

python codes

```
1
2 from IPython import display
3 from IPython.core.interactiveshell import
  InteractiveShell
4 InteractiveShell.ast_node_interactivity = "all"
5 import warnings
6 warnings.filterwarnings("ignore")
7 import numpy as np
8 import matplotlib.pyplot as plt
9
10
11 # ## *(b) i**
12
13
14 # define the matrix A and vector b
15 def get_A_b(z, U):
16     '''
17     params:
18         z: the vector of the labels (N, 1)
19         U: the extended features (N, d)
20     return:
21         A: shape (N + 1, 1)
22         b: shape (N + 1, 1)
23     '''
24     zTz = z @ z.T
25     UTU = U @ U.T
26     A = np.concatenate((np.concatenate((zTz * UTU, -
  z), axis=1),
```

```

27         np.concatenate((z.T,
np.zeros((1, 1))), axis=1)),
28         axis=0)
29     b = np.concatenate((np.ones((z.shape[0], 1)),
np.zeros((1, 1))), axis=0)
30     return A, b
31
32
33 def get_lamda_mu(A, b):
34     '''
35     params:
36         A: matrix shape (N + 1, N + 1)
37         b: vector shape (N + 1, 1)
38     '''
39     sol = (np.linalg.inv(A) @ b).ravel()
40     lamda, mu = sol[:A.shape[0]-1], sol[-1]
41     ## check lamda is greater than zero
42     while np.any(lamda < 0):
43         idx1 = [i for i in range(lamda.shape[0]) if
lamda[i] >= 0]
44         idx = idx1 + [sol.shape[0] - 1]
45         A = A[idx, :][:, idx]
46         b = b[idx, :]
47         sol = (np.linalg.inv(A) @ b).ravel()
48         tmp, mu = sol[:A.shape[0]-1], sol[-1]
49         lamda[lamda<0] = 0
50         lamda[idx1] = tmp
51
52     print("lamda is ", lamda)
53     print("mu is ", mu)
54
55     return lamda, mu
56
57

```

```

58 # ## **(b) ii**
59
60
61 def check_KKT12(lamda, z):
62     '''
63     params:
64         lamda: lagaurange multiplier. vector shape
65         (1, N)
66         z: the vector of the labels (N, 1)
67     '''
68     print("KKT conditions 1 is satisfied: ",
69 np.allclose(lamda@z, 0))
70
71     print("KKT conditions 2 is satisfied: ",
72 np.all(lamda>=0) or np.allclose(lamda, 0))
73
74
75 # ## **(b) iii**
76
77
78
79 def get_w_w0(lamda, z, U):
80     '''
81     params:
82         lamda: lagaurange multiplier. vector shape
83         (1, N)
84         z: the vector of the labels (N, 1)
85         U: the extended features (N, d)
86     returns:
87         w: the weights, vector shape (1, N)
88         w0: the bias, scalar
89     '''
90     w = lamda.reshape((1, -1)) @ (z * U)
91     w0 = 1 / z[0, 0] - w @ U[0:1].T
92     print("w is {}, w0 is {}".format(w, w0))
93
94

```

```

88         return w, w0
89
90
91     # ## **(b) iv**
92
93
94     def check_KKT3(z, U, w, w0):
95         '''
96         params:
97             z: the vector of the labels (N, 1)
98             U: the extended features (N, d)
99             w: the weights, vector shape (1, N)
100             w0: the bias, scalar
101         '''
102         z_hat = U @ w.T + w0
103         cond3 = z * z_hat - 1
104         if np.all(cond3 >= 0) or np.allclose(cond3, 0):
105             res = True
106         else:
107             res = False
108         print("KKT conditions 3 is satisfied: ", res)
109
110
111     # ## **(c)**
112
113
114     z = np.array([[1], [1], [-1]])
115     U = np.array([[1, 2], [2, 1], [0, 0]])
116     A, b = get_A_b(z, U)
117     lamda, mu = get_lamda_mu(A, b)
118     check_KKT12(lamda, z)
119     w, w0 = get_w_w0(lamda, z, U)
120     check_KKT3(z, U, w, w0)
121

```

```

122
123 # ## **(d)**
124
125
126 def plot_decision_boundary(training, label_train, w,
w0):
127     # Total number of classes
128     classes = np.unique(label_train)
129     nclass = len(classes)
130
131     class_names = []
132     for c in classes:
133         class_names.append('Class ' + str(int(c)))
134
135     # Set the feature range for plotting
136     max_x1 = np.ceil(np.max(training[:, 0])) + 1.0
137     min_x1 = np.floor(np.min(training[:, 0])) - 1.0
138     max_x2 = np.ceil(np.max(training[:, 1])) + 1.0
139     min_x2 = np.floor(np.min(training[:, 1])) - 1.0
140
141     xrange = (min_x1, max_x1)
142     yrange = (min_x2, max_x2)
143
144     # step size for how finely you want to visualize
the decision boundary.
145     inc = 0.005
146
147     # generate grid coordinates. This will be the
basis of the decision boundary visualization.
148     (x1, x2) = np.meshgrid(np.arange(xrange[0],
xrange[1] + inc / 100, inc),
149                             np.arange(yrange[0],
yrange[1] + inc / 100, inc))
150

```

```

151     # size of the (x1, x2) image, which will also be
the size of the
152     # decision boundary image that is used as the
plot background.
153     image_size = x1.shape
154     # make (x1, x2) pairs as a bunch of row vectors.
155     grid_2d = np.hstack((x1.reshape(x1.shape[0] *
x1.shape[1], 1, order='F'),
156                           x2.reshape(x2.shape[0] *
x2.shape[1], 1, order='F'))))
157
158     pred_label = np.where(grid_2d @ w.T + w0 > 0, 1,
-1)
159     # reshape the idx (which contains the class
label) into an image.
160     decision_map = pred_label.reshape(image_size,
order='F')
161
162     # create fig
163     fig, ax = plt.subplots()
164     ax.imshow(decision_map, vmin=np.min(classes),
vmax=9, cmap='Pastel1',
165               extent=[xrange[0], xrange[1],
yrange[0], yrange[1]],
166               origin='lower')
167
168     # plot the class training data.
169     data_point_styles = ['rx', 'bo', 'g*']
170     for i in range(nclass):
171         ax.plot(training[label_train == classes[i],
0],
172               training[label_train == classes[i],
1],

```

```

173         data_point_styles[int(classes[i]) -
174         1],
175         label=class_names[i])
176     ax.legend()
177     plt.tight_layout()
178     plt.show()
179
180     return fig
181
182
183 fig = plot_decision_boundary(U, z.ravel(), w, w0)
184 fig.savefig("./figs/2d.png", dpi=200)
185
186
187 # ## *(f)**
188
189
190 z = np.array([[1], [1], [-1]])
191 U = np.array([[1, 2], [2, 1], [1, 1]])
192 A, b = get_A_b(z, U)
193
194 # (i)
195 lamda, mu = get_lamda_mu(A, b)
196
197 # (ii)
198 check_KKT12(lamda, z)
199
200 # (iii)
201 w, w0 = get_w_w0(lamda, z, U)
202
203 # (iv)
204 check_KKT3(z, U, w, w0)
205

```

```
206
207 fig = plot_decision_boundary(U, z.ravel(), w, w0)
208 fig.savefig("./figs/2f4.png", dpi=200)
209
210
211 # ##  $g$ 
212
213
214 z = np.array([[1], [1], [-1]])
215 U = np.array([[1, 2], [2, 1], [0, 1.5]])
216 A, b = get_A_b(z, U)
217
218 # (i)
219 lamda, mu = get_lamda_mu(A, b)
220
221 # (ii)
222 check_KKT12(lamda, z)
223
224 # (iii)
225 w, w0 = get_w_w0(lamda, z, U)
226
227 # (iv)
228 check_KKT3(z, U, w, w0)
229
230
231 fig = plot_decision_boundary(U, z.ravel(), w, w0)
232 fig.savefig("./figs/2g4.png", dpi=200)
233
234
235
```