$. In this problem, we will be using the labels as $Y = \{-1, 1\}$ as it will be easier to derive the likelihood of the $P(Y|X)$.

- Show that if Y $\in \{-1, 1\}$ the probability of Y given X can be written as

$$P(Y|X) = \frac{1}{1 + e^{-y(\beta_0+\beta_1\mathbf{x})}}$$

- We have learned that the coefficients β_0 and β_1 can be found using MLE estimates. Show that the Log Likelihood function for m data points can be written as

$$\ln \mathcal{L}(\beta_0, \beta_1) = -\sum_{i=1}^{m} \ln\left(1 + e^{-y_i(\beta_0+\beta_1\mathbf{x}_i)}\right)$$

- Plot a 3d figure showing the relation between Log Likelihood function and variable β_0 , β_1 when $X = 1, Y = -1$ and $X = 1, Y = 1$. Take values between $[-2, 2]$ for both β_0 and β_1 with a step size of 0.1 to plot the 3d plot.

- Based on the graph, is it possible to maximize this function?

Problem 2. (Numerically Solving Logistic Regression) In general, there is no closed form expression for the optimal β_0, β_1 that maximize the above log-likelihood in Problem 1. 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,

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

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 $y_i \in \{0, 1\}$.

- 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 x 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.
  Why is random splitting better than sequential splitting in our case?

- 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 epoch.
- Plot the loss for each iteration for the train and test sets
- 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. (Not a programming question)
- 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. Ideally 50 iterations should run in 10 seconds or less.

Problem 3. 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).

1. 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.
2. 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.
3. 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?
4. Redo parts 2,3 with $n = 1000$. Are the results any different than part 3? Why?

Problem 4.

K-means clustering can be viewed as an optimization problem that attempts to minimize some objective function. For the given objectives, determine the update rule for the centroid, c_k of the k-th cluster C_k . In other word, find the optimal c_k that minimizes the objective function. The data x contains p features.

1. Show that setting the objective to the sum of the squared Euclidean distances of points from the center of their clusters,

$$\sum_{k=1}^{K} \sum_{x \in C_k} \sum_{i=1}^{p} (c_{ki} - x_i)^2$$

   results in an update rule where the optimal centroid is the mean of the points in the cluster.

2. Show that setting the objective to the sum of the Manhattan distances of points from the center of their clusters,

$$\sum_{k=1}^{K} \sum_{x \in C_k} \sum_{i=1}^{p} |c_{ki} - x_i|$$

   results in an update rule where the optimal centroid is the median of the points in the cluster.