CS498 AML,AMO HW2

WORAWICH CHAIYAKUNAPRUK, Worapol Setwipatanachai

TOTAL POINTS

100 / 100

QUESTION 1

1 Screenshot of best accuracy 15 / 15

+ **10.5 pts** Accuracy in [70.0, 76.0) if they show SVM update equations in code snippet

+ **15 pts** Accuracy in range [78.4, 100]

+ **12 pts** Accuracy in [76.0, 78.4) {Since the test accuracy is 75.9 with all positive labels}

+ **6 pts** Other accuracy values if they show SVM update equations in code snippet

- **15 pts** Otherwise

+ 15 Point adjustment

QUESTION 2

2 Plot of accuracy for different regularization constants 20 / 20

- **5 pts** One plot is missing.

- **5 pts** One plot is not correct, e.g., curves not converged etc.

- **2 pts** If one doesn't plot the curves as required, i.e, [1,1e-1,1e-2,1e-3], but plots enough curves.

+ **20 pts** Full points.

- **20 pts** No graphs (incorrect homework submission)

+ 20 Point adjustment

QUESTION 3

3 Plot of magnitude of the coeff for different regularization constants 20 / 20

- **5 pts** One plot is missing.

- **5 pts** One plot is not correct, e.g., curves not converged etc.

- **2 pts** If one doesn't plot the curves as required, i.e, [1,1e-1,1e-2,1e-3], but plots enough number of curves.

- **0 pts** Correct

- **20 pts** Incorrect homework

+ 20 Point adjustment

QUESTION 4

4 Best Estimate of reg constant and learning rate + explanation 25 / 25

- **0 pts** Correct

- **0.5 pts** If one only says trying several lrs (without any detail lr values) but the one mentioned in the text book is the best.

- **1 pts** If one just simply says that lr = 1/(0.01*season +50) seems to give a better accuracy without anymore analysis and explanation.

- **12.5 pts** Analysis for the learning rate is missing.

- **12.5 pts** Analysis for lambda is missing.

- **25 pts** Incorrect homework.

+ 25 Point adjustment

QUESTION 5

5 Screenshot of Code 20 / 20

- **0 pts** Correct

- **5 pts** SGD is incorrect

+ 20 Point adjustment

QUESTION 6

6 Late 0 / 0

- **5 pts** 1 day

- **10 pts** 2 days

- **15 pts** 3 days

- **20 pts** 4 days

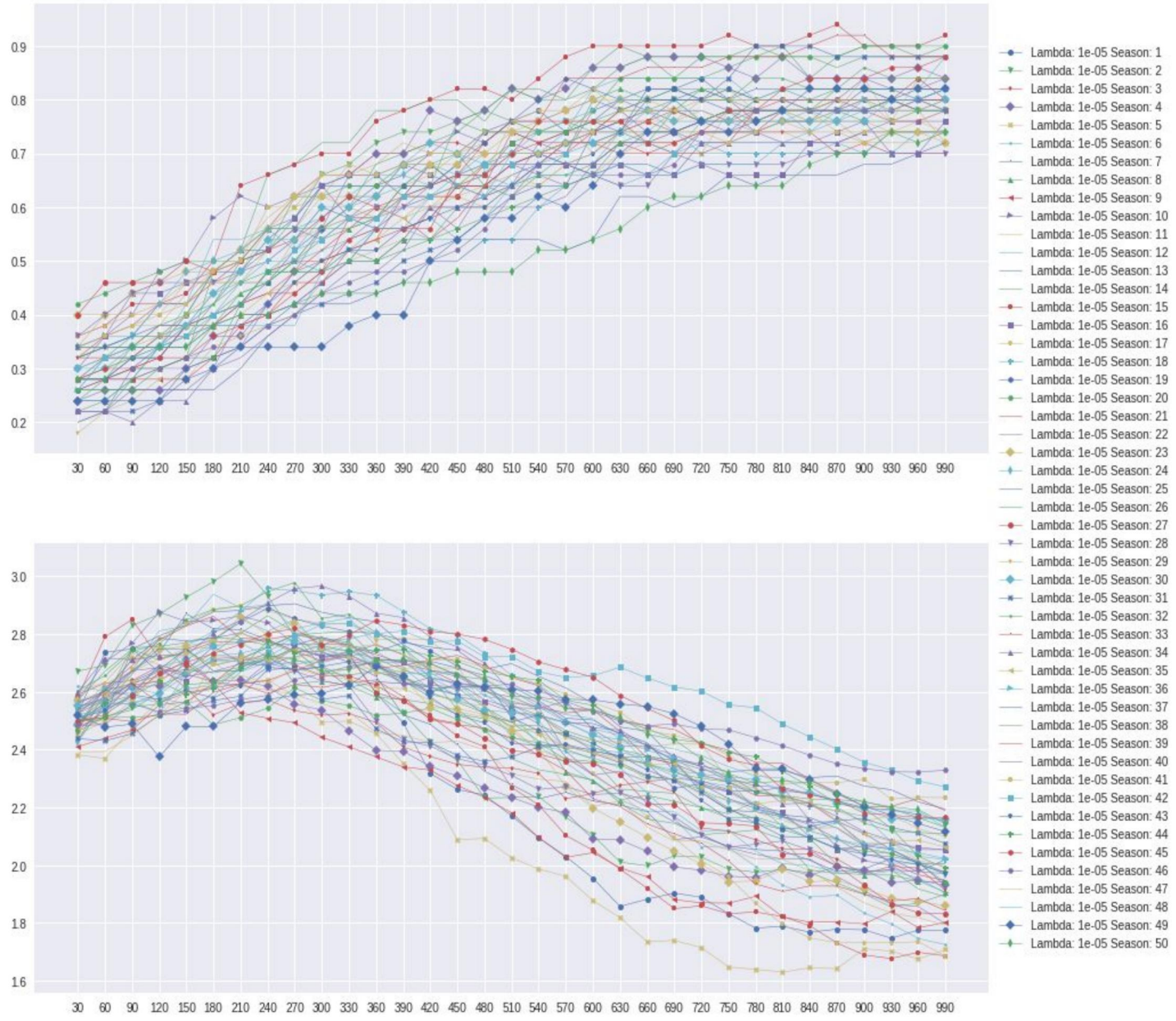
- **25 pts** 5 days

- **30 pts** 6 days

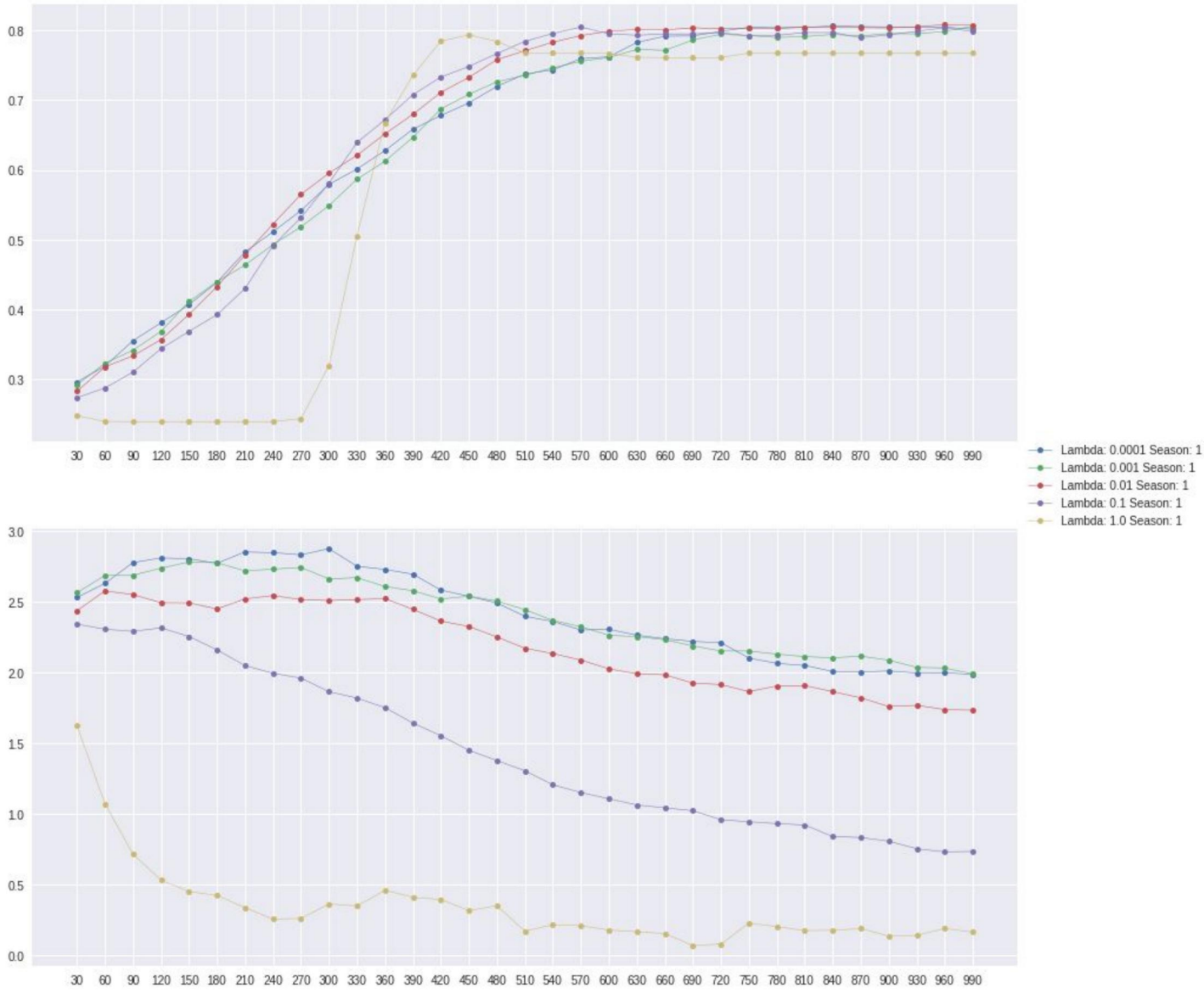
✓ **- 0 pts On time.**

Page 1: Test Accuracy (from Autograder)

Page 2: Learning Rate Validation



Page 3: Regularization Constant Validation



Page 4 Your estimate of the best value of the regularization constant, together with a brief description of why you believe that is a good value. What was your choice for the learning rate and why did you choose it?

Lambda = 0.01
Accuracy = 0.8080072793448589
a = [[0.37460604],
[0.1551405],
[0.55226118],
[1.46628975],
[0.498473],
[0.37187249]]
b = [-0.953125]

First of all, we found the *learning rate (eta)* value by iterating eta values based on a fixed *regularization constant (lambda)*. Our *eta* variable is defined as *m/(k+n)* where *m = 1, k = season iteration by index, and n = 50*. Then we iterated *eta* over 50 seasons, each season we calculated accuracy of *eta* validation set. Once we plotted accuracy of *eta* validation set versus number of steps, we had three matrices to determine the best learning rate: 1) sum of accuracy in each season, 2) sum of accuracy in the second half of each season, and 3) final accuracy of each season. From these three matrices, we picked season number 15 as shown in Page 9, which yielded learning rate of **1/64**.

Next, we used *regularization constants (lambda)* of [1e-4, 1e-3, 1e-2, 1e-1, 1] and plotted accuracy of *lambda* validation set versus *number of steps* for each *lambda* at a constant learning rate of 1/64. By applying the same matrices as the previous step, we found that *regularization constant* of 0.01 was the best. Then, we could obtain the correlating *a* and *b* from this best case to be used in the final prediction model.

Page 5+: A screenshot of your code.

```
from google.colab import drive
drive.mount('/content/drive/')
```

Mounted at /content/drive/

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

# Zero mean unit variance
def zmuv(a):
 a = (a-a.mean(axis=0))/a.std(axis=0)
 return a

# Import training data (f for features, l for labels)
train_f = np.genfromtxt('drive/My Drive/CS 498 AML/HW2/Data/train.txt', dtype=None, delimiter=",",
usecols=(0,2,4,10,11,12))
train_l = np.genfromtxt('drive/My Drive/CS 498 AML/HW2/Data/train.txt', dtype=np.string_, delimiter=",",
usecols=(14))

train_l[train_l==b' <=50K'] = -1
train_l[train_l==b' >50K'] = 1
train_l = np.asarray([train_l], dtype=int)
train_f = zmuv(train_f)

# Make a single array of standardized features and corresponding labels
train = np.concatenate([train_f,train_l.T],axis=1)

# Import test data (f for features, l for labels)
test_f = np.genfromtxt('drive/My Drive/CS 498 AML/HW2/Data/test.txt', dtype=None, delimiter=",",
usecols=(0,2,4,10,11,12))
test_f = zmuv(test_f)
```

Part 1: Finding an appropriate learning rate (eta)

```
# xi is a column vector of features
# yi is a label
# a is a column vector
# b is a scalar

# Predict and return sign
def predict(u, x):
 s = np.dot(u[0].T,x.T)+u[1]
 s = np.sign(s)
```

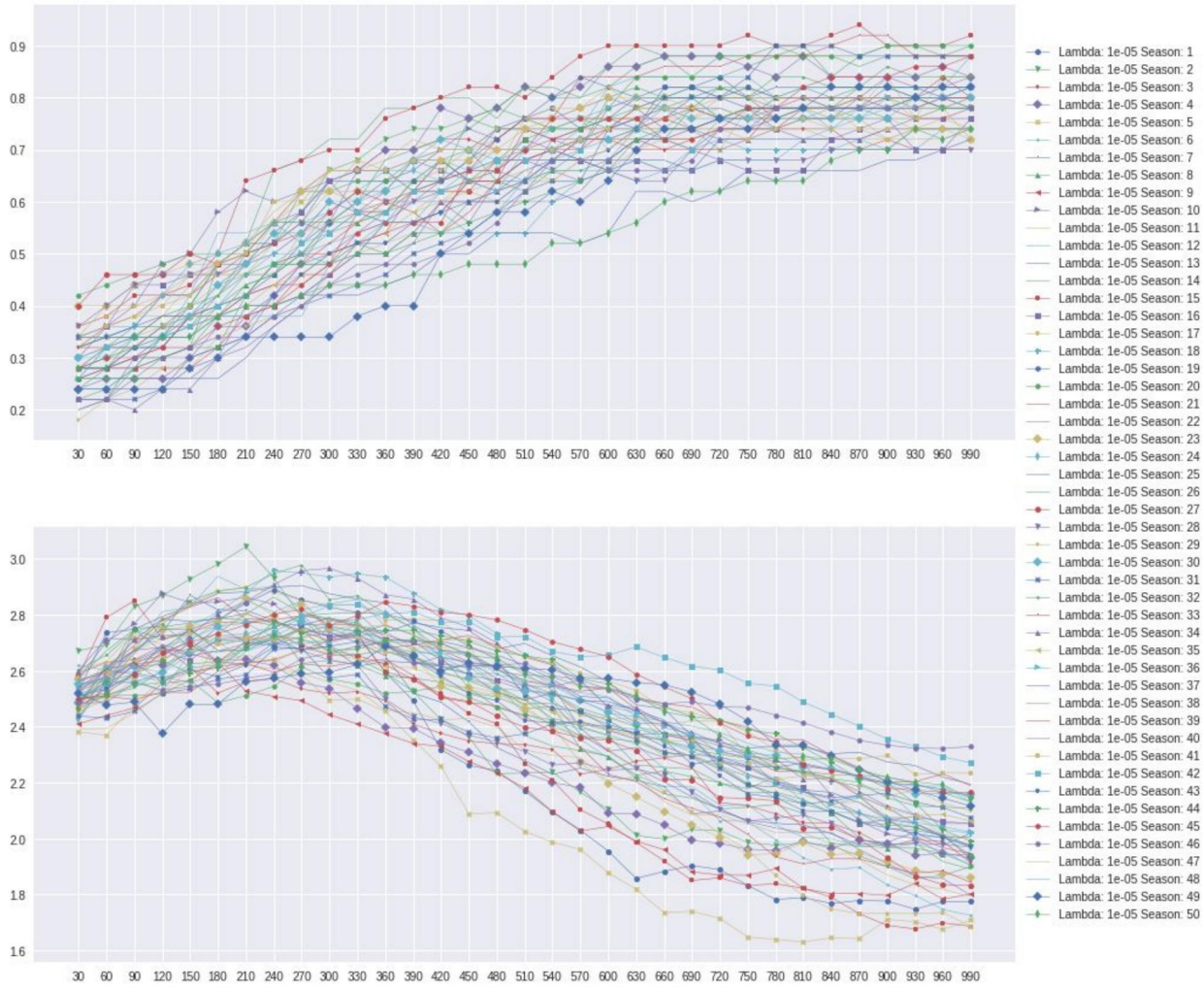
```
  return s.T

# Compare real and predicted labels and get accuracy
def accuracy(u, x, y):
  s = predict(u, x)
  acc = np.sum(y == s)/y.size
  return acc

# Calculate gradient of u. Return gradient w.r.t. to a and b
def gradient(yi, xi, u, lam=1e-3):
  crit = yi[0,0]*gamma(u, xi)
  if crit >= 1:
    grad_b = 0
    grad_a = lam * u[0]
  else:
    grad_b = -yi[0,0]
    grad_a = lam * u[0] - yi[0,0]*xi
  return [grad_a,grad_b]

# Taking a step with eta and gradient. Return new value of u
def step(u, grad, k):
  a_new = u[0] - eta(k)*grad[0]
  b_new = u[1] - eta(k)*grad[1]
  return [a_new, b_new]

# Calculate gamma
def gamma(u, xi):
  gam = np.dot(u[0].T,xi) + u[1]
  return gam

# Returns training cost
def training_cost(u, yi, xi, gam, lam=1e-3):
  cost = (1/len(yi))*np.amax([0,1-np.dot(yi,gamma(u, xi))])+(lam/2)*np.dot(u[0].T,u[0])
  return cost

# Calculate eta based on season#
def eta(k, m=1, n=50):
  return m/(k+n)
```

```
# Prepare data based on size of lambda validation set of 4396 (10%)
# Pick 10% of all training data for lambda validation set
# Pick 50 samples from training data for eta validation set
# Pick 1 sample from the remaining training data for learning
# ind is 10% of all training data
```

```
ind = 4396

shuffled = train.copy()
np.random.shuffle(shuffled)

lam_val = shuffled[0:ind,:]
lam_val_f = lam_val[:,0:6]
lam_val_l = np.array([lam_val[:,6]]).T

training_lesslam = shuffled[ind:,:]

# SVM Setup
num_step = 990
season = 50
lam = np.array([1e-5])

label = list()
step_acc_list = list()
acc_list = list()
new_ab_list = list()
markers=['o','v','*','D','X','.',',','^','<','>','1','2','3','4','8','s','p','P','h','H','+','x','D','d','|','_','o','v','*','D','X','.',',','^','<','>'
,'1','2','3','4','8','s','p','P','h','H','+','x','D','d','|','_']
plt.figure(figsize=(15,15))

# Loop through all regularization constants
for j in range(len(lam)):
#   Loop through all seasons
 for k in range(season):
#    Shuffle to get new validation sets and new training data
   foreta = training_lesslam.copy()
   np.random.shuffle(foreta)

   eta_val = foreta[0:50,:]
   eta_val_f = eta_val[:,0:6]
   eta_val_l = np.array([eta_val[:,6]]).T

#    Reinitialize a and b
   a = np.full((6,1),1)
   b = 4
   u = [a,b]
   new_ab = u.copy()
#    Reinitialize accuracy matrix
   acc = np.zeros((1,int(num_step/30)))
   step_acc = np.array([np.arange(0,num_step,30)])+30
```

```
    mag_a = np.zeros((1,int(num_step/30)))

#     Loop through all steps
    for i in range(num_step):
      rand = int(np.random.randint(foreta.shape[0]-50))
      training_sample = foreta[rand+50,:]
      training_sample_f = np.array([training_sample[0:6]])
      training_sample_l = np.array([[training_sample[6]]])

      xi = training_sample_f.T
      yi = training_sample_l
      grad = gradient(yi, xi, new_ab)
      new_ab = step(new_ab, grad, k)
#       Calculate accuracy every 30 steps
      if (1+i)%30 == 0:
        m = int((1+i)/30)
        acc[0,m-1] = accuracy(new_ab, eta_val_f, eta_val_l)
        mag_a[0,m-1] = np.linalg.norm(new_ab[0])

    acc_list.append((acc[0]))
    step_acc_list.append('Lambda: %s Season: %s' %(lam[j], k+1))
    new_ab_list.append(new_ab[:])

    plt.subplot(211)
    label.append(plt.plot(step_acc[0],acc[0],linewidth=0.5, label = 'Lambda: %s Season: %s' %(lam[j], k+1),
marker = markers[j+k+j+j+j],  markersize=5))
    plt.xticks(np.arange(min(step_acc[0]), max(step_acc[0])+1, 30))

    plt.subplot(212)
    label.append(plt.plot(step_acc[0],mag_a[0],linewidth=0.5, label = 'Lambda: %s Season: %s' %(lam[j], k+1),
marker = markers[j+k+j+j+j],  markersize=5))
    plt.xticks(np.arange(min(step_acc[0]), max(step_acc[0])+1, 30))

plt.legend(label)
plt.legend(loc='center left', bbox_to_anchor=(1, 1.1))
plt.show()

acc_list = np.array(acc_list)
step_acc_list = np.array(step_acc_list)
new_ab_list = np.array(new_ab_list)

print("max sum in >>>>", step_acc_list[sum(acc_list.T).argmax()] , "\na, b = ",
new_ab_list[sum(acc_list.T[:]).argmax()])
print("max half sum in >>>>", step_acc_list[sum(acc_list.T[7:]).argmax()], "\na, b = ",
new_ab_list[sum(acc_list.T[7:]).argmax()])
```

```
print("max single last step in >>>> ", step_acc_list[(acc_list.T[-1]).argmax()], "at: ",max(acc_list.T[-1]), "\na, b = ", new_ab_list[np.unravel_index((acc_list).argmax(),acc_list.shape)[0]])
```




```
max sum in >>>> Lambda: 1e-05 Season: 15
a, b =  [array([[0.31804914],
       [0.14026951],
       [0.53318245],
       [1.52159757],
       [0.22748698],
       [0.26621688]])
 -0.90625]
max half sum in >>>> Lambda: 1e-05 Season: 15
a, b =  [array([[0.31804914],
       [0.14026951],
       [0.53318245],
       [1.52159757],
       [0.22748698],
       [0.26621688]])
```

```
 -0.90625]
max single last step in >>>>  Lambda: 1e-05 Season: 15 at:  0.92
a, b =  [array([[0.31804914],
       [0.14026951],
       [0.53318245],
       [1.52159757],
       [0.22748698],
       [0.26621688]])
 -0.90625]
```

Part 2: Finding an appropriate regularization constant (lambda)

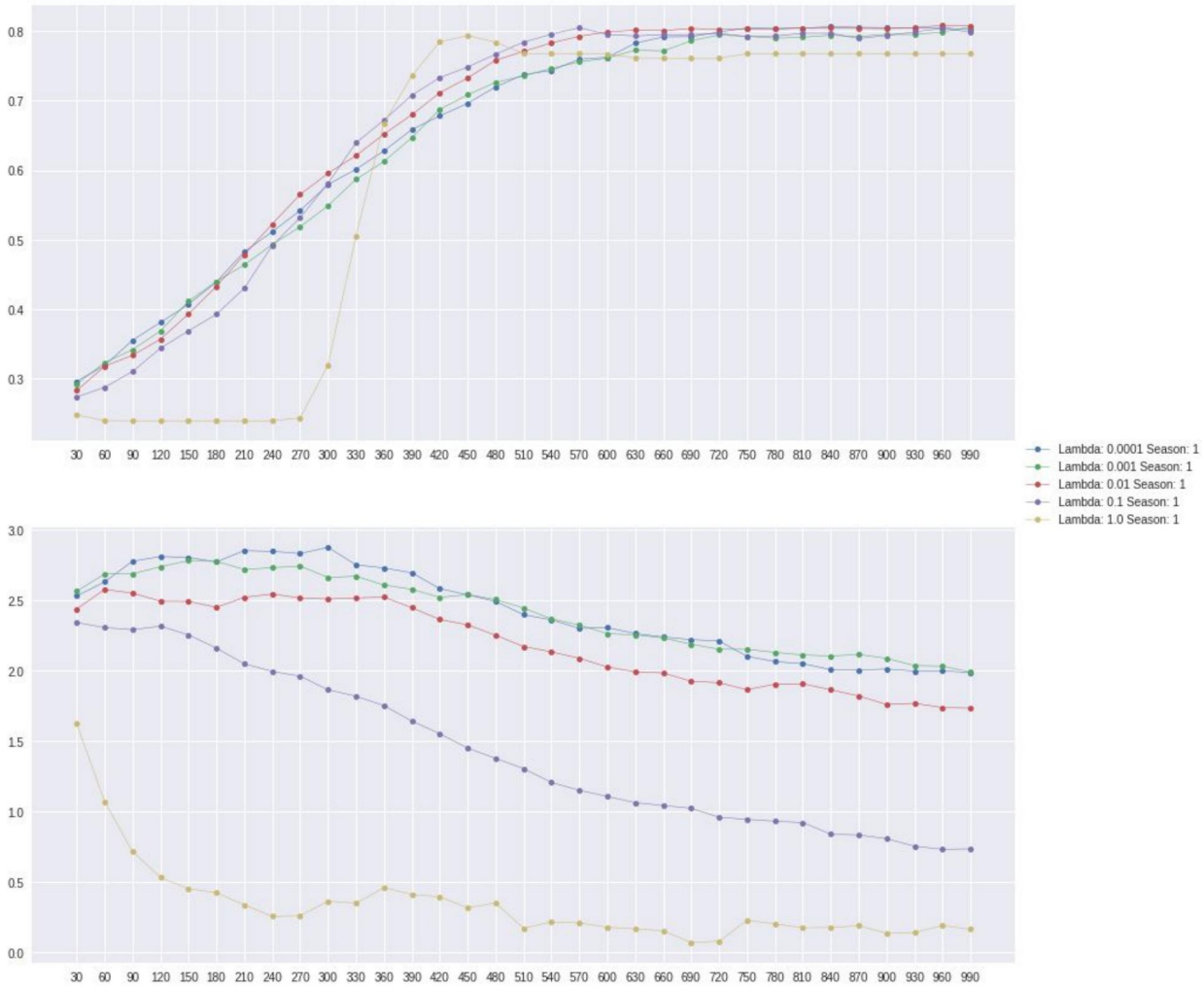
```
# xi is a column vector of features
# yi is a label
# a is a column vector
# b is a scalar

# Predict and return sign
def predict(u, x):
  s = np.dot(u[0].T,x.T)+u[1]
  s = np.sign(s)
  return s.T

# Compare real and predicted labels and get accuracy
def accuracy(u, x, y):
  s = predict(u, x)
  acc = np.sum(y == s)/y.size
  return acc

# Calculate gradient of u. Return gradient w.r.t. to a and b
def gradient(yi, xi, u, lam=1e-3):
  crit = yi[0,0]*gamma(u, xi)
  if crit >= 1:
    grad_b = 0
    grad_a = lam * u[0]
  else:
    grad_b = -yi[0,0]
    grad_a = lam * u[0] - yi[0,0]*xi
  return [grad_a,grad_b]

# Taking a step with eta and gradient. Return new value of u
def step(u, grad, k):
  a_new = u[0] - eta(k)*grad[0]
  b_new = u[1] - eta(k)*grad[1]
  return [a_new, b_new]
```

```
# Calculate gamma
def gamma(u, xi):
 gam = np.dot(u[0].T,xi) + u[1]
 return gam

# Returns training cost
def training_cost(u, yi, xi, gam, lam=1e-3):
 cost = (1/len(yi))*np.amax([0,1-np.dot(yi,gamma(u, xi))])+(lam/2)*np.dot(u[0].T,u[0])
 return cost

# Use fixed eta
def eta(k, m=1, n=50):
 return 1/64 #m/(2*k+n)
```

```
# Prepare data based on size of lambda validation set of 4396 (10%)
# Pick 10% of all training data for lambda validation set
# Pick 50 samples from training data for eta validation set
# Pick 1 sample from the remaining training data for learning
# ind is 10% of all training data

ind = 4396

shuffled = train.copy()
np.random.shuffle(shuffled)

lam_val = shuffled[0:ind,:]
lam_val_f = lam_val[:,0:6]
lam_val_l = np.array([lam_val[:,6]]).T

training_lesslam = shuffled[ind:,:]

# SVM Setup
num_step = 990
season = 1
lam = np.array([1e-4, 1e-3, 1e-2, 1e-1, 1])

label = list()
label2 = list()

step_acc_list = list()
acc_list = list()
new_ab_list = list()
markers=['o','v','*','D','X','.',',','^','<','>','1','2','3','4','8','s','p','P','h','H','+','x','D','d','|','_']
plt.figure(figsize=(15,15))
```

```
# Loop through all lambdas
for j in range(len(lam)):
#   Loop through all seasons
 for k in range(season):
#    Shuffle to get new validation sets and new training data
   foreta = training_lesslam.copy()
   np.random.shuffle(foreta)

   eta_val = foreta[0:50,:]
   eta_val_f = eta_val[:,0:6]
   eta_val_l = np.array([eta_val[:,6]]).T

#    Reinitialize a and b
   a = np.full((6,1),1)
   b = 4
   u = [a,b]
   new_ab = u.copy()
#    Reinitialize accuracy matrix
   acc = np.zeros((1,int(num_step/30)))
   mag_a = np.zeros((1,int(num_step/30)))
   step_acc = np.array([np.arange(0,num_step,30)])+30
#    Loop through all steps
   for i in range(num_step):
    rand = int(np.random.randint(foreta.shape[0]-50))
    training_sample = foreta[rand+50,:]
    training_sample_f = np.array([training_sample[0:6]])
    training_sample_l = np.array([[training_sample[6]]])

    xi = training_sample_f.T
    yi = training_sample_l
    grad = gradient(yi, xi, new_ab,lam[j])
    new_ab = step(new_ab, grad, k)
#     Calculate accuracy every 30 steps
    if (1+i)%30 == 0:
     m = int((1+i)/30)
     acc[0,m-1] = accuracy(new_ab, lam_val_f, lam_val_l)
     mag_a[0,m-1] = np.linalg.norm(new_ab[0])

   acc_list.append((acc[0]))
   step_acc_list.append('Lambda: %s Season: %s' %(lam[j], k+1))
   new_ab_list.append(new_ab[:])

   plt.subplot(211)
   label.append(plt.plot(step_acc[0],acc[0],linewidth=0.5, label = 'Lambda: %s Season: %s' %(lam[j], k+1),
```

```
marker = markers[k],  markersize=5))
   plt.xticks(np.arange(min(step_acc[0]), max(step_acc[0])+1, 30))

   plt.subplot(212)
   label.append(plt.plot(step_acc[0],mag_a[0],linewidth=0.5, label = 'Lambda: %s Season: %s' %(lam[j], k+1),
marker = markers[k],  markersize=5))
   plt.xticks(np.arange(min(step_acc[0]), max(step_acc[0])+1, 30))

plt.legend(label)
plt.legend(loc='center left', bbox_to_anchor=(1, 1.1))
plt.show()

acc_list = np.array(acc_list)
step_acc_list = np.array(step_acc_list)
new_ab_list = np.array(new_ab_list)

print("max sum in >>>>", step_acc_list[sum(acc_list.T).argmax()] , "\na, b = ",
new_ab_list[sum(acc_list.T[:]).argmax()])
print("max half sum in >>>>", step_acc_list[sum(acc_list.T[7:]).argmax()], "\na, b = ",
new_ab_list[sum(acc_list.T[7:]).argmax()])
print("max single last step in >>>> ", step_acc_list[(acc_list.T[-1]).argmax()], "at: ",max(acc_list.T[-1]), "\na, b
= ", new_ab_list[np.unravel_index((acc_list).argmax(),acc_list.shape)[0]])
```



max sum in >>>> Lambda: 0.01 Season: 1
a, b = [array([[0.37460604],
[0.1551405],
[0.55226118],
[1.46628975],
[0.498473],
[0.37187249]])
-0.953125]
max half sum in >>>> Lambda: 0.01 Season: 1
a, b = [array([[0.37460604],
[0.1551405],
[0.55226118],
[1.46628975],
[0.498473],
[0.37187249]])
-0.953125]
max single last step in >>>> Lambda: 0.01 Season: 1 at: 0.8080072793448589
a, b = [array([[0.37460604],
[0.1551405],
[0.55226118],

```
    [1.46628975],
    [0.498473  ],
    [0.37187249]])
-0.953125]
```

Part 3: Generating test set prediction file for autograder based on trained SVM model

```
#   Generate test set prediction file for autograder
a1 = np.array([0.37460604,\
      0.1551405 ,\
      0.55226118,\
      1.46628975,\
      0.498473  ,\
      0.37187249])
b1 = -0.953125

u1 = np.array((a1,b1))
pre = predict(u1, test_f)
pretxt = ""
for i in range(pre.size):
  if pre[i]==-1:
    pretxt=pretxt+'<=50K\n'
  else:
    pretxt=pretxt+'>50K\n'
f = open('drive/My Drive/CS 498 AML/HW2/submission6.txt', 'w')
f.write(pretxt)
f.close()
```