**CS498 AMO Homework 2**

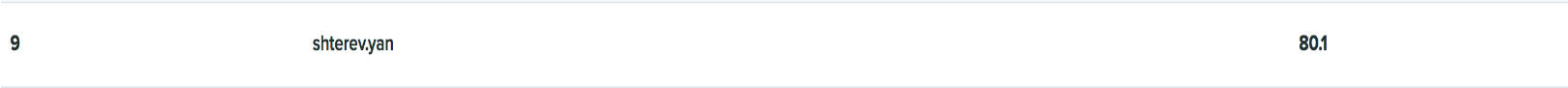
Team :

Minyuan Gu ([minyuan3@illinois.edu](mailto:minyuan3@illinois.edu), netid minyuan3)

Yanislav Shterev ([shterev2@illinois.edu](mailto:shterev2@illinois.edu), netid shterev2)

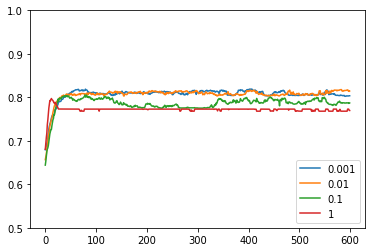
**Page 1 (15 points)**





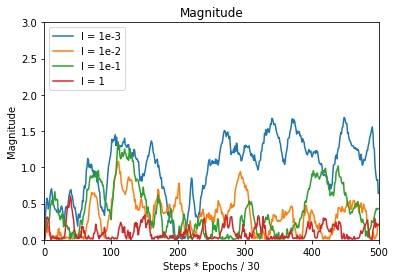
**Page 2 (20 points)**

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 best value of the regularization rate was while lambda is 0.01 achieving 81.84% accuracy on the testing data. The corresponding m and n parameters being part of the learning rate formula were 1 and 300. Which produced a smooth training without huge fluctuations during the different epochs and steps. In that case the stochastic gradient descent was smoothly updating matrix A and the scalar value b. By increasing lambda, the penalization factor increases and features contribution to the learning was decreasing which caused model underfitting. Respectively having low value of lambda was causing overfitting of the model not being able to penalize enough.

**Page 5 (20 points)**

* SVM training with (stochastic) gradient descent updating

def update\_weight(self):  
 weight\_to\_update=self.X\*self.Y.reshape((self.Y.shape[0],1))  
 zero\_cost\_matrix = 1 - (np.dot(self.X, self.weight)+self.b)\*self.Y  
 zero\_cost\_matrix[zero\_cost\_matrix<0]=0  
 weight\_to\_update = weight\_to\_update\*((zero\_cost\_matrix!=0).reshape((zero\_cost\_matrix.shape[0],1)))  
 weight\_to\_update = -(1/self.X.shape[0])\*np.sum(weight\_to\_update,axis=0)+self.reg\_lambda\*self.weight  
 self.weight = self.weight - self.step\_length\*weight\_to\_update  
 self.b = self.b - self.step\_length\*(-np.dot(self.Y, 1\*(zero\_cost\_matrix!=0))/self.X.shape[0])  
  
def StochasticGradientDesc(self, X, Y):  
 self.X = X  
 self.Y = Y  
 self.update\_weight()

* Label prediction

def predict(self, X):  
 return 2\*((np.dot(X, self.weight)+self.b)>0)-1

* Calculation of the accuracies

if j % 30 == 0:  
 print("--->Step: ", j, " <----")  
 validation\_result = svm.predict(hyperParmSearch\_data[:, :-1])  
 validation\_accuracy = sum(validation\_result == hyperParmSearch\_data[:, -1]) / hyperParmSearch\_data.shape[0]  
 print(" Validation accuracy is ", validation\_accuracy\*100, "%")  
 accuracy\_history[idx\_lambda, int((i\*steps+j)/30)-1]=validation\_accuracy  
 if validation\_accuracy >= max\_achieved\_accuracy:  
 max\_achieved\_accuracy = validation\_accuracy  
 max\_achieved\_weight = svm.weight

**Page 6+**

import numpy as np  
import csv  
import matplotlib.pyplot as plt  
  
  
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):  
 weight\_to\_update=self.X\*self.Y.reshape((self.Y.shape[0],1))  
 zero\_cost\_matrix = 1 - (np.dot(self.X, self.weight)+self.b)\*self.Y  
 zero\_cost\_matrix[zero\_cost\_matrix<0]=0  
 weight\_to\_update = weight\_to\_update\*((zero\_cost\_matrix!=0).reshape((zero\_cost\_matrix.shape[0],1)))  
 weight\_to\_update = -(1/self.X.shape[0])\*np.sum(weight\_to\_update,axis=0)+self.reg\_lambda\*self.weight  
 self.weight = self.weight - self.step\_length\*weight\_to\_update  
 self.b = self.b - self.step\_length\*(-np.dot(self.Y, 1\*(zero\_cost\_matrix!=0))/self.X.shape[0])  
  
 def StochasticGradientDesc(self, X, Y):  
 self.X = X  
 self.Y = Y  
 self.update\_weight()  
  
 def predict(self, X):  
 return 2\*((np.dot(X, self.weight)+self.b)>0)-1  
  
 def set\_learningRate(self, lr):  
 self.step\_length = lr  
  
  
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:, :]  
  
#regularisation\_lambda = [1e-7, 1e-5, 1e-3, 1e-2, 1e-1, 1]  
regularisation\_lambda = [1e-3, 1e-2, 1e-1, 1]  
step\_length\_m = 1  
step\_length\_n = 300  
total\_epoch = 60  
steps = 300  
batch\_size = 1  
accuracy\_history = np.zeros((len(regularisation\_lambda), int(steps\*total\_epoch/30)))  
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\_epoch):  
 print("\*\*\*\*\*\*epoch: ", i," \*\*\*\*\*\*\*")  
 lr = step\_length\_m/(0.01\*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(hyperParmSearch\_data[:, :-1])  
 validation\_accuracy = sum(validation\_result == hyperParmSearch\_data[:, -1]) / hyperParmSearch\_data.shape[0]  
 print(" Validation accuracy is ", validation\_accuracy\*100, "%")  
 accuracy\_history[idx\_lambda, int((i\*steps+j)/30)-1]=validation\_accuracy  
 if validation\_accuracy >= max\_achieved\_accuracy:  
 max\_achieved\_accuracy = validation\_accuracy  
 max\_achieved\_weight = svm.weight  
x\_axis = range(int(steps\*total\_epoch/30))  
plt.plot(x\_axis, accuracy\_history[0])  
plt.plot(x\_axis, accuracy\_history[1])  
plt.plot(x\_axis, accuracy\_history[2])  
plt.plot(x\_axis, accuracy\_history[3])  
plt.ylim((0.5,1))  
plt.legend([regularisation\_lambda[0], regularisation\_lambda[1], regularisation\_lambda[2], regularisation\_lambda[3]], loc='lower right')  
plt.show()  
  
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()  
  
  
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)