

```
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
8
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
17 def compute_accuracy(predicted_y, y):
18     acc = 100.0
19     acc = np.sum(predicted_y == y)/predicted_y.shape[0]
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):
26     Y = X if Y is None else Y
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):
53             kernel_matrix[i,j] = (np.dot(X[i], Y[j]) + 1) ** degree
54     #=====#
55     # END YOUR CODE HERE #
56     #=====#
57     return kernel_matrix
58
```

```

60 Y = X if Y is None else Y
61 m = X.shape[0]
62 n = Y.shape[0]
63 assert X.shape[1] == Y.shape[1]
64 kernel_matrix = np.zeros((m, n))
65 #=====#
66 # STRART YOUR CODE HERE #
67 #=====#
68 for i in range(m):
69     for j in range(n):
70         kernel_matrix[i,j] = gaussian_kernel_point(X[i], Y[j])
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
79 # function above,
80 # then implement and uncomment this one.
81 # def gaussian_kernel(X, Y=None, sigma=5.0):
82 #     return
83
84 class SVM(object):
85     def __init__(self):
86         self.train_x = pd.DataFrame()
87         self.train_y = pd.DataFrame()
88         self.test_x = pd.DataFrame()
89         self.test_y = pd.DataFrame()
90         self.kernel_name = None
91         self.kernel = None
92
93     def load_data(self, train_file, test_file):
94         self.train_x, self.train_y = getDataframe(train_file)
95         self.test_x, self.test_y = getDataframe(test_file)
96
97     def train(self, kernel_name='linear_kernel', C=None):
98         self.kernel_name = kernel_name
99         if(kernel_name == 'linear_kernel'):
100             self.kernel = linear_kernel
101         elif(kernel_name == 'polynomial_kernel'):
102             self.kernel = polynomial_kernel
103         elif(kernel_name == 'gaussian_kernel'):
104             self.kernel = gaussian_kernel
105         else:
106             raise ValueError("kernel not recognized")
107
108         self.C = C
109         if self.C is not None:
110             self.C = float(self.C)
111
112         self.fit(self.train_x, self.train_y)
113
114     # predict labels for test dataset
115     def predict(self, X):
116         if self.w is not None: ## linear case
117             n = X.shape[0]

```

```

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

```

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