CS559- Homework Assignment 3

Problem 1.1

Given: 9 data instances with 2 attributes and classes

Goal: Show which attribute will be the first splitting for the decision tree

Solution:

0) Before splitting: the parent class(P)

Class	#of instances
+	4
-	5

Entropy(P) = $-\frac{4}{9}\log_2\frac{4}{9} - \frac{5}{9}\log_2\frac{5}{9}$ = 0.9911

1) Splitting the attribute 1(a_1)

a11:Class	#of instances
+	3
-	1

a12: Class	#of instances
+	1
-	4

2) Calculate the entropy for each leaf and the information gain

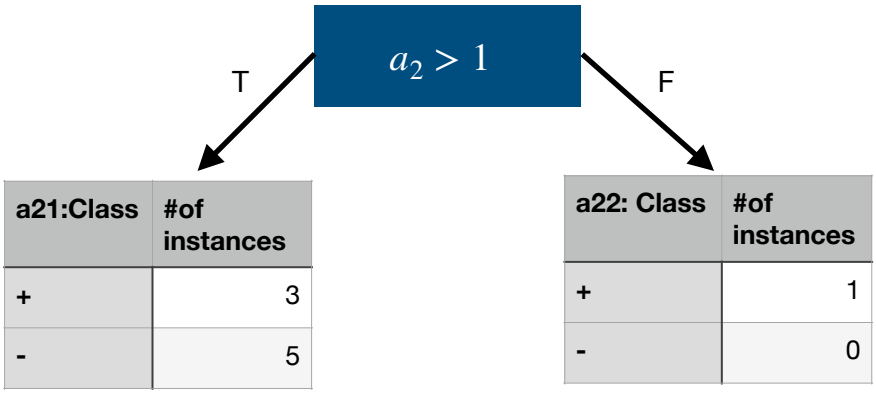
Entropy(a_{11}) = $-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4}$ = 0.8113

Entropy(a_{12}) = $-\frac{1}{5}\log_2\frac{1}{5} - \frac{4}{5}\log_2\frac{4}{5}$ = 0.7219

Entropy(a_1) = $\frac{4}{9}$Entropy(a_{11}) + $\frac{5}{9}$Entropy(a_{12}) = 0.7616

Information Gain = Entropy(P) - Entropy(a_1) = 0.9911 - 0.7616 = 0.2295

3) Splitting the attribute 2(a_2) and repeat 2) procedure



a21:Class	#of instances
+	3
-	5

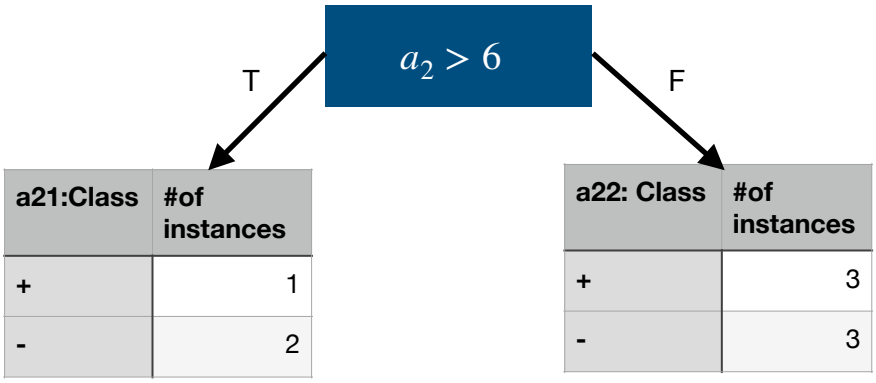
a22: Class	#of instances
+	1
-	0

Entropy(a_{21}) = $-\frac{3}{8}\log_2\frac{3}{8} - \frac{5}{8}\log_2\frac{5}{8}$ = 0.9544

Entropy(a_{22}) = $-\frac{1}{1}\log_2\frac{1}{1} - \frac{0}{1}\log_2\frac{0}{1}$ = 0

Entropy(a_2) = $\frac{8}{9}$Entropy(a_{21}) + $\frac{1}{9}$Entropy(a_{22}) = 0.8484

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.8484 = 0.1427



a21:Class	#of instances
+	1
-	2

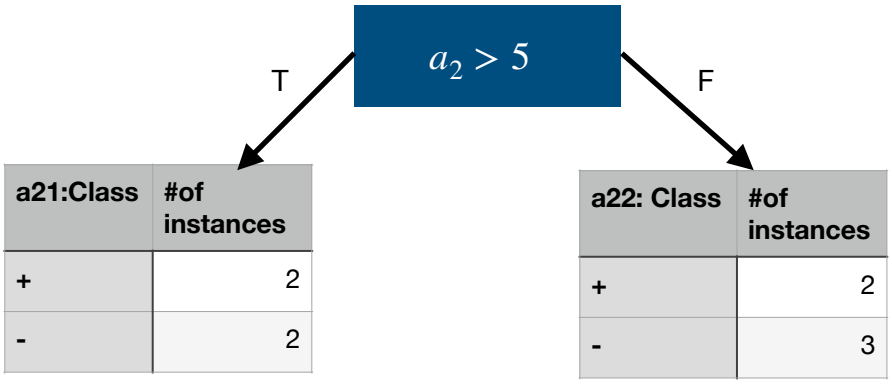
a22: Class	#of instances
+	3
-	3

Entropy(a_{21}) = $-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3}$ = 0.9183

Entropy(a_{22}) = $-\frac{3}{6}\log_2\frac{3}{6} - \frac{3}{6}\log_2\frac{3}{6}$ = 1

Entropy(a_2) = $\frac{3}{9}$Entropy(a_{21}) + $\frac{6}{9}$Entropy(a_{22}) = 0.9728

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.9728 = 0.0183



a21:Class	#of instances
+	2
-	2

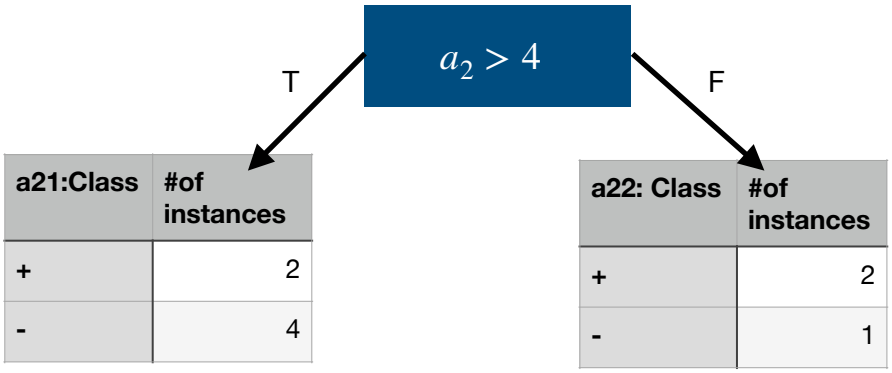
a22: Class	#of instances
+	2
-	3

Entropy(a_{21}) = $-\frac{2}{4}\log_2\frac{2}{4} - \frac{2}{4}\log_2\frac{2}{4}$ = 1

Entropy(a_{22}) = $-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}$ = 0.9710

Entropy(a_2) = $\frac{4}{9}$Entropy(a_{21}) + $\frac{5}{9}$Entropy(a_{22}) = 0.9839

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.9839 = 0.0072



a21:Class	#of instances
+	2
-	4

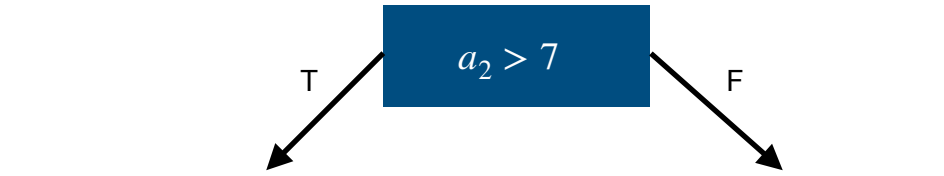
a22: Class	#of instances
+	2
-	1

Entropy(a_{21}) = $-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}$ = 0.9183

Entropy(a_{22}) = $-\frac{2}{3}\log_2\frac{2}{3} - \frac{1}{3}\log_2\frac{1}{3}$ = 0.9183

Entropy(a_2) = $\frac{6}{9}$Entropy(a_{21}) + $\frac{3}{9}$Entropy(a_{22}) = 0.9183

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.9183 = 0.0728



a21:Class	#of instances
+	0
-	1

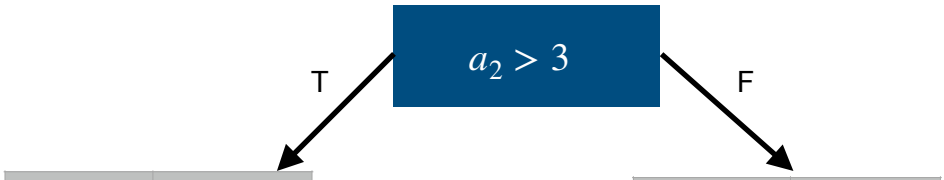
a22: Class	#of instances
+	4
-	4

Entropy(a_{21}) = $-\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1} = 0$

Entropy(a_{22}) = $-\frac{4}{8}\log_2\frac{4}{8} - \frac{4}{8}\log_2\frac{4}{8} = 1$

Entropy(a_2) = $\frac{1}{9}$Entropy(a_{21}) + $\frac{8}{9}$Entropy(a_{22}) = 0.8889

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.8889 = 0.1022



a21:Class	#of instances
+	3
-	4

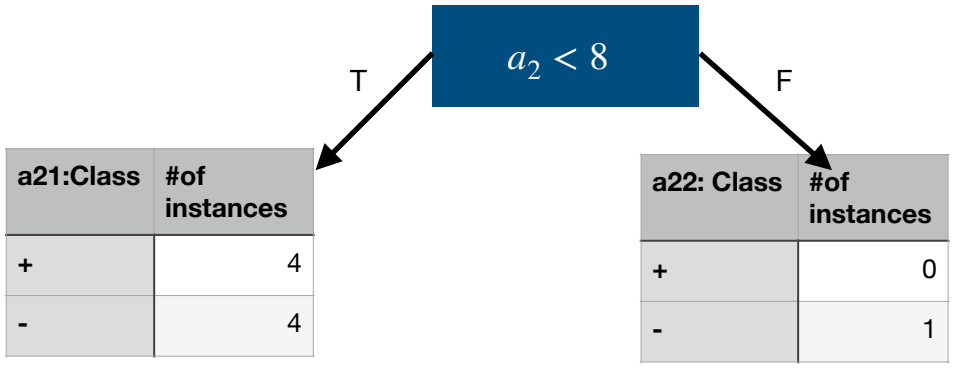
a22: Class	#of instances
+	1
-	1

Entropy(a_{21}) = $-\frac{3}{7}\log_2\frac{3}{7} - \frac{4}{7}\log_2\frac{4}{7} = 0.9852$

Entropy(a_{22}) = $-\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{2}\log_2\frac{1}{2} = 1$

Entropy(a_2) = $\frac{7}{9}$Entropy(a_{21}) + $\frac{2}{9}$Entropy(a_{22}) = 0.9885

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.9885 = 0.0026



a21:Class	#of instances
+	4
-	4

a22: Class	#of instances
+	0
-	1

$$\text{Entropy}(a_{21}) = -\frac{4}{8}\log_2\frac{4}{8} - \frac{4}{8}\log_2\frac{4}{8} = 1$$

$$\text{Entropy}(a_{22}) = -\frac{0}{1}\log_2\frac{0}{1} - \frac{1}{1}\log_2\frac{1}{1} = 0$$

$$\text{Entropy}(a_2) = \frac{8}{9}\text{Entropy}(a_{21}) + \frac{1}{9}\text{Entropy}(a_{22}) = 0.8889$$

Information Gain = Entropy(P) - Entropy(a_2) = 0.9911 - 0.8889 = 0.1022

As computing, the split for a_2 using the distinct value threshold, splitting $a_1 > 1$ has the highest information gain. However, by comparing to the attribute a_1, the information gain from splitting a_1 has the highest value. Therefore a_1 will be chosen as the first split for the decision tree.

Problem 1.2

Solution

If we use “Instance” as another attribute, we are ‘too’ overfitting the data as we split too many weak classifiers. Therefore, this attribute should not be used in the tree.

Problem 2.1

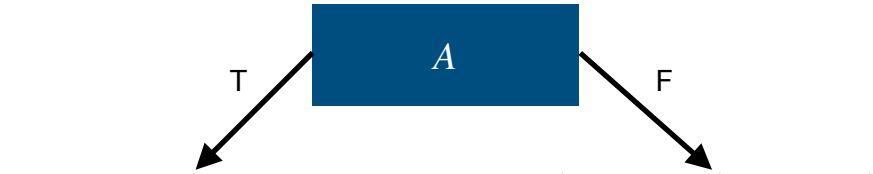
Given: the dataset with two attributes and two classes, the cost matrix
Goal: obtain the first attribute to split using the GINI index

Solution

0) Before splitting: the parent class(P)

Class	#of instances
+	35
-	65

$$\text{GINI}(P) = 1 - \left[\left(\frac{7}{20}\right)^2 + \left(\frac{13}{20}\right)^2\right] = 0.455$$



A1:Class	#of instances
+	20
-	30

A2: Class	#of instances
+	15
-	35

$$\text{GINI}(A_1) = 1 - \left[\left(\frac{2}{5}\right)^2 + \left(\frac{3}{5}\right)^2\right] = 0.48$$

$$\text{GINI}(A_2) = 1 - \left[\left(\frac{3}{10}\right)^2 + \left(\frac{7}{10}\right)^2\right] = 0.42$$

$$\text{GINI}(A) = \frac{1}{2}\text{GINI}(A_1) + \frac{1}{2}\text{GINI}(A_2) = 0.45$$

Information gain A = GINI(P) - GINI(A) = 0.455 - 0.45 = 0.005

B1:Class	#of instances
+	15
-	20

B2: Class	#of instances
+	20
-	45

$$\text{GINI}(B_1) = 1 - \left[\left(\frac{3}{7}\right)^2 + \left(\frac{4}{7}\right)^2\right] = 0.4898$$

$$\text{GINI}(B_2) = 1 - \left[\left(\frac{4}{13}\right)^2 + \left(\frac{9}{13}\right)^2\right] = 0.4260$$

$$\text{GINI}(B) = \frac{7}{20}\text{GINI}(B_1) + \frac{13}{20}\text{GINI}(B_2) = 0.4483$$

Information gain B = GINI(P) - GINI(B) = 0.455 - 0.4483 = 0.0067

According to the information gain, gain B > gain A; therefore, choose B as the first attribute to be split.

Problem 2.2

Solution

1) Find the confusion matrices for attributes A and B

Confusion matrix for A

Actual Class	Attribute A: T	Attribute A: F
+	20	15
-	30	35

Confusion matrix for B

Actual Class	Attribute B: T	Attribute B: F
+	15	20
-	20	45

2) Applying the given Cost matrix to the confusion matrices to compute the cost of misclassification.

For A:

Cost A = 20*(-1) + 15*(100) + 0 + 35*(-10) = 1130

For B:

Cost B = 15*(-1) + 20*(100) + 0 + 45*(-10) = 1535

According to the cost calculation, Cost A < Cost B; therefore, A is the first attribute to be split first.

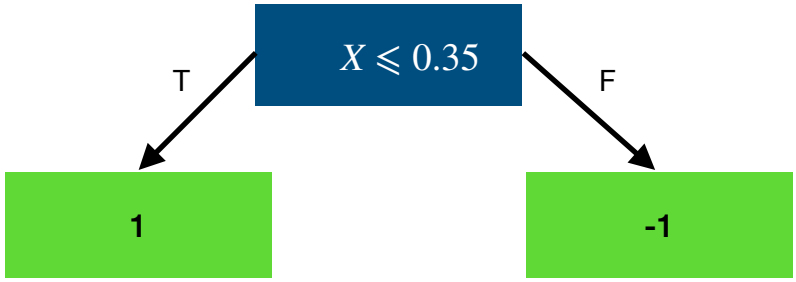
Problem 3

Given: 1 attribute and 2 label classes
Goal:

1) compute the weight of all the instances after each weak classifier's first round of the AdaBoost algorithm (no need to normalize the weight).

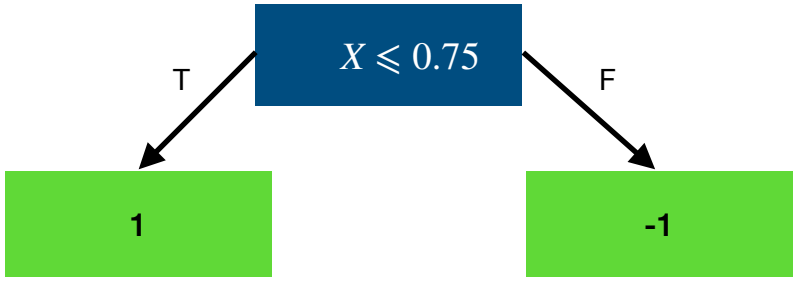
2) specify the data instances which will be reweighed after the first round.

1) Consider H1



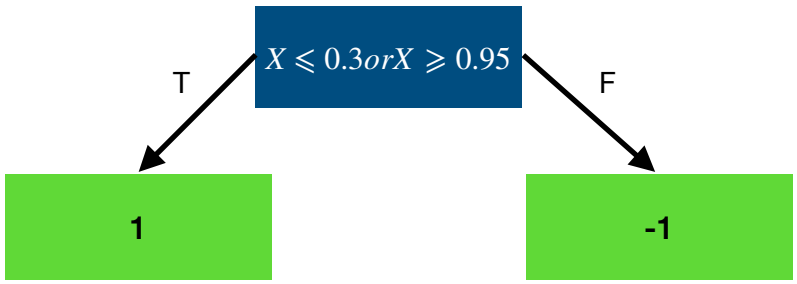
- initial weight; $D_1(i) = 0.1$
- According to the dataset, ID9 and ID10 are misclassified (2 misclassified), the total error is calculated by $\epsilon_1 = 0.1 * 2 = 0.2$
- $\alpha_1 = \frac{1}{2}\ln(\frac{1-0.2}{0.2}) = 0.6931$
- Update weight: $D_2(i) = 0.1e^{-0.6931} = 0.05$ for the correct prediction, $D_2(i) = 0.1e^{0.6931} = 0.2$ for the incorrect prediction
- Reweighed data: 'All of them'

2) Consider H2



- initial weight; $D_1(i)$ = 0.1
- According to the dataset, ID1, ID2, ID3, and ID8 are misclassified (4 misclassified), the total error is calculated by $\epsilon_1 = 0.1 * 4 = 0.4$
- $\alpha_1 = \frac{1}{2}\ln(\frac{1-0.4}{0.4}) = 0.2027$
- Update weight: $D_2(i) = 0.1e^{-0.2027} = 0.0817$ for the correct prediction, $D_2(i) = 0.1e^{0.2027} = 0.1225$ for the incorrect prediction
- Reweighed data: 'All of them'

3) Consider H3



- initial weight; $D_1(i)$ = 0.1
- According to the dataset, ID9 is misclassified (1 misclassified), the total error is calculated by $\epsilon_1 = 0.1 * 1 = 0.1$
- $\alpha_1 = \frac{1}{2}\ln(\frac{1-0.1}{0.1}) = 1.0986$
- Update weight: $D_2(i) = 0.1e^{-1.0986} = 0.0333$ for the correct prediction, $D_2(i) = 0.1e^{1.0986} = 0.3$ for the incorrect prediction
- Reweighed data: 'All of them'

Problem 4

Part I
Given: 2D dataset and label classes
Goal: classify the test point following the two strategies

Solution

1) 5-Nearest neighbor

From the plot, 5 neighboring data points are identified as the nearest to the test point(5,4). Those 5 points are shown in the following table.

X	Y	Class
6	5	+
7	3	+
4	4	-
5	1	-
4	1	-

Since the distances are weighted uniformly, the class for the test point(5,4) is -.

2) Manhattan distance weighted 3-nearest neighbors ($\frac{1}{d^2}$)

From the same plot, 3 neighboring data points are identified as the nearest to the test point(5,4). By this time the Manhattan distance is calculated, and the weight factor according to the distance is calculated for the majority vote. Those 3 points, along with the distance are shown in the following table. Note that point(7,3) has the same distance as point(5,1). By considering 3-neighbor, 2 cases are satisfied.

X	Y	Class	Distance(d)	Wi = 1/d^2
4	4	-	|5-4|+|4-4| = 1	1
6	5	+	|5-6|+|4-5| = 2	1/4
7	3	+	|5-7|+|4-3| = 3	1/9
5	1	-	|5-5|+|4-1| = 3	1/9

Case I: (4,4)|-, (6,5)|+, (7,3)|+

For - : w|- = 1
For + : w|+ = 1/4 + 1/9 = 0.361

w| - > w|+; therefore, the test point is classified as -.

Case II: (4,4)|-, (6,5)|+, (5,1)|-

For - : w|- = 1+1/9 = 1.11
For + : w|+ = 1/4 = 0.25

w| - > w|+; therefore, the test point is classified as -.

Part II
Solution

1) Import train.csv file

```
#read training data
train_set = pd.read_csv('train.csv', header=0)
train_set
```

	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

2) Separate features from labels and normalize features using standardization method from sklearn.

```
#separate features from labels
features = train_set.loc[:,['x','y','z']]
labels_train = train_set.loc[:,'class']

#normalize features
features_train = StandardScaler().fit_transform(features)
print('feature shape: ',features_train.shape)
features_train #display features
```

```
feature shape:  (1000, 3)

array([[ 1.2909373 ,  1.64214887,  0.49931501],
       [ 0.58765781, -1.74022951, -1.04276611],
       [ 0.23836075,  1.6723947 , -1.71388907],
       ...,
       [-0.82816586,  1.07888402, -0.35147993],
       [-0.59800699, -0.59737263,  0.98276143],
       [ 0.70588507, -0.80765324, -0.10670655]])
```

3) Train the 3-nearest neighbor model, specifying the argument n_neighbors = 3, and the distance metric is ‘euclidean’ by default.

```
#train 3-nearest neighbor classifier from the training dataset

threeNeighbors = KNeighborsClassifier(n_neighbors=3)
threeClassifier = threeNeighbors.fit(features_train,labels_train)
```

4) Import ‘test.csv’ file for testing the dataset.
5) Separate features from labels and then normalize features accordingly.

```
#separate features from labels in testing set
features_test = test_set.loc[:,['x','y','z']]
features_test = StandardScaler().fit_transform(features_test)
labels_test = test_set.loc[:,'actual-class']
```

6) Classify the test dataset and also estimate the class probability for each datapoints

```
#classify classes for the testing data
labels_pred = threeClassifier.predict(features_test)
labels_pred
```

```
array([1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0])
```

```
#estimate the probability for the testing data
labels_prob = threeClassifier.predict_proba(features_test)
labels_prob
```

```
array([[0.        , 1.        ],
       [0.33333333, 0.66666667],
       [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.        , 1.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ]])
```

7) Generate the confusion matrix and classification report. Note from the confusion matrix that the model misclassifies 2 data points.

```
#create confusion metrix and classification report
confusion_metrix_1 = confusion_matrix(labels_test,labels_pred)
confusion_metrix_1
```

```
array([[12,  2],
       [ 0,  6]])
```

```
print(classification_report(labels_test,labels_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        14
           1       0.75      1.00      0.86         6

    accuracy                           0.90        20
   macro avg       0.88      0.93      0.89        20
weighted avg       0.93      0.90      0.90        20
```

8) Calculate the weight from the user-defined function. The weight of each data point is 1/d^2. This function takes a distance matrix as an input and outputs the weight factor matrix.

```
#calculate the weight from a user-defined function
def weighted_dist(distance):
    #weight = np.zeros((distance.shape[0],1))
    weight = np.zeros(distance.shape)
    for i in range(0,len(distance)):
        d_square = np.power(distance[i][0] - distance[i][1],2) + np.power(distance[i][0] - distance[i][2],2) + np.po
        weight[i] = 1/d_square

    return weight
```

Note that the size of the weight factor is the same as the distance matrix below.

```
#Question(2): create a classifier using Euclidean distance with weighted 3-nearest neighbors(1/d^2)
kpoints = threeClassifier.kneighbors()
distance = kpoints[0]
distance #display the distance
```

```
array([[0.21388347, 0.29744062, 0.30803045],
       [0.14763209, 0.28948863, 0.3027134 ],
       [0.27955823, 0.33040975, 0.4966061 ],
       ...,
       [0.23978173, 0.2421378 , 0.28354917],
       [0.08039089, 0.26069554, 0.29814222],
       [0.13211408, 0.15813885, 0.28982672]])
```

9) Train the model n_neighbors = 3, weights = weighted_dist. The euclidean matric is by default.

```
#classifier for 3-nearest neighbors with weight (1/d^2)
threeNeighborsWeighted = KNeighborsClassifier(n_neighbors=3, weights=weighted_dist)
threeClassifierWeighted = threeNeighborsWeighted.fit(features_train,labels_train)
```

10) Classify the test data points and estimate the class probability.

```
#classify classes for the testing data
labels_pred_weighted = threeClassifierWeighted.predict(features_test)
labels_pred_weighted
```

```
array([1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0])
```

```
#estimate the weighted probability for the testing data
labels_prob_weighted = threeClassifierWeighted.predict_proba(features_test)
labels_prob_weighted
```

```
array([[0.        , 1.        ],
       [0.33333333, 0.66666667],
       [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.        , 1.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ]])
```

11) Generate the confusion matrix and classification report. Note that the prediction is similar to the previous model.

```
#create confusion metrix and classification report
confusion_metrix_2 = confusion_matrix(labels_test,labels_pred_weighted)
confusion_metrix_2
```

```
array([[12,  2],
       [ 0,  6]])
```

```
print(classification_report(labels_test,labels_pred_weighted))
```

```
              precision    recall  f1-score   support

           0       1.00      0.86      0.92        14
           1       0.75      1.00      0.86         6

    accuracy                           0.90        20
   macro avg       0.88      0.93      0.89        20
weighted avg       0.93      0.90      0.90        20
```