CSC 535 Data Mining

## Assignment 4 Report

### Submitted to:

### Dr. Jamil Saquer

### Author(s):

### Khaled Hossain

**Perceptron**

**Introduction**

Chart, scatter chart

Description automatically generatedFor this assignment I used the dataset provided by the instructor. Two datasets were provided, training data and testing data. The train data consists of 80 rows and 3 columns, the test data consists of 20 rows and 3 columns. My approach for solving the problem was first by reading the data and converting into X\_train, X\_test, y\_train, y\_test. Then I implemented the perceptron algorithm. The classes to be classified had the values 1 or -1. The dataset provide was linearly separable.

**Fig:- Scatter plot of training data**

**Background**

The perceptron algorithm is a supervised learning algorithm for binary classifiers. This algorithm is a type of linear classifier that makes its predictions based on a linear predictor function by combining a set of weights with the feature vector. A simple perceptron can be used to separate any two sets of numbers that are linearly separable. The threshold activation function used for this algorithm was the unit step activation function. This function outputs 1 where x is greater than or equal to 1. If it is less than 1 it outputs -1. The stopping criterion used for the perceptron algorithm was a certain number of iterations.

**Implementation**

The perceptron algorithm was implemented using the unit step activation function and the stopping criterion was a certain number of iterations. The algorithm update’s weight based on error. The learning rate controls how fast the network learns. I used a learning rate of 0.02. If a learning rate is too large, the change in weights become large and the perceptron become unstable. If the learning rate is too small that the train is very slow, however, the percepron is more stable and resistant to noise. We initialize the weights that repeat for each train sample first we use the linear function than we use the unit step function, we continue this until the stopping condition is met.

**Chart, scatter chart

Description automatically generatedExperimental Setup and Results**

**A close up of a newspaper

Description automatically generatedFig:- Testing data set scatter plot and learned hyperplane**

The accuracy of the perceptron was calculated to be 100%, the learned weights were: [0.7688 -0.6938]. The learned bias was calculated to be 0.96

**Conclusion**

The real dataset called “Bank Note Authentication” used was taken from Kaggle. It consists of data that was extracted from images that were taken from genuine and forged banknote-like specimens. The attributes of the dataset were:-

1. variance of Wavelet Transformed image (continuous)
2. skewness of Wavelet Transformed image (continuous)
3. curtosis of Wavelet Transformed image (continuous)
4. entropy of image (continuous)
5. class (integer)

After running the perceptron algorithm, I was able to get 99.7% accuracy. After trying with different number of iterations and learning rates I learned that if the learning rate used too big or small it decreases the accuracy. For, example when I increased the accuracy the learning rate from 0.05 to 0.09 the accuracy decreased to 97.66%

**References**

Shantanu. (2020, January 26). BankNote Authentication UCI. Retrieved November 07, 2020, from https://www.kaggle.com/shantanuss/banknote-authentication-uci

**Code**

The program should be run using jupyter notebook

"""

Program: hw4.ipynb

Programmed By: Khaled Hossain

Description: Implementation of Perceptron.

"""

5

6 #-----------------------------------------------Imports---------------------------------------

7 import numpy as np

8 import pandas as pd

9 import matplotlib.pyplot as plt

10 from sklearn.datasets import make\_blobs

11 #---------------------------------------------------------------------------------------------

12 # In[152]:

13

14

15 train\_data = pd.read\_csv('training\_data.txt', sep=" ", header=None)

16 train\_data.columns = ["a", "b", "label"]

17 test\_data = pd.read\_csv('testing\_data.txt', sep=" ", header=None)

18 test\_data.columns = ["a", "b", "label"]

19

20 X\_train = np.array(train\_data[['a','b']])

21 y\_train = np.array(train\_data['label'])

22

23 X\_test = np.array(test\_data[['a','b']])

24 y\_test = np.array(test\_data['label'])

25

26 test\_data

27

28

29 # In[153]:

30

31

32 class Perceptron:

33 # initialize learning rate, number of iterations and the activation function

34 def \_\_init\_\_(self, learning\_rate=0.02, n\_iters=10):

35 self.lr = learning\_rate

36 self.n\_iters = n\_iters

37 self.activation\_func = self.\_unit\_step\_func

38

39 def fit(self, X, y):

40 n\_samples, n\_features = X.shape

41 # print(X.shape)

42 # Initialize weights

43 self.weights = np.zeros(n\_features)

44 self.bias = 0

45 # we make sure y consists of only class 1 and -1

46 y\_ = []

47 for i in y:

48 if i > 0:

49 y\_.append(1)

50 else:

51 y\_.append(-1)

52

53 for \_ in range(self.n\_iters):

54 # first we apply linear function which is:-

55 # w\*transpose(x) \* bias then we use the unit step function

56 for index, x\_i in enumerate(X):

57 # print(x\_i)

58 linear\_output = np.dot(x\_i, self.weights) + self.bias

59 y\_predicted = self.activation\_func(linear\_output)

60

61 # Perceptron update rule

62 update = self.lr \* (y\_[index] - y\_predicted)

63

64 self.weights += update \* x\_i

65 #print("weights: ", self.weights)

66 self.bias += update

67 #print(self.bias)

68

69

70

71

72 def accuracy(self, y\_true, y\_pred):

73 accuracy = np.sum(y\_true == y\_pred) / len(y\_true)

74 return accuracy

75

76 def predict(self, X):

77 #print(x)

78 #print(self.weights)

79 #print(self.bias)

80 linear\_output = np.dot(X, self.weights) + self.bias

81 y\_predicted = self.activation\_func(linear\_output)

82 return y\_predicted

83

84 # def sigmoid(self,X):

85 # return 1/(1+np.exp(-X))

86

87 def \_unit\_step\_func(self, x):

88 # print(np.where(x>=1, 1, -1))

89 return np.where(x>=1, 1, -1)

90

91

92 # In[154]:

93

94

95 p = Perceptron(learning\_rate = 0.01, n\_iters = 10)

96 p.fit(X\_train, y\_train)

97 predictions = p.predict(X\_test)

98

99 obj = []

100 for i in range(len(X\_test)):

101 obj.append(X\_test[i])

102

103 # print(obj)

104

105 for i in range(len(predictions)):

106 print(obj[i], "Actual label", y\_test[i], "Predicted label: ", predictions[i])

107

108 accuracy = p.accuracy(y\_test, predictions)

109 print("Accuracy: ", accuracy\*100, '%')

110 print("Learned weights are: ", p.weights)

111 print("Learned bias: ", p.bias)

112

113

114 # In[155]:

115

116

117 # training data set and learned hyperplane

118

119 fig = plt.figure()

120 ax = fig.add\_subplot(1,1,1)

121 plt.scatter(X\_train[:,0], X\_train[:,1],marker='o',c=y\_train)

122

123 x0\_1 = np.amin(X\_train[:,0])

124 # print(x0\_1)

125 x0\_2 = np.amax(X\_train[:,0])

126 # print(x0\_2)

127 # print(p.weights)

128 x1\_1 = (-p.weights[0] \* x0\_1 - p.bias) / p.weights[1]

129 # print(x1\_1)

130 x1\_2 = (-p.weights[0] \* x0\_2 - p.bias) / p.weights[1]

131 # print(x1\_2)

132

133 ax.plot([x0\_1, x0\_2], [x1\_1, x1\_2])

134

135 ymin = np.amin(X\_train[:,1])

136 ymax = np.amax(X\_train[:,1])

137 ax.set\_ylim([ymin-3,ymax+3])

138

139 plt.show()

140

141

142 # In[156]:

143

144

145 fig = plt.figure()

146 ax = fig.add\_subplot(1,1,1)

147 plt.scatter(X\_test[:,0], X\_test[:,1],marker='o',c=y\_test)

148

149 plt.show()

150

151

152 # In[157]:

153

154

155 # Testing data set scatter plot and learned hyperplane

156

157 X, y = make\_blobs(n\_samples=len(y\_test), centers=2, random\_state=6)

158 #p.fit(X,y)

159 p.fit(X, y)

160

161 plt.scatter(X[:, 0], X[:, 1], c=y, s=55, cmap=plt.cm.Paired)

162

163 # plot the decision function

164 ax = plt.gca()

165 xlim = ax.get\_xlim()

166 ylim = ax.get\_ylim()

167

168 # plot decision boundary and margins

169 ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,

170 linestyles=['--', '-', '--'])

171

172 # plot support vectors

173 ax.scatter(clf.support\_vectors\_[:, 0], clf.support\_vectors\_[:, 1], s=100,

174 linewidth=1, facecolors='none', edgecolors='k')

175 plt.show()

176

177

178 # In[158]:

179

180

181 # Real dataset

182 train\_data = pd.read\_csv('BankNoteAuthentication.csv')

183 train\_data

184

185

186 # In[159]:

187

188

189 test\_data = train\_data.sample(frac =.25)

190 test\_data.count()

191 test\_data

192

193

194 # In[160]:

195

196

197 class Perceptron:

198

199 def \_\_init\_\_(self, learning\_rate=0.02, n\_iters=25):

200 self.lr = learning\_rate

201 self.n\_iters = n\_iters

202 self.activation\_func = self.\_unit\_step\_func

203

204 def fit(self, X, y):

205 n\_samples, n\_features = X.shape

206 self.weights = np.zeros(n\_features)

207 self.bias = 0

208

209 y\_ = []

210 for i in y:

211 if i > 0:

212 y\_.append(1)

213 else:

214 y\_.append(0)

215 #print(y\_)

216

217

218 for \_ in range(self.n\_iters):

219

220 for index, x\_i in enumerate(X):

221 # print(x\_i)

222 linear\_output = np.dot(x\_i, self.weights) + self.bias

223 y\_predicted = self.activation\_func(linear\_output)

224

225 # Perceptron update rule

226 update = self.lr \* (y\_[index] - y\_predicted)

227

228 self.weights += update \* x\_i

229 #print("weights: ", self.weights)

230 self.bias += update

231 #print(self.bias)

232

233

234 def accuracy(self, y\_true, y\_pred):

235 accuracy = np.sum(y\_true == y\_pred) / len(y\_true)

236 return accuracy

237

238 def predict(self, X):

239 #print(x)

240 #print(self.weights)

241 #print(self.bias)

242 linear\_output = np.dot(X, self.weights) + self.bias

243 y\_predicted = self.activation\_func(linear\_output)

244 return y\_predicted

245

246 def \_unit\_step\_func(self, x):

247 # print(np.where(x>=1, 1, -1))

248 return np.where(x>=1, 1, 0)

249

250

251 # In[163]:

252

253

254 X\_train = np.array(train\_data[['variance', 'skewness', 'curtosis', 'entropy']])

255 y\_train = np.array(train\_data['class'])

256

257 X\_test = np.array(test\_data[['variance', 'skewness', 'curtosis', 'entropy']])

258 y\_test = np.array(test\_data['class'])

259

260 p = Perceptron(learning\_rate = 0.05, n\_iters = 25)

261 p.fit(X\_train, y\_train)

262 predictions = p.predict(X\_test)

263

264 obj = []

265 for i in range(len(X\_test)):

266 obj.append(X\_test[i])

267

268 # print(obj)

269

270 for i in range(len(predictions)):

271 print(obj[i], "Actual label", y\_test[i], "Predicted label: ", predictions[i])

272

273 accuracy = p.accuracy(y\_test, predictions)

274 print("Accuracy: ", accuracy\*100, '%')

275 print("Learned weights are: ", p.weights)

276 print("Learned bias: ", p.bias)

277

278

279 # In[ ]:

280

281

282

283

284

285 # In[ ]:

286

287

288

289

290