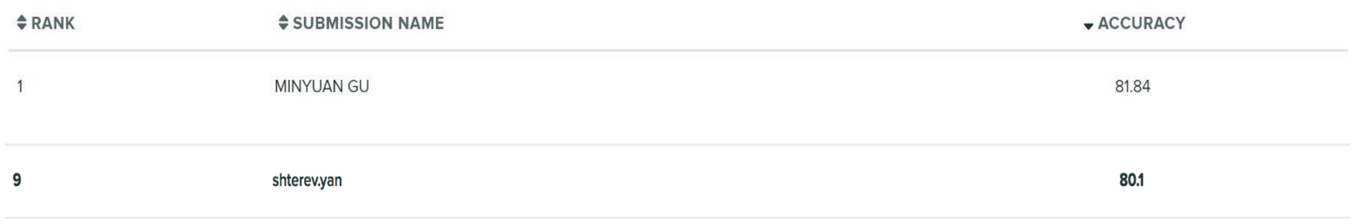
CS498 AMO Homework 2

Team :
Minyuan Gu (minyuan3@illinois.edu, netid minyuan3)
Yanislav Shterev (shterev2@illinois.edu, netid shterev2)

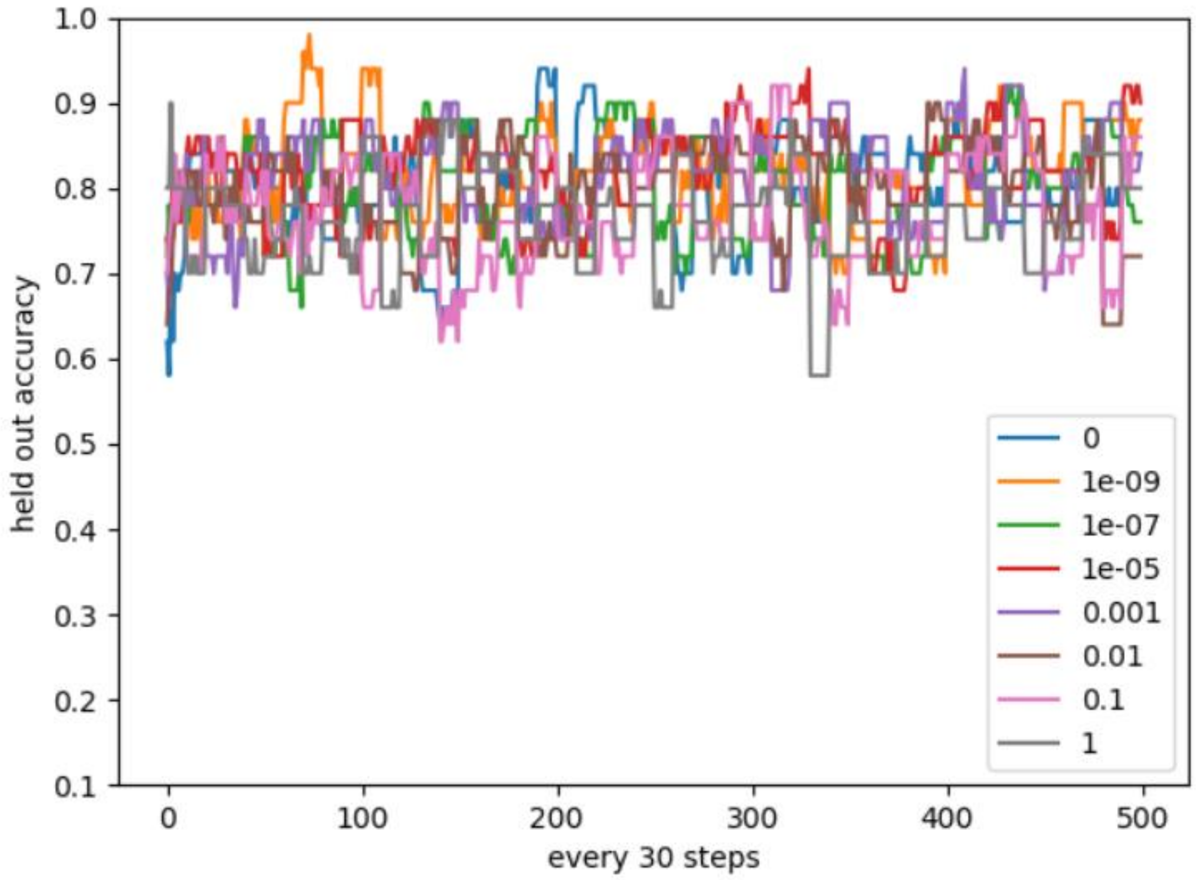
Page 1 (15 points)



RANK	SUBMISSION NAME	ACCURACY
1	MINYUAN GU	81.84
9	shterev.yan	80.1

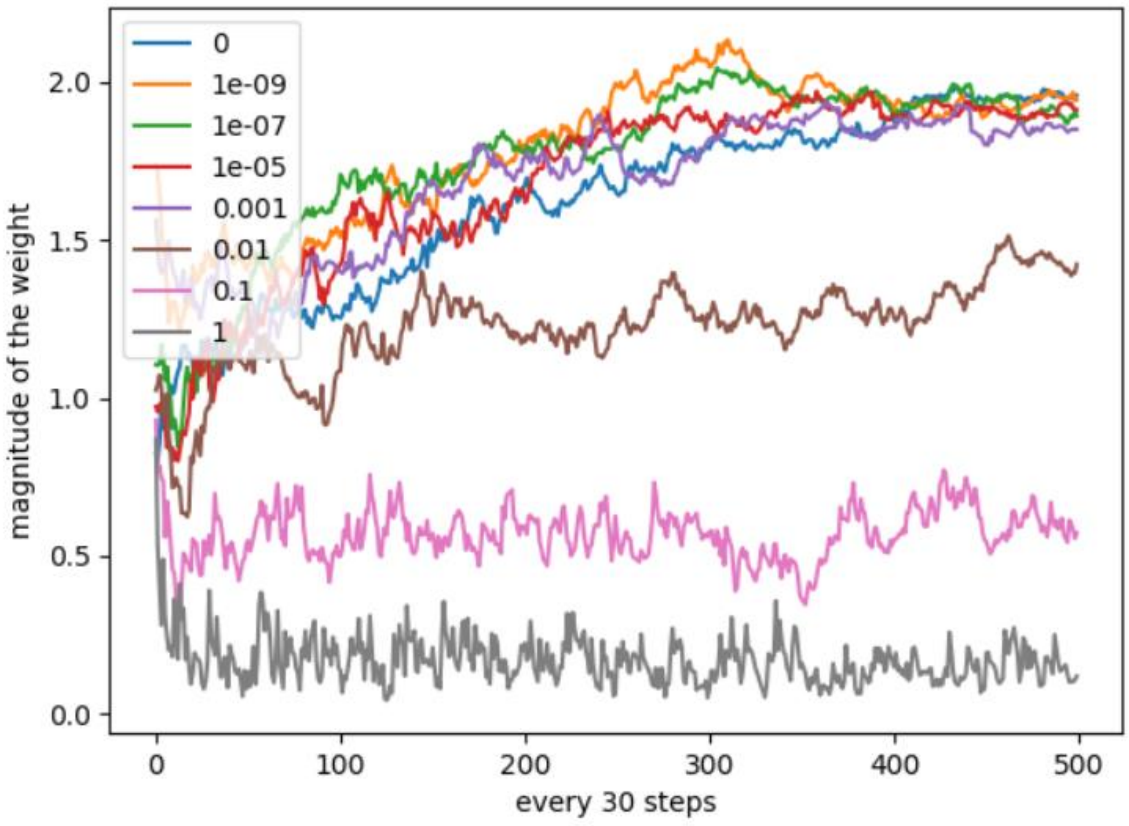
Page 2 (20 points)

A plot of the validation accuracy every 30 steps, for each value of the regularization constant.



Page 3 (20 points)

A plot of the magnitude of the coefficient vector every 30 steps, for each value of the regularization constant.



Page 4 (25 points)

The lambda consideration

The best value of the regularization rate was while lambda is 0.01 (1e-2) based on the validation set accuracy we had:

```
Test data accuracy is for lambda  0  is:  80.0820250284414 %
Test data accuracy is for lambda  1e-09  is:  79.95449374288964 %
Test data accuracy is for lambda  1e-07  is:  80.09101251422071 %
Test data accuracy is for lambda  1e-05  is:  80.13651877133105 %
Test data accuracy is for lambda  0.001  is:  80.25028441410694 %
Test data accuracy is for lambda  0.01  is:  80.25028441410694 %
Test data accuracy is for lambda  0.1  is:  78.77133105802048 %
Test data accuracy is for lambda  1  is:  76.35949943117178 %
```

Actually 1e-2, 1e-3 and 1e-5 all have very close performance; 1e-2 and 1e-3 have the same accuracy. We decided to choose 1e-2 over 1e-3 because we think larger regularization constant assists in preventing overfitting and reduce the variance for future data.

We also tested the extreme cases, for example lambda=0 (disabled regularization) and lambda=1 (lean more to regularization); we observed both cases are far from ideal.

By increasing lambda, the penalization factor increased on the weight magnitude, while features contribution to the learning was decreased. It could start causing model under-fitting if too large. On the other side, having low value of lambda caused the model over-fitting and resulted in lower accuracy on validation set.

The learning rate (step length) consideration.

The corresponding m and n parameters are part of the learning rate formula in our code: lr = step_length_m/(1*i+step_length_n)

where we have step_length_m = 1 and step_length_n = 50, i is the current season. For the 1 in from of i (**1***i+….), 1 is chose as the multiplying factor to trim down the learning rate at later season (we also tried 0.1, 0.01).

From above it means we will have a starting learning rate of 0.02 (1/50) and slowly trimming down to 0.01(1/50+50) if total season is 50. We also tried other step_length_n, e.g. 10, 100, 150, 200, and they didn't provide expected results.

If learning rate is too small, we will require more seasons to train the model until it is stable; if the learning rate is too large, it will become hard to converge at later stage since larger value caused the oscillating of the weights (e.g. making occasional large, bad, moves).

Especially in the case of batch size =1 (Stochastic Gradient Descent), smaller learning rate are recommended due to the fact that gradient is calculated on only one sample which means more variance. Initial weights also played a role on the final accuracy and learning rate, we used randomly initialized weight and choose small learning rate to avoid over shooting on the initial bad weights.

Having that said, we also noticed if we put a stop of the training once we reached a certain accuracy (e.g. record high) on the validation set, with batch size = 2, 3 or 5, it seems initial learning rate of 0.2 (step_length_n = 5) is quite a good choice; but this is out of scope of the discussion of this home work.

Libraries used & Reference:

David Forsyth's book - Probability and Statistics for Computer Science
David Forsyth's book - Applied Machine Learning
Trevor Walker's lecture and sample code – CS-498 Lecture videos
csv – for reading data from csv format: https://docs.python.org/3/library/csv.html
Adult dataset - **training dataset**
https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/train.txt
Testing dataset https://courses.engr.illinois.edu/cs498aml/sp2019/homeworks/test.txt
Numpy - http://www.numpy.org/
matplotlib - to plot the accuracy and magnitude: https://matplotlib.org/

Page 5+

```
import numpy as np
import csv
import matplotlib.pyplot as plt
import math


class supportVectorMachine:
    def __init__(self, weight, b=0.0, reg_lambda=1e-1, step_length=0.1):
        self.reg_lambda = reg_lambda
        self.weight = np.array(weight)
        self.b = b
        self.X = np.array([])
        self.Y = np.array([])
        self.cost = 0
        self.training_cost = np.array([])
        self.step_length = step_length

    def cost_function(self, X, Y):
        self.training_cost = 1 - (np.dot(X, self.weight)+self.b)*Y
        self.training_cost[self.training_cost<0] = 0
        self.cost = np.mean(self.training_cost) + self.reg_lambda*np.dot(self.weight.T, self.weight)/2
        print(" training cost is ", self.cost)

    def update_weight(self):
        #print("current weight: ", self.weight)
        weight_to_update=self.X*self.Y.reshape((self.Y.shape[0],1))
        #print("X is:", self.X)
        #print("Y is:", self.Y)
        #print("weight_to_update of yx", weight_to_update)
        zero_cost_matrix = 1 - (np.dot(self.X, self.weight)+self.b)*self.Y
        zero_cost_matrix[zero_cost_matrix<0]=0
        #print("zero cost maxtrix as filter: ", zero_cost_matrix)
        weight_to_update = weight_to_update*((zero_cost_matrix!=0).reshape((zero_cost_matrix.shape[0],1)))
        #print("masking 'cost = 0', weight_to_update", weight_to_update)
        #print("regularization term is ", self.reg_lambda*self.weight)
        weight_to_update = -(1/self.X.shape[0])*np.sum(weight_to_update,axis=0)+self.reg_lambda*self.weight
        #print("final weight_to_update: ",weight_to_update)
        self.weight = self.weight - self.step_length*weight_to_update
        #print("new weight: ", self.weight)
        # to update b
        #print("current b: ", self.b)
        self.b = self.b - self.step_length*(-np.dot(self.Y, 1*(zero_cost_matrix!=0))/self.X.shape[0])
        #print("updated_b: ", self.b)

    def StochasticGradientDesc(self, X, Y):
        self.X = X
        self.Y = Y
        self.update_weight()
    def predict(self, X):
        #print("raw output is: ", np.dot(X, self.weight)+self.b)
        return 2*((np.dot(X, self.weight)+self.b)>0)-1

    def set_learningRate(self, lr):
        self.step_length = lr


def save_for_submission(results):
    fobj = open('./homework2/submission.txt', 'a+')
    for i in results:
        if i >= 1:
            fobj.write('>50K\n')
        else:
            fobj.write('<=50K\n')
    fobj.close()


#################################################################
#     Import training data, shuffle, rescale & split            #
#################################################################
data = []
X = []
Y = []
# import the data from the csv.
with open('./homework2/train.txt', newline='') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        data.append(row)

data = np.array(data)
np.random.shuffle(data)
# extract only continuous variable values to form X
X = data[:, (0,2,4,10,11,12)].astype(float)
# extract last col to form classes of 1 for >50K and -1 for <=50K
Y = 2*(data[:, 14] == ' >50K')-1
# rescale the features to same variance and zero means.
X = (X - np.mean(X, axis=0))/np.std(X,axis=0)
rescaled_data = np.column_stack((X,Y))

split_idx = int(data.shape[0]*0.1)
hyperParmSearch_data = rescaled_data[:split_idx, :]
train_data = rescaled_data[split_idx:, :]

#################################################################
#                    Hyper Parameters definition                #
#################################################################
regularisation_lambda = [0,1e-9, 1e-7, 1e-5, 1e-3, 1e-2, 1e-1, 1]
step_length_m = 1
step_length_n = 50
total_season = 50
steps = 300
batch_size = 1

#################################################################
#             Training using different lambda values            #
#################################################################
# following history record the held out accuracy every 30 steps.
accuracy_history = np.zeros((len(regularisation_lambda), int(steps*total_season/30)))
weight_magnitude_history = np.zeros((len(regularisation_lambda), int(steps*total_season/30)))
# following accuracy report each lambda's performance against validation set.
final_validation_accuracy_history = np.zeros(len(regularisation_lambda))
svm = None
max_achieved_accuracy = 0
max_achieved_weight = []
for idx_lambda in range(len(regularisation_lambda)):
    weight = np.random.rand(X.shape[1])
    svm = supportVectorMachine(weight=weight, reg_lambda=regularisation_lambda[idx_lambda])
    for i in range(total_season):
        print("******season: ", i," *******")
        lr = step_length_m/(1*i+step_length_n)
        svm.set_learningRate(lr)

        np.random.shuffle(train_data)
        held_out = train_data[:50, :]
        train = train_data[50:, :]
        for j in range(1, steps+1):
            selected = np.random.randint(train.shape[0], size=batch_size)
            svm.StochasticGradientDesc(train[selected, :-1], train[selected, -1])
            if j % 30 == 0:
                print("--->Step: ", j, " <----")
                validation_result = svm.predict(held_out[:, :-1])
                validation_accuracy = sum(validation_result == held_out[:, -1]) / held_out.shape[0]
                print(" Validation accuracy is ", validation_accuracy*100, "%")
                #svm.cost_function(held_out[:, :-1], held_out[:, -1])
                accuracy_history[idx_lambda, int((i*steps+j)/30)-1] = validation_accuracy
                weight_magnitude_history[idx_lambda, int((i*steps+j)/30)-1] = math.sqrt(np.sum(svm.weight **
2))

    test_result = svm.predict(hyperParmSearch_data[:, :-1])
    test_accuracy = sum(test_result == hyperParmSearch_data[:, -1]) / hyperParmSearch_data.shape[0]
    final_validation_accuracy_history[idx_lambda] = test_accuracy
    if test_accuracy >= max_achieved_accuracy:
        max_achieved_accuracy = test_accuracy
        max_achieved_weight = (svm.weight, svm.b, idx_lambda)

#################################################################
#            Plot the graph for different lambdas               #
#################################################################
x_axis = range(int(steps*total_season/30))
plt.subplot(1, 2, 1)
for idx_lambda in range(len(regularisation_lambda)):
    print(" Test data accuracy is for lambda ",regularisation_lambda[idx_lambda]," is: ",
final_validation_accuracy_history[idx_lambda]*100, "%")
    plt.plot(x_axis, accuracy_history[idx_lambda])
plt.ylim((0.1,1))
plt.legend(regularisation_lambda, loc='lower right')
plt.xlabel('every 30 steps')
plt.ylabel('held out accuracy')
plt.subplot(1, 2, 2)
for idx_lambda in range(len(regularisation_lambda)):
    plt.plot(x_axis, weight_magnitude_history[idx_lambda])
plt.xlabel('every 30 steps')
plt.ylabel('magnitude of the weight')
plt.legend(regularisation_lambda, loc='upper left')
plt.show()


#################################################################
#          Predict on the test set for submission               #
#################################################################
svm.weight = max_achieved_weight[0]
svm.b = max_achieved_weight[1]
grader_data = []
with open('./homework2/test.txt', newline='') as f:
    reader = csv.reader(f, delimiter=',')
    for row in reader:
        grader_data.append(row)

grader_data = np.array(grader_data)
grader_X = grader_data[:, (0,2,4,10,11,12)].astype(float)
# rescale the features to same variance and zero means.
grader_X = (grader_X - np.mean(grader_X, axis=0))/np.std(grader_X,axis=0)
grader_result = svm.predict(grader_X)
save_for_submission(grader_result)
```