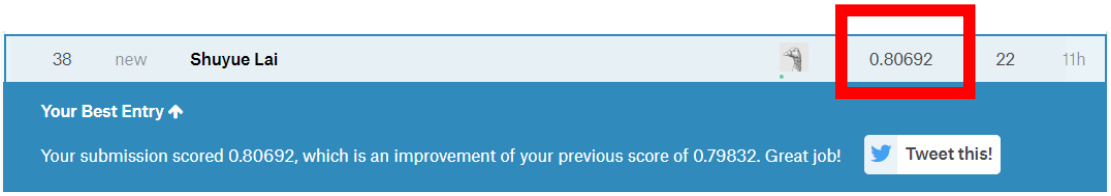
Homework 2

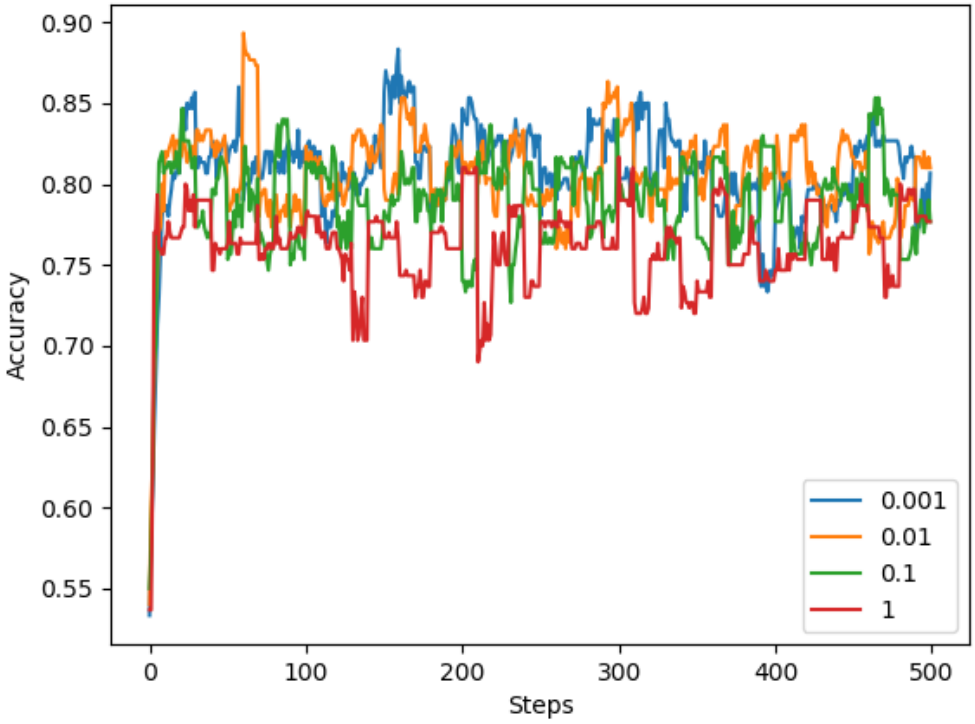
Problem 1：

1. Screenshot of leaderboard accuracy

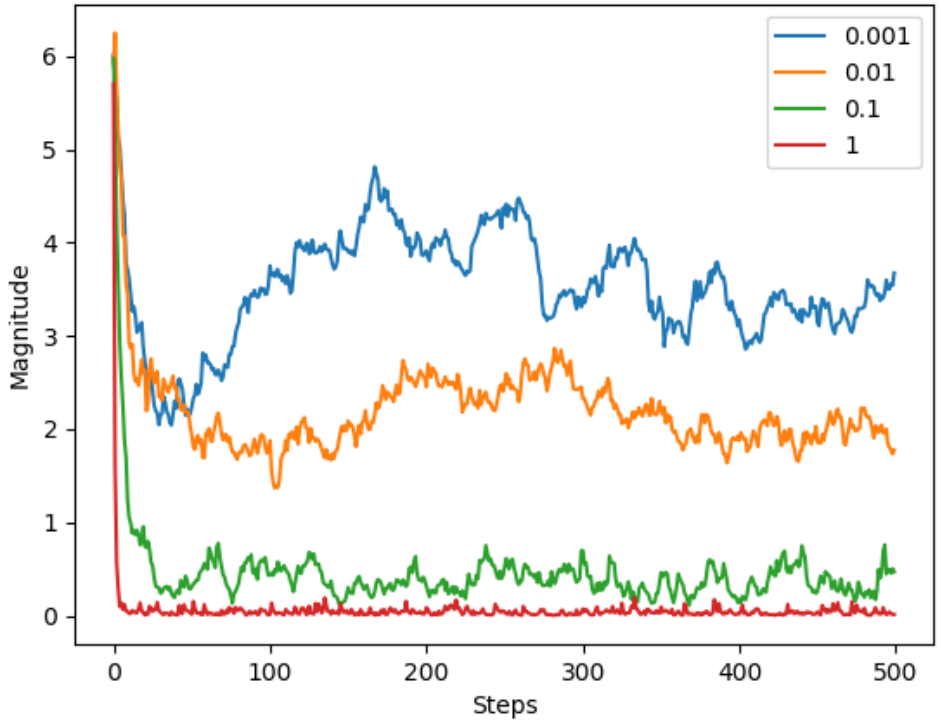


Best test dataset accuracy: 80.692%

2. Plot of the accuracy:



3. Plot of the magnitude:



4.

a) Best value of the regularization constant:

0.01. The best value means the regularization generate the highest accuracy of validation after training. In my project, I compare the result with each regularization constant and the result turns out to be 0.01.

```
    ## VALIDATION TO GET BEST REGULARIZER
    best_accuracy = 0
    best_a = np.array([1, 1, 1, 1, 1, 1])
    best_b = 1.00
    best_regularizer = 0
    accuray = []
    magnitude = []
    for regularizer in train_regularizer:
        [current_a, current_b, current_accuracy, list_accuracy, list_
        if current_accuracy > best_accuracy:
            best_a = current_a
            best_b = current_b
            best_accuracy = current_accuracy
            best_regularizer = regularizer

        accuray.append(list_accuracy)
        magnitude.append(list_magnitude)
    print("best regularizer:", best_regularizer)
```

```
if __name__=="__main__"
hw2 ×
/home/layla/anaconda3/envs/cs498/bin/python /home/layla/Desktop/cs498/hw2/shuyuel2_hw2.py
/home/layla/anaconda3/envs/cs498/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataCo
  warnings.warn(msg, DataConversionWarning)
/home/layla anaconda3/env /cs498/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataCo
  warnings. arn(msg, Data onversionWarning)
best regula izer: 0.01
/home/layla anaconda3/env /cs498/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: Ma
  warnings. plDeprecation, stacklevel=1)
```

b) Learning rate:

```
# step_length
step_length = 1.0 / ((0.01 * iter_epoch) + 50)
```

The learning rate I chose is negative relate to the current Round of epoch. At the beginning, the learning rate should be large to speed up to train. After a while, the learning rate is closer to the best value. Thus, the learning rate should be smaller in case of miss the best value.

5. Screenshot of code:

a) Library:

```
import csv
from sklearn import preprocessing
import numpy as np
from matplotlib import pyplot as plt
```

b) Stochastic Gradient Descent:

```
def stochasticGradientDescent(train_input_x, train_input_y, regularizer, train_sample_amount):

    ## INITIALIZE LIST OF ACCURACY ANF MAGNITUDE
    list_accuracy = []
    list_magnitude = []

    ## INITIALIZE a AND b
    a = np.array([1, 1, 1, 1, 1, 1])
    b = 1.00

    # GRADIENT DESCENT
    ## TRAIN
    amount_epoch = 50
    amount_step = 300
    amount_validation = 50

    for iter_epoch in range(amount_epoch):
        # step_length
        step_length = 1.0 / ((0.01 * iter_epoch) + 50)

        # data of the whole epoch
        index_epoch = np.random.choice(train_sample_amount, size=amount_step + amount_validation, replace=False)
        train_input_x_epoch = train_input_x[index_epoch, :]
        train_input_y_epoch = train_input_y[index_epoch]

        # training data and validation data
        index_step = np.random.choice(train_input_x_epoch.shape[0], size=amount_step, replace=False)
        train_input_x_step = train_input_x_epoch[index_step, :]
        train_input_y_step = train_input_y_epoch[index_step]
        train_input_x_validation = np.delete(train_input_x_epoch, index_step, axis=0)
        train_input_y_validation = np.delete(train_input_y_epoch, index_step, axis=0)

        # renew a and b
        for iter_step in range(amount_step):
            xi = train_input_x_step[iter_step, :]
            yi = train_input_y_step[iter_step]
            gi = yi * ((a).dot(xi) + b)

            if (gi >= 1):
                a = a - step_length * regularizer * a
            else:
                a = a - step_length * (regularizer * a - yi * xi)
                b = b + step_length * yi

            # EVERY 30 STEPS
            if(iter_step % 30 == 0):
                # predict label of training set and get accuracy
                correct_amount = 0
                for iter_y in range(amount_step):
                    if train_input_y_step[iter_y] * ((a).dot(train_input_x_step[iter_y, :]) + b) > 0:
                        correct_amount = correct_amount + 1
                accuracy = float(correct_amount / amount_step)
                list_accuracy.append(accuracy)

                # get magnitude
                magnitude = (a).dot(a.T)
                list_magnitude.append(magnitude)

    # predict label of validation set
    correct_amount = 0
    for iter_y in range(amount_validation):
        if train_input_y_validation[iter_y] * ((a).dot(train_input_x_validation[iter_y, :]) + b) > 0:
            correct_amount = correct_amount + 1

    # calculate accuracy
    accuracy_validation = correct_amount / amount_validation

    return a, b, accuracy_validation, list_accuracy, list_magnitude
```