**ID5030 :: Assignment-1**

Varun V (AE14B050)

**Solution : Multi Variable Logistic Regression:**

We follow these steps in our code:

* Import data into numpy arrays and assign classes according to the scheme given.
* Scale the training data features (not the dependent class variable) using a standard scalar and apply the same the test data.
* Create one hot encoding for our class variable and split it into 2/3 binary columns.
* Declare placeholders in Tensorflow for our training and test data sets.
* Declare the Weight and bias variables.
* Define logits and cross entropy loss equations using the above placeholders and variables.
* Declare the optimization algo along with the method and corresponding variable to work on.
* Initiate a new Tensorflow session and initialize the variables for this session.
* Calculate initial cross entropy loss.
* Run the session with the optimization algorithm while simultaneous calculating the cross entropy loss for both training and test data.
* Calculate the final classes from the model and report the statistical data required.

**(a) Binary Classification(<300=0; >300=1):**

**Code:**

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Mon Mar  5 10:30:51 2018
5. @author: Varun
6. """
8. **import** numpy as np
9. **import** pandas as pd
10. **import** matplotlib.pyplot as plt
11. **import** tensorflow as tf
12. **from** sklearn.model\_selection **import** train\_test\_split
13. **from** sklearn.preprocessing **import** StandardScaler,MinMaxScaler
14. **from** matplotlib **import** style
15. #statics
16. a = 0.8#learning rate
17. maxitr = 1000#maximum number of iterations
18. con = 0.000001#convergence criterion
20. #import data, assign classes, scale
21. data = pd.read\_csv('E:/8thsem/ML/ass3/30\_train\_features.csv')
22. test = pd.read\_csv('E:/8thsem/ML/ass3/30\_test\_features.csv')
23. data = np.array(data)
24. test = np.array(test)
26. data[:,-1][data[:,-1]<=300]=0
27. data[:,-1][data[:,-1]>=300]=1
29. test[:,-1][test[:,-1]<=300]=0
30. test[:,-1][test[:,-1]>=300]=1
32. scaler = StandardScaler()
33. data[:,:-1] = scaler.fit\_transform(data[:,:-1])
34. test[:,:-1] = scaler.transform(test[:,:-1])
36. features = data.shape[1]-1
37. samples = data.shape[0]
38. samples\_t = test.shape[0]
39. classes = np.unique(data[:,-1]).shape[0]
41. onehoty = np.eye(classes)[data[:,-1].astype('int64')]
42. onehoty\_t = np.eye(classes)[test[:,-1].astype('int64')]

45. batchsize = samples

48. #placeholders for input varaibles
49. x = tf.placeholder(tf.float32, [batchsize,features], name = "x")
50. y = tf.placeholder(tf.float32, [batchsize,classes], name = "y")
52. x\_t = tf.placeholder(tf.float32, [samples\_t,features], name = "x\_t")
53. y\_t = tf.placeholder(tf.float32, [samples\_t,classes], name = "y\_t")
55. #variables to optimize
56. w = tf.Variable(tf.random\_normal(shape = [features,classes], stddev = 0.01) , name = "weights")
57. b = tf.Variable(tf.zeros([1,classes]), name = "bias")
59. #prediction equation
60. logit = tf.matmul(x,w) + b
61. logit\_t = tf.matmul(x\_t,w) + b
62. entropy = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits = logit, labels = y)
63. entropy\_t = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits = logit\_t, labels = y\_t)
64. #loss
65. loss = tf.reduce\_mean(entropy,name= 'loss')
66. loss\_t = tf.reduce\_mean(entropy\_t,name= 'loss\_t')
68. #algorithm to use
69. algo = tf.train.GradientDescentOptimizer(learning\_rate = a).minimize(loss)
71. #session initiate
72. sess = tf.Session()
73. #initialize variables with default values already given
74. sess.run(tf.global\_variables\_initializer())


78. #initialize the losses to plot mean cross entropy, BGD variant
79. losses = np.empty(shape=[1],dtype=float)
80. losses[0] = sess.run(loss,feed\_dict={x:data[:,:-1].reshape(samples,features),y:onehoty.reshape(samples,classes)})
81. losses\_t = np.empty(shape=[1],dtype=float)
82. losses\_t[0] = sess.run(loss\_t,feed\_dict={x\_t:test[:,:-1].reshape(samples\_t,features),y\_t:onehoty\_t.reshape(samples\_t,classes)})



87. #train to optimize variables, BGD variant
88. **for** i **in** range(maxitr):
89. sess.run(algo,feed\_dict={x:data[:,:-1].reshape(batchsize,features),y:onehoty.reshape(batchsize,classes)})
90. losses = np.append(losses,sess.run(loss,feed\_dict={x:data[:,:-1].reshape(batchsize,features),y:onehoty.reshape(batchsize,classes)}))
91. losses\_t = np.append(losses\_t,sess.run(loss\_t,feed\_dict={x\_t:test[:,:-1].reshape(samples\_t,features),y\_t:onehoty\_t.reshape(samples\_t,classes)}))
92. **if**(abs(losses[-1]-losses[-2])<con):
93. **break**

96. #calculate final outcomes
97. finalw,finalb = sess.run([w,b])
98. total = np.exp(np.dot(data[:,:-1],finalw)+finalb)
99. total\_t = np.exp(np.dot(test[:,:-1],finalw)+finalb)
101. #manipulate for prediction
102. **for** i **in** range(classes):
103. dum = total[:,i]/np.sum(total,1)
104. dum\_t = total\_t[:,i]/np.sum(total\_t,1)
105. **if**(i==0):
106. y\_predicted = dum
107. y\_predicted\_t = dum\_t
108. **else**:
109. y\_predicted = np.c\_[y\_predicted,dum]
110. y\_predicted\_t = np.c\_[y\_predicted\_t,dum\_t]
112. maxcol = np.max(y\_predicted,1)
113. maxcol\_t = np.max(y\_predicted\_t,1)
115. **for** i **in** range(samples):
116. y\_predicted[i,:][y\_predicted[i,:]!=maxcol[i]]=0
118. y\_predicted[i,:][y\_predicted[i,:]==maxcol[i]]=1
119. **for** i **in** range(samples\_t):
120. y\_predicted\_t[i,:][y\_predicted\_t[i,:]!=maxcol\_t[i]]=0
121. y\_predicted\_t[i,:][y\_predicted\_t[i,:]==maxcol\_t[i]]=1
123. diff = np.sum((onehoty - y\_predicted)\*\*2,1)/(classes-1)
124. diff\_t = np.sum((onehoty\_t - y\_predicted\_t)\*\*2,1)/(classes-1)
125. accuracy = diff[diff==0].shape[0]/diff.shape[0]
126. accuracy\_t = diff\_t[diff\_t==0].shape[0]/diff\_t.shape[0]
128. yfinal = np.dot(y\_predicted,np.unique(data[:,-1]))
129. yfinal\_t = np.dot(y\_predicted\_t,np.unique(test[:,-1]))
131. #correlation coefficients
132. corr\_coefs = np.corrcoef(yfinal,data[:,-1],rowvar = False)
133. corr\_coefs\_t = np.corrcoef(yfinal\_t,test[:,-1],rowvar = False)
135. #sensitivity and specificity
136. sensitivity\_t,specificity\_t = [],[]
137. positives\_t = test[:,-1][test[:,-1]==0].shape[0]
138. sensitivity\_t.append(yfinal\_t[:positives\_t][yfinal\_t[:positives\_t]==0].shape[0]/positives\_t)
139. specificity\_t.append(yfinal\_t[positives\_t:][yfinal\_t[positives\_t:]!=0].shape[0]/(samples\_t-positives\_t))

142. #plt.plot(losses)
143. #plt.plot(losses\_t)

**Results:**

|  |  |
| --- | --- |
| **Final Cross Entropy Loss** | 0.38271129 |
| **Prediction accuracy-train** | 83.673469% |
| **Prediction accuracy-test** | 72.727273% |
| **Correlation coefficient-test** | 0.4380479 |
| **Sensitivity-test-class0** | 0.5 |
| **Specificity-test-class0** | 0.89473684 |
| **Sensitivity -test-class1** | 0.89473684 |
| **Specificity-test-class1** | 0.5 |

**(a) Multinomial Classification(<300=0; 300-450=1; >450=2):**

**Code:**

1. # -\*- coding: utf-8 -\*-
2. """
3. Created on Mon Mar  5 10:30:51 2018
5. @author: Varun
6. """
8. **import** numpy as np
9. **import** pandas as pd
10. **import** matplotlib.pyplot as plt
11. **import** tensorflow as tf
12. **from** sklearn.model\_selection **import** train\_test\_split
13. **from** sklearn.preprocessing **import** StandardScaler,MinMaxScaler
14. **from** matplotlib **import** style
16. #statics
17. a = 0.1 #learning rate
18. maxitr = 1000#maximum number of iterations
19. con = 0.000001#convergence criterion
20. variant = 'BGD'#SDG or BGD as required
21. #import data, assign classes, scale
22. data = pd.read\_csv('E:/8thsem/ML/ass3/30\_train\_features.csv')
23. test = pd.read\_csv('E:/8thsem/ML/ass3/30\_test\_features.csv')
24. data = np.array(data)
25. test = np.array(test)
27. data[:,-1][data[:,-1]<=300]=0
28. data[:,-1][np.logical\_and(data[:,-1]>300,data[:,-1]<450)]=1
29. data[:,-1][data[:,-1]>=450]=2
31. test[:,-1][test[:,-1]<=300]=0
32. test[:,-1][np.logical\_and(test[:,-1]>300,test[:,-1]<450)]=1
33. test[:,-1][test[:,-1]>=450]=2
35. scaler = StandardScaler()
36. data[:,:-1] = scaler.fit\_transform(data[:,:-1])
37. test[:,:-1] = scaler.transform(test[:,:-1])
39. features = data.shape[1]-1
40. samples = data.shape[0]
41. samples\_t = test.shape[0]
42. classes = np.unique(data[:,-1]).shape[0]
44. onehoty = np.eye(classes)[data[:,-1].astype('int64')]
45. onehoty\_t = np.eye(classes)[test[:,-1].astype('int64')]

48. batchsize = samples

51. #placeholders for input varaibles
52. x = tf.placeholder(tf.float32, [batchsize,features], name = "x")
53. y = tf.placeholder(tf.float32, [batchsize,classes], name = "y")
55. x\_t = tf.placeholder(tf.float32, [samples\_t,features], name = "x\_t")
56. y\_t = tf.placeholder(tf.float32, [samples\_t,classes], name = "y\_t")
58. #variables to optimize
59. w = tf.Variable(tf.random\_normal(shape = [features,classes], stddev = 0.01) , name = "weights")
60. b = tf.Variable(tf.zeros([1,classes]), name = "bias")
62. #prediction equation
63. logit = tf.matmul(x,w) + b
64. logit\_t = tf.matmul(x\_t,w) + b
65. entropy = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits = logit, labels = y)
66. entropy\_t = tf.nn.softmax\_cross\_entropy\_with\_logits\_v2(logits = logit\_t, labels = y\_t)
67. #loss
68. loss = tf.reduce\_mean(entropy,name= 'loss')
69. loss\_t = tf.reduce\_mean(entropy\_t,name= 'loss\_t')
71. #algorithm to use
72. algo = tf.train.GradientDescentOptimizer(learning\_rate = a).minimize(loss)
74. #session initiate
75. sess = tf.Session()
76. #initialize variables with default values already given
77. sess.run(tf.global\_variables\_initializer())


81. #initialize the losses to plot mean cross entropy, BGD variant
82. losses = np.empty(shape=[1],dtype=float)
83. losses[0] = sess.run(loss,feed\_dict={x:data[:,:-1].reshape(samples,features),y:onehoty.reshape(samples,classes)})
84. losses\_t = np.empty(shape=[1],dtype=float)
85. losses\_t[0] = sess.run(loss\_t,feed\_dict={x\_t:test[:,:-1].reshape(samples\_t,features),y\_t:onehoty\_t.reshape(samples\_t,classes)})

88. #train to optimize variables, BGD variant
89. **for** i **in** range(maxitr):
90. sess.run(algo,feed\_dict={x:data[:,:-1].reshape(batchsize,features),y:onehoty.reshape(batchsize,classes)})
91. losses = np.append(losses,sess.run(loss,feed\_dict={x:data[:,:-1].reshape(batchsize,features),y:onehoty.reshape(batchsize,classes)}))
92. losses\_t = np.append(losses\_t,sess.run(loss\_t,feed\_dict={x\_t:test[:,:-1].reshape(samples\_t,features),y\_t:onehoty\_t.reshape(samples\_t,classes)}))
93. **if**(abs(losses[-1]-losses[-2])<con):
94. **break**

97. #calculate final outcomes
98. finalw,finalb = sess.run([w,b])
99. total = np.exp(np.dot(data[:,:-1],finalw)+finalb)
100. total\_t = np.exp(np.dot(test[:,:-1],finalw)+finalb)
102. #manipulate for prediction
103. **for** i **in** range(classes):
104. dum = total[:,i]/np.sum(total,1)
105. dum\_t = total\_t[:,i]/np.sum(total\_t,1)
106. **if**(i==0):
107. y\_predicted = dum
108. y\_predicted\_t = dum\_t
109. **else**:
110. y\_predicted = np.c\_[y\_predicted,dum]
111. y\_predicted\_t = np.c\_[y\_predicted\_t,dum\_t]
113. maxcol = np.max(y\_predicted,1)
114. maxcol\_t = np.max(y\_predicted\_t,1)
116. **for** i **in** range(samples):
117. y\_predicted[i,:][y\_predicted[i,:]!=maxcol[i]]=0
119. y\_predicted[i,:][y\_predicted[i,:]==maxcol[i]]=1
120. **for** i **in** range(samples\_t):
121. y\_predicted\_t[i,:][y\_predicted\_t[i,:]!=maxcol\_t[i]]=0
122. y\_predicted\_t[i,:][y\_predicted\_t[i,:]==maxcol\_t[i]]=1
124. diff = np.sum((onehoty - y\_predicted)\*\*2,1)/(classes-1)
125. diff\_t = np.sum((onehoty\_t - y\_predicted\_t)\*\*2,1)/(classes-1)
126. accuracy = diff[diff==0].shape[0]/diff.shape[0]
127. accuracy\_t = diff\_t[diff\_t==0].shape[0]/diff\_t.shape[0]
129. yfinal = np.dot(y\_predicted,np.unique(data[:,-1]))
130. yfinal\_t = np.dot(y\_predicted\_t,np.unique(test[:,-1]))
132. #correlation coefficients
133. corr\_coefs = np.corrcoef(yfinal,data[:,-1],rowvar = False)
134. corr\_coefs\_t = np.corrcoef(yfinal\_t,test[:,-1],rowvar = False)
136. #sensitivity and specificity
137. sensitivity\_t,specificity\_t = [],[]

140. positives\_t = test[:,-1][test[:,-1]==0].shape[0]
141. positives\_t2 = test[:,-1][test[:,-1]==1].shape[0]
142. positives\_t3 = test[:,-1][test[:,-1]==2].shape[0]
144. sensitivity\_t.append(yfinal\_t[:positives\_t][yfinal\_t[:positives\_t]==0].shape[0]/positives\_t)
145. specificity\_t.append(yfinal\_t[positives\_t:][yfinal\_t[positives\_t:]!=0].shape[0]/(samples\_t-positives\_t))
147. sensitivity\_t.append(yfinal\_t[positives\_t:positives\_t+positives\_t2][yfinal\_t[positives\_t:positives\_t+positives\_t2]==1].shape[0]/positives\_t2)
148. specificity\_t.append(np.concatenate((yfinal\_t[:positives\_t],yfinal\_t[-positives\_t3:]))[np.concatenate((yfinal\_t[:positives\_t],yfinal\_t[-positives\_t3:]))!=1].shape[0]/(samples\_t-positives\_t2))
150. sensitivity\_t.append(yfinal\_t[positives\_t+positives\_t2:][yfinal\_t[positives\_t+positives\_t2:]==2].shape[0]/positives\_t3)
151. specificity\_t.append(yfinal\_t[:-positives\_t3][yfinal\_t[:-positives\_t3]!=2].shape[0]/(samples\_t-positives\_t3))
153. #plt.plot(losses)
154. #plt.plot(losses\_t)

**Results:**

|  |  |
| --- | --- |
| **Final Cross Entropy Loss** | 0.57090735 |
| **Prediction accuracy-train** | 79.591837% |
| **Prediction accuracy-test** | 0.4848485% |
| **Correlation coefficient-test** | 0.27970349 |
| **Sensitivity-test-class0** | 0.57142857 |
| **Specificity-test-class0** | 0.84210526 |
| **Sensitivity -test-class1** | 0.3 |
| **Specificity-test-class1** | 0.86956521 |
| **Sensitivity-test-class2** | 0.55555556 |
| **Specificity-test-class2** | 0.54166667 |