10/30/2020 svm.py

```
1 import numpy as np
2 from numpy import linalg
3 import cvxopt
4 import cvxopt.solvers
5 import sys
6 import pandas as pd
7 cvxopt.solvers.options['show_progress'] = False
9 # Reads the data from CSV files, converts it into Dataframe and returns x and y
  dataframes
10 def getDataframe(filePath):
11
      dataframe = pd.read_csv(filePath)
12
      y = dataframe['y']
13
      x = dataframe.drop('y', axis=1)
14
      y = y*2 -1.0
15
      return x.to_numpy(), y.to_numpy()
16
  def compute_accuracy(predicted_y, y):
17
18
      acc = 100.0
      acc = np.sum(predicted_y == y)/predicted_y.shape[0]
19
20
      return acc
21
22 def gaussian_kernel_point(x, y, sigma=5.0):
23
      return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))
24
25
  def linear_kernel(X, Y=None):
      Y = X if Y is None else Y
26
27
      m = X.shape[0]
28
      n = Y.shape[0]
29
      assert X.shape[1] == Y.shape[1]
30
      kernel matrix = np.zeros((m, n))
31
      #=======#
32
      # STRART YOUR CODE HERE #
33
      #=======#
34
      for i in range(m):
35
          for j in range(n):
36
              kernel_matrix[i,j] = np.dot(X[i], Y[j])
37
      #=======#
38
          END YOUR CODE HERE
39
      #=======#
40
      return kernel_matrix
41
42
  def polynomial_kernel(X, Y=None, degree=3):
43
      Y = X if Y is None else Y
44
      m = X.shape[0]
45
      n = Y.shape[0]
46
      assert X.shape[1] == Y.shape[1]
47
      kernel matrix = np.zeros((m, n))
48
      #=======#
49
      # STRART YOUR CODE HERE #
50
      #=======#
51
      for i in range(m):
52
          for j in range(n):
              kernel_matrix[i,j] = (1+np.dot(X[i],Y[j])) ** degree
53
54
      #=============================
         END YOUR CODE HERE
55
56
      #=======#
57
      return kernel matrix
58
59 def gaussian_kernel(X, Y=None, sigma=5.0):
```

10/30/2020 svm.py Y = X if Y is None else Y 60 61 m = X.shape[0]62 n = Y.shape[0]assert X.shape[1] == Y.shape[1] 63 64 kernel matrix = np.zeros((m, n)) #=======# 65 # STRART YOUR CODE HERE 66 #=======# 67 68 for i in range(m): 69 for j in range(n): kernel_matrix[i,j] = np.exp((-np.linalg.norm(X[i]-Y[j])**2)/ (2* 70 (sigma**2))) 71 #=======# 72 END YOUR CODE HERE 73 #=======# 74 return kernel matrix 75 76 77 # Bonus question: vectorized implementation of Gaussian kernel 78 # If you decide to do the bonus question, comment the gaussian kernel function above, 79 # then implement and uncomment this one. 80 # def gaussian_kernel(X, Y=None, sigma=5.0): 81 # return 82 83 class SVM(object): def __init__(self): 84 85 self.train_x = pd.DataFrame() 86 self.train y = pd.DataFrame() 87 self.test_x = pd.DataFrame() self.test_y = pd.DataFrame() 88 self.kernel name = None 89 self.kernel = None 90 91 92 def load_data(self, train_file, test_file): 93 self.train_x, self.train_y = getDataframe(train_file) 94 self.test_x, self.test_y = getDataframe(test_file) 95 96 def train(self, kernel_name='linear_kernel', C=None): 97 98 self.kernel_name = kernel_name if(kernel_name == 'linear_kernel'): 99 100 self.kernel = linear_kernel elif(kernel name == 'polynomial kernel'): 101 102 self.kernel = polynomial_kernel elif(kernel_name == 'gaussian_kernel'): 103 self.kernel = gaussian_kernel 104 105 else: raise ValueError("kernel not recognized") 106 107 self.C = C108 if self.C is not None: 109 110 self.C = float(self.C) 111 112 self.fit(self.train_x, self.train_y) 113 # predict labels for test dataset 114 115 def predict(self, X): 116 if self.w is not None: ## linear case 117 n = X.shape[0]118 predicted_y = np.zeros(n)

```
10/30/2020
                                             svm.py
119
               #==================================
120
               # STRART YOUR CODE HERE #
121
               #=======#
               predicted_y = np.dot(self.w, np.transpose(X)) + self.b
122
123
               #=======#
124
                   END YOUR CODE HERE
125
               #========#
126
               return predicted_y
127
           else: ## non-linear case
128
               n = X.shape[0]
129
130
               predicted_y = np.zeros(n)
               #=======#
131
132
               # STRART YOUR CODE HERE #
133
               #=======#
134
               kernel result = self.kernel(X,self.sv)
135
               for i in range(n):
136
                   for j in range(self.sv.shape[0]):
137
                      predicted_y[i] += self.a[j]*self.sv_y[j]*kernel_result[i,j]
138
               #=======#
139
                   END YOUR CODE HERE
140
               #========#
141
               return predicted_y
142
143
        144
        # Please DON'T change any code below this line!
        145
146
        def fit(self, X, y):
147
           n_samples, n_features = X.shape
148
           # Kernel matrix
149
           K = self.kernel(X)
150
           # dealing with dual form quadratic optimization
151
152
           P = cvxopt.matrix(np.outer(y,y) * K)
153
           q = cvxopt.matrix(np.ones(n_samples) * -1)
154
           A = cvxopt.matrix(y, (1,n_samples),'d')
155
           b = cvxopt.matrix(0.0)
156
157
           if self.C is None:
158
               G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
               h = cvxopt.matrix(np.zeros(n_samples))
159
           else:
160
               tmp1 = np.diag(np.ones(n samples) * -1)
161
162
               tmp2 = np.identity(n_samples)
               G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
163
               tmp1 = np.zeros(n_samples)
164
165
               tmp2 = np.ones(n_samples) * self.C
166
               h = cvxopt.matrix(np.hstack((tmp1, tmp2)))
167
           # solve QP problem
168
           solution = cvxopt.solvers.qp(P, q, G, h, A, b)
169
170
           # Lagrange multipliers
171
           a = np.ravel(solution['x'])
172
173
           # Support vectors have non zero lagrange multipliers
174
           sv = a > 1e-5
175
           ind = np.arange(len(a))[sv]
176
           self.a = a[sv]
177
           self.sv = X[sv]
178
           self.sv_y = y[sv]
```

localhost:4649/?mode=python 3/4

10/30/2020 svm.py 179 print("%d support vectors out of %d points" % (len(self.a), n_samples)) 180 181 182 # Intercept via average calculating b over support vectors 183 self.b = 0for n in range(len(self.a)): 184 185 self.b += self.sv_y[n] 186 self.b -= np.sum(self.a * self.sv_y * K[ind[n],sv]) 187 self.b /= len(self.a) 188 189 # Weight vector if self.kernel_name == 'linear_kernel': 190 self.w = np.zeros(n_features) 191 192 for n in range(len(self.a)): self.w += self.a[n] * self.sv_y[n] * self.sv[n] 193 194 else: self.w = None 195 196 197 def test(self): 198 199 accuracy = self.classify(self.test_x, self.test_y) 200 return accuracy 201 def classify(self, X, y): 202 203 predicted y = np.sign(self.predict(X))

accuracy = compute_accuracy(predicted_y, y)

return accuracy

204

205