Lab 7

December 30, 2021

Machine Learning Lab

Lab 07

Importing Packages

[1]:
```
import os                                       #Importing OS
import pandas as pd                             #Importing Pandas
import numpy as np                              #Importing Numpy
import matplotlib.pyplot as plt                 #Importing Matplotlib
import random                                   #Importing Random
from scipy.spatial.distance import pdist        #Importing Scipy pdist
```

0.0.1 Exercise 0: Dataset Preprocessing

[2]:
```
#Initializing a random seed to be used for sampling
random_seed = 3116
```

Function to check for Padding and if needed pad with 0s

[3]:
```
def check_for_padding(df):

    #Getting the Maximum row size to match for
    max_row_size = df.shape[1]

    #Iterating through all rows of the provided dataframe
    for index, row in df.iterrows():

        #Extracting the last index till where there is any valid value and␣
↪adding 1 to convert this into total valid values
        total_valid_values = row.last_valid_index() + 1

        #If total valid values in a row is less than maximum number values then␣
↪we do padding
        if total_valid_values < max_row_size:

            #Padding from the left with 0s
            df.iloc[index,:] = row.shift(max_row_size - total_valid_values,␣
↪fill_value=0)
```

```
    #Returning the modified dataframe
    return df
```

Function to Merge Two Dataframes having same columns and Structure

```
[4]: def merge_dataframes(df1 ,df2):
         #Checking if both the dataframe have same number of columns
         if len(df1.columns) != len(df2.columns):
             raise Exception('For Merging Two Dataframes, the number of columns in␣
      ↪both Dataframes should be equal')

         #Checking for variable sample length in dataframe 1 and doing padding if␣
      ↪necessary
         df1 = check_for_padding(df1)

         #Checking for variable sample length in dataframe 2 and doing padding if␣
      ↪necessary
         df2 = check_for_padding(df2)

         #Returning the Concatenation of Dataframe 1 and Dataframe 2 and reassigning␣
      ↪the index values
         return pd.concat([df1 ,df2], ignore_index = True)
```

Function to do splitting into Training and Test set using Stratified Sampling

```
[5]: def stratify_split_dataframe(df ,train_ratio ,validation_ratio, test_ratio):
         #Shuffling the dataset to induce the Randomization in the dataset
         df = df.sample(frac = 1, random_state = random_seed).reset_index(drop =␣
      ↪True)

         #Creating Matrix X by droping the class column
         X = df.drop(0,axis=1)

         #Creating the Y column vector by only taking the class column
         Y = df.iloc[:,0].to_numpy()

         #Appending the bias column of 1 in the starting of Matrix X
         X = np.append(np.ones(shape=(len(X),1)),X,axis=1)

         #Calculating the total rows for Train and Validation sets
         total_rows_train = int(len(X)*train_ratio)
         total_rows_validation = int(len(X)*validation_ratio)

         #Splitting the dataset into Train, Validation and Testing sets
         X_train , X_val , X_test = X[:total_rows_train,:] , X[total_rows_train:
      ↪total_rows_train + total_rows_validation,:] , X[total_rows_train +␣
      ↪total_rows_validation:,:]
```

```
    Y_train , Y_val , Y_test = Y[:total_rows_train].reshape(-1,1) ,␣
↪Y[total_rows_train:total_rows_train + total_rows_validation].reshape(-1,1) ,␣
↪Y[total_rows_train + total_rows_validation:].reshape(-1,1)

    #Returing the Splitted datasets
    return X_train, Y_train, X_test, Y_test, X_val, Y_val
```

Function to Normalize and Standardize the dataset by removing the mean and scaling to unit-variance

[6]:
```
def normalize_dataset(df):
    #Returing (x-mean)/standard Deviation
    return (df - df.mean())/df.std()
```

Preprocessing the datasets

[7]:
```
#Base path for all the datasets
base_path = 'UCRArchive_2018/'

#Train file Suffix
train_file_suffix = '_TRAIN.tsv'

#Test File Suffix
test_file_suffix = '_TEST.tsv'

#List containing the folder names to exclude from processing
exclude_folders = ['Missing_value_and_variable_length_datasets_adjusted']
```

[8]:
```
#A dictionary to contain the statistics of all the datasets
dataset_stats = {'Name' : [],
                 'Total Columns' : [],
                 'Total Rows' : [],
                 'Total Classes' : [],
                 'Null Values' : []}
```

[9]:
```
#A dictionary to contain the processed dataframes for each datasets
datasets = {'Name' : [],
            'Dataset' : []}
```

Scanning the Dataset Folders and extracting all the train and test data files for processing

[10]:
```
#Using the os package to scan the base dataset folder
with os.scandir(base_path) as entries:
    for entry in entries:
        #Checking if entry is a directory and is not in the excluded folders␣
↪list
        if (entry.is_dir()) and (entry.name not in exclude_folders):
```

```
            #Formating the training and testing file name
            train_filename = '{}/{}'.format(entry.path, entry.name +␣
↪train_file_suffix)
            test_filename = '{}/{}'.format(entry.path, entry.name +␣
↪test_file_suffix)

            #Reading the Train and Test CSV file using Pandas
            train_df , test_df = pd.read_csv(train_filename , sep='\t' , header␣
↪= None) , pd.read_csv(test_filename , sep='\t' , header = None)

            #Merging the Train and Test Dataframes
            merged_df = merge_dataframes(train_df,test_df)

            #Storing the stats of this dataset into our Dictionary
            dataset_stats['Name'].append(entry.name)
            dataset_stats['Total Columns'].append(len(merged_df.columns))
            dataset_stats['Total Rows'].append(merged_df.shape[0])
            dataset_stats['Total Classes'].append(len(merged_df.iloc[:,0].
↪unique()))
            dataset_stats['Null Values'].append(merged_df.isnull().values.any())

            #Storing the Processed Dataframe into the created dictionary
            datasets['Name'].append(entry.name)
            datasets['Dataset'].append(merged_df)
```

Converting the Dictionary to the Dataframe for Displaying purposes

[11]:
```
dataset_info = pd.DataFrame.from_dict(dataset_stats)
dataset_info
```

[11]:

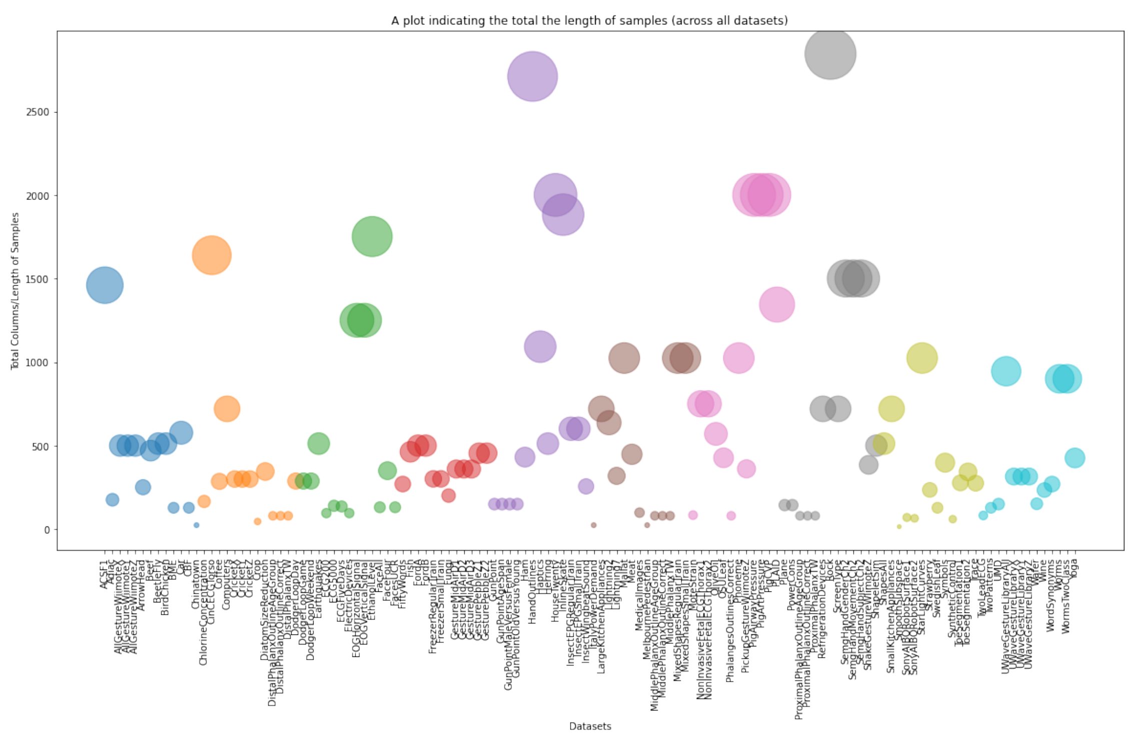
	Name	Total Columns	Total Rows	Total Classes	Null Values
0	ACSF1	1461	200	10	False
1	Adiac	177	781	37	False
2	AllGestureWiimoteX	501	1000	1	False
3	AllGestureWiimoteY	501	1000	1	False
4	AllGestureWiimoteZ	501	1000	1	False
..	…	…	…	…	…
123	Wine	235	111	2	False
124	WordSynonyms	271	905	25	False
125	Worms	901	258	5	False
126	WormsTwoClass	901	258	2	False
127	Yoga	427	3300	2	False

[128 rows x 5 columns]

Plotting interesting statistics Plotting Datasets versus Total Columns

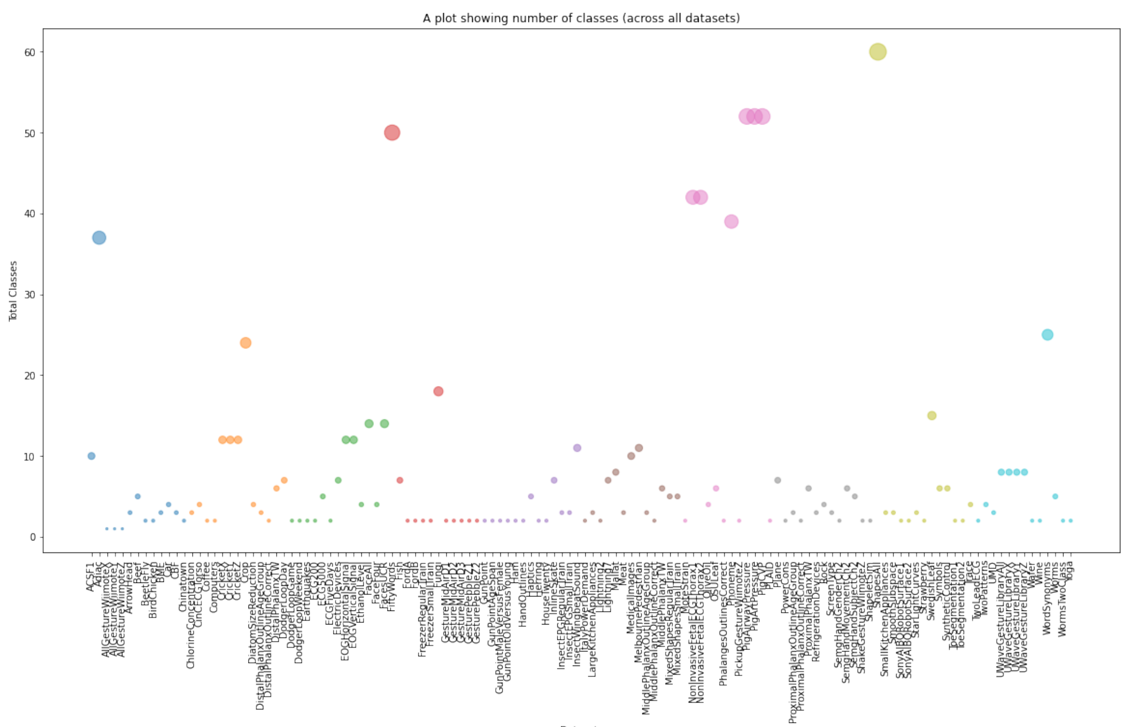
[12]:

```python
#Creating a Figure
fig = plt.figure(figsize=(20,10))

#Creating an array containing marker size of each dataset
marker_size = dataset_stats['Total Columns']

#Creating an array containing the color code for each dataset
colors = [plt.cm.tab10(i/float(len(dataset_stats['Total Columns'])-1)) for i in range(len(dataset_stats['Total Columns']))]

#Plotting the scatter plot
plt.scatter(dataset_stats['Name'],dataset_stats['Total Columns'], s = marker_size,c = colors,alpha = 0.5)

#Setting the parameters for the graph
plt.xlabel('Datasets')
plt.ylabel('Total Columns/Length of Samples')
plt.title('A plot indicating the total the length of samples (across all datasets)')
plt.xticks(rotation=90)

#Showing the Graph
plt.show()
```



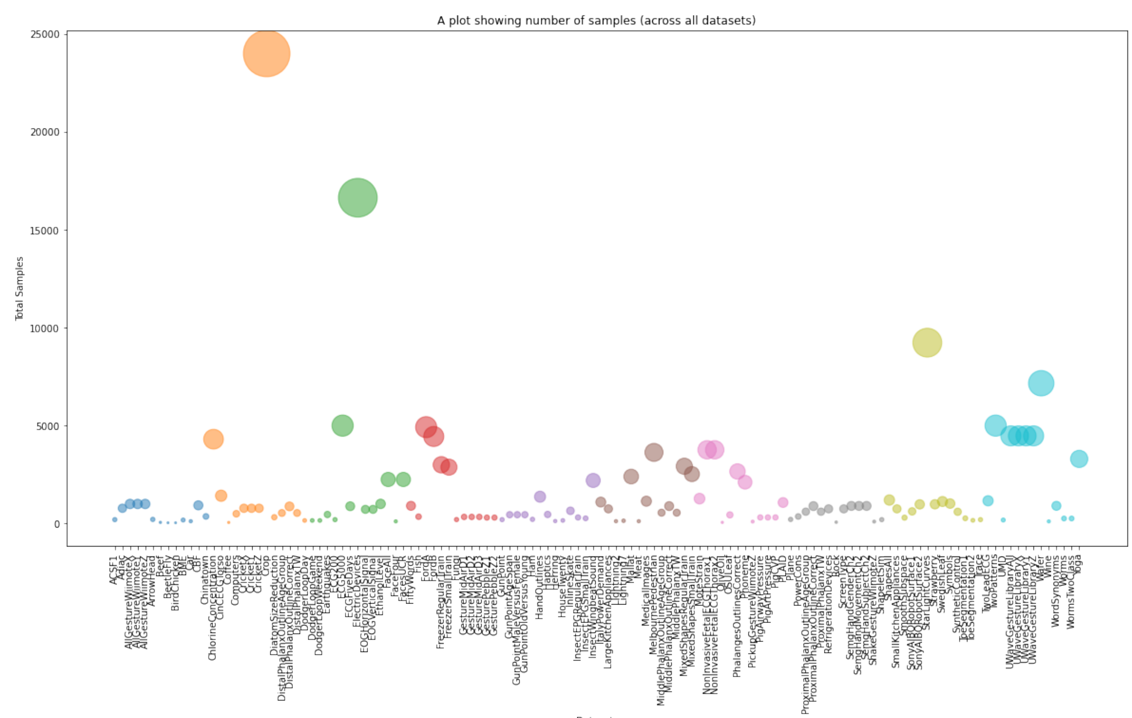
Plotting Datasets versus Total Classes

[13]:
```python
#Creating a Figure
fig = plt.figure(figsize=(20,10))

#Creating an array containing marker size of each dataset
marker_size = np.multiply(dataset_stats['Total Classes'] , 5)

#Creating an array containing the color code for each dataset
colors = [plt.cm.tab10(i/float(len(dataset_stats['Total Classes'])-1)) for i in␣
↪range(len(dataset_stats['Total Classes']))]

#Plotting the scatter plot
plt.scatter(dataset_stats['Name'],dataset_stats['Total Classes'], s =␣
↪marker_size,c = colors,alpha = 0.5)

#Setting the parameters for the graph
plt.xlabel('Datasets')
plt.ylabel('Total Classes')
plt.title('A plot showing number of classes (across all datasets)')
plt.xticks(rotation=90)

#Showing the Graph
plt.show()
```



Plotting Datasets versus Total Rows

[50]:
```python
#Creating a Figure
fig = plt.figure(figsize=(20,10))

#Creating an array containing marker size of each dataset
marker_size = np.multiply(dataset_stats['Total Rows'] , 0.1)

#Creating an array containing the color code for each dataset
colors = [plt.cm.tab10(i/float(len(dataset_stats['Total Rows'])-1)) for i in range(len(dataset_stats['Total Rows']))]

#Plotting the scatter plot
plt.scatter(dataset_stats['Name'],dataset_stats['Total Rows'], s = marker_size,c = colors,alpha = 0.5)

#Setting the parameters for the graph
plt.xlabel('Datasets')
plt.ylabel('Total Samples')
plt.title('A plot showing number of samples (across all datasets)')
plt.xticks(rotation=90)

#Showing the Graph
plt.show()
```



0.0.2 Exercise 1: Dataset Imputation with KNN

List the datasets having missing values

[15]:
```
#Extracting Datasets having Some Missing Values
missing_value_datasets = dataset_info[dataset_info['Null Values'] == True]
missing_value_datasets
```

[15]:

	Name	Total Columns	Total Rows	Total Classes	Null Values
25	DodgerLoopDay	289	158	7	True
26	DodgerLoopGame	289	158	2	True
27	DodgerLoopWeekend	289	158	2	True
71	MelbournePedestrian	25	3633	11	True

For each dataset with missing values, and for each feature (timestep) of it that has missing values imputing the value by calculating the mean of its nearest K neighbors. Function to calculate the euclidean distance between two rows in a dataframe

[16]:
```
def calculate_euclidean_distance_df(curr_row ,other_rows):
    sum_square = np.sum(np.square(curr_row - other_rows) ,axis = 1)
    return np.sqrt(sum_square)
```

Function to calculate the euclidean distance between two points in numpy

[17]:
```
def calculate_euclidean_distance_numpy(point1 ,point2):
    sum_square = np.sum(np.square(point1 - point2))
    return np.sqrt(sum_square)
```

Function to calculate the mean value from k rows values

[18]:
```
def calculate_average_value(other_row ,distance,k):
    #Extracting the k nearest rows with minimum distance
    nearest_rows = other_row.iloc[np.argsort(list(distance))[:k]]

    #Returning the mean of all the rows over k rows
    return np.sum(nearest_rows ,axis = 0) / k
```

Function to calculate the accuracy of the KNN model

[19]:
```
def accuracy_knn_classification(predicted_classes ,actual_classes):
    #Checking if the size of both predicted classes and actual classes is same
    if len(predicted_classes) != len(actual_classes):
        raise Exception('The size of both Predicted Classes array and Actual␣
 ↪Classes array should be same')

    #Initializing variable to 0
    correct = 0

    #Iterating through all values to match these values
    for i in range(len(predicted_classes)):
```

```
        #If values are equal then incrementing the correct variable
        if predicted_classes[i] == actual_classes[i]:
            correct += 1

    #Returning the accuracy as the precentage
    return (correct / len(predicted_classes)) * 100
```

Function to impute the missing data in the dataset using KNN approximation

```
[20]: def impute_data_with_KNN(df ,k):
    #Iterating over all rows and finding rows with missing values
    for index, row in df.iterrows():

        #Checking if the current row contains any null values
        if row.isnull().values.any():

            #Calculating the distance between current index row and all other␣
 ↪rows
            distance = calculate_euclidean_distance_df(df.iloc[index] ,df.
 ↪iloc[~df.index.isin([index])])

            #Calculating the mean value of k nearest rows
            average_values = calculate_average_value(df.iloc[~df.index.
 ↪isin([index])] ,distance ,k)

            #Replacing the Null values with the new calculated values
            row.fillna(average_values ,inplace = True)

            #Replacing the row with the updated row values
            df.iloc[index ,:] = row

    #Returning the modified dataframe
    return df
```

Function the predict the class of observation using KNN Classific

```
[21]: def predict_knn_class(trainX ,trainY ,q ,k ,use_scipy, scipy_distance_metric):
    #Initializing an empty distance array
    distance_array = np.zeros(shape=(len(trainX) ,))

    #Iterating through all rows of the train X rows for calculating the␣
 ↪euclidean distance
    for index, observation in enumerate(trainX):

        #If scipy is allowed to use
        if use_scipy:
            try:
```

```
                #Creating a query matrix with one row of query and one row of␣
↪training observation
                query_matrix = np.append(q.reshape(1,-1),observation.
↪reshape(1,-1),axis=0)

                #Calculating the distance using Scipy using the provided␣
↪distance metric
                distance_array[index] =␣
↪pdist(query_matrix,scipy_distance_metric)[0]
            except:
                #If for any reason, the scipy distance cannot be calculate,␣
↪then replacing it with positive infinity indicating the maximum distance
                #Replacing it with positive infinity because in the end it will␣
↪ignored by k nearest neighbour calculation
                distance_array[index] = np.iinfo(np.int32).max
        else:
            #If Scipy is not allowed then
            #Calculating the distance between the queried and the current␣
↪indexed row using euclidean and saving it in an array
            distance_array[index] = calculate_euclidean_distance_numpy(q␣
↪,observation)

    #Extracting the k nearest rows based on distance value
    k_min_distance = np.argsort(distance_array)[:k]

    #Getting the classes of these nearest rows
    k_nearest_class = trainY[k_min_distance,:]

    #The predicted class is then the maximum occuring class
    values, counts = np.unique(k_nearest_class.ravel(), return_counts=True)
    predicted_class = values[np.argmax(counts)]

    #Returning the Predicted class
    return predicted_class
```

[22]:

```
imputation_dataset_accuracy = {
                    'Name' : [],                #Contains the Name of Datasets
                    'Classification K' : [],    #Contains the Optimum K value␣
↪for Classification found using Grid Search for each Dataset
                    'Imputation K' : [],        #Contains the Optimum K value␣
↪for Imputation found using Grid Search for each Dataset
                    'Validation Accuracy' : [], #Contains the Best Validation␣
↪accuracy achieved using the Optimum Classification K value and Optimum␣
↪Imputation K
                    'Test Accuracy' : []        #Contains the Test Accuracy␣
↪achieved using the Optimum Classification K value and Optimum Imputation K
```

```
}
```

Applying Grid Search on K for classification and k for Imputation on every Dataset

```
[23]: #Iterating/Enumerating through all the datasets
for index , dataset in enumerate(datasets['Dataset']):

    #Creating a accuracy matrix which will contain the accuracy for each␣
 ↪combination of K classification and K Imputation
    accuracy_matrix = np.zeros(shape=(5, 5))

    #Splitting the dataset into Train, Validation and Test sets
    X_train, Y_train, X_test, Y_test, X_val, Y_val =␣
 ↪stratify_split_dataframe(dataset, 0.7, 0.15, 0.15)

    #Iterating through different values of K for KNN Classification, Picking K␣
 ↪from [1,2,3,4,5]
    for k_classification in range(1, 6):

        #Iterating through different values of K for Data Imputation, Picking K␣
 ↪from [1,2,3,4,5]
        for k_impute in range(1, 6):

            #Checking if Dataset has any null values in it
            if dataset.isnull().values.any():
                #Imputing the missing values in dataset using KNN data␣
 ↪Imputation with specific K
                dataset = impute_data_with_KNN(dataset, k_impute)

            #Creating an array to store the predicated classes for validation␣
 ↪set
            predicted_class_val = []

            #Iterating through each observation in the Validation set
            for x_val in X_val:

                #Predicting the class of validation row using the specific K␣
 ↪and Train set
                pred_class = predict_knn_class(X_train, Y_train, x_val,␣
 ↪k_classification, False, '')

                #Appending the predicted class to the created array
                predicted_class_val.append(pred_class)

            #Now calculating the total accuracy of the Validation set
            accuracy = accuracy_knn_classification(predicted_class_val, Y_val.
 ↪ravel())
```

```
            #Saving the accuracy in our Accuracy Matrix for current Combination
            accuracy_matrix[k_classification - 1, k_impute - 1] = accuracy

    #Finding the combination with the highest accuracy for the current Dataset
    optimum_point = np.where(accuracy_matrix == np.amax(accuracy_matrix))
    optimum_k_classification , optimum_k_imputation = optimum_point[0][0] ,␣
↪optimum_point[1][0]

    #Creating an empty array to store the predicted classes for the Test set
    predicted_class_test = []

    #Iterating through all the observations in the Test set
    for x_test in X_test:

        #Predicting the class of Test row using the specific K and Train set
        pred_class = predict_knn_class(X_train, Y_train, x_test,␣
↪optimum_k_classification + 1, False, '')

        #Appending the predicted class to the created array
        predicted_class_test.append(pred_class)

    #After predicting the classes for all Test set observation, now calculating␣
↪it overall accuracy
    test_accuracy = accuracy_knn_classification(predicted_class_test, Y_test.
↪ravel())

    #Checking if Dataset has any null values in it
    if dataset.isnull().values.any():
        #Imputing the missing values in dataset using KNN data Imputation with␣
↪Optimum K for Imputation
        dataset = impute_data_with_KNN(dataset, optimum_k_imputation + 1)

        #Replacing the old dataset with the Imputed Dataset with best K
        datasets['Dataset'][index] = dataset

    #Appending the Optimum combination of K for classification and k for␣
↪Imputation for the current Dataset in the dictionary
    imputation_dataset_accuracy['Name'].append(datasets['Name'][index])
    imputation_dataset_accuracy['Classification K'].
↪append(optimum_k_classification + 1)
    imputation_dataset_accuracy['Imputation K'].append(optimum_k_imputation + 1)
    imputation_dataset_accuracy['Validation Accuracy'].
↪append(accuracy_matrix[optimum_k_classification,optimum_k_imputation])
    imputation_dataset_accuracy['Test Accuracy'].append(test_accuracy)
```

```
    #Printing the Optimum Combination for the current Dataset
    print('The Best Classification K value and the Best Imputation K value for␣
↪Dataset:\nDataset Name: {}\t Validation Accuracy: {}\tTest Accuracy:␣
↪{}\nBest Classification K: {}\tBest Imputation K: {}\n'.format(
          datasets['Name'][index],␣
↪accuracy_matrix[optimum_k_classification,optimum_k_imputation],␣
↪test_accuracy, optimum_k_classification + 1, optimum_k_imputation + 1))
```

```
The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ACSF1      Validation Accuracy: 56.666666666666664        Test
Accuracy: 56.666666666666664
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Adiac      Validation Accuracy: 64.1025641025641  Test Accuracy:
63.559322033898304
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: AllGestureWiimoteX         Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: AllGestureWiimoteY         Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: AllGestureWiimoteZ         Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ArrowHead  Validation Accuracy: 87.09677419354838 Test Accuracy:
81.81818181818183
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Beef       Validation Accuracy: 77.77777777777779 Test Accuracy:
44.44444444444444
Best Classification K: 2        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: BeetleFly  Validation Accuracy: 66.66666666666666 Test Accuracy:
50.0
Best Classification K: 1        Best Imputation K: 1
```

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: BirdChicken   Validation Accuracy: 66.66666666666666 Test Accuracy: 83.33333333333334
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: BME   Validation Accuracy: 100.0   Test Accuracy: 100.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Car   Validation Accuracy: 88.88888888888889 Test Accuracy: 66.66666666666666
Best Classification K: 4   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: CBF   Validation Accuracy: 97.84172661870504 Test Accuracy: 97.14285714285714
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Chinatown Validation Accuracy: 98.14814814814815 Test Accuracy: 100.0
Best Classification K: 5   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ChlorineConcentration   Validation Accuracy: 97.67801857585138 Test Accuracy: 98.145285935085
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: CinCECGTorso   Validation Accuracy: 99.53051643192488 Test Accuracy: 100.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Coffee   Validation Accuracy: 100.0   Test Accuracy: 100.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Computers Validation Accuracy: 66.66666666666666 Test Accuracy: 52.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: CricketX   Validation Accuracy: 63.24786324786324 Test Accuracy: 64.1025641025641
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: CricketY   Validation Accuracy: 62.39316239316239 Test Accuracy: 50.427350427350426
Best Classification K: 5   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: CricketZ   Validation Accuracy: 64.1025641025641 Test Accuracy: 58.97435897435898
Best Classification K: 2   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Crop   Validation Accuracy: 76.91666666666667 Test Accuracy: 76.08333333333334
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DiatomSizeReduction   Validation Accuracy: 100.0   Test Accuracy: 100.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DistalPhalanxOutlineAgeGroup   Validation Accuracy: 83.75 Test Accuracy: 81.70731707317073
Best Classification K: 5   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DistalPhalanxOutlineCorrect   Validation Accuracy: 77.86259541984732 Test Accuracy: 81.81818181818183
Best Classification K: 3   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DistalPhalanxTW   Validation Accuracy: 76.25   Test Accuracy: 76.82926829268293
Best Classification K: 5   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DodgerLoopDay   Validation Accuracy: 60.86956521739131 Test Accuracy: 36.0
Best Classification K: 4   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DodgerLoopGame   Validation Accuracy: 86.95652173913044 Test Accuracy: 84.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: DodgerLoopWeekend Validation Accuracy: 100.0   Test Accuracy: 96.0

```
Best Classification K: 2        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Earthquakes        Validation Accuracy: 81.15942028985508 Test
Accuracy: 84.28571428571429
Best Classification K: 4        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ECG200     Validation Accuracy: 90.0      Test Accuracy: 90.0
Best Classification K: 2        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ECG5000    Validation Accuracy: 94.93333333333334 Test Accuracy:
94.8
Best Classification K: 2        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ECGFiveDays        Validation Accuracy: 97.72727272727273 Test
Accuracy: 99.25373134328358
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ElectricDevices    Validation Accuracy: 72.70541082164328 Test
Accuracy: 72.16659991990389
Best Classification K: 3        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: EOGHorizontalSignal        Validation Accuracy: 65.74074074074075
Test Accuracy: 66.36363636363637
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: EOGVerticalSignal  Validation Accuracy: 57.407407407407405
Test Accuracy: 57.27272727272727
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: EthanolLevel       Validation Accuracy: 29.333333333333332
Test Accuracy: 32.89473684210527
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FaceAll    Validation Accuracy: 94.06528189910979 Test Accuracy:
92.3076923076923
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FaceFour   Validation Accuracy: 100.0     Test Accuracy:
```

```
83.33333333333334
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FacesUCR   Validation Accuracy: 94.65875370919882 Test Accuracy:
95.56213017751479
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FiftyWords         Validation Accuracy: 65.18518518518519 Test
Accuracy: 70.07299270072993
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Fish       Validation Accuracy: 92.3076923076923  Test Accuracy:
81.48148148148148
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FordA      Validation Accuracy: 70.46070460704607 Test Accuracy:
67.65899864682002
Best Classification K: 5        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FordB      Validation Accuracy: 67.26726726726727 Test Accuracy:
66.61676646706587
Best Classification K: 4        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FreezerRegularTrain        Validation Accuracy: 96.44444444444444
Test Accuracy: 94.66666666666667
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: FreezerSmallTrain  Validation Accuracy: 96.98375870069606 Test
Accuracy: 96.99769053117782
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Fungi      Validation Accuracy: 100.0     Test Accuracy: 100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GestureMidAirD1    Validation Accuracy: 96.0      Test Accuracy:
98.07692307692307
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
```

```
Dataset Name: GestureMidAirD2    Validation Accuracy: 100.0     Test Accuracy:
100.0
Best Classification K: 4        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GestureMidAirD3    Validation Accuracy: 98.0      Test Accuracy:
96.15384615384616
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GesturePebbleZ1    Validation Accuracy: 100.0     Test Accuracy:
100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GesturePebbleZ2    Validation Accuracy: 100.0     Test Accuracy:
100.0
Best Classification K: 3        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GunPoint   Validation Accuracy: 96.66666666666667 Test Accuracy:
93.33333333333333
Best Classification K: 3        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GunPointAgeSpan    Validation Accuracy: 100.0     Test Accuracy:
100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GunPointMaleVersusFemale   Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: GunPointOldVersusYoung     Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Ham        Validation Accuracy: 81.25     Test Accuracy:
81.81818181818183
Best Classification K: 2        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: HandOutlines       Validation Accuracy: 89.75609756097562 Test
Accuracy: 86.95652173913044
Best Classification K: 4        Best Imputation K: 1
```

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Haptics   Validation Accuracy: 55.072463768115945   Test Accuracy: 47.14285714285714
Best Classification K: 3   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Herring   Validation Accuracy: 52.63157894736842 Test Accuracy: 60.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: HouseTwenty   Validation Accuracy: 69.56521739130434 Test Accuracy: 76.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: InlineSkate   Validation Accuracy: 48.45360824742268 Test Accuracy: 48.484848484848484
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: InsectEPGRegularTrain   Validation Accuracy: 100.0   Test Accuracy: 100.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: InsectEPGSmallTrain   Validation Accuracy: 100.0   Test Accuracy: 100.0
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: InsectWingbeatSound   Validation Accuracy: 66.96969696969697 Test Accuracy: 60.303030303030305
Best Classification K: 4   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ItalyPowerDemand   Validation Accuracy: 98.17073170731707 Test Accuracy: 97.57575757575758
Best Classification K: 5   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: LargeKitchenAppliances   Validation Accuracy: 57.14285714285714 Test Accuracy: 60.17699115044248
Best Classification K: 1   Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Lightning2   Validation Accuracy: 66.66666666666666 Test

Accuracy: 73.68421052631578
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Lightning7 Validation Accuracy: 47.61904761904761 Test Accuracy: 54.54545454545454
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Mallat Validation Accuracy: 98.05555555555556 Test Accuracy: 98.61111111111111
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Meat Validation Accuracy: 100.0 Test Accuracy: 100.0
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MedicalImages Validation Accuracy: 79.53216374269006 Test Accuracy: 68.02325581395348
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MelbournePedestrian Validation Accuracy: 89.70588235294117 Test Accuracy: 91.75824175824175
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MiddlePhalanxOutlineAgeGroup Validation Accuracy: 68.67469879518072 Test Accuracy: 72.61904761904762
Best Classification K: 5 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MiddlePhalanxOutlineCorrect Validation Accuracy: 84.21052631578947 Test Accuracy: 74.81481481481481
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MiddlePhalanxTW Validation Accuracy: 68.29268292682927 Test Accuracy: 58.333333333333336
Best Classification K: 5 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MixedShapesRegularTrain Validation Accuracy: 93.60730593607306 Test Accuracy: 94.0909090909091
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:

Dataset Name: MixedShapesSmallTrain Validation Accuracy: 93.12169312169311
Test Accuracy: 93.42105263157895
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: MoteStrain Validation Accuracy: 92.10526315789474 Test
Accuracy: 91.66666666666666
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: NonInvasiveFetalECGThorax1 Validation Accuracy:
86.87943262411348 Test Accuracy: 86.21908127208481
Best Classification K: 5 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: NonInvasiveFetalECGThorax2 Validation Accuracy:
91.48936170212765 Test Accuracy: 90.45936395759718
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: OliveOil Validation Accuracy: 88.88888888888889 Test Accuracy:
100.0
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: OSULeaf Validation Accuracy: 60.60606060606061 Test Accuracy:
64.17910447761194
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PhalangesOutlinesCorrect Validation Accuracy: 79.39698492462311
Test Accuracy: 81.25
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Phoneme Validation Accuracy: 11.708860759493671 Test
Accuracy: 14.826498422712934
Best Classification K: 5 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PickupGestureWiimoteZ Validation Accuracy: 100.0 Test
Accuracy: 100.0
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PigAirwayPressure Validation Accuracy: 26.08695652173913 Test
Accuracy: 8.333333333333332
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PigArtPressure Validation Accuracy: 30.434782608695656
Test Accuracy: 22.916666666666664
Best Classification K: 4 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PigCVP Validation Accuracy: 19.565217391304348 Test
Accuracy: 16.666666666666664
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PLAID Validation Accuracy: 100.0 Test Accuracy: 100.0
Best Classification K: 2 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Plane Validation Accuracy: 96.7741935483871 Test Accuracy:
100.0
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: PowerCons Validation Accuracy: 100.0 Test Accuracy:
96.36363636363636
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ProximalPhalanxOutlineAgeGroup Validation Accuracy:
82.22222222222221 Test Accuracy: 80.43478260869566
Best Classification K: 5 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ProximalPhalanxOutlineCorrect Validation Accuracy:
83.45864661654136 Test Accuracy: 87.4074074074074
Best Classification K: 4 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ProximalPhalanxTW Validation Accuracy: 84.44444444444444 Test
Accuracy: 84.78260869565217
Best Classification K: 4 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: RefrigerationDevices Validation Accuracy: 43.75 Test
Accuracy: 50.442477876106196
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Rock Validation Accuracy: 80.0 Test Accuracy:
63.63636363636363

Best Classification K: 2	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ScreenType	Validation Accuracy: 50.0	Test Accuracy: 42.47787610619469
Best Classification K: 5	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SemgHandGenderCh2	Validation Accuracy: 94.81481481481482 Test Accuracy: 91.85185185185185
Best Classification K: 1	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SemgHandMovementCh2	Validation Accuracy: 69.62962962962963 Test Accuracy: 69.62962962962963
Best Classification K: 1	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SemgHandSubjectCh2	Validation Accuracy: 85.92592592592592 Test Accuracy: 82.96296296296296
Best Classification K: 3	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ShakeGestureWiimoteZ	Validation Accuracy: 100.0	Test Accuracy: 100.0
Best Classification K: 1	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ShapeletSim	Validation Accuracy: 46.666666666666664 Test Accuracy: 53.333333333333336
Best Classification K: 5	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ShapesAll	Validation Accuracy: 78.88888888888889 Test Accuracy: 78.88888888888889
Best Classification K: 1	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SmallKitchenAppliances	Validation Accuracy: 36.607142857142854 Test Accuracy: 38.05309734513274
Best Classification K: 1	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SmoothSubspace	Validation Accuracy: 97.77777777777777 Test Accuracy: 95.55555555555556
Best Classification K: 1	Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:

Dataset Name: SonyAIBORobotSurface1 Validation Accuracy: 100.0 Test Accuracy: 98.93617021276596
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SonyAIBORobotSurface2 Validation Accuracy: 99.31972789115646 Test Accuracy: 98.63945578231292
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: StarLightCurves Validation Accuracy: 88.23104693140795 Test Accuracy: 87.66233766233766
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Strawberry Validation Accuracy: 97.27891156462584 Test Accuracy: 97.2972972972973
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SwedishLeaf Validation Accuracy: 79.76190476190477 Test Accuracy: 77.05882352941177
Best Classification K: 4 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Symbols Validation Accuracy: 96.73202614379085 Test Accuracy: 96.73202614379085
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: SyntheticControl Validation Accuracy: 94.44444444444444 Test Accuracy: 91.11111111111111
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ToeSegmentation1 Validation Accuracy: 77.5 Test Accuracy: 65.85365853658537
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: ToeSegmentation2 Validation Accuracy: 83.33333333333334 Test Accuracy: 92.3076923076923
Best Classification K: 4 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Trace Validation Accuracy: 80.0 Test Accuracy: 80.0
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: TwoLeadECG Validation Accuracy: 98.27586206896551 Test Accuracy: 100.0
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: TwoPatterns Validation Accuracy: 98.26666666666667 Test Accuracy: 98.26666666666667
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: UMD Validation Accuracy: 92.5925925925926 Test Accuracy: 92.85714285714286
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: UWaveGestureLibraryAll Validation Accuracy: 96.1251862891207 Test Accuracy: 95.0965824665676
Best Classification K: 2 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: UWaveGestureLibraryX Validation Accuracy: 78.53949329359166 Test Accuracy: 77.56315007429421
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: UWaveGestureLibraryY Validation Accuracy: 73.47242921013412 Test Accuracy: 69.09361069836552
Best Classification K: 3 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: UWaveGestureLibraryZ Validation Accuracy: 70.6408345752608 Test Accuracy: 67.16196136701336
Best Classification K: 5 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Wafer Validation Accuracy: 99.81378026070763 Test Accuracy: 99.62825278810409
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Wine Validation Accuracy: 81.25 Test Accuracy: 94.44444444444444
Best Classification K: 1 Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: WordSynonyms Validation Accuracy: 65.92592592592592 Test Accuracy: 66.42335766423358

```
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Worms      Validation Accuracy: 50.0      Test Accuracy: 45.0
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: WormsTwoClass      Validation Accuracy: 65.78947368421053 Test
Accuracy: 62.5
Best Classification K: 1        Best Imputation K: 1

The Best Classification K value and the Best Imputation K value for Dataset:
Dataset Name: Yoga      Validation Accuracy: 92.72727272727272 Test Accuracy:
92.92929292929293
Best Classification K: 1        Best Imputation K: 1
```

Reporting the final test accuracy for each dataset by using the optimal K found for imputation and the optimal K found for classification.

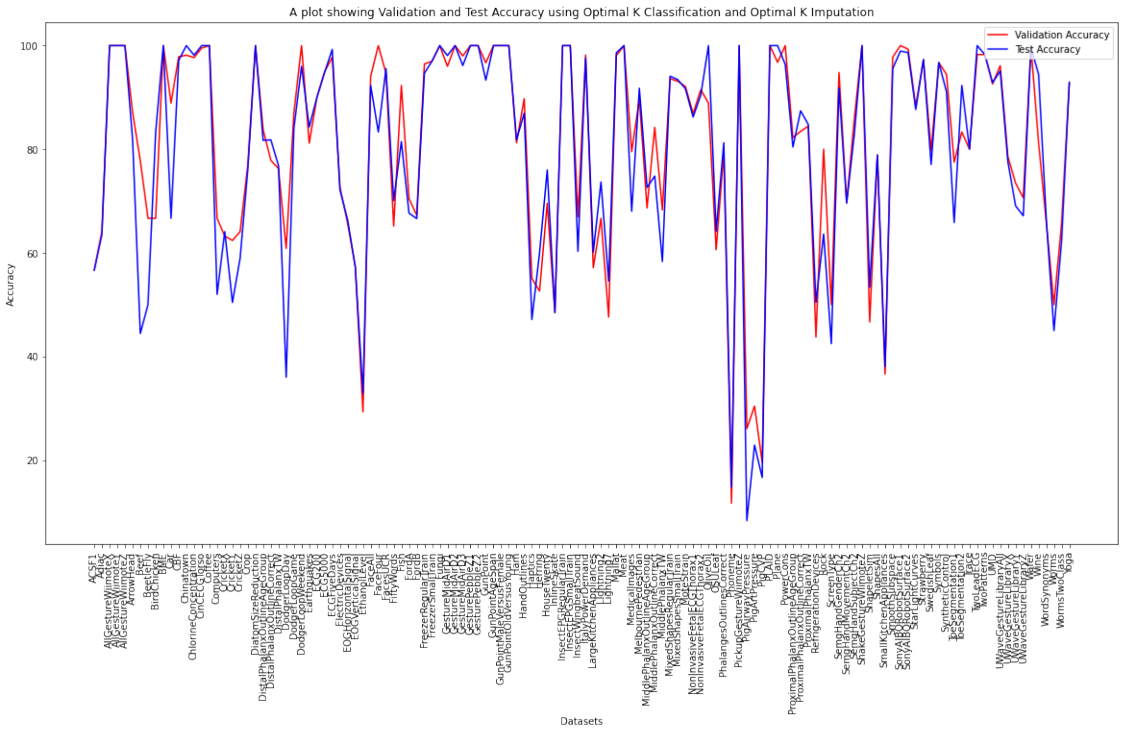
[28]:
```python
#Converting dictionary to the dataframe for displaying the results of Accuracy␣
↪of Validation and Test accross datasets
classification_summary = pd.DataFrame.from_dict(imputation_dataset_accuracy)
classification_summary
```

[28]:
```
                  Name  Classification K  Imputation K  Validation Accuracy  \
0                ACSF1                 1             1            56.666667
1                Adiac                 1             1            64.102564
2     AllGestureWiimoteX               1             1           100.000000
3     AllGestureWiimoteY               1             1           100.000000
4     AllGestureWiimoteZ               1             1           100.000000
..                 …               …           …                 …
123                 Wine               1             1            81.250000
124         WordSynonyms               1             1            65.925926
125                Worms               1             1            50.000000
126        WormsTwoClass               1             1            65.789474
127                 Yoga               1             1            92.727273

     Test Accuracy
0        56.666667
1        63.559322
2       100.000000
3       100.000000
4       100.000000
..             …
123      94.444444
124      66.423358
125      45.000000
```

```
126      62.500000
127      92.929293

[128 rows x 5 columns]
```

Plotting Validation and Test Accuracy in one Plot

[25]:
```
#Creating a Figure
fig = plt.figure(figsize=(20, 10))

#Plotting the Line plot for Validation Accuracy
plt.plot(classification_summary.iloc[:, 0], classification_summary.iloc[:, 3],␣
 ↪c = 'r', label = 'Validation Accuracy')

#Plotting the Line plot for Test Accuracy
plt.plot(classification_summary.iloc[:, 0], classification_summary.iloc[:, 4],␣
 ↪c = 'b', label = 'Test Accuracy')

#Setting the parameters for the graph
plt.xlabel('Datasets')
plt.ylabel('Accuracy')
plt.title('A plot showing Validation and Test Accuracy using Optimal K␣
 ↪Classification and Optimal K Imputation')
plt.xticks(rotation=90)
plt.legend(loc=1)

#Showing the Graph
plt.show()
```



From the above graph we can observe that our validation accuracy and testing accuracy is approximately similar for all datasets.Some Datasets gives really good validation and testing accuracy but some datasets struggle for classifying it correctly. This accuracies can be further improved if we try different and more advanced Hyperparameter tuning methods like hyperband.

0.0.3 Exercise 2: Time Series Classification with Various Distance Measures

Downscaling the number of datasets and taking first 10 datasets for processing

[29]:
```
#Taking the first 10 dataset for this experimentation
ten_datasets = datasets['Dataset'][:10]
```

Listing different Distance measures from Scipy Package

[30]:
```
distance_metrics = ['braycurtis', 'canberra', 'chebyshev', 'cityblock', 'correlation', 'cosine', 'dice', 'euclidean', 'hamming',
                    'jaccard', 'jensenshannon', 'kulsinski', 'mahalanobis', 'matching', 'minkowski', 'rogerstanimoto', 'russellrao',
                    'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule']
```

Creating a Dictionary to store all the statistics of different datasets for this experiment

[31]:
```
dataset_accuracy = {'Name' : [],               #Contains the Name of Datasets
                    'K' : [],                  #Contains the Optimum K value found using Grid Search for each Dataset
                    'Distance Metric' : [],    #Contains the Best Distance Measure found using Grid Search for each Dataset
```

```
                    'Validation Accuracy' : [], #Contains the Best Validation␣
↪accuracy achieved using the Optimum K value and Optimum Distance Measure
                    'Test Accuracy' : []        #Contains the Test Accuracy␣
↪achieved using Optimum K and Distance Measure from Validation set
                   }
```

For each distance measure and for each dataset in the consideration, using the validation samples to tune the parameter K and selecting one best distance measure.

[32]:
```
#Iterating/Enumerating through all the selected datasets
for index ,dataset in enumerate(ten_datasets):

    #Creating a accuracy matrix which will contain the accuracy for each␣
↪combination of k and distance measure
    accuracy_matrix = np.zeros(shape=(len(distance_metrics) ,5))

    #Splitting the dataset into Train, Validation and Test splits
    X_train, Y_train, X_test, Y_test, X_val, Y_val =␣
↪stratify_split_dataframe(dataset, 0.7, 0.15, 0.15)

    #Iterating through the range of different K values, taking K from␣
↪[1,2,3,4,5]
    for k_classification in range(1 ,6):

        #Iterating/Enumerating through different distance measures
        for dist_index ,dist_met in enumerate(distance_metrics):

            #Creating an empty array to store the predicted classes for the␣
↪validation set
            predicted_class_val = []

            #Iterating through all the observations in the Validation set
            for x_val in X_val:

                #Predicting the class for the single validation set observation␣
↪using scipy distance measure
                pred_class = predict_knn_class(X_train, Y_train, x_val,␣
↪k_classification, True, dist_met)

                #Appending the predicted class to the created array
                predicted_class_val.append(pred_class)

            #After predicting the classes for all validation set observation,␣
↪now calculating it overall accuracy
            accuracy = accuracy_knn_classification(predicted_class_val, Y_val.
↪ravel())
```

```
            #Saving the accuracy in our Accuracy Matrix for current Combination
            accuracy_matrix[dist_index,k_classification - 1] = accuracy

    #Finding the combination with the highest accuracy for the current Dataset
    optimum_point = np.where(accuracy_matrix == np.amax(accuracy_matrix))
    optimum_dist_met , optimum_k = optimum_point[0][0] , optimum_point[1][0]

    #Creating an empty array to store the predicted classes for the Test set
    predicted_class_test = []

    #Iterating through all the observations in the Test set
    for x_test in X_test:

        #Predicting the class for the single Test set observation using Optimum␣
↪K and Optimum scipy distance measure
        pred_class = predict_knn_class(X_train, Y_train, x_test, optimum_k + 1,␣
↪True, distance_metrics[optimum_dist_met])

        #Appending the predicted class to the created array
        predicted_class_test.append(pred_class)

    #After predicting the classes for all Test set observation, now calculating␣
↪it overall accuracy
    test_accuracy = accuracy_knn_classification(predicted_class_test, Y_test.
↪ravel())

    #Appending the Optimum combination of K and Distance measure for the␣
↪current Dataset in the dictionary
    dataset_accuracy['Name'].append(datasets['Name'][index])
    dataset_accuracy['K'].append(optimum_k + 1)
    dataset_accuracy['Distance Metric'].
↪append(distance_metrics[optimum_dist_met])
    dataset_accuracy['Validation Accuracy'].
↪append(accuracy_matrix[optimum_dist_met,optimum_k])
    dataset_accuracy['Test Accuracy'].append(test_accuracy)

    #Printing the Optimum Combination for the current Dataset
    print('The Best K value and the Best Distance Metric for Dataset:\nDataset␣
↪Name: {}\t Validation Accuracy: {}\tTest Accuracy: {}\nBest K: {}\tBest␣
↪Distance Metric: {}\n'.format(
          datasets['Name'][index], accuracy_matrix[optimum_dist_met,optimum_k],␣
↪test_accuracy, optimum_k + 1, distance_metrics[optimum_dist_met]))
```

```
The Best K value and the Best Distance Metric for Dataset:
Dataset Name: ACSF1      Validation Accuracy: 83.33333333333334 Test Accuracy:
73.33333333333333
Best K: 1       Best Distance Metric: braycurtis
```

```
The Best K value and the Best Distance Metric for Dataset:
Dataset Name: Adiac      Validation Accuracy: 70.08547008547008 Test Accuracy:
68.64406779661016
Best K: 1       Best Distance Metric: chebyshev

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: AllGestureWiimoteX         Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best K: 1       Best Distance Metric: braycurtis

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: AllGestureWiimoteY         Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best K: 1       Best Distance Metric: braycurtis

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: AllGestureWiimoteZ         Validation Accuracy: 100.0     Test
Accuracy: 100.0
Best K: 1       Best Distance Metric: braycurtis

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: ArrowHead  Validation Accuracy: 90.32258064516128 Test Accuracy:
75.75757575757575
Best K: 1       Best Distance Metric: braycurtis

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: Beef       Validation Accuracy: 88.88888888888889 Test Accuracy:
44.44444444444444
Best K: 4       Best Distance Metric: chebyshev

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: BeetleFly  Validation Accuracy: 66.66666666666666 Test Accuracy:
66.66666666666666
Best K: 1       Best Distance Metric: braycurtis

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: BirdChicken        Validation Accuracy: 83.33333333333334 Test
Accuracy: 83.33333333333334
Best K: 1       Best Distance Metric: chebyshev

The Best K value and the Best Distance Metric for Dataset:
Dataset Name: BME        Validation Accuracy: 100.0     Test Accuracy: 100.0
Best K: 1       Best Distance Metric: canberra
```

Aggregating the results across all 10 datasets, and rank all distance metrics according to the test accuracy

[34]:

```
#Converting dictionary to the dataframe for displaying the results of Accuracy␣
↪of Validation and Test accross datasets
scipy_summary = pd.DataFrame.from_dict(dataset_accuracy)
scipy_summary
```

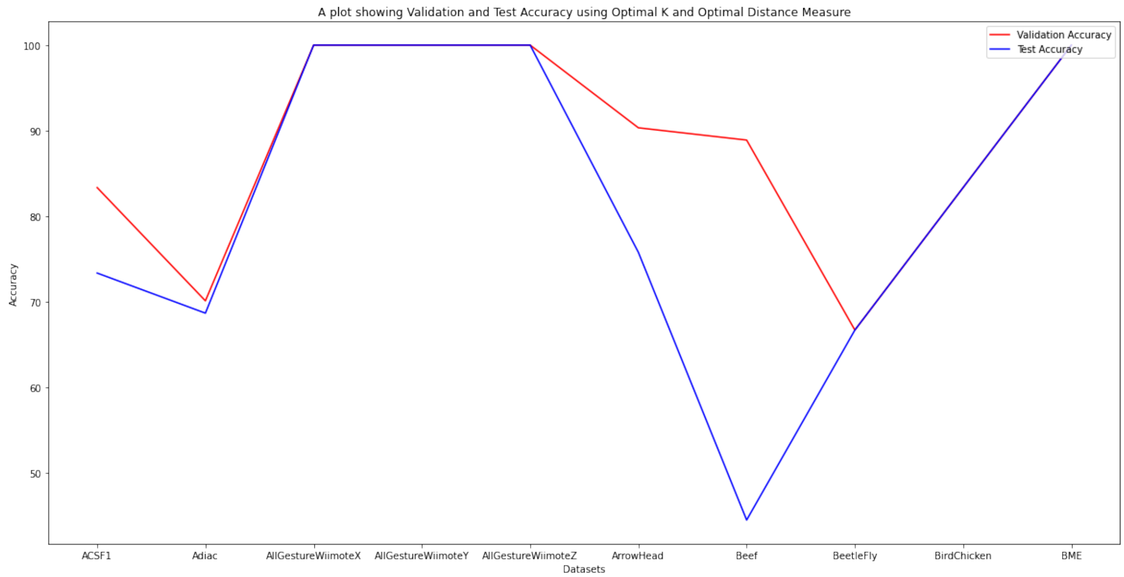
[34]:

	Name	K	Distance Metric	Validation Accuracy	Test Accuracy
0	ACSF1	1	braycurtis	83.333333	73.333333
1	Adiac	1	chebyshev	70.085470	68.644068
2	AllGestureWiimoteX	1	braycurtis	100.000000	100.000000
3	AllGestureWiimoteY	1	braycurtis	100.000000	100.000000
4	AllGestureWiimoteZ	1	braycurtis	100.000000	100.000000
5	ArrowHead	1	braycurtis	90.322581	75.757576
6	Beef	4	chebyshev	88.888889	44.444444
7	BeetleFly	1	braycurtis	66.666667	66.666667
8	BirdChicken	1	chebyshev	83.333333	83.333333
9	BME	1	canberra	100.000000	100.000000

Plotting Validation and Test Accuracy in one Plot

[35]:

```
#Creating a Figure
fig = plt.figure(figsize=(20, 10))

#Plotting the Line plot for Validation Accuracy
plt.plot(scipy_summary.iloc[:, 0], scipy_summary.iloc[:, 3], c = 'r', label =␣
↪'Validation Accuracy')

#Plotting the Line plot for Test Accuracy
plt.plot(scipy_summary.iloc[:, 0], scipy_summary.iloc[:, 4], c = 'b', label =␣
↪'Test Accuracy')

#Setting the parameters for the graph
plt.xlabel('Datasets')
plt.ylabel('Accuracy')
plt.title('A plot showing Validation and Test Accuracy using Optimal K and␣
↪Optimal Distance Measure')
plt.legend(loc=1)

#Showing the Graph
plt.show()
```



From the above graph we can observe that our validation accuracy and testing accuracy is approximately similar for all datasets.Some Datasets gives really good validation and testing accuracy but some datasets struggle for classifying it correctly. This accuracies can be further improved if we try different and more advanced Hyperparameter tuning methods like hyperband.

0.0.4 Exercise 3: Accelerating K-Nearest Neighbour Classifier

Partial Distances/Lower Bounding Function to perform KNN classification using partial distance and Lower bounding

[36]:
```
def partial_distance(trainX,trainY, query, k):

    #Creating an array to store distances of k nearest neighbours
    distance_array_k = np.zeros(shape=(k,), dtype='i,i')

    #Iterating first through the first K rows to find the nearest neighbour to␣
↪the quering row
    for index, row in enumerate(trainX[:k,:]):

        #Appending the calculated euclidean distance to the created distance␣
↪array
        distance_array_k[index] = (index,␣
↪calculate_euclidean_distance_numpy(query ,row))

    #Sorting the distance array based on distance so that we can check for␣
↪better neighbour later
    distance_array_k = sorted(distance_array_k, key=lambda x:x[1])

    #Now Iterating through all rows after first K rows
    for index, row in enumerate(trainX[k:,:]):
```

```
        #Initializing variable for distance and taken features
        d = 0
        m = 1

        #we have to iterate until we have calculate the distance with respect␣
↪to every feature and the calculated distance is smaller than previous
        while m < trainX.shape[1] & d < ␣
↪calculate_euclidean_distance_numpy(query[:m], row[:m]):

            #Adding the distance to the variable
            d += np.sum(np.square(query[:m] , row[:m]))

            #Incrementing the variable m which indicates that in next iteration␣
↪we have to take one more feature for calculating distance
            m += 1

        #Checking if our calculated distance is better than the distance␣
↪calculated before with all features
        if d < calculate_euclidean_distance_numpy(query ,row):

            #If yes, than appending that row to our K nearest neighbour␣
↪distance array
            temp_arr = distance_array_k.copy()
            temp_arr.append((index,d))
            temp_arr = sorted(temp_arr, key=lambda x:x[1])
            distance_array_k = temp_arr[:-1]

    #Getting the classes of these nearest rows
    k_nearest_class = trainY[[i[0] for i in distance_array_k]]

    #The predicted class is then the maximum occuring class
    values, counts = np.unique(k_nearest_class, return_counts=True)
    predicted_class = values[np.argmax(counts)]

    #Returning the Predicted class
    return predicted_class
```

Finding the Dataset with the largest number of Samples

[37]:
```
print('The Dataset with the largest number of Samples is: \n')
print(dataset_info.iloc[dataset_info['Total Rows'].idxmax(),:])
```

```
The Dataset with the largest number of Samples is:

Name              Crop
Total Columns       47
Total Rows       24000
```

```
Total Classes        24
Null Values       False
Name: 20, dtype: object
```

Extracting the biggest Dataset from the list

[38]:
```
biggest_dataset = datasets['Dataset'][datasets['Name'].index('Crop')]
biggest_dataset.head()
```

[38]:
```
   0      1      2      3        4        5        6      7        8  \
0  1  0.240  0.257  0.274  0.25700  0.27700  0.29700  0.317  0.29325
1  1  0.219  0.209  0.199  0.21567  0.23233  0.24900  0.237  0.18500
2  1  0.281  0.240  0.199  0.20000  0.23400  0.26800  0.264  0.32200
3  1  0.125  0.218  0.311  0.26900  0.33000  0.39100  0.237  0.15300
4  1  0.117  0.144  0.171  0.18975  0.20850  0.22725  0.246  0.21300

         9  …       37       38       39     40     41     42      43  \
0  0.26950  …  0.41475  0.38850  0.36225  0.336  0.324  0.312  0.3310
1  0.20300  …  0.36900  0.36200  0.35500  0.348  0.405  0.351  0.2950
2  0.29933  …  0.32300  0.32800  0.33300  0.259  0.308  0.264  0.2750
3  0.14675  …  0.54367  0.43733  0.33100  0.251  0.249  0.277  0.2795
4  0.21700  …  0.40200  0.32500  0.24800  0.171  0.167  0.144  0.1485

      44     45     46
0  0.350  0.333  0.316
1  0.239  0.218  0.197
2  0.286  0.306  0.326
3  0.282  0.277  0.272
4  0.153  0.152  0.151

[5 rows x 47 columns]
```

Doing Grid search on K with Partial Distance KNN and the Biggest Dataset: ‘Crop’

[39]:
```
#Creating a accuracy matrix which will contain the accuracy for each␣
↪combination of K classification and K Imputation
accuracy_arr = np.zeros(shape=(5,))

#Splitting the dataset into Train, Validation and Test sets
Crop_X_train, Crop_Y_train, Crop_X_test, Crop_Y_test, Crop_X_val, Crop_Y_val =␣
↪stratify_split_dataframe(biggest_dataset, 0.7, 0.15, 0.15)

#Iterating through different values of K for KNN Classification, Picking K from␣
↪[1,2,3,4,5]
for k_classification in range(1, 6):

    #Creating an array to store the predicated classes for validation set
    predicted_class_val = []
```

```
    #Iterating through each observation in the Validation set
    for x_val in Crop_X_val:

        #Predicting the class of validation row using the specific K and Train␣
 ↪set
        pred_class = partial_distance(Crop_X_train, Crop_Y_train, x_val,␣
 ↪k_classification)

        #Appending the predicted class to the created array
        predicted_class_val.append(pred_class)

    #Now calculating the total accuracy of the Validation set
    accuracy = accuracy_knn_classification(predicted_class_val, Crop_Y_val.
 ↪ravel())

    #Saving the accuracy in our Accuracy Matrix for current Combination
    accuracy_arr[k_classification - 1] = accuracy

#Finding the combination with the highest accuracy for the current Dataset
optimum_point = np.where(accuracy_arr == np.amax(accuracy_arr))
optimum_k_classification = optimum_point[0][0]

#Creating an empty array to store the predicted classes for the Test set
predicted_class_test = []

#Iterating through all the observations in the Test set
for x_test in Crop_X_test:

    #Predicting the class of Test row using the specific K and Train set
    pred_class = partial_distance(Crop_X_train, Crop_Y_train, x_test,␣
 ↪optimum_k_classification + 1)

    #Appending the predicted class to the created array
    predicted_class_test.append(pred_class)

#After predicting the classes for all Test set observation, now calculating it␣
 ↪overall accuracy
test_accuracy = accuracy_knn_classification(predicted_class_test, Crop_Y_test.
 ↪ravel())

#Printing the Optimum Combination for the current Dataset
print('The Best Classification K value for Dataset:\nDataset Name: Crop\t␣
 ↪Validation Accuracy: {}\tTest Accuracy: {}\nBest Classification K: {}\n'.
 ↪format(
```

```
    accuracy_arr[optimum_k_classification], test_accuracy,␣
↪optimum_k_classification + 1))
```

```
The Best Classification K value for Dataset:
Dataset Name: Crop       Validation Accuracy: 11.833333333333334        Test
Accuracy: 11.805555555555555
Best Classification K: 3
```

From the above experimentation on partial Distances with lower bounding, I observe that it run's really quickly as compared to KNN classification without Partial Distance but the only issue that it faces is its accuracy. The accuracy that it achieves is really low as compared to KNN without Partial Distance (12 vs 76). It think this can be improved if we try to do good hyperparameter tuning for K

Locality Sensitive Hashing Reference Website: https://towardsdatascience.com/locality-sensitive-hashing-for-music-search-f2f1940ace23

Function to calculate the Hash value using Random Projection/ Simhash

[40]:
```python
def simHash(random_vector, observation):
    #For each value of dot product, we will check if the value is greater than␣
↪0 or not and appending it as a string
    bools = ''.join((np.dot(random_vector, observation) > 0).astype('int').
↪astype('str'))

    #Returning the final hash value
    return bools
```

Function to Create and Populate the hashtable with train rows data

[41]:
```python
def create_populate_hashtables(trainX, k):
    #Creating k different Random vectors to denote k different hyperplanes
    random_vectors = np.random.randn(k,len(trainX[0]))

    #Creating and Initializing an empty hash table to store my hash values for␣
↪training rows
    hashtable = []

    ##Iterating through all training rows and storing there hash values in the␣
↪hash table
    for index , row in enumerate(trainX):

        #Calculating hash value of training row using SimHash
        hashvalue = simHash(random_vectors, row)

        #Appending the hash value to the hash table
        hashtable.append(hashvalue)
```

```
    #Returning the hash table and the random vectors
    return hashtable , random_vectors
```

Function to perform KNN classification using Locally Sensitive Hashing

```
[42]: def locally_sensitive_hashing(trainX, trainY, validationX, k):
          #Creating and Populating Hash tables for training data and Creating the␣
       ↪random vectors for Hashing
          hashtable, random_vectors = create_populate_hashtables(trainX, k)

          #Creating and Initializing an empty array to store the predicted classes␣
       ↪for the validation set
          predicted_classes_val = []

          #Iterating through all the rows in Validation set to Predict there classes
          for index, val_row in enumerate(validationX):

              #Calculating the hash value of the validation row
              val_hash = simHash(random_vectors, val_row)

              #Initializing the predicted class variable with None
              val_predicted_class = None

              try:
                  #Searching the hash table to find the index with the same hash␣
       ↪value as the validation row
                  train_index = hashtable.index(val_hash)

                  #Extracting the class of training row which is close to the␣
       ↪validation found using hash table
                  val_predicted_class = trainY[train_index]
              except:

                  #Assiging class to infinity since we were not able find the class␣
       ↪using LSH method
                  val_predicted_class = np.iinfo(np.int32).max

              #Appending the predicted class to the array
              predicted_classes_val.append(val_predicted_class)

          #Returning the predicted class array
          return predicted_classes_val
```

Doing Grid search on K with Locally Sensitive Hashing KNN and the Biggest Dataset: 'Crop'

```
[43]: #Creating a accuracy matrix which will contain the accuracy for each␣
       ↪combination of K classification and K Imputation
      accuracy_arr = np.zeros(shape=(5,))
```

```python
#Splitting the dataset into Train, Validation and Test sets
Crop_X_train, Crop_Y_train, Crop_X_test, Crop_Y_test, Crop_X_val, Crop_Y_val =␣
↪stratify_split_dataframe(biggest_dataset, 0.7, 0.15, 0.15)

#Iterating through different values of K for KNN Classification, Picking K from␣
↪[1,2,3,4,5]
for k_classification in range(1, 6):

    #Creating an array to store the predicated classes for validation set
    predicted_class_val = []

    #Predicting the class of validation row using the specific K and Train set
    predicted_class_val = locally_sensitive_hashing(Crop_X_train, Crop_Y_train,␣
↪Crop_X_val, k_classification)

    #Now calculating the total accuracy of the Validation set
    accuracy = accuracy_knn_classification(predicted_class_val, Crop_Y_val.
↪ravel())

    #Saving the accuracy in our Accuracy Matrix for current Combination
    accuracy_arr[k_classification - 1] = accuracy

#Finding the combination with the highest accuracy for the current Dataset
optimum_point = np.where(accuracy_arr == np.amax(accuracy_arr))
optimum_k_classification = optimum_point[0][0]

#Creating an empty array to store the predicted classes for the Test set
predicted_class_test = []

#Predicting the class of Test row using the specific K and Train set
predicted_class_test = locally_sensitive_hashing(Crop_X_train, Crop_Y_train,␣
↪Crop_X_test, optimum_k_classification + 1)

#After predicting the classes for all Test set observation, now calculating it␣
↪overall accuracy
test_accuracy = accuracy_knn_classification(predicted_class_test, Crop_Y_test.
↪ravel())

#Printing the Optimum Combination for the current Dataset
print('The Best Classification K value for Dataset:\nDataset Name: Crop\t␣
↪Validation Accuracy: {}\tTest Accuracy: {}\nBest Classification K: {}\n'.
↪format(
      accuracy_arr[optimum_k_classification], test_accuracy,␣
↪optimum_k_classification + 1))
```

```
The Best Classification K value for Dataset:
```

```
Dataset Name: Crop       Validation Accuracy: 11.25     Test Accuracy:
4.861111111111112
Best Classification K: 4
```

From the above experimentation on locally sensitive hashing, I observe that it run's really quickly as compared to KNN classification and KNN classification without Partial Distance but the only issue that it faces is its accuracy. The accuracy that it achieves is really low as compared to KNN (11 vs 76). It think this can be improved if we try to do good hyperparameter tuning for K