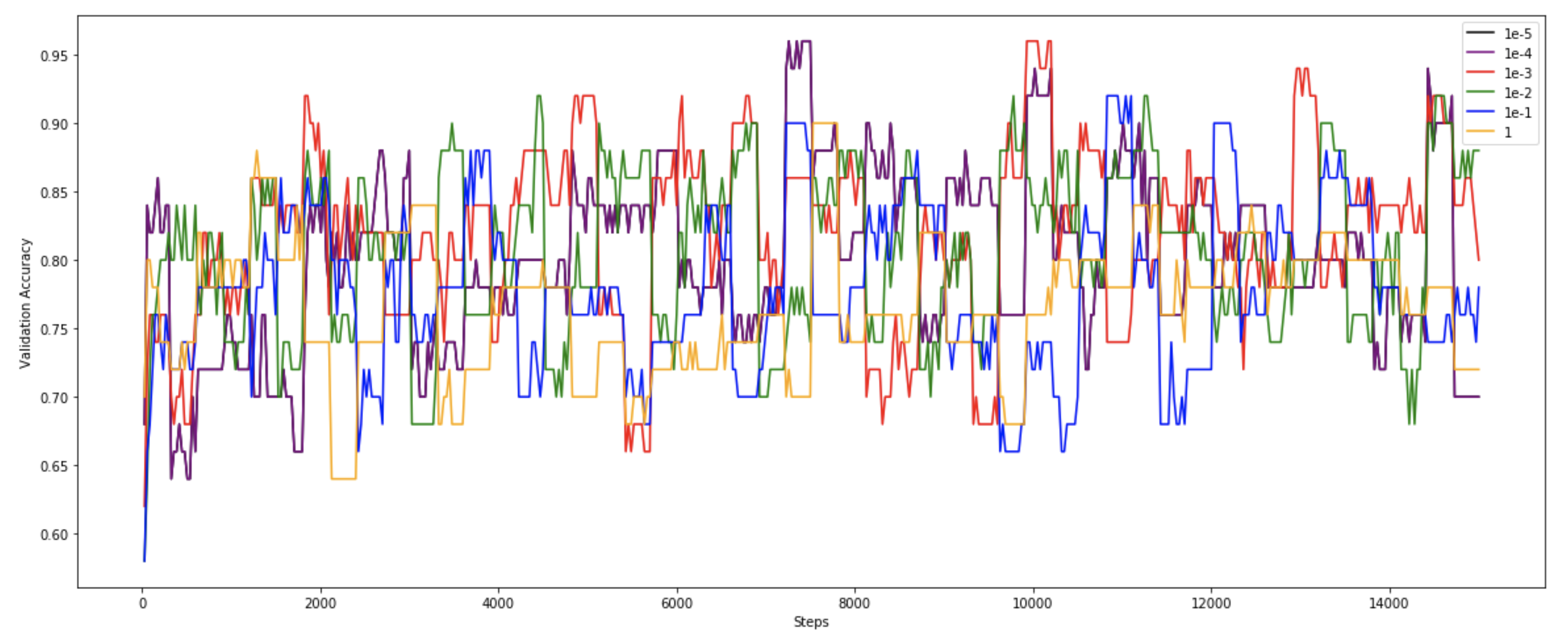
# Best Accuracy on the Test Set

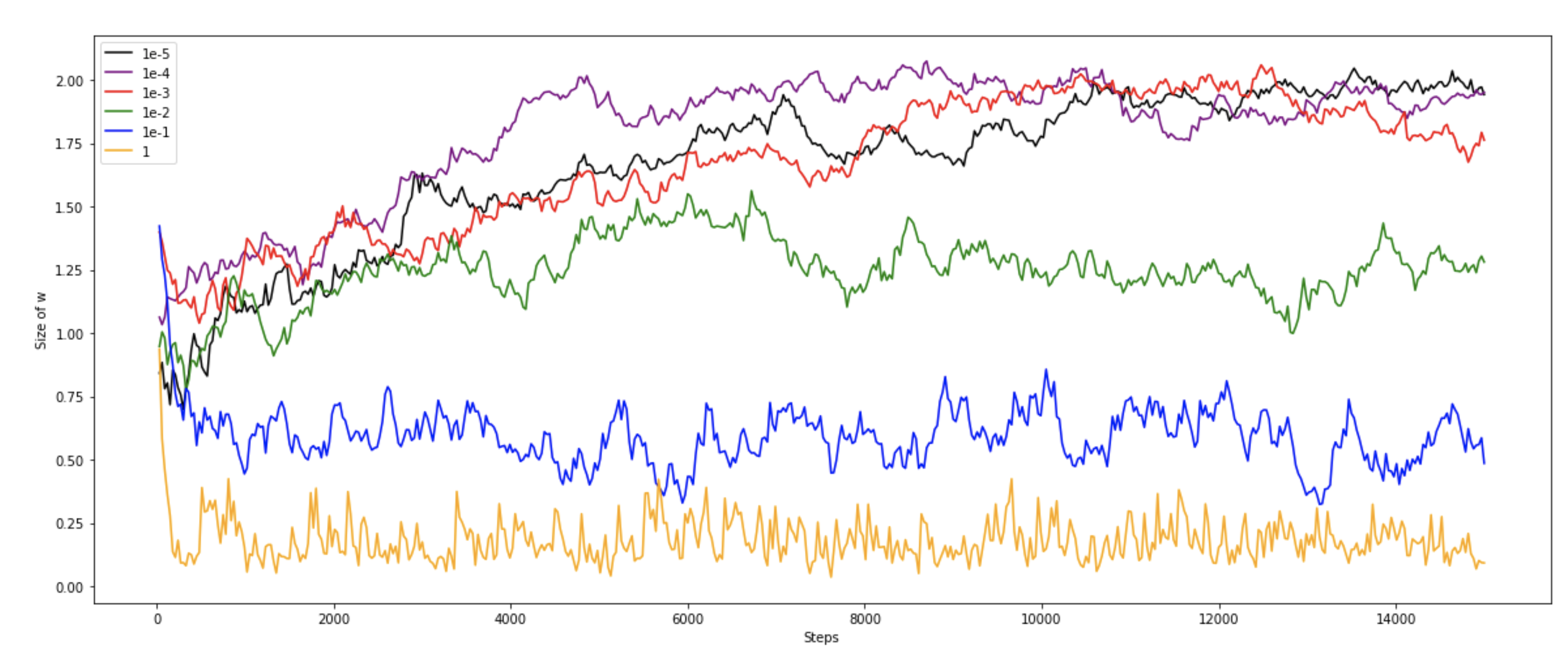
The Best Score = 80.9 (Lambda = 0.0001, learning rate = 0.01)



# Validation Accuracy every 30 steps each Regularization Constant

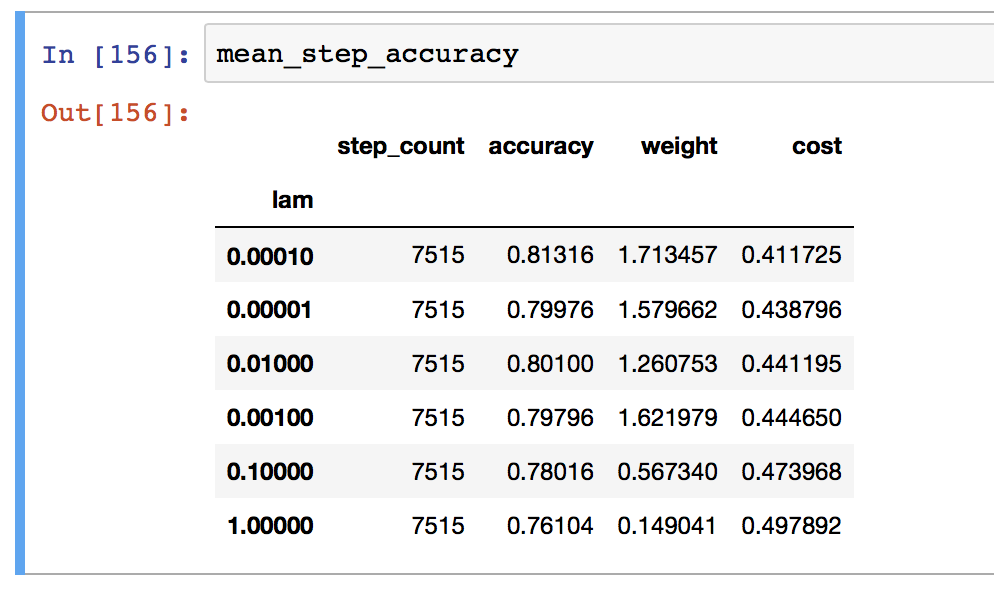


# Magnitude of the coefficient vector for every 30 steps for each Regularization Constant



# Best Value of Regularization Constant

The best value of Regularization constant is **0.0001**. This regularization constant is being chosen, because the cost seems to be **lowest cost** among other regularization constant (0.001, 0.00001 etc). The cost has been average over the 50 validation examples taken from the shuffle training set runs which extract per each session(50 session total), each have 300 steps , cost being taken after each 30 steps for each session per regularization constant)



The best Learning rate seems to be **0.01** (&gradually decrease in every steps). The reason is because of low average cost (below screen shot). The learning rate 0.001 seems to be slow and takes lots of steps to reach global minimum, however we are not going that many steps for each session (also can be seen from the Held Out error rate – 0.48). The 0.02 and 0.03 seems to be close in terms of average error rate, however 0.01 seems to be good enough

|  |  |  |  |
| --- | --- | --- | --- |
| Learning Rate | Error Rate | Learning Rate | Error Rate |
| learning\_rate = 0.01  **Min Cost = 0.41** |  | learning\_rate = 0.001  **Min Cost = 0.48** |  |
| learning\_rate = 0.02  **Min Cost = 0.42** |  | learning\_rate = 0.03  **Min Cost = 0.42** |  |

# Entire Code

|  |
| --- |
| **import** **numpy** **as** **np**  **import** **pandas** **as** **pd**  **import** **matplotlib.pyplot** **as** **plt**  %**matplotlib** inline  **def** load\_dataset(filepath='data/',print\_ind=**False**):  columns = ['age','workclass','fnlwgt','education','education-num','marital-status','occupation','relationship','race','sex','capital-gain','capital-loss','hours-per-week','native-country','target']  train = pd.read\_csv("data/train.txt",names=columns)  test = pd.read\_csv("data/test.txt",names=columns[:-1])    train['target'].replace(' <=50K',-1,inplace=**True**)  train['target'].replace(' >50K',1,inplace=**True**)  label=np.array(train['target']).reshape(len(train['target']),1)  train.drop('target',axis=1,inplace=**True**)  train = np.array(train)  test = np.array(test)  label = label.astype(int)  **if** (print\_ind):  print ("Train Shape: **{}** Test Shape**{}**".format(train.shape,test.shape))  **return** train,test,label  **def** preprocessing(data):  *#train,test,label=load\_dataset()*  train=extract\_contineous(data)  *#test=extract\_contineous(test)*  train\_scale=feature\_scaling(train)  test\_scale=feature\_scaling(test)  train\_with\_label = np.append(train\_scale,label,axis=1)  **return** train\_with\_label,test\_scale  **def** extract\_contineous(data):  cont\_columns = [0,2,4,10,11,12]  **return** (data[:,cont\_columns]).astype(float)  **def** feature\_scaling(data,print\_ind=**False**):  feature\_mean=data.mean(axis=0).astype(float)  feature\_var =data.var(axis=0).astype(float)  data = (data - feature\_mean) / np.sqrt(feature\_var)  **if** (print\_ind):  print ("Scale Shape:**{}**".format(input\_df\_scale.shape))  **return** data  **def** penalty\_term(a):  **return** 1/2 \* np.asscalar(np.transpose(a).dot(a))  **def** obj\_func(a,b,data):  obj = np.dot(a.T,data)+b  **return** obj  **def** gradient\_calc(w,lam,data):  X = data[:-1]  y = data[-1]  a = w[:-1]  b = w[-1]  diff = y \* obj\_func(a,b,X)  a\_delta = np.array([])  b\_delta = 0  **if** (diff >= 1):  a\_delta = lam\*a  b\_delta = 0  **else**:  a\_delta = np.subtract(lam \* a, (y \* X).reshape(6, 1))  b\_delta = -y  gradient = (np.append(np.array(a\_delta), np.array([b\_delta]))).reshape(7, 1)  **return** gradient      **def** cost\_function(w,lam,data):  a = w[:-1]  b = w[-1:]  m=len(data)  temp\_max\_val=0    **for** e **in** data:  X = e[:-1]  y = e[-1:]  obj = obj\_func(a,b,X)  error = 1 - y \* obj  temp\_max\_val+=max(0, np.asscalar(error))    max\_val = ((1/m)\*temp\_max\_val) + lam \* penalty\_term(a)  **return** max\_val  **def** pred\_calc(w,X):  a = w[:-1]  b = w[-1][0]  obj = obj\_func(a,b,X)  pred = np.sign(obj)[0]  **return** pred  **def** evaluate\_model(w,lam,data):  num\_correct = 0  **for** d **in** data:  X = d[:-1]  y = d[-1]  pred = pred\_calc(w,X)  **if** (pred == y):  num\_correct += 1  **return** (num\_correct/len(data))  **def** train\_test\_split(data,eval\_percent):  np.random.shuffle(data)  end\_loc = len(data)//eval\_percent  eval\_data=data[:end\_loc]  train\_data=data[end\_loc:]  **return** train\_data,eval\_data  **def** train\_model(train):  w = np.random.rand(7,1) *#initialize weight*  weight\_cost = {}  step\_count=0  num\_epochs = 50 *#initialize number of epochs*  num\_steps = 300 *#initialize number of steps*  *#l\_rate = 0.001 #initialize learning rate*  *#l\_rate = (1/(0.01\*i+50))*  costs = []  accuracy\_step\_wise = []  accuracy\_lam\_wise = []  *#train,test=preprocessing()*  np.random.shuffle(train) *# Shuffle train Dataset*  train\_set,eval\_set=train\_test\_split(train,10) *#|--10%(valid\_set)---|-----------------90%(train\_set)-----------------|*  epoch\_data = train\_set[:50] *#|-(50 epoch\_data)--|--------------90%-50 Example(train\_data)-----------------|*  train\_data = train\_set[50:] *#|-(50 epoch\_data)--|--------------90%-50 Example(train\_data)-----------------|*  **for** l **in** [1e-5,1e-4,1e-3,1e-2,1e-1,1]:  **for** i **in** range(num\_epochs):  **for** j **in** range(num\_steps):  step\_count += 1  gradient = gradient\_calc(w,l,train\_data[j])  l\_rate = (1/(0.01\*i+100))  step = l\_rate \* gradient  w = np.subtract(w, step)  **if** (step\_count % 30 == 0): *#Each Step = 30*  acuuracy\_step=evaluate\_model(w,l,epoch\_data) *#Each Step Level (epoch\_data)*  cost\_step = cost\_function(w, l, epoch\_data) *#Each Step Level (epoch\_data)*  accuracy\_step\_wise.append([l,step\_count,acuuracy\_step,np.sqrt(np.sum(w[:-1]\*\*2)),cost\_step]) *#Each Step Level*  np.random.shuffle(train\_set) *#|-----------------90%(train\_set)(Shuffle)-----------------|*  epoch\_data = train\_set[:50] *#|-(50 epoch\_data)--|--------------90%-50 Example(train\_data)-----------------|*  train\_data = train\_set[50:] *#|-(50 epoch\_data)--|--------------90%-50 Example(train\_data)-----------------|*  acuuracy\_lam=evaluate\_model(w,l,eval\_set) *#Each lamda level (epoch\_data)*  cost\_lam = cost\_function(w, l, eval\_set) *#Each lambda level (epoch\_data)*  weight\_cost[l] = {'W':w,'Accuracy':acuuracy\_lam, 'Cost':cost\_lam}  accuracy\_lam\_wise.append([l,step\_count,acuuracy\_lam,np.sqrt(np.sum(w[:-1]\*\*2)),cost\_lam]) *#Each Step Level*  step\_count=0  np.random.shuffle(train) *# Shuffle train Dataset*  train\_set,eval\_set=train\_test\_split(train,10) *#|--10%(valid\_set)---|-----------------90%(train\_set)-----------------|*  epoch\_data = train\_set[:50] *#|-(50 epoch\_data)--|--------------90%-50 Example(train\_data)-----------------|*  train\_data = train\_set[50:] *#|-(50 epoch\_data)--|--------------90%-50 Example(train\_data)-----------------|*  w = np.random.rand(7,1) *#initialize weight*  *#step\_accuracy=np.array(accuracy\_step\_wise)*  *#lam\_accuracy=np.array(accuracy\_lam\_wise)*  *#return step\_accuracy,lam\_accuracy*  **return** accuracy\_step\_wise,accuracy\_lam\_wise,weight\_cost  **def** plot\_val\_accuracy(step\_accuracy):  plt.subplots(figsize=(20,8))  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.00001][:,1],step\_accuracy[step\_accuracy[:,0] == 0.0001][:,2],color='black')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.0001][:,1],step\_accuracy[step\_accuracy[:,0] == 0.0001][:,2],color='purple')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.001][:,1],step\_accuracy[step\_accuracy[:,0] == 0.001][:,2],color='red')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.01][:,1],step\_accuracy[step\_accuracy[:,0] == 0.01][:,2],color='green')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.1][:,1],step\_accuracy[step\_accuracy[:,0] == 0.1][:,2],color='blue')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 1][:,1],step\_accuracy[step\_accuracy[:,0] == 1][:,2],color='orange')  plt.legend(['1e-5','1e-4','1e-3','1e-2','1e-1','1'])  plt.xlabel('Steps')  plt.ylabel('Validation Accuracy')    **def** plot\_magnitude\_w(step\_accuracy):  plt.subplots(figsize=(20,8))  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.00001][:,1],step\_accuracy[step\_accuracy[:,0] == 0.00001][:,3],color='black')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.0001][:,1],step\_accuracy[step\_accuracy[:,0] == 0.0001][:,3],color='purple')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.001][:,1],step\_accuracy[step\_accuracy[:,0] == 0.001][:,3],color='red')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.01][:,1],step\_accuracy[step\_accuracy[:,0] == 0.01][:,3],color='green')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 0.1][:,1],step\_accuracy[step\_accuracy[:,0] == 0.1][:,3],color='blue')  plt.plot(step\_accuracy[step\_accuracy[:,0] == 1][:,1],step\_accuracy[step\_accuracy[:,0] == 1][:,3],color='orange')  plt.legend(['1e-5','1e-4','1e-3','1e-2','1e-1','1'])  plt.xlabel('Steps')  plt.ylabel('Size of w')  **def** pred\_test(w,test):  sr\_pred\_test=[]  **for** data **in** test:  pred\_test\_val=pred\_calc(w,data)  **if** (pred\_test\_val == -1):  pred = '<=50K'  **elif**(pred\_test\_val == 1):  pred = '>50K'  sr\_pred\_test.append(pred)  *#sr\_pred\_test.append(pred\_test\_val)*  pd.DataFrame(sr\_pred\_test).to\_csv("submission.txt",index=**False**,header=**False**)    **def** main(show):  train,test,label=load\_dataset()  train\_contineous=extract\_contineous(train)  train\_scale=feature\_scaling(train\_contineous)  train\_with\_label = np.append(train\_scale,label,axis=1)  test\_contineous=extract\_contineous(test)  test\_scale=feature\_scaling(test\_contineous)  *#train\_with\_label = np.append(train\_scale,label,axis=1)*  *#train,test=preprocessing()*  step\_accuracy,lam\_accuracy,weight\_cost=train\_model(train\_with\_label)  **if** (show):  plot\_val\_accuracy(np.array(step\_accuracy))  plot\_magnitude\_w(np.array(step\_accuracy))  pred\_test(pd.DataFrame(weight\_cost).T.loc[0.0001].loc['W'],test\_scale)  **return** step\_accuracy,pd.DataFrame(lam\_accuracy,columns=['lam','step\_count','accuracy','weight','cost']).sort\_values(by='cost'),pd.DataFrame(weight\_cost).T,pd.DataFrame(step\_accuracy,columns=['lam','step\_count','accuracy','weight','cost']).groupby('lam').mean().sort\_values(by='cost')  step\_accuracy,lam\_accuracy,weight\_cost,mean\_step\_accuracy=main(**False**) |