

# Python codes

```
1
2 import warnings
3 warnings.filterwarnings("ignore")
4
5
6 import numpy as np
7 import pandas as pd
8 import matplotlib.pyplot as plt
9 from tqdm import tqdm
10
11
12 # ## **(d) Code a 2-class kernel nearest means classifier**
13
14
15 class KernelNearestMeansClassifier:
16     def __init__(self, gamma=0.01, kernel_type='rbf'):
17         self.gamma = gamma
18         self.kernel_type = kernel_type
19
20     def fit(self, X, y):
21         self.X_train = X
22         self.y_train = y
23
24
25     def predict(self, X):
26         X1 = self.X_train[self.y_train==0]
27         X2 = self.X_train[self.y_train==1]
28         if self.kernel_type == 'rbf':
29             dist1 = np.zeros((X1.shape[0], X1.shape[0]))
30             for i in range(X1.shape[0]):
31                 dist1[i, :] = np.sum((X1[i] - X1)**2, axis=1)
32                 self.K1 = np.mean(np.exp(-self.gamma * dist1))
33
34             dist2 = np.zeros((X2.shape[0], X2.shape[0]))
35             for i in range(X2.shape[0]):
36                 dist2[i, :] = np.sum((X2[i] - X2)**2, axis=1)
37                 self.K2 = np.mean(np.exp(-self.gamma * dist2))
38
39             res = np.zeros((X.shape[0], 2))
40             for i in tqdm(range(X.shape[0])):
41                 res[i, 0] = -self.K1 + 2 * np.mean(np.exp(-self.gamma *
42 np.sum((X[i] - X1)**2, axis=1)))
43                 res[i, 1] = -self.K2 + 2 * np.mean(np.exp(-self.gamma *
44 np.sum((X[i] - X2)**2, axis=1)))
45             elif self.kernel_type == "linear":
46                 self.K1 = np.mean(X1 @ X1.T)
47                 self.K2 = np.mean(X2 @ X2.T)
48
49                 res = np.zeros((X.shape[0], 2))
50                 for i in range(X.shape[0]):
51                     res[i, 0] = -self.K1 + 2 * np.mean(X[i] @ X1.T)
52                     res[i, 1] = -self.K2 + 2 * np.mean(X[i] @ X2.T)
```

```

51
52         else:
53             raise NotImplementedError
54         y_pred = np.argmax(res, axis=1)
55         return y_pred
56
57     def score(self, x, y):
58         y_pred = self.predict(x)
59         acc = np.mean(y_pred==y)
60         return acc
61
62
63 train_df1 = pd.read_csv("./Pr1_dataset1/train.csv", header=None)
64 val_df1 = pd.read_csv("./Pr1_dataset1/val.csv", header=None)
65 test_df1 = pd.read_csv("./Pr1_dataset1/test.csv", header=None)
66 X_train1, y_train1 = train_df1.iloc[:, :-1].values, train_df1.iloc[:,
67 -1].values
68 X_val1, y_val1 = val_df1.iloc[:, :-1].values, val_df1.iloc[:, -1].values
69 X_test1, y_test1 = test_df1.iloc[:, :-1].values, test_df1.iloc[:,
70 -1].values
71 y_train1[y_train1==2] = 0
72 y_val1[y_val1==2] = 0
73 y_test1[y_test1==2] = 0
74
75 train_df2 = pd.read_csv("./Pr1_dataset2/train_2.csv", header=None)
76 val_df2 = pd.read_csv("./Pr1_dataset2/val_2.csv", header=None)
77 test_df2 = pd.read_csv("./Pr1_dataset2/test_2.csv", header=None)
78 X_train2, y_train2 = train_df2.iloc[:, :-1].values, train_df2.iloc[:,
79 -1].values
80 X_val2, y_val2 = val_df2.iloc[:, :-1].values, val_df2.iloc[:, -1].values
81 X_test2, y_test2 = test_df2.iloc[:, :-1].values, test_df2.iloc[:,
82 -1].values
83
84
85
86
87 # ## **(e) plot the errors**
88
89
90 ticks = np.linspace(-2, 2, 5)
91 gammas = np.array([1 / (10**(-i)) for i in ticks if i<0] + [10**i for i in
92 ticks if i>=0])
93 rbf_val_errors1 = []
94 for gamma in gammas:
95     clf = KernelNearestMeansClassifier(gamma)
96     clf.fit(X_train1, y_train1)
97     rbf_val_errors1.append(1 - clf.score(X_val1, y_val1))
98 plt.figure(figsize=(20,16))
99 _ = plt.xticks(ticks, list(map(lambda x:round(x,2), gammas)), rotation=45)
100 _ = plt.plot(ticks, rbf_val_errors1)
101 _ = plt.xlabel("gamma (log scale)", fontsize=18)
102 _ = plt.ylabel("Validation error", fontsize=18)
103 plt.title("validation dataset 1", fontsize=20)
104 plt.savefig("./figs/1d1.png", dpi=200)
105 plt.show()

```

```

104 ticks = np.linspace(-2, 2, 5)
105 gammas = np.array([1 / (10**(-i)) for i in ticks if i<0] + [10**i for i in
106 ticks if i>=0])
107 rbf_val_errors2 = []
108 for gamma in gammas:
109     clf = KernelNearestMeansClassifier(gamma)
110     clf.fit(X_train2, y_train2)
111     rbf_val_errors2.append(1 - clf.score(X_val2, y_val2))
112 plt.figure(figsize=(20,16))
113 _ = plt.xticks(ticks, list(map(lambda x:round(x,1), gammas)), rotation=45)
114 _ = plt.plot(ticks, rbf_val_errors2)
115 _ = plt.xlabel("gamma (log scale)", fontsize=18)
116 _ = plt.ylabel("Validation error", fontsize=18)
117 plt.title("Validation dataset 2", fontsize=20)
118 plt.savefig("./figs/1d2.png", dpi=200)
119 plt.show()
120
121 # ## *(f) compare the test error of two kernel type*
122
123
124 clf = KernelNearestMeansClassifier(kernel_type='linear')
125 clf.fit(X_train1, y_train1)
126 print("The test error of linear kernel on the dataset1 is {:.3f}".format(1
127 - clf.score(X_test1, y_test1)))
128
129 clf = KernelNearestMeansClassifier(kernel_type='linear')
130 clf.fit(X_train2, y_train2)
131 print("The test error of linear kernel on the dataset2 is {:.3f}".format(1
132 - clf.score(X_test2, y_test2)))
133
134 clf = KernelNearestMeansClassifier(gamma=0.1, kernel_type='rbf')
135 clf.fit(X_train1, y_train1)
136 print("The test error of rbf kernel on the dataset1 is {:.3f}".format(1 -
137 clf.score(X_test1, y_test1)))
138
139 clf = KernelNearestMeansClassifier(gamma=100, kernel_type='rbf')
140 clf.fit(X_train2, y_train2)
141 print("The test error of rbf kernel on the dataset2 is {:.3f}".format(1 -
142 clf.score(X_test2, y_test2)))
143
144 # ## *(g, h) plot the training data, decision region and boundary*
145
146
147 def plot_decision_boundary(training, label_train, clf):
148     # Total number of classes
149     classes = np.unique(label_train)
150     nclass = len(classes)
151
152     class_names = []
153     for c in classes:
154         class_names.append('class ' + str(int(c)))
155
156     # Set the feature range for plotting
157     max_x1 = np.ceil(np.max(training[:, 0])) + 1.0
158     min_x1 = np.floor(np.min(training[:, 0])) - 1.0
159     max_x2 = np.ceil(np.max(training[:, 1])) + 1.0

```

```

157     min_x2 = np.floor(np.min(training[:, 1])) - 1.0
158
159     xrange = (min_x1, max_x1)
160     yrange = (min_x2, max_x2)
161
162     # step size for how finely you want to visualize the decision boundary.
163     inc = 0.01
164
165     # generate grid coordinates. This will be the basis of the decision
166     # boundary visualization.
167     (x1, x2) = np.meshgrid(np.arange(xrange[0], xrange[1] + inc / 100,
168                                     inc),
169                             np.arange(yrange[0], yrange[1] + inc / 100,
170                                     inc))
171
172     # size of the (x1, x2) image, which will also be the size of the
173     # decision boundary image that is used as the plot background.
174     image_size = x1.shape
175     # make (x1, x2) pairs as a bunch of row vectors.
176     grid_2d = np.hstack((x1.reshape(x1.shape[0] * x1.shape[1], 1,
177                                     order='F'),
178                           x2.reshape(x2.shape[0] * x2.shape[1], 1,
179                                     order='F'))))
180
181     pred_label = clf.predict(grid_2d)
182     # reshape the idx (which contains the class label) into an image.
183     decision_map = pred_label.reshape(image_size, order='F')
184
185     # create fig
186     fig, ax = plt.subplots()
187     ax.imshow(decision_map, vmin=np.min(classes), vmax=9, cmap='Pastel2',
188               extent=[xrange[0], xrange[1], yrange[0], yrange[1]],
189               origin='lower')
190
191     # plot the class training data.
192     data_point_styles = ['rx', 'bo', 'g*']
193     for i in range(nclass):
194         ax.plot(training[label_train == classes[i], 0],
195                 training[label_train == classes[i], 1],
196                 data_point_styles[int(classes[i]) - 1],
197                 label=class_names[i])
198     ax.legend()
199
200     plt.tight_layout()
201     plt.show()
202
203     return fig
204
205
206
207 clf = KernelNearestMeansClassifier(kernel_type='linear')
208 clf.fit(X_train1, y_train1)
209 plot_decision_boundary(X_train1, y_train1, clf)
210
211
212
213 clf = KernelNearestMeansClassifier(kernel_type='linear')
214 clf.fit(X_train2, y_train2)
215 plot_decision_boundary(X_train2, y_train2, clf)

```

```
210
211
212 clf = KernelNearestMeansClassifier(gamma=0.1, kernel_type='rbf')
213 clf.fit(X_train1, y_train1)
214 plot_decision_boundary(X_train1, y_train1, clf)
215
216
217 clf = KernelNearestMeansClassifier(gamma=100, kernel_type='rbf')
218 clf.fit(X_train2, y_train2)
219 plot_decision_boundary(X_train2, y_train2, clf)
220
221
222 # ## *(i) Repeat (h) using different  $\gamma$  **
223
224
225 for gamma in [0.01, 0.1, 0.3, 3, 10, 100]:
226     clf = KernelNearestMeansClassifier(gamma=gamma*0.1, kernel_type='rbf')
227     clf.fit(X_train1, y_train1)
228     fig = plot_decision_boundary(X_train1, y_train1, clf)
229     fig.savefig('./figs/1i1_r_{}.png'.format(gamma), dpi=200)
230
231
232 for gamma in [0.01, 0.1, 0.3, 3, 10, 100]:
233     clf = KernelNearestMeansClassifier(gamma=gamma*100, kernel_type='rbf')
234     clf.fit(X_train2, y_train2)
235     fig = plot_decision_boundary(X_train2, y_train2, clf)
236     fig.savefig('./figs/1i2_r_{}.png'.format(gamma), dpi=200)
237
238
239
```