

```

1 import numpy as np
2 import pandas as pd
3 from statistics import mode
4 pd.set_option("display.max_rows", None, "display.max_columns",
5               None)
6
7 # load the dataset(sl = sepal length, sw = sepal width, pl =
8   petal length, pw = petal width, class = Target)
9 iris = pd.read_csv('C:\\Users\\Authorized User1\\Desktop\\Data
10   mining\\assignment 2\\iris.txt',
11                   names=["sl", "sw", "pl", "pw", "class"])
12
13 # 0 = "Iris-setosa", 1 = "Iris-versicolor", 2 = "Iris-virginica"
14 iris['class_n'] = np.select([(iris['class'] == "Iris-setosa"), (
15   iris['class'] == "Iris-versicolor"), (iris['class'] == "Iris-
16   virginica")], [0, 1, 2], default="none")
17
18 print(iris.info())
19 print (iris)
20
21 #1. Make a function of a KNN classifier to perform
22   classification based on the majority
23   voting of K nearest neighbors. Use the Minkowski distance
24   function you made in HW2 as the distance measure.
25
26 def minkowski(data_frame, array, r):
27     col_name = ['sl', 'sw', 'pl', 'pw']
28     b = {}
29     for (w, i, j, k, l) in zip(data_frame.index, data_frame[
30   col_name[0]], data_frame[col_name[1]], data_frame[col_name[2]
31   ], data_frame[col_name[3]]):
32         b[w] = round((((abs(i - array[0])) ** r) + ((abs(j -
33   array[1])) ** r) + ((abs(k - array[2])) ** r) + ((abs(l - array
34   [3])) ** r)) ** (1 / r)), 2)
35     a = dict(sorted(b.items(), key=lambda item: item[1], reverse=
36   False))
37     return (a)
38
39 def knn(frame, array, r, k):
40     d = {}
41     for i in array.index:
42         c = (minkowski(frame, (list(array.drop('class_n', axis =
43   1).loc[i])), r))
44         pairs = {k: c[k] for k in list(c)[:k]}
45         d[i] = (classification(list(pairs.keys())))
46     k = pd.DataFrame(list(d.items()), columns = ['index', '
47   predicted'])
48     k = k.set_index('index')
49     return (pd.concat([array, k], axis=1, join="inner"))
50

```

```

39 def classification(j):
40     h = []
41     for i in range(len(j)):
42         if j[i] in range(50):
43             h.append(0)
44         elif j[i] in range(50,100):
45             h.append(1)
46         else:
47             h.append(2)
48     return(mode(h))
49
50 def accuracy_and_confusion(host):
51     confusion_matrix = pd.crosstab(host['class_n'], host['
predicted'], rownames=['Actual'], colnames=['Predicted'])
52     #print ('the confusion matrix')
53     #print(confusion_matrix)
54     l = 0
55     for (i,j) in zip(host['class_n'],host['predicted']):
56         if i == j:
57             l += 1
58         else:
59             pass
60     print ('Accuracy score in %')
61     print((l/len(host))*100)
62
63
64 #2. For the IRIS dataset, prepare a training dataset and a
    testing dataset for classification model training and testing.
65 # For each class, take the first 40 samples into the training
    dataset, and the remaining 10 samples into the testing dataset.
66 # The resulting training dataset has 120 samples and testing
    dataset has 30 samples.
67 df = iris.drop('class',axis = 1)
68
69 x_train1 = (df.iloc[0:40,:])
70 x_train2 = (df.iloc[50:90,:])
71 x_train3 = (df.iloc[100:140,:])
72
73
74
75 x_test1 = (df.iloc[40:50,:])
76 x_test2 = (df.iloc[90:100,:])
77 x_test3 = (df.iloc[140:150,:])
78
79 x_train = pd.concat([x_train1,x_train2,x_train3])
80 x_test = pd.concat([x_test1,x_test2,x_test3])
81
82 print (len(x_train))
83 print (len(x_test))
84
85
86 #3.1    For each KNN parameter setting, report classification

```

```

86 accuracy and the confusion matrix.
87
88 print ('FOR FULL DATASET TRAINING_SET 120 AND TESTING_SET 30')
89
90 for e in [3,5,7]:
91     for t in [1, 2, 4]:
92         x_train = pd.concat([x_train1,x_train2,x_train3])
93         x_test = pd.concat([x_test1,x_test2,x_test3])
94         final = (knn(x_train,x_test,t,e))
95         final["class_n"] = pd.to_numeric(df["class_n"])
96         print ('for r = %d and k = %d'%(t,e))
97         accuracy_and_confusion(final)
98         print('=====')
99
100
101 #3.2 Calculate and report the classification accuracy for
each class at each parameter setting.
102
103 print ('FOR IRIS-SETOSA DATASET TRAINING_SET 40 AND
TESTING_SET 10')
104
105 for e in [3,5,7]:
106     for t in [1, 2, 4]:
107         x_train = (iris.iloc[0:40, :])
108         x_test = (iris.iloc[40:50, :])
109         final = (knn(x_train,x_test,t,e))
110         final["class_n"] = pd.to_numeric(df["class_n"])
111         print ('for r = %d and k = %d'%(t,e))
112         accuracy_and_confusion(final)
113         print('=====')
114
115 print ('FOR IRIS-VERSICOLOR DATASET TRAINING_SET 40 AND
TESTING_SET 10')
116
117 for e in [3,5,7]:
118     for t in [1, 2, 4]:
119         x_train = (iris.iloc[50:90, :])
120         x_test = (iris.iloc[90:100, :])
121         final = (knn(x_train,x_test,t,e))
122         final["class_n"] = pd.to_numeric(df["class_n"])
123         print ('for r = %d and k = %d'%(t,e))
124         accuracy_and_confusion(final)
125         print('=====')
126
127 print ('FOR IRIS-VERGINICA DATASET TRAINING_SET 40 AND
TESTING_SET 10')
128
129 for e in [3,5,7]:
130     for t in [1, 2, 4]:
131         x_train = (iris.iloc[100:140, :])
132         x_test = (iris.iloc[140:150, :])
133         final = (knn(x_train,x_test,t,e))

```

```

134         final["class_n"] = pd.to_numeric(df["class_n"])
135         print ('for r = %d and k = %d'%(t,e))
136         accuracy_and_confusion(final)
137         print('=====')
138
139 #3.3 Assume we use the average accuracy of each class as
the overall model performance measure,
140 # find the best parameter setting that generated the highest
average accuracy for 3 classes.
141
142 def ggg():
143     frame = pd.DataFrame(columns=['k', 'p', 'acc_score'])
144     for q in [0,50,100]:
145         x_train = iris.drop('class',axis=1).iloc[q:q+40,:]
146         x_test = iris.drop('class', axis=1).iloc[q+40:q + 50
147         , :]
148         for e in [3, 5, 7]:
149             for t in [1, 2, 4]:
150                 final = (knn(x_train, x_test, t, e))
151                 final["class_n"] = pd.to_numeric(df["class_n"
152                 ])
153                 frame = frame.append({ "k": e, 'p': t, '
154                 acc_score': (accuracy_and_confusion(final))},ignore_index=True
155                 )
156     return(frame)
157
158 ggg()
159
160 #4. As shown in the plot below, a simple decision tree is
constructed to classify two iris flowers: Versicolor and
161 Virginica using two features of petal width and petal length
162 . Assume the binary decision boundary on Petal Length is 4.8,
163 # and the decision boundary on Petal Width is 1.7. Make a
function to implement this simple decision tree and use your
function to
164 # classify the 100 iris samples of Versicolor and Virginica.
Report the classification accuracy, sensitivity, and
specificity.
165 # Here we define sensitivity = accuracy for class Versicolor,
and specificity = accuracy of class Virginica.
166
167 df_c = iris.drop('class_n',axis = 1).iloc[100:150,:]
168
169 df_c['predicted'] = np.select([(df_c["pl"] <= 4.8) & (df_c["pw"
170 " ] <= 1.7),
171                               (df_c["pl"] > 4.8) & (df_c["pw"
172                               ] > 1.7),
173                               (df_c["pl"] >= 4.8) & (df_c["pw"
174                               ] <= 1.7),
175                               (df_c["pl"] <= 4.8) & (df_c["pw"
176                               ] >= 1.7)], ["Iris-versicolor", "Iris-virginica", "Iris-

```

```
168 virginica", "Iris-virginica"], default='')
169 print (df_c)
170 confusion_matrix = pd.crosstab(df_c['class'], df_c['predicted']
    ], rownames=['Actual'], colnames=['Predicted'])
171 print ('the confusion matrix')
172 print(confusion_matrix)
173 l = 0
174 for (i,j) in zip(df_c['class'],df_c['predicted']):
175     if i == j:
176         l += 1
177     else:
178         pass
179 print ('Accuracy score in %')
180 print((l/len(df_c))*100)
181
```