Homework Assignment 4

COGS 118A: Introduction to Machine Learning I

Due: 11:59pm, Sunday, Nov. 4th, 2018 (Pacific Time).

Instructions: Use **Jupyter Notebook** to answer the questions below, include all your code and figures in the notebook. Export the notebook as PDF file and submit it on Gradescope. You may look up the information on the Internet, but you must write the final homework solutions by yourself.

Please Note: Writing homework without using Jupyter Notebook will lead to point deduction.

Late Policy: 5% of the total points will be deducted on the first day past due. Every 10% of the total points will be deducted for every extra day past due.

Grade: ____ out of 100 points

1 (35 points) Parabola Estimation

We are given the data $S = \{(x_i, y_i), i = 1, \ldots, n\}$. Here, $x_i, i = 1, \ldots, n$ are one dimensional scalars. We will try to fit the data with a parabola, i.e. $y_i = w_1 x_i + w_2 x_i^2 + w_3$. To solve this problem with matrix manipulation, we represent the data as matrices $X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T$ and $Y = [y_1, y_2, \ldots, y_n]^T$, where $\mathbf{x}_i$ is a feature vector corresponding to the data x_i: $\mathbf{x}_i = [1, x_i, x_i^2]^T$ in this problem. $W = [w_0, w_1, w_2]^\top$. We are finding W that minimizes the sum-of-squares error function $g(W)$.

Please download `HW4_2018.ipynb` from website

1.1 L2 norm (5 points)

Consider L_2 norm as your loss function:

$$g(W) = \|XW - Y\|_2^2 = (XW - Y)^T (XW - Y). \tag{1}$$

Use the **closed form solution** to compute W and plot the scatter graph of data and estimated parabola. Report the parabola function and the figures.

Ans:

$$y = 51.07 - 16.06x + 2.36x^2$$

```
X_and_Y = np.load('./hw4-q1-parabola.npy')
X = X_and_Y[:, 0]  # Shape: (300,)
Y = X_and_Y[:, 1]  # Shape: (300,)

import numpy as np
from numpy.linalg import inv
ones = np.ones((300,1))
new_X = np.hstack((ones, np.reshape(X, (300,1)), np.reshape
        ((X * X), (300,1))))
new_Y = np.reshape(Y, (300,1))
A = np.dot(np.transpose(new_X), new_X)
W = np.dot(inv(A), np.dot(np.transpose(new_X), new_Y))
print(W)

# Plot the scatter plot of the data and the estimated parabola
plt.scatter(X, Y, c = 'blue')
plt.plot(X, np.dot(new_X, W), c = 'red')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Estimated Parabola using L2 norm')
plt.show()
```

1.2 L1 norm (15 points)

Consider L_1 norm as your loss function:

$$g(W) = \|XW - Y\|_1 = \sum_{i=1}^{n} |y_i - f(x_i; W)| \tag{2}$$

1. (5 points) Derive the gradient of the cost function $g(W)$ with respect to W and you should have gradient as in the following form: (refer to lecture slides for hints)

$$\frac{\partial g(W)}{\partial W} = \Big(\big(\text{sign}(XW - Y)\big)^\top X\Big)^\top$$

where

$$\text{sign}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0. \end{cases}$$

and if X is a matrix, $\text{sign}(X)$ means performing element-wise $\text{sign}(x_{ij})$ over all element x_{ij} in X.

2. (10 points) Use the **gradient descent method** to find W^* and plot the scatter graph of data and estimated parabola. Report the parabola function and the figures.

Hint 1: In NumPy, you can use `np.sign(x)` to compute the sign of matrix `x`.

Hint 2: You may need to change the number of iterations to 300000, learning rate to 0.000001 and error threshold for W to 0.00001. Same settings may apply to sub-problem **(c)**.

Ans:

$$\frac{d|x|}{dx} = \text{sign}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0. \end{cases}$$

$$\begin{aligned} \frac{d|y_i - f(x_i;W)|}{dW} &= \begin{cases} \frac{-df(x_i;W)}{dW}, & y_i - f(x_i;W) > 0 \\ 0, & y_i - f(x_i;W) = 0 \\ \frac{df(x_i;W)}{dW}, & y_i - f(x_i;W) < 0. \end{cases} \\ &= \text{sign}(y_i - f(x_i;W)) \cdot \frac{-df(x_i;W)}{dW} \\ &= \text{sign}(f(x_i;W) - y_i) \cdot \mathbf{x}_i \end{aligned}$$

$$\begin{aligned} \frac{dg(W)}{dW} &= \sum_{i=1}^{n} \frac{d|y_i - f(x_i;W)|}{dW} \\ &= \sum_{i=1}^{n} \Big(\text{sign}(f(x_i;W) - y_i) \cdot \mathbf{x}_i\Big) \\ &= \Big(\text{sign}(X \cdot W - Y)^\top \cdot X\Big)^\top \end{aligned}$$

```
def gradient(X, Y, W):
    sign = np.sign(X.dot(W) - Y.T)
    grad = np.dot(sign,X)
    return grad[0]

def gradient_descent(X, Y, W, lam, max_iter, theta):
    cost_hist = []
    w_hist = []
    w_hist.append(W)
    for it in range(max_iter):
        cost_hist.append(sum(abs(X.dot(W) - Y)))
        W_new = w_hist[it] - lam * gradient(X, Y, w_hist[it])
        w_hist.append(W_new)
        diff = abs(w_hist[it+1] - w_hist[it])
        if sum(diff) < theta:
            break
    return W_new

optimal_W = gradient_descent(X = new_X,
```

```
                                    Y = new_Y,
                                    W = [0, 0, 0],
                                    lam = 0.00001,
                                    max_iter = 300000,
                                    theta = 0.00001)
print(optimal_W)

# Plot the scatter plot of the data and the estimated parabola
plt.scatter(X, Y, c = 'blue')
plt.plot(X, np.dot(new_X, optimal_W), c = 'red')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Estimated Parabola using L1 norm')
plt.show()
```

1.3 L1 and L2 norm (10 points)

Consider L_1 and L_2 norm as your loss function:

$$g(W) = \alpha \|XW - Y\|_2^2 + (1-\alpha)\|XW - Y\|_1 \tag{3}$$
$$= \sum_{i=1}^{n} \left(\alpha \big(y_i - f(x_i; W)\big)^2 + (1-\alpha)|y_i - f(x_i; W)| \right)$$

Use the **gradient descent method** to find W^* when $\alpha = 0.3$, $\alpha = 0.5$, and $\alpha = 0.7$ respectively and plot the scatter graph of data and estimated parabola. Report the parabola function and the figures.

Ans:

```
def gradient(X, Y, W, alpha):
    # Compute L1 norm
    sign = np.sign(X.dot(W) - Y.T)
    norm_1 = np.dot(sign,new_X)
    # Compute L2 norm
    norm_2 = 2 * ((X.dot(W) - Y.T).dot(X))
    grad = norm_1[0] + alpha*norm_2[0]
    return grad

def gradient_descent(X, Y, W, lam, max_iter, alpha, theta):
    cost_hist = []
    w_hist = []
    w_hist.append(W)
    for it in range(max_iter):
        cost_hist.append(sum(abs(X.dot(W) - Y)))
        W_new = w_hist[it] - lam * gradient(X, Y, w_hist
```

```
                [it], alpha)
        w_hist.append(W_new)
        diff = abs(w_hist[it+1] - w_hist[it])
        if sum(diff) < theta:
            break
    return W_new

# Plot the scatter plot of the data and the estimated parabola
plt.scatter(X, Y, c = 'blue')

# Search through all possible regularizer value (alpha)
for a in [0.3, 0.5, 0.7]:
        optimal_W = gradient_descent(X = new_X,
                                     Y = new_Y,
                                     W = [0, 0, 0],
                                     lam = 0.000001,
                                     max_iter = 300,
                                     alpha = a,
                                     theta = 0.00001)
        print(optimal_W)
        plt.plot(X, np.dot(new_X, optimal_W),
                 label = 'alpha = ' + str(a))

plt.legend()
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Estimated Parabola using L1 norm')
plt.show()
```

1.4 Comparison (5 points)

Compare the result from above three subsections (1.1-1.3) and plot all curves in one figure along with scatter graph of data. Try to explain the reason to **(1)** the position of each curve compared to the position of valid data points and outliers **(2)** difference between L_2 curve and L_1 curve **(3)** similarity among L_2 curve and $L_1 + L_2$ curves.

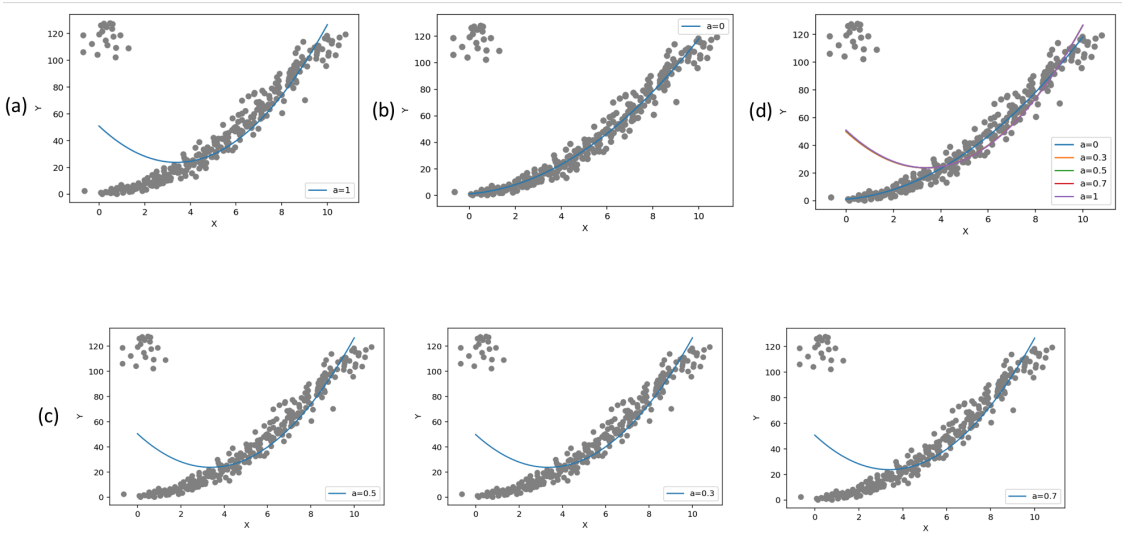
Ans:

1. L_1 curve models the valid data points well. L_2 and L_1 + L_2 curves lie between outliers and valid data points.

2. L_2 loss causes too much penalty from outliers, which makes outliers "pull" the curve far away from valid points, while L_1 loss' penalty from outliers is relatively less to keep the curve lying among the valid data points.

3. Ideally $L_1 + L_2$ curves should lie between L_1 curve and L_2 curve (and indeed it is), but due to the L_2 loss $>>$ L_1 loss, the L_2 part is dominating in the $L_1 + L_2$ loss, which makes the shape of $L_1 + L_2$ curves more similar to L_2 curve (almost overlapped) than L_1 curve.

Ans: Plots for questions above



2 (15 points) Logistic Regression (1)

For a logistic regression function with input $x \in \mathbb{R}$ and output $y \in \{0, 1\}$, the probability of $P(y = 1|x) = \dfrac{e^{\alpha+\beta x}}{1 + e^{\alpha+\beta x}}$.

1. Please write down the formulation of $P(y = 0|x)$.
2. Show the equality $[P(y = 1|x)]^y \times [P(y = 0|x)]^{1-y} = \dfrac{1}{1 + e^{-(2y-1)(\alpha+\beta x)}}$ when $y \in \{0, 1\}$.
3. What is the decision boundary for classifier:

$$y = \begin{cases} 1, & \alpha + \beta x \geq 0, \\ 0, & else \end{cases}$$

   and how is it related to $P(y = 1|x)$.

Ans:

1. $P(y = 0|x) = 1 - P(y = 1|x) = \frac{1}{1+e^{\alpha+\beta x}}$
2. When $y = 0$,

$$LHS = P(y = 0|x) = \frac{1}{1 + e^{-(2*0-1)(\alpha+\beta x}} = RHS$$

   And when $y = 1$,

$$LHS = P(y = 1|x) = \frac{1}{1 + e^{-(\alpha+\beta x}} = RHS$$

   Note: Many of you tried to directly solve the problem, but the equation only holds when $y = 0$ or $y = 1$. If you didn't specifically mention this condition, you cannot receive points from this problem.

3. The decision boundary is $\alpha + \beta x = 0$. It relates to $P(y = 1|x)$ in that when the point is on the decision boundary, $p(y = 1|x) = \frac{e^0}{1+e^0} = \frac{1}{2}$.

3 (10 points) Logistic Regression (2)

For a logistic regression function with input $\mathbf{x} \in \mathbb{R}^m$, i.e. $\mathbf{x}$ is a m-dimensional vector, and output $y \in \{0, 1\}$, the probability of $P(y = 1|\mathbf{x}) = \dfrac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$.

1. Please show that $P(y|\mathbf{x}) = \dfrac{1}{1 + e^{-(2y-1)\times(\mathbf{w}^T\mathbf{x}+b)}}$
2. Please analyze the decision boundary for this logistic regression classifier. (On what condition of $\mathbf{x}$, y will be predicated as 1 or 0?)

Ans:

1. $P(y = 0|x) = 1 - p(y + 1|x) = \frac{e^{-(w^T x+b)}}{1+e^{(-w^T x+b)}} = \frac{1}{1+e^{(W^T x+b)}}$ because $y \in \{0, 1\}$. Then consider the case when $y = 0$ and $y = 1$, the proof to show the equation holds is very similar to the solution above (2.2).
2. The decision boundary is $w^T x + b = 0$ (Bascially this question tells you logistic function behaves the same for $x \in R$ or $x \in R^k$; thus you can use matrix operation instead of single point operation for next question.)

4 (40 points) Logistic Regression (3)

In logistic regression model, y is the label for each data point, and it can be either 0 or 1. Here, we define our approximate function to

$$p(y = 1|\mathbf{x}) = h(\mathbf{x}; \mathbf{w}, b) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}$$

where $\mathbf{x} \in \mathbb{R}^m$ is the feature, and b is the bias, and $\mathbf{w} \in \mathbb{R}^K$ contains the set of parameters $\mathbf{w} = (w_0, w_1, ..., w_m)$. The output of $h(\mathbf{x}; \mathbf{w}, b)$ is called the confidence. When $h(\mathbf{x}; \mathbf{w}, b) >= 0.5$, the classifier outputs 1 for the given $\mathbf{x}$; otherwise, the classifier outputs 0.

Dataset

Download the data file **Q4_data.txt** from the course website. This dataset is modified from **Iris** dataset. The dataset contains 2 classes of 50 instances each, and the two classes are *Iris-versicolour* and *Iris-virginica*. The dataset currently contains 100 instances. The first 15 instances of each class are used for testing, and the rest of data are used for training. The code for the above preprocessing is provided.

Model

In this task, the first 4 attributes and the 5th attribute of i-th instance are considered as its feature $\mathbf{x}_i \in \mathbb{R}^4$ and its label $y_i \in \mathbb{R}$.
The goal is to train a classifier based on logistic regression to predict the correct label y_i from the given feature $\mathbf{x}_i$. We define a loss function which measures the distance between the correct label and the prediction from the classifier, as is shown below:

$$\mathcal{L}(\mathbf{w}) = -\sum_i \ln p(y_i|\mathbf{x}_i; \mathbf{w}, b)$$

where

$$p(y_i|\mathbf{x}_i; \mathbf{w} + b) = \frac{1}{1 + e^{-(2y_i-1)\times(\mathbf{w}^T\mathbf{x}_i+b)}},$$

which is called the sigmoid function.

4.1 Gradient Descent (10 pts)

The training procedure is to minimize the loss function on the training set. Consider ***gradient descent*** method to find the optimal $\mathbf{w}^*$ in $h(\mathbf{x}; \mathbf{w}, b)$.

1. Derive $\dfrac{\partial\mathcal{L}(\mathbf{w})}{\partial\mathbf{w}}$.

2. Suppose that the learning rate is denoted as α. Write down the update rule for $\mathbf{w}$.

Ans:

$$\frac{\partial L(w)}{\partial w} = \sum -(2y_i - 1)x_i(1 - p(y_i|x_i; w, b)$$

$$w = w - a * \frac{\partial L(w)}{\partial w}$$

4.2 Training (10 pts)

Implement your own **gradient descent** algorithm to train a binary classifier based on logistic regression. You might need to vectorize your algorithm in order to run efficiently. Not that b is also supposed to be learned in your gradient descent algorithm. **Plot the training curve: cost function vs. # iterations**

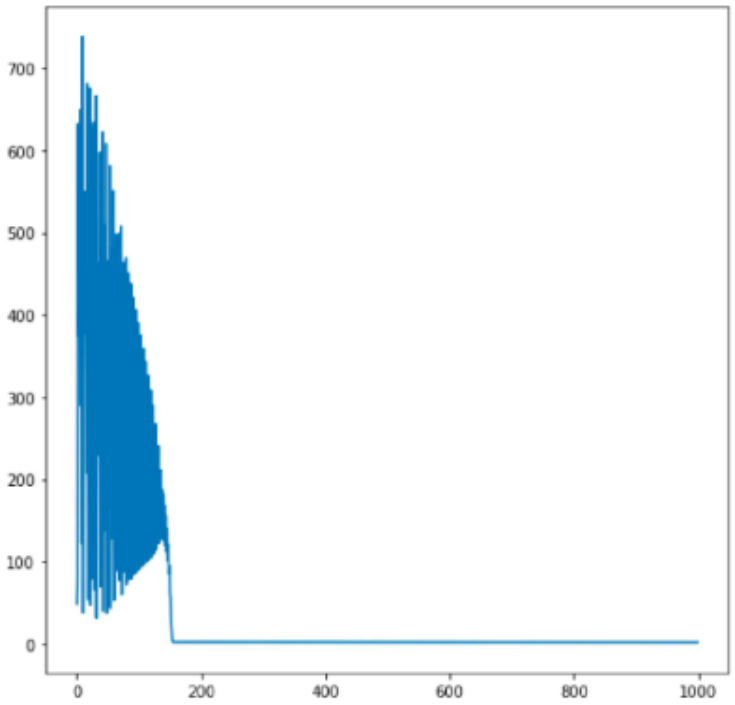
Ans:

```
# probability
def prob(X,Y,W):
    return 1/(1+np.exp(-(2*Y-1)*np.dot(X,W)))

# gradient of loss function
def gradient_logisitic(X,Y,W):
    return np.dot(np.transpose(-(2*Y-1)*X),(1-prob(X,Y,W)))

next_w = w
train_err = []
for _ in range(n_iter):
    next_w = w - alpha * gradient_logistic(X_train,Y_train,w)
    cost = np.sum(-1*np.log(prob(X_train,y_train,w)))
    train_err.append(cost)
    w = next_w

plt.plot(train_err)
plt.show()
```



4.3 Decision Boundary (10 pts)

Derive the equation of the decision boundary for your trained classifier

Ans: Decision boundary is:

$$-4.63 + -7.71x_1 + -7.39x_2 + 11.81x_3 + 10.57x_4 = 0$$

The answer is not unique; it based on your w learned above, but the decision boundary should be $wX = 0$ in general.

4.4 Test (10 pts)

Classify the data in the test set and report the accuracy. **Ans:**

```
prediction = np.dot(X_test,w) > 0
print(sum(prediction==y_test)/len(y_test))

0.86666666666
```

Again this answer is not unique, but you should have a similar value.