INF552: Programming Assignment 4

Yi-Li Chen

Part 1: Implementation

- All imported library

```
import csv
import numpy as np
from sklearn.metrics import accuracy_score
import random
import matplotlib.pyplot as plt
```

Perceptron Learning algorithm

- Read text file into a list and convert to array

```
data = []
with open('classification.txt') as f:
    file = csv.reader(f)
    for line in file:
        data.append(line)
data = np.array(data).astype('float64')
```

- Firstly, initialize the w. Secondly, make a loop that when there is one violated. Lastly, update the w by two cases.

```
def perceptron_learning_ALG(x, y_true, alpha):
    w = np.zeros(len(x[0]))
    iteration = 0
    yhat = -1*(x.dot(w) < 0) + 1*(x.dot(w) >= 0)
    while np.any(yhat != y_true):
        for idx in range(len(x)):
            z = x[idx].dot(w)
            if y_true[idx]==1 and z<0:
                w = w + alpha*x[idx]
                break
            if y_true[idx]==-1 and z>=0:
                w = w - alpha*x[idx]
                break
        yhat = -1*(x.dot(w) < 0) + 1*(x.dot(w) >= 0)
        iteration+=1
    acc = accuracy_score(y_true, yhat)
    return iteration, w, acc
```

- The result of Perceptron Learning

```
1. data_points = data[:,:3]
2. yi = data[:,3]
3. xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
4. iteration, w, acc = perceptron_learning_ALG(xi, yi, 0.1)
```

After 23884 iterations:

$$\text{weights} = [\ 0 \quad 11.33717993 \quad -9.11213384 \quad -6.78511535\ ]$$
$$\text{accuracy} = 1.0$$

Pocket algorithm

In pocket algorithm, I tried two kind of methods. The first one is only update w when updated w makes fewer mistakes than previous w. The another one is randomly pick one that is wrong and update w every time, but remember the w that have least wrong prediction.

- Read text file into a list and convert to array

```
1. data = []
2. with open('classification.txt') as f:
3.     file = csv.reader(f)
4.     for line in file:
5.         data.append(line)
6. data = np.array(data).astype('float64')
```

- If $\vec{w}^T\vec{x}$ less than zero, then the estimated $\hat{y}$ will be -1. Otherwise, $\hat{y}$ will be 1.

```
1. def sign(z):
2.     yhat = -1*(z < 0) + 1*(z >= 0)
3.     return yhat
```

- Find all the index of misclassification data.

```
1. def find_all_diff(w, x, y):
2.     zi = x.dot(w)
3.     yhat = sign(zi)
4.     all_diff_idx = np.where(yhat!=y)[0]
5.     return all_diff_idx
```

- The 1st method of algorithm:
  Firstly, initialize the w. Secondly, make a loop of finding the new w and the index of misclassification for given number of iterations. Lastly, only update the w when the new w makes fewer mistakes than previous w. In addition, create a `violated_amount` list to store the amount of violated data.

```
def pocket_ALG(x, y_true, alpha, iteration_num):
    w = np.zeros(len(x[0]))
    w_diff_idx = find_all_diff(w, x, y_true)
    violated_amount = []
    for i in range(iteration_num):
        violated_amount.append(len(w_diff_idx))
        diff_idx = random.choice(w_diff_idx)
        w2 = w + alpha*y_true[diff_idx]*x[diff_idx]
        w2_diff_idx = find_all_diff(w2, x, y_true)
        if len(w2_diff_idx) < len(w_diff_idx):
            w = w2.copy()
            w_diff_idx = w2_diff_idx.copy()
    zi = x.dot(w)
    yhat = sign(zi)
    acc = accuracy_score(y_true, yhat)
    return w, w_diff_idx, acc, violated_amount, yhat
```

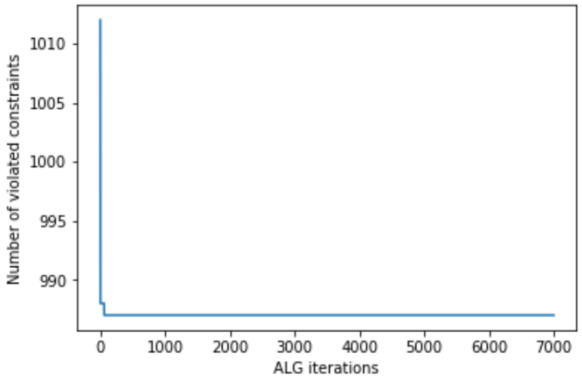
- The result of the 1st method for pocket algorithm

```
data_points = data[:,:3]
y = data[:,4]
x = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w, w_diff_idx, acc, violated_amount, yhat = pocket_ALG(data_points, y, 0.1, 7000)
```

After 7000 iterations:

$$\text{weights} = [\ 0 \quad -0.02579255 \quad 0.00195799 \quad 0.05170381\ ]$$
$$\text{accuracy} = 0.5065$$

```
x = [i for i in range(1,7001)]
y = violated_amount
plt.plot(x,y)
plt.xlabel('ALG iterations')
plt.ylabel('Number of violated constraints')
plt.show()
```



- The 2nd method of algorithm:

  Firstly, initialize the w. Secondly, make a loop of finding the new w and the index of misclassification for given number of iterations. Lastly, randomly pick one that is wrong and update w every time, but only remember the w that have least wrong prediction.

```
def pocket_ALG(x, y_true, alpha, iteration_num):
    w = np.zeros(len(x[0]))
    w_diff_idx = find_all_diff(w, x, y_true)
    violated_amount = []
    w_diff_least = len(y)
    for i in range(iteration_num):
        violated_amount.append(len(w_diff_idx))
        diff_idx = random.choice(w_diff_idx)
        w = w + alpha*y_true[diff_idx]*x[diff_idx]
        w_diff_idx = find_all_diff(w, x, y_true)
        if len(w_diff_idx) < w_diff_least:
            w_diff_least = len(w_diff_idx)
            w_best = w
    zi = x.dot(w)
    yhat = sign(zi)
    acc = accuracy_score(y_true, yhat)
    return w_best, w_diff_idx, acc, violated_amount, yhat
```

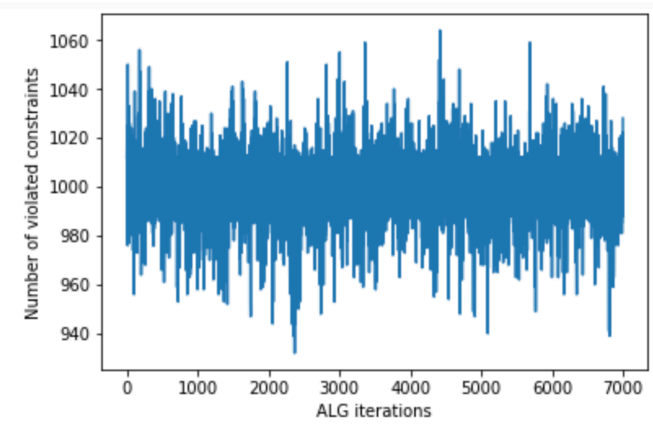
- The result of the 2nd method for pocket algorithm

```
data_points = data[:,:3]
y = data[:,4]
x = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w_best, w_diff_idx, acc, violated_amount, yhat = pocket_ALG(xi, yi, 0.1, 7000)
```

After 7000 iterations:

$$\text{weights} = [\ 0 \quad -0.15457602 \quad 0.12474349 \quad 0.02682324\ ]$$
$$\text{accuracy} = 0.506$$

```
x = [i for i in range(1,7001)]
y = violated_amount
plt.plot(x,y)
plt.xlabel('ALG iterations')
plt.ylabel('Number of violated constraints')
plt.show()
```



Logistic Regression

- Read text file into a list and convert to array

```
data = []
with open('classification.txt') as f:
    file = csv.reader(f)
    for line in file:
        data.append(line)
data = np.array(data).astype('float64')
```

- Build a sigmoid function

```
def sigmoid(s):
    z = np.exp(s)/(1+np.exp(s))
    return z
```

- Calculate the gradient of cost function

```
def gradient_Ein(w, xi, yi):
    sum2=0
    for x,y in zip(xi,yi):
        sum2 += y*x*(1/(1+np.exp(y*x.dot(w))))
    return sum2*(-1/len(xi))
```

- Firstly, initialize the w. Secondly, make a loop to update the w for given number of iterations. Lastly, use sigmoid function and classify the data by their probabilities.

```
def Logistic_Reg(xi, yi, alpha, iteration_num):
    w = np.zeros(len(xi[0]))
    for i in range(iteration_num):
        w = w - alpha*gradient_Ein(w, xi, yi)
    s = xi.dot(w)
```

```
    theta = sigmoid(s)
    yhat = -1*(theta < 0.5) + 1*(theta >= 0.5)
    acc = accuracy_score(yi, yhat)
    return w, acc, yhat, theta
```

- The result of logistic regression

```
data_points = data[:,:3]
yi = data[:,4]
xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w, acc, yhat, theta = Logistic_Reg(xi, yi, 0.1, 7000)
```

After 7000 iterations:

$$\text{weights} = [\ -0.03149498 \quad -0.17769975 \quad 0.11444872 \quad 0.07669738\ ]$$
$$\text{accuracy} = 0.5295$$

Linear Regression

- Read text file into a list and convert to array

```
data = []
with open('linear-regression.txt') as f:
    file = csv.reader(f)
    for line in file:
        data.append(line)
data = np.array(data).astype('float64')
```

- Using the matrix solution, $\vec{w}^* = (DD^T)^{-1}D\vec{y}$, to get the optimal w.

```
def Linear_Reg(xi, yi):
    DDT = xi.transpose().dot(xi)
    DDT_inv = np.linalg.inv(DDT)
    w = DDT_inv.dot(xi.transpose()).dot(yi)
    return w
```

- The result of linear regression

```
data_points = data[:,:2]
yi = data[:,2]
xi = np.concatenate((np.ones((len(data_points),1)), data_points), axis=1)
w = Linear_Reg(xi, yi)
```

$$\text{weights} = [\ 0.01523535 \quad 1.08546357 \quad 3.99068855\ ]$$

Before I start doing this assignment, I thought the logistic regression will be the most difficult part. However, the most challenging problem is to build the pocket algorithm. Since the professor have clearly explain the algorithm of the others. Therefore, it is not hard to finish perceptron learning, logistic algorithm and linear algorithm. But when building the function of pocket algorithm, it's kind of hard to decide which method it's better. Thus, I tried two method to make it.