ASSIGNMENT – 2

DATA MINING

K Nearest Neighbour

In the Pa2 : Classification – Nearest Neighbours assignment , we were asked to apply different classifications on the given data set to put us in the best position to be able to mine the data in the most efficient manner.

In an attempt to achieve what was asked of us for the same, we have been given certain sub tasks which are :

- Load dataset_NN.csv dataset. [1 points]
- Pre-processing. [3 points]
- Select best 3 attributes for training and testing your model. [2 points]
- Find the best K using elbow method. [3 points]
- Split your dataset 75% for training, and 25% for testing the classifier. [2 points] • Use Euclidean distance.[3 points]
- Test the classifier with three different numbers for neighbors and record the results.[3 points]
- Use comments to explain your code and variable names.[2 points]
- Calculate and print the confusion matrix, and the classification Report (includes:precision, recall, f1-score, and support) for all three different numbers. **Plot the Error rate vs. K-value**.[6 points]

These all tasks all lead up to the concept in data mining called K Nearest Neighbour.

K Nearest Neighbour:

In the K nearest Neighbour, which is an algorithm, it is based on the metric of similarity of data between each aspect of data. KNN is known to give the best accuracy of the available models over huge sets of data. The application of KNN requires certain pre-requisites to be met before we start the process. One of the most basic steps as a data miner is to learn to pre-process the data for maintaining and gaining the most optimal values as we move ahead into the assignment. The next crucial step is to find the best K value. As asked in our project we achieved the best value of K using elbow method from the other methods available. Since knn works on the basis of distance between the attributes , we are also asked to use a metric for the same. We have been given a task to use “Euclidean distance”. This utilizes a combination of the elbow method and Euclidean distance by assigning a value for k that maps it to its nearest K. On selection of the best k we have calculated the distance from this k to all the values of k is minimum.

Jotting down the steps in a gest :

1. Loading the csv file which is unclassified.
2. Take a measure from the newly imported to the already classified data.
3. Attain a value for K where distance is minimal.
4. We check the attributes with best metrics .
5. We then test the classifier with the best attributes which we selected. (For comparison we have also selected to test the classifier with all the attributes)
6. For our understanding we print the confusion matrix and the classification matrix and then plot the error rate with respect to k-value.

Task#1 - Load dataset_NN.csv dataset. [1 points]

The initial step towards working on any data mining assignment, is to read the file into our assignment. We have achieved this using the function "pd.read_csv".

```
In [58]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import seaborn as sns

         from sklearn.preprocessing import LabelEncoder
         from sklearn.model_selection import train_test_split
         from sklearn.cluster import KMeans
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import classification_report
         from sklearn.neighbors import KNeighborsClassifier

         #Load dataset_NN.csv dataset. [1 points]

         ship_data = pd.read_csv('dataset_NN.csv')
```

Using this function we have read our dataset file by the name "dataset_NN.csv".

We have imported all the required libraries and packages. **Task#2**

- Pre-processing. [3 points]

We have to firstly pre-process the data before we proceed to mine. On exploring the given data set we have found that the attribute age contains missing values. Further more , in order to find k value and implement K value and confusion matrix , it is required to convert the string values to integer values.
We achieve both these by using the function fillna and fit_transform .

```
In [15]: #Pre-processing. [3 points]
         #converting all values to numericals for better handling
         encode = LabelEncoder()
         encoded_set = ship_data[ship_data.columns[:]].apply(encode.fit_transform)

         #replacing missing values in Age with NA

         ship_data.fillna(np.mean)[:1]
         ship_data.Age.fillna("NA")
```

```
Out[15]: 0      22.0
         1      38.0
         2      26.0
         3      35.0
         4      35.0
                ...
         886    27.0
         887    19.0
         888      NA
         889    26.0
         890    32.0
         Name: Age, Length: 891, dtype: object
```

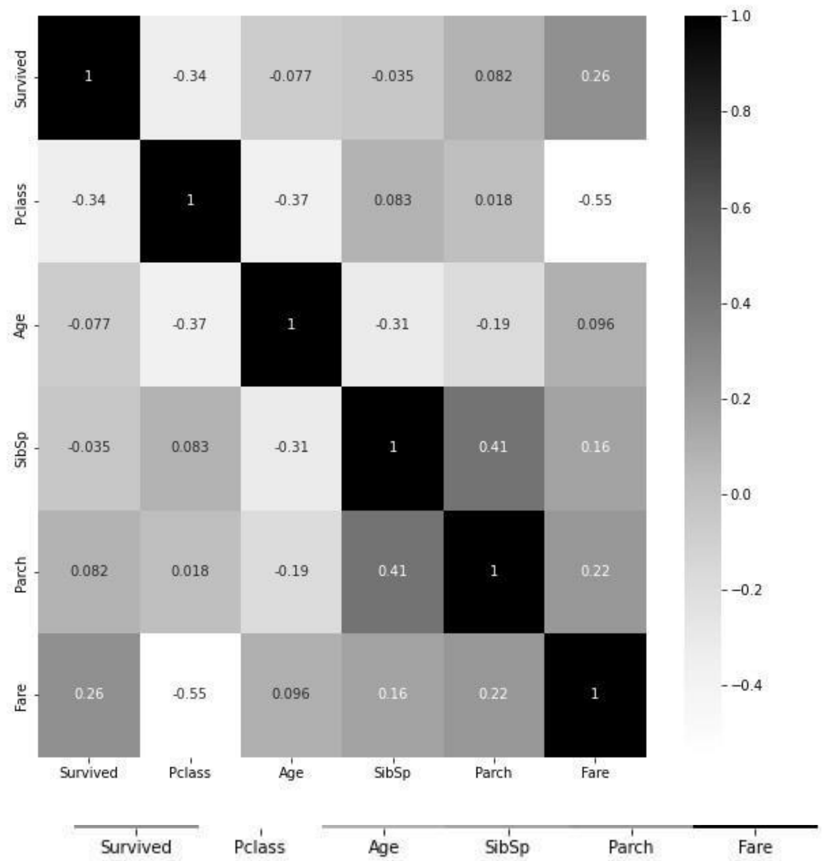
In the attached image we can see that the values that are missing have been replaced with the string "NA".

TASK#3 - Select best 3 attributes for training and testing your model. [2 points]

It is important to select the best attributes based on the fact of their uniformity and the level to which they are influencing the results of our tests and trainings.

```
In [16]: #Select best 3 attributes for training and testing your model. [2 points]
plt.figure(figsize=(10,10))
correaltion = ship_data.corr()
sns.heatmap(correaltion, annot=True, cmap=plt.cm.Greys)
plt.show()

ship_data.corr()['Survived']
```




```
Out[16]: Survived    1.000000
Pclass     -0.338481
Age        -0.077221
SibSp      -0.035322
Parch       0.081629
Fare        0.257307
Name: Survived, dtype: float64
```

```
In [17]: #for finding K value
output_variable = abs(cor["Survived"])
#printing top 3 correlated features
output_variable = output_variable.sort_values(ascending=False)
output_variable = output_variable.iloc[:4]

print(output_variable.head())
```

```
Survived    1.000000
Pclass      0.338481
Fare        0.257307
Parch       0.081629
Name: Survived, dtype: float64
```

Observation : We used output_variable to print the top three best attributes. The best three attributes are Pcass , Fare and Parch

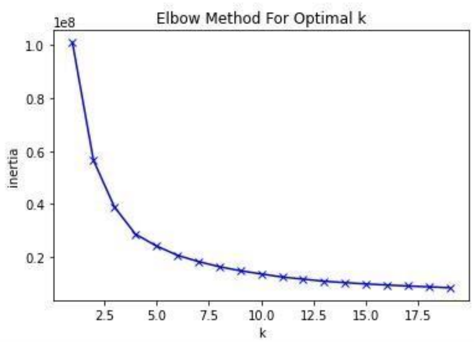
TASK#4 - Find the best K using elbow method. [3 points]:

Inertia is used to understand how well the clustering is done with respect to the values of their centroids. An opitmal inertia is one with a low value and a lower number of clusters.

```
In [18]: #Finding the best K using elbow method (Optimal K)

inertia = []
K = range(1,20)
for k in K:
    km = KMeans(n_clusters=k,)
    km = km.fit(encoded_set)
    inertia.append(km.inertia_)

plt.plot(K, inertia, 'bx-')
plt.xlabel('k')
plt.ylabel('inertia')
plt.title('Elbow Method For Optimal k')
plt.show()
```



After reading from the encoded_set we use the functions from matplotlib package to plot a graph [passing the required parameters.

TASK#5-Split your dataset 75% for training, and 25% for testing the classifier. [2 points]

As part our testing ad training processes we primarily need to split the dataset over the required attributes with a ratio of 75 is to 25.

In order to perform our tasks as well as possible we have chosen to take two sets of variables, X,y and W,v while passing the best attributes in to X while passing all columns into W.

We used the object ship_data to pass the best three attributes Pclass, Fare and Parch.

While passing into W we passed all columns using the output variable and function .isin

```
In [35]: #Split your dataset 75% for training, and 25% for testing the classifier. [2 points]
X = ship_data[['Pclass','Fare','Parch']]
y = ship_data['Survived']

W = ship_data.loc[:, ship_data.columns.isin(output_variable.keys())]
v = ship_data['Survived']

training_X,testing_X,training_y,testing_y = train_test_split(X,y,test_size=0.25,random_state=2022)
training_W,testing_W,training_v,testing_v = train_test_split(W,v,test_size=0.25,random_state=2022)
```

TASK#6 -Use Euclidean distance.[3 points] :

TASK#7 Test the classifier with three different numbers for neighbors and record the results.[3 points]

In Euclidean distance, we measure the distance between two points and utilize this as part of measuring cluster analysis.   KNN algorithms is one of the most used application of Euclidean distance for cluster analysis. This is where we test the classifier by the metric of Euclidean over the Neighbours 10-14.

```
In [42]: error_rate = []
for i in range(10,14):
 knn = KNeighborsClassifier(n_neighbors=i,metric="euclidean")
 knn.fit(training_X,training_y)
 predicted = knn.predict(testing_X)
 matrix = confusion_matrix(testing_y, predicted)
 report = classification_report(testing_y, predicted)
 print("Confusion Matrix for n_neighbors=",i,"\n")
 print(matrix)
 print("\nClassification Report for n_neighbors=",i,"\n")
 print(report)
 error_rate.append(np.mean(predicted != testing_y))
```

```
Confusion Matrix for n_neighbors= 10

[[118  25]
 [ 47  33]]

Classification Report for n_neighbors= 10

              precision    recall  f1-score   support

           0       0.72      0.83      0.77       143
           1       0.57      0.41      0.48        80

    accuracy                           0.68       223
   macro avg       0.64      0.62      0.62       223
weighted avg       0.66      0.68      0.66       223

Confusion Matrix for n_neighbors= 11

[[113  30]
 [ 44  36]]

Classification Report for n_neighbors= 11

              precision    recall  f1-score   support

           0       0.72      0.79      0.75       143
           1       0.55      0.45      0.49        80

    accuracy                           0.67       223
   macro avg       0.63      0.62      0.62       223
weighted avg       0.66      0.67      0.66       223

Confusion Matrix for n_neighbors= 12

[[121  22]
 [ 48  32]]

Classification Report for n_neighbors= 12

              precision    recall  f1-score   support

           0       0.72      0.85      0.78       143
           1       0.59      0.40      0.48        80

    accuracy                           0.69       223
   macro avg       0.65      0.62      0.63       223
weighted avg       0.67      0.69      0.67       223

Confusion Matrix for n_neighbors= 13

[[116  27]
 [ 43  37]]

Classification Report for n_neighbors= 13

              precision    recall  f1-score   support

           0       0.73      0.81      0.77       143
           1       0.58      0.46      0.51        80

    accuracy                           0.69       223
   macro avg       0.65      0.64      0.64       223
weighted avg       0.68      0.69      0.68       223
```

```
Confusion Matrix for n_neighbors= 10

[[126  17]
 [ 13  67]]

Classification Report for n_neighbors= 10

              precision    recall  f1-score   support

           0       0.91      0.88      0.89       143
           1       0.80      0.84      0.82        80

    accuracy                           0.87       223
   macro avg       0.85      0.86      0.86       223
weighted avg       0.87      0.87      0.87       223

Confusion Matrix for n_neighbors= 11

[[122  21]
 [  7  73]]

Classification Report for n_neighbors= 11

              precision    recall  f1-score   support

           0       0.95      0.85      0.90       143
           1       0.78      0.91      0.84        80

    accuracy                           0.87       223
   macro avg       0.86      0.88      0.87       223
weighted avg       0.89      0.87      0.88       223

Confusion Matrix for n_neighbors= 12

[[126  17]
 [  9  71]]

Classification Report for n_neighbors= 12

              precision    recall  f1-score   support

           0       0.93      0.88      0.91       143
           1       0.81      0.89      0.85        80

    accuracy                           0.88       223
   macro avg       0.87      0.88      0.88       223
weighted avg       0.89      0.88      0.88       223

Confusion Matrix for n_neighbors= 13

[[126  17]
 [  8  72]]

Classification Report for n_neighbors= 13

              precision    recall  f1-score   support

           0       0.94      0.88      0.91       143
           1       0.81      0.90      0.85        80

    accuracy                           0.89       223
   macro avg       0.87      0.89      0.88       223
weighted avg       0.89      0.89      0.89       223
```

Observation: We can observe here that the accuracy is at a good rate of 90 + for all three values of k that we took.

We chose an array by the name error_rate and proceeded to select a range of i = 10 to 14 .

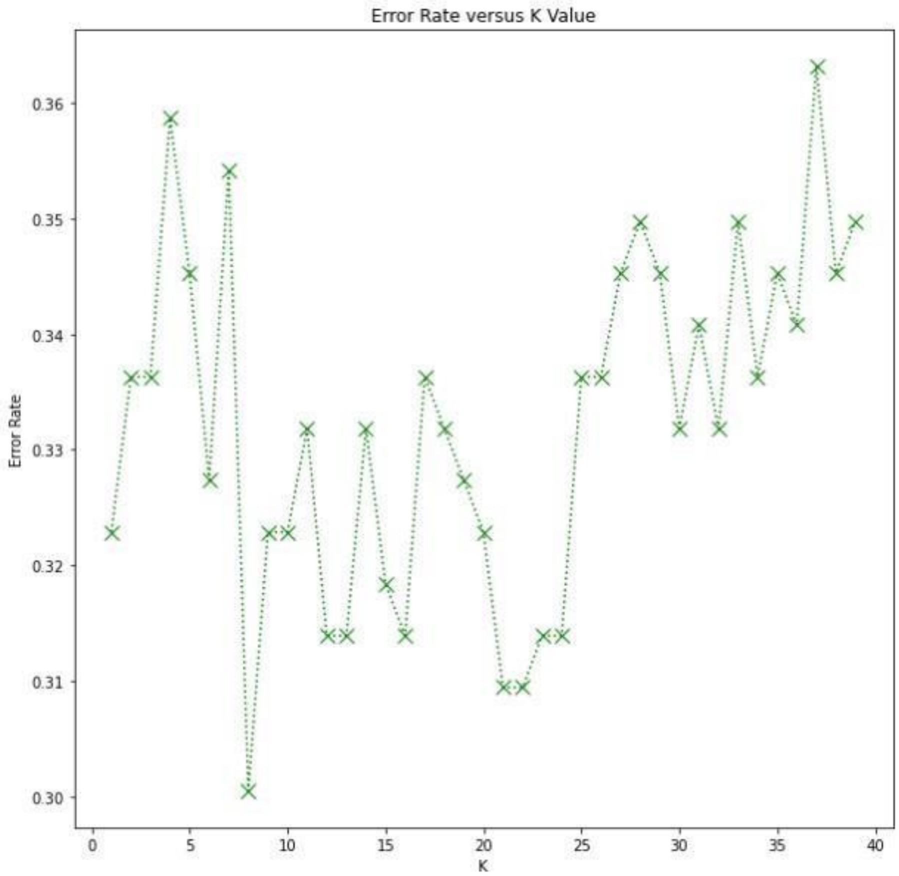
This is because we are only required to perform the task of three values for nearest Neighbours.

TASK#8 : Calculate and print the confusion matrix, and the classification Report (includes:precision, recall, f1-score, and support) for all three different numbers. Plot the Error rate vs. K-value.[6 points]

In task 8 we were asked to print the confusion matrix which we had finished above. The classification has also be done above. Now to plot error vs k-value , We have implemented the same using the following lines of code :

In [55]:
```
#Plot the Error rate vs. K-value.
error_rate = []
for i in range(1,40):
  knn = KNeighborsClassifier(n_neighbors=i)
  knn.fit(training_X,training_y)
  pred_i = knn.predict(testing_X)
 error_rate.append(np.mean(pred_i != testing_y))
plt.figure(figsize=(10,10))
plt.plot(range(1,40),error_rate,color='green', linestyle='dotted',
marker='x',markerfacecolor='red', markersize=10)
plt.title('Error Rate versus K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("The least/minimal error value = ",min(error_rate),".","\nThe minimal error value is obtained at K =",e
```

```
The least/minimal error value =  0.3004484304932735 .
The minimal error value is obtained at K = 7
```



Observation : From the graph we can see that most optimal value is at k=7 which is the lowest drop of the graph.

We have utilized the default methods provided by the knnneighbour package.

We have printed the minimal error value using min(error_rate) function while passing the parameter error_rate.

We have further printed the error value at its minimal at value of k using min(error_rate.index(min()) where error rate is again the required variable.