CSE 4020 - MACHINE LEARNING

Lab 29+30

K-Nearest Neighbour

Submitted by: Alokam Nikhitha(19BCE2555)

KNN

Question:

1. Load the data

2. Initialize K to your chosen number of neighbors

3. For each example in the data

3.1 Calculate the distance between the query example and the current example from the data.

3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels of the selected K entries

7. If regression, return the mean of the K labels

8. If classification, return the mode of the K labels

Dataset Used:

diabetes dataset from https://www.kaggle.com/uciml/pima-indians-diabetesdatabase/version/1

Procedure:

-Using pandas, we first import the dataset into our workspace.

-The independent and dependent attributes to be employed in our classification model must then be decided.

- After that, we divided our data into two sets: training and test.

- After that, we must Feature Scale our dataset.

-Scaling concerns should be accounted for in many attributes.

- Next, we determine the k value for which the classifier has the lowest error.

- After that, we use the best k value to fit our classifier model.

- After that, we construct a variable to record our expected result.

-the X test set's classifier

- Last, we compute our assessment metrics.

Code Snippets and Explanation:

In [1]:
```
#Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
#Importing the dataset
dataset = pd.read_csv("diabetes.csv")
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

Importing the Libraries and Dataset

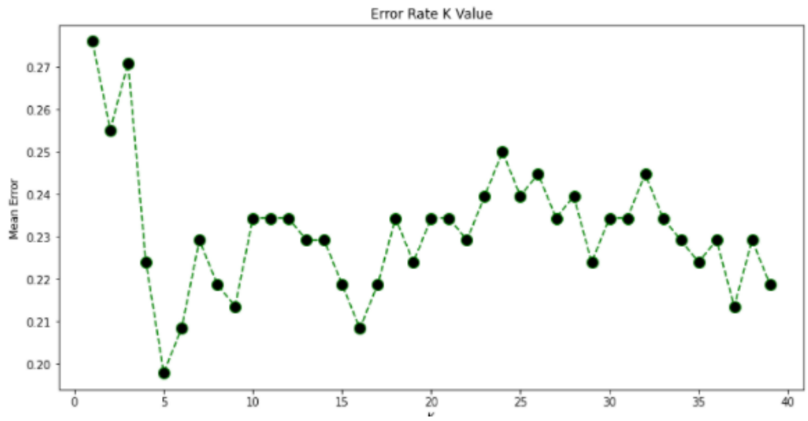
In [3]:
```
#Splitting the datasets into training and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state=0)
```

In [4]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Splitting the dataset into Training and testing sets and Feature scaling

Here we are splitting our dataset into training set and test set with 25% of our dataset values in test set and remaining 75% in training set.

In [5]:
```
from sklearn.neighbors import KNeighborsClassifier
error = []
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='green', linestyle='dashed', marker='o', markerfacecolor='black', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K')
plt.ylabel('Mean Error')
plt.show()
```



We're trying to figure out what the best value for K is. When we choose K as 5, we can observe that the error value is the lowest.

In [6]:
```
#Fitting Classification model to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric='minkowski')
classifier.fit(X_train, y_train)
```

Out[6]:
```
KNeighborsClassifier()
```

We're using training sets to fit our KNN classifier. Because of the previous outcome, we chose K value of 5.

In [7]:
```
#Predicting the X_test result
y_pred = classifier.predict(X_test)
```

On the test set, we're creating a list of predictions based on the classifier's predictions. Our Confusion Matrix has also been created.

In [8]:
```
#Printing the confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

Out[8]:
```
array([[114,  16],
       [ 22,  40]], dtype=int64)
```

```
In [9]: #Printing the Accuracy and Mean Squared Error Value
        from sklearn.metrics import accuracy_score, mean_squared_error
        print("Accuracy value: \t", accuracy_score(y_test, y_pred))
        print("MSE value: \t", mean_squared_error(y_test, y_pred))

Accuracy value:         0.8020833333333334
MSE value:      0.19791666666666666
```

```
In [10]: #Printing the classification report
         from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred))

              precision    recall  f1-score   support

           0       0.84      0.88      0.86       130
           1       0.71      0.65      0.68        62

    accuracy                           0.80       192
   macro avg       0.78      0.76      0.77       192
weighted avg       0.80      0.80      0.80       192
```

We've created our numerous evaluation matrixes here. Precision, recall, and f1 score are all included, as well as accuracy and mean squared error value. Our model has an accuracy of 80 percent and an MSE score of 0.1979.

def knn(data, query, k, distance_fn, choice_fn):

neighbor_distances_and_indices = []

 # 3. For each example in the data

for index, example in enumerate(data):

 # 3.1 Calculate the distance between the query example and t he current

 # example from the data.

distance = distance_fn(example[:-1], query)

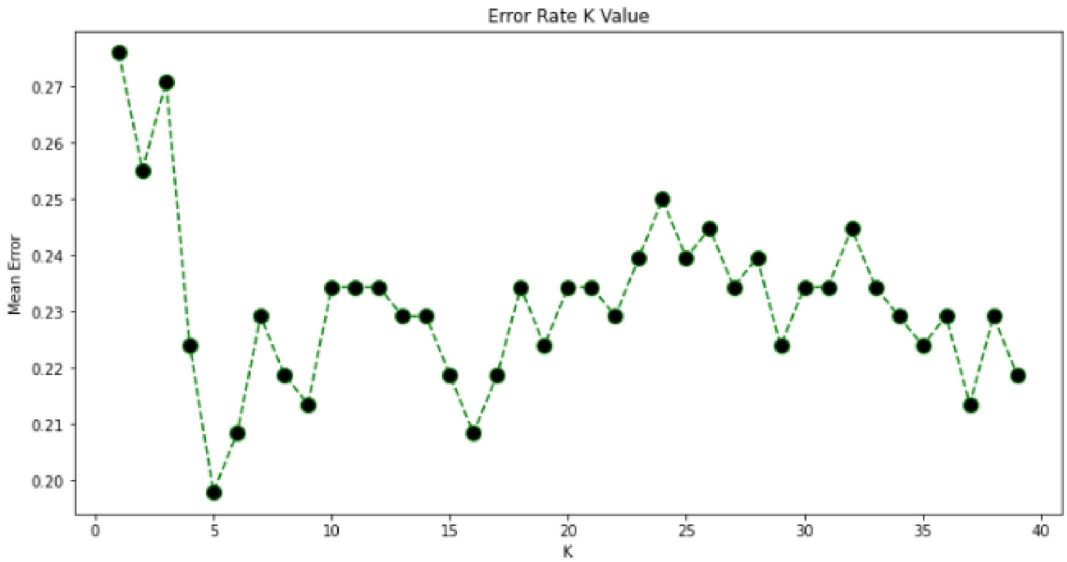
 # 3.2 Add the distance and the index of the example to an orde red collection

neighbor_distances_and_indices.append((distance, index))

```
    # 4. Sort the ordered collection of distances and indices from
    # smallest to largest (in ascending order) by the distances
    sorted_neighbor_distances_and_indices = sorted(neighbor_dist
ances_and_indices)

    # 5. Pick the first K entries from the sorted collection
    k_nearest_distances_and_indices = sorted_neighbor_distances_
and_indices[:k]

    # 6. Get the labels of the selected K entries
    k_nearest_labels = [data[i][1] for distance, i in k_nearest_distanc
es_and_indices]

    # 7. If regression (choice_fn = mean), return the average of the K
 labels
    # 8. If classification (choice_fn = mode), return the mode of the K
 labels
    return k_nearest_distances_and_indices , choice_fn(k_nearest_l
abels)

#function to calculate the mean used in case of regression
def mean(labels):
    return sum(labels) / len(labels)

#function to calculate the mode used in case of classification
def mode(labels):
    return Counter(labels).most_common(1)[0][0]
```

```
#function to calculate the distance between two data points
def euclidean_distance(point1, point2):
    sum_squared_distance = 0
    for i in range(len(point1)):
        sum_squared_distance += math.pow(point1[i] - point2[i], 2)
    return math.sqrt(sum_squared_distance)
```

Result and Conclusion:



	precision	recall	f1-score	support
0	0.84	0.88	0.86	130
1	0.71	0.65	0.68	62
accuracy			0.80	192
macro avg	0.78	0.76	0.77	192
weighted avg	0.80	0.80	0.80	192

Modal Accuracy:80%