Problem 4. K-nearest Neighbor Classification

Part II.

Preparing the dataset

Importing Libraries

In [190…

```
import pandas as pd
```

Importing the training dataset

In [191…

```
training = pd.read_csv('/Users/dakshbhuva/Desktop/CS-559 ML/HW_3/train.csv')
training
```

Out[191…

	x	y	z	class
0	8.599291	9.729418	6.432371	1
1	6.592955	0.082556	1.969544	1
2	5.596471	9.815682	0.027295	1
3	2.743639	8.783177	4.041946	0
4	4.458362	5.750222	0.099070	0
...	...	...	...	...
995	4.617314	7.700236	5.907128	0
996	5.453472	1.798360	1.992616	0
997	2.553853	8.122934	3.970146	0
998	3.210456	3.342092	7.831479	0
999	6.930237	2.742352	4.678527	1

1000 rows × 4 columns

In [192…

```
tr_values = training.values
train_X, train_y = tr_values[:, :-1], tr_values[:, -1]
```

Importing the testing dataset

In [193…

```
testing = pd.read_csv('/Users/dakshbhuva/Desktop/CS-559 ML/HW_3/test.csv')
testing
```

Out[193…

	ID	x	y	z	actual-class
0	1	8.074807	5.988044	3.844979	1
1	2	4.952249	5.823205	1.612045	0
2	3	4.773178	0.078757	4.209442	0
3	4	9.845919	2.055448	3.525702	1
4	5	1.612492	1.320515	8.200455	0
5	6	7.987555	9.188111	7.222228	1
6	7	0.311558	3.974680	7.897371	0
7	8	1.219113	0.266045	2.741136	0
8	9	0.636340	1.831257	6.767459	0
9	10	0.890168	8.613714	2.884227	0
10	11	7.226514	9.852794	7.373560	1
11	12	2.709551	3.719191	5.743540	0
12	13	2.842368	1.902145	2.216614	0
13	14	3.610773	4.589548	7.714008	0
14	15	4.888200	6.720637	7.261562	0
15	16	8.857224	9.056900	8.862604	1
16	17	8.239402	9.347802	5.277351	1
17	18	3.219759	2.980960	6.646886	0
18	19	2.146974	5.328725	5.801703	0
19	20	1.156302	8.542813	1.859447	0

Extracting the required columns from the testing dataset

In [194…

```
cols_to_use = ['x','y','z','actual-class',]
testing = testing[cols_to_use]
testing
```

Out[194...

	x	y	z	actual-class
0	8.074807	5.988044	3.844979	1
1	4.952249	5.823205	1.612045	0
2	4.773178	0.078757	4.209442	0
3	9.845919	2.055448	3.525702	1
4	1.612492	1.320515	8.200455	0
5	7.987555	9.188111	7.222228	1
6	0.311558	3.974680	7.897371	0
7	1.219113	0.266045	2.741136	0
8	0.636340	1.831257	6.767459	0
9	0.890168	8.613714	2.884227	0
10	7.226514	9.852794	7.373560	1
11	2.709551	3.719191	5.743540	0
12	2.842368	1.902145	2.216614	0
13	3.610773	4.589548	7.714008	0
14	4.888200	6.720637	7.261562	0
15	8.857224	9.056900	8.862604	1
16	8.239402	9.347802	5.277351	1
17	3.219759	2.980960	6.646886	0
18	2.146974	5.328725	5.801703	0
19	1.156302	8.542813	1.859447	0

In [195...

```
te_values = testing.values
test_X, test_y = te_values[:, :-1], te_values[:, -1]
```

K-Nearest Neighbors Algorithm

Problem 4_Part II

(1)

k = 3

In [196…
```
# Importing KNN from sklearn packages
from sklearn.neighbors import KNeighborsClassifier

# Implenting KNN
model = KNeighborsClassifier(n_neighbors=3)
```

In [197…
```
# Training the KNN model
model.fit(train_X, train_y)
```

Out[197…
```
KNeighborsClassifier(n_neighbors=3)
```

In [198…
```
#Predicting the Output
predicted = model.predict(test_X)
print(predicted)
```

```
[1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
```

Probability Estimates for the final decision

In [199…
```
P_E = model.predict_proba(test_X)
print("Probability Estimates:\n", P_E)
```

```
Probability Estimates:
 [[0.         1.        ]
 [0.66666667 0.33333333]
 [1.         0.        ]
 [0.         1.        ]
 [1.         0.        ]
 [0.         1.        ]
 [1.         0.        ]
 [1.         0.        ]
 [1.         0.        ]
 [1.         0.        ]
 [0.         1.        ]
 [1.         0.        ]
 [1.         0.        ]
 [1.         0.        ]
 [0.33333333 0.66666667]
 [0.         1.        ]
 [0.         1.        ]
 [1.         0.        ]
 [1.         0.        ]
 [1.         0.        ]]
```

Problem 4_Part II

(2)

Euclidean distance weighted 3-nearest neighbors

In [200…

```
# Importing KNN from sklearn packages
from sklearn.neighbors import KNeighborsClassifier

# Implenting Euclidean distance weighted KNN
weighted_knn = KNeighborsClassifier(n_neighbors=3, weights='distance', p=2, m
```

In [201…

```
# Training the KNN model
weighted_knn.fit(train_X, train_y)
```

Out[201…

```
KNeighborsClassifier(metric='euclidean', n_neighbors=3, weights='distance')
```

In [202…

```
#Predicting the Output
predict = weighted_knn.predict(test_X)
print(predict)
```

```
[1. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 1. 1. 0. 0. 0.]
```

=>Hence, the predicted label for each point remains the same as that in question (1)

Problem 4_Part II

(3)

Confusion Matrix for (1)

In [203…

```
from sklearn.metrics import confusion_matrix

print("Confusion Matrix:\n",confusion_matrix(test_y, predicted))
```

```
Confusion Matrix:
 [[13  1]
 [ 0  6]]
```

Confusion Matrix for (2)

In [204…

```
from sklearn.metrics import confusion_matrix

print("Confusion Matrix:\n",confusion_matrix(test_y, predict))
```

```
Confusion Matrix:
 [[13  1]
 [ 0  6]]
```

Accuracy for (1)

In [205…

```
#Importing sklearn package for calculating the accuracy
from sklearn.metrics import accuracy_score

print("Accuracy:",accuracy_score(test_y, predicted))
```

```
Accuracy: 0.95
```

Accuracy for (2)

In [206…

```
#Importing sklearn package for calculating the accuracy
from sklearn.metrics import accuracy_score

print("Accuracy:",accuracy_score(test_y, predict))
```

```
Accuracy: 0.95
```

Precision for (1)

In [207…

```
from sklearn.metrics import precision_score

print("Precision:",precision_score(test_y, predicted))
```

```
Precision: 0.8571428571428571
```

Precision for (2)

In [208…

```
from sklearn.metrics import precision_score

print("Precision:",precision_score(test_y, predict))
```

```
Precision: 0.8571428571428571
```

F-Measure for (1)

In [209…

```python
from sklearn.metrics import f1_score

print("F-Measure:",f1_score(test_y, predicted))
```

```
F-Measure: 0.923076923076923
```

F-measure for (2)

In [210…

```python
from sklearn.metrics import f1_score

print("F-Measure:",f1_score(test_y, predict))
```

```
F-Measure: 0.923076923076923
```

=> From above, it is clear that the results of (1) and (2) are exactly the same and hence both the methods have the same performance