# 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.

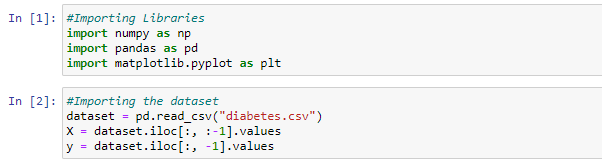
- After that, we use the best k value to fit our classifier model.

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

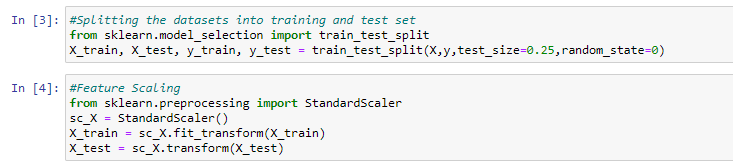
-the X test set's classifier

- Last, we compute our assessment metrics.

**Code Snippets and Explanation:**

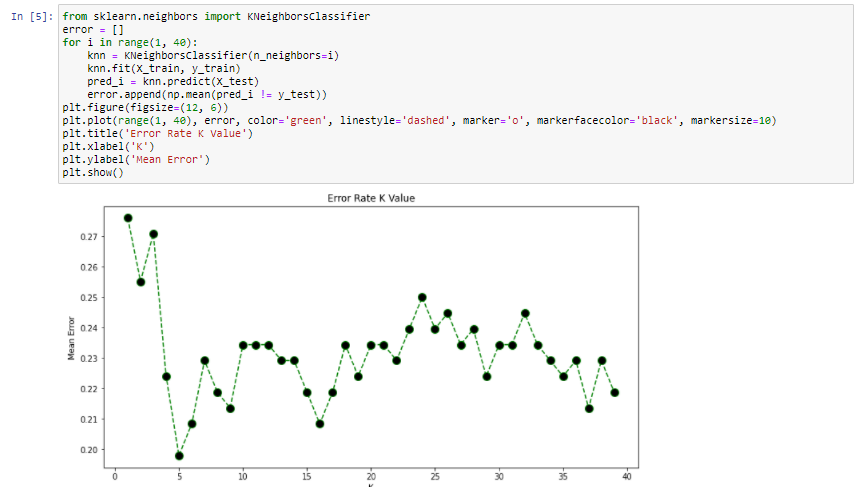


Importing the Libraries and Dataset

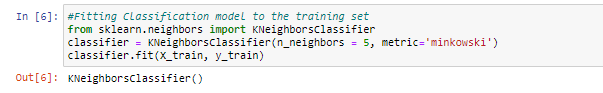


Splitting the dataset into Training and testing sets and Feature scaling

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.



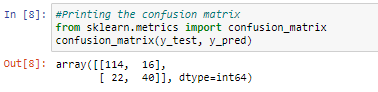
We're trying to figure out what the best value for K is. When we choose K as 5, we can observe that the error value is the lowest.

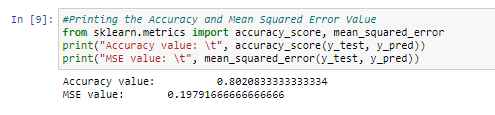


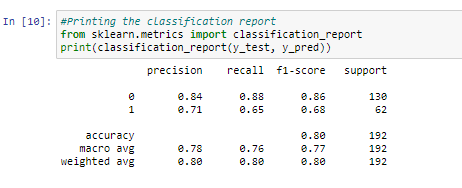
We're using training sets to fit our KNN classifier. Because of the previous outcome, we chose K value of 5.



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







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 the current

        # example from the data.

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

        # 3.2 Add the distance and the index of the example to an ordered 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\_distances\_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\_distances\_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\_labels)

#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):

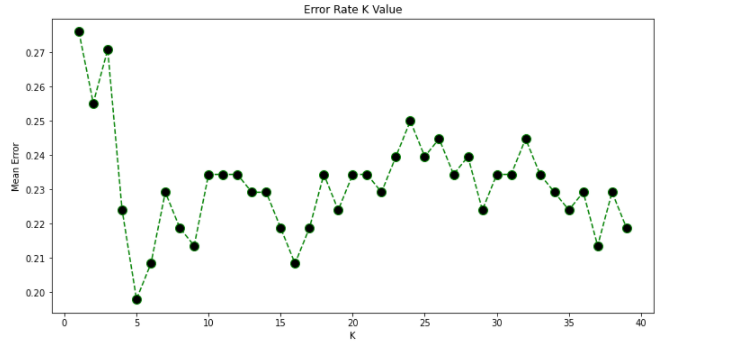
    sum\_squared\_distance = 0

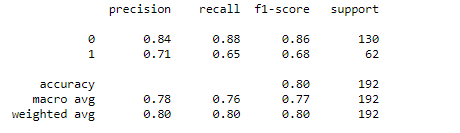
    for i in range(len(point1)):

        sum\_squared\_distance += math.pow(point1[i] - point2[i], 2)

    return math.sqrt(sum\_squared\_distance)

**Result and Conclusion:**

****

****

**Modal Accuracy:80%**