CS-512

Project Report

Topic : K-Nearest Neighbours

Project Group 52

Group Members:

Manasvini Nittala (mn777)

Sahithi Reddy Sakinala (ss4362)

Jahnavi Manchala (jm2658)

INDEX

1. Introduction to KNN

K-Nearest Neighbors Algorithm is one of the simple, easy-to-implement, and yet effective supervised machine learning algorithms. We can use it in any classification (This or That) or regression (How much of This or That) scenario. It predicts responses for new data (testing data) based upon its similarity with other known data (training) samples. It assumes that data with similar traits sit together.

It is called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

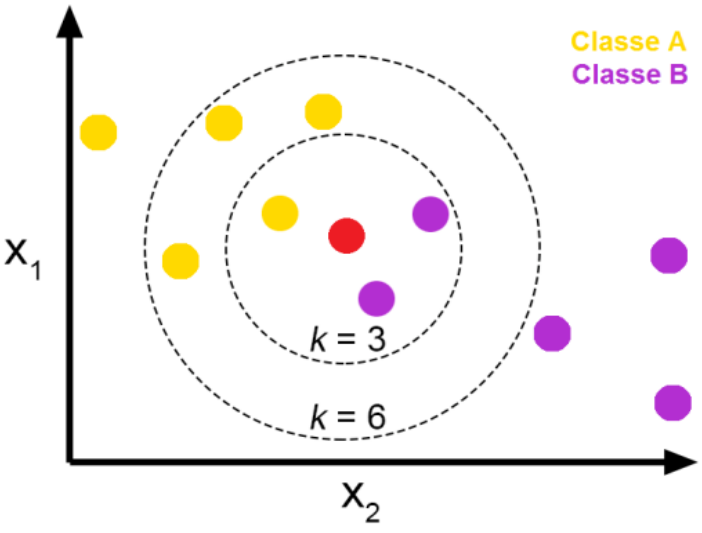
There are many ways to calculate the distance in KNN:

- Manhattan Method
- Euclidean Method
- Minkowski Method
- mahalanobis distance, etc

Here we will be using Euclidean distance to calculate the distance of a new data point from each point in our training dataset.

Euclidean Distance = (x2−x1)2+(y2−y1)2

Let's say that we want to predict the class label of query point(Red color star shown in the below attached image)



Case-1:

If we consider that the k value in KNN is 3 as shown in the above circle the query point belongs to class B as class B data points are more in the k(3) neighbors

Case-2:

In the same way as explained above if the value of k is 6(6 neighbors) is considered the query point belongs to class A as the majority of k neighbors belong to class A.

3. How does the KNN classifier work?

A query point in KNN is classified to which class it belongs based on the majority vote in the chosen value of K(nearest neighbors). As told earlier the neighbors are known from the distance matrices used, majorly Euclidean distance which is also known as L2 norm is used. So we can make KNN work through data points that are not given and instead similarity matrices or distance matrices are given.

Now comes the interesting question of how we could choose the right value of K???

This is the most important thing to be known inorder to make KNN work perfectly i.e If we observe the above mentioned cases in the intuition we could observe that the query point belongs to class B when k=3 and class A when k=6. Then how could we say it belongs to a particular class mainly depends on the right K value. Now let's see how we could determine the right value of K.

We would be splitting our dataset given to train and test dataset basically but here to determine the right value of K we would split the dataset given in to train,cross validation, test datasets(preferably 60%,20%,20% respectively).After a model is trained using the train dataset we would be using cross validation dataset in order to test the model using different values of K and cross validation accuracy is obtained for every value of K and the right value of K is the one with highest accuracy.

Steps to be followed in KNN algorithm are:

1)Split the dataset into train,cross validation,test datasets.

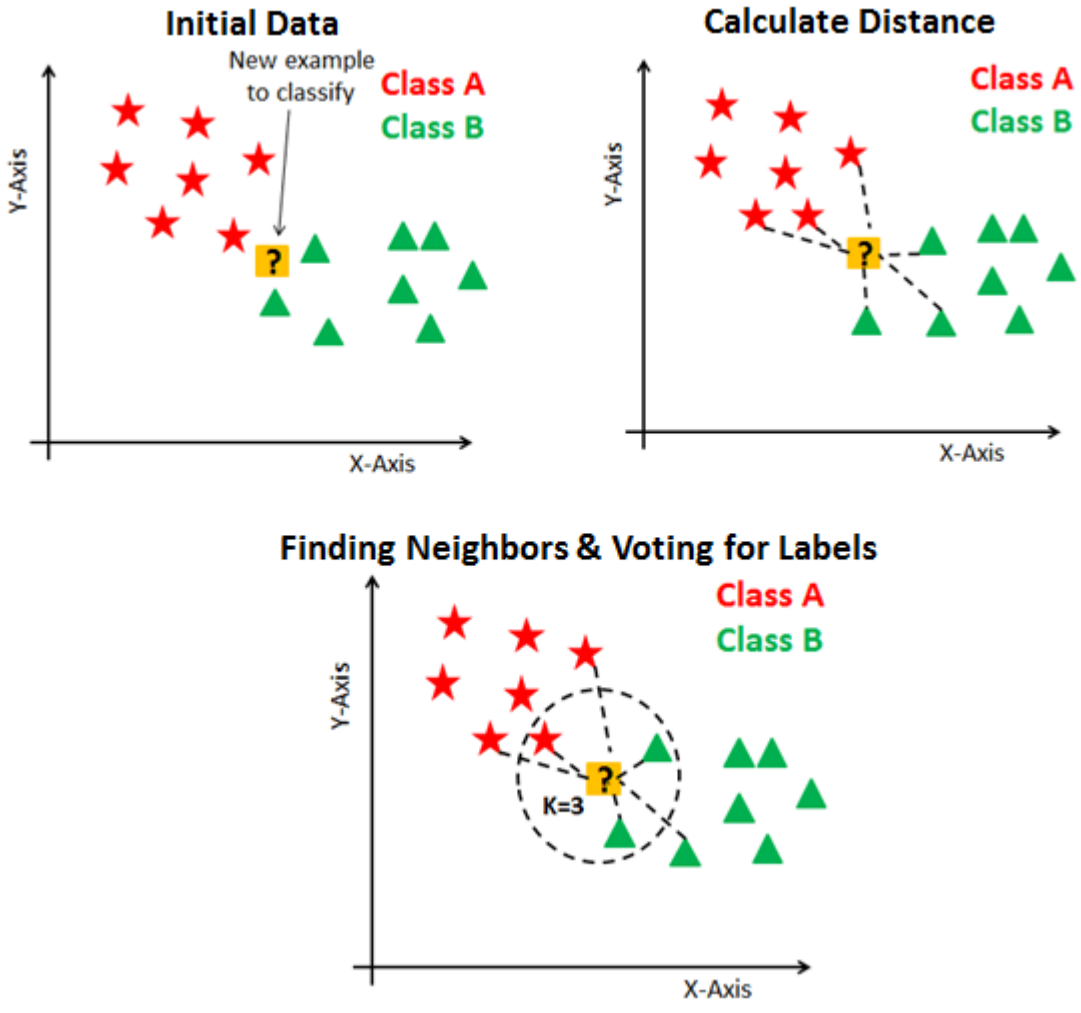
2)Choose the distance metric that is to be used.

3)Make a model using train dataset

4)Determine the right value of K using cross validation dataset accuracy.

5)For the query point given, calculate the distance from all data points and sort them in ascending order and apply the value of K obtained i.e k nearest neighbors.

6)Assign the query point to the class based on the majority vote of its k neighbors.



2. Time and space complexity:

Training Phase: In the training phase we need to store all the data points.The space complexity would be O(n*d) where n represents the number of datapoints and d represents the number of features that determine each datapoint.The time that it takes to calculate the n data points with d features would would be O(n*d).

Testing Phase: Testing phase or Run time complexity is O(n*k*d) where k represents the number of nearest neighbors that needs to be considered.

3. Pseudo Code:

KNN

```
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))

# k_nearest_neighbors class
class KNN:

    # constructor for k nearest neighbors

    def __init__(self, k=3):
        self.y_train = None
        self.X_train = None
        self.k = k

    # fit method for machine learning

    def fit(self, X, y):
        """
        X: training samples
        y: training lables --target
        """
        self.X_train = X
        self.y_train = y
```

```
    # predict method
    def predict(self, X):
        predicted_labels = [self._helper(x) for x in X]
        return np.array(predicted_labels)

    # helper method
    def _helper(self, x):
        # compute distances
        distances = [euclidean_distance(x, x_train) for x_train in
self.X_train]

        # get k nearest samples,labels
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in k_indices]

        # majority vote, most common class label
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
```

KNN_Test

```
iris = datasets.load_iris()
X, y = iris.data, iris.target
X_train:training samples
X_test: test samples
y_train: training lables
y_test : test lables
"""
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# classifier clf
clf = KNN.KNN(3)

# fit method training data
clf.fit(X_train, y_train)

#  predict test sample
predictions = clf.predict(X_test)

# test accuracy
```

```
accuracy = np.sum(predictions == y_test) / len(y_test)
print(f"My KNN Model Accuracy is :  {accuracy:.3}")

#  sklearn KNN
s = KNeighborsClassifier(n_neighbors=3)
s.fit(X_train, y_train)
predict = s.predict(X_test)


def tune_parameter(X_train, y_train, X_test, y_test, k_num):
    accuracy = []
    # y_test = y_testtt
    for i in range(1, k_num):
        ms = KNeighborsClassifier(n_neighbors=i)
        # model_sklearn.fit(X, y)
        ms.fit(X_train, y_train)
        y_pred = ms.predict(X_test)
        accuracy.append(accuracy_score(y_test, y_pred))
    return np.array(accuracy)


accuracy_1 = tune_parameter(X_train, y_train, X_test, y_test, 100)

error = [1 - x for x in error_rate]
optimal_n = error.index(min(error))

m = KNeighborsClassifier(n_neighbors=optimal_n)
m.fit(X_train, y_train)
y_pred1 = m.predict(X_test)
accuracy_score(y_test, y_pred1)
acc = accuracy_score(y_test, y_pred1) * 100
```

4. Advantages of KNN:

No Training Period: KNN is called **Lazy Learner (Instance based learning).** It does not learn anything in the training period. It does not derive any discriminative function from the training data. In other words, there is no training period for it. It stores the training dataset and learns from it only at the time of making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g. SVM, Linear Regression etc.

Since the KNN algorithm requires no training before making predictions, **new data can be added seamlessly** which will not impact the accuracy of the algorithm.

KNN is very easy to implement. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

- No assumptions about data distribution, useful in real world application
- Simple algorithm to explain and understand
- It can use for both classification and regression

5. Disadvantages of KNN:

Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.

Does not work well with high dimensions: The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

Need feature scaling: We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

Sensitive to noisy data, missing values and outliers: KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

- Computationally expensive, because the algorithm stores all of the training data
- High memory requirement, again, it stores all of the training data
- Prediction stage might be slow (with a big N).

6. Graphs

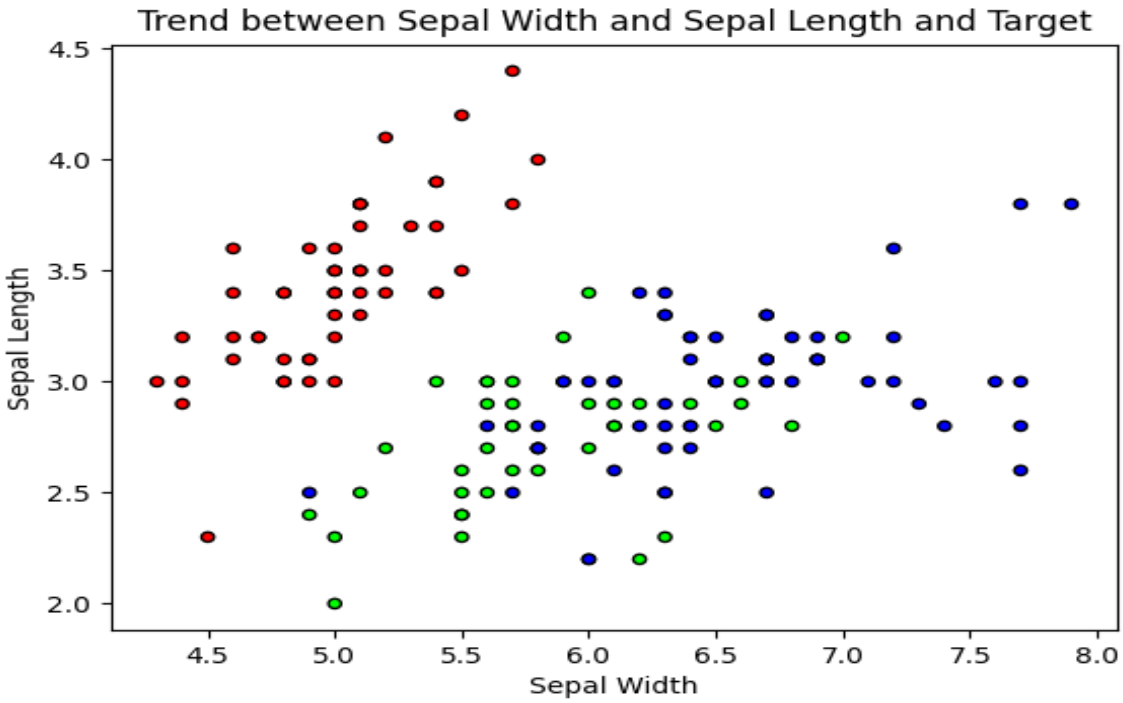


Fig: Trend between sepal width and sepal length and target

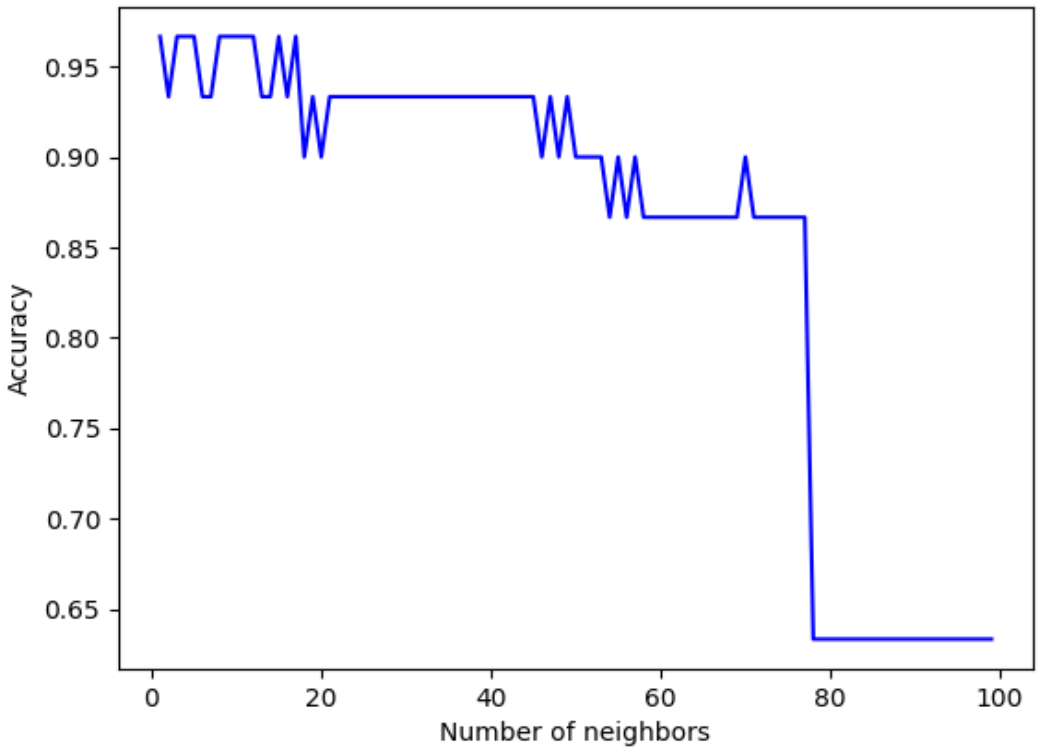


Fig: Trend between Accuracy and Number of neighbors

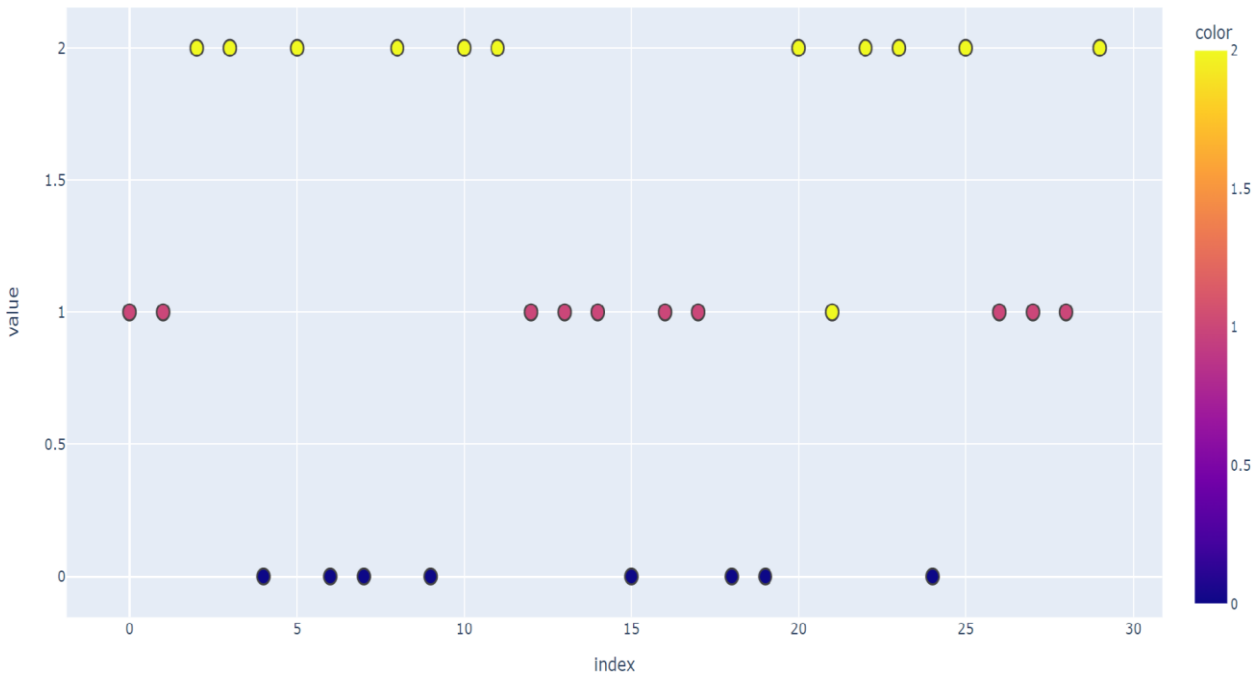


Fig: Trend between predictions and Target class

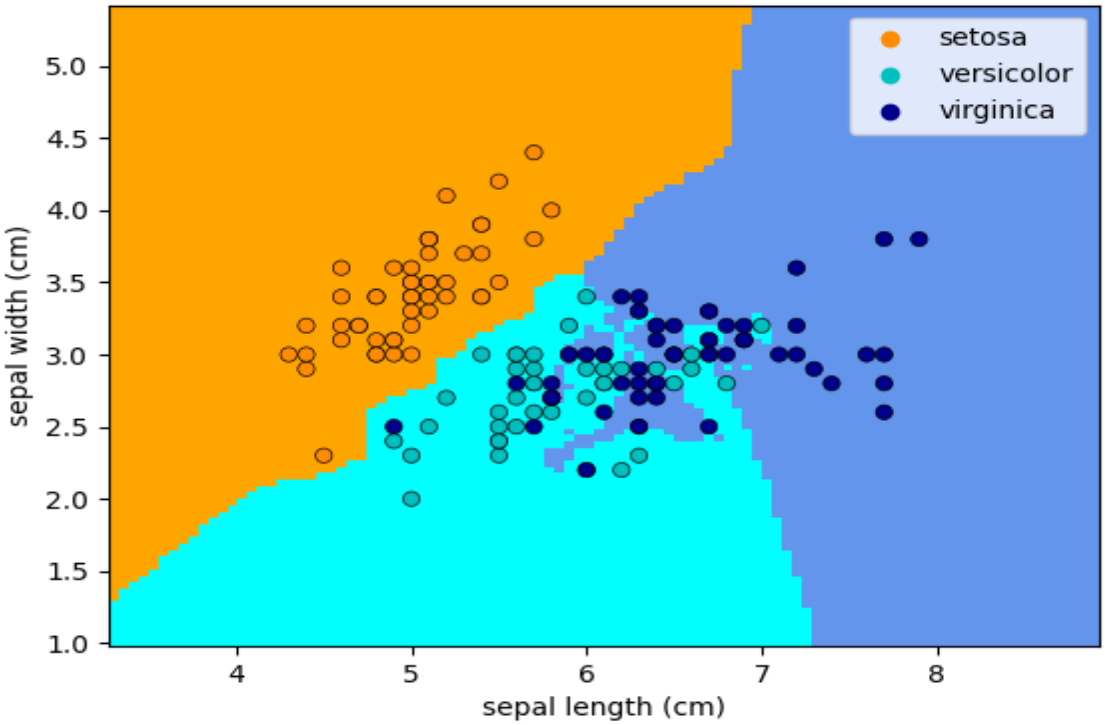


Fig: Trend between sepal width and sepal length in cm

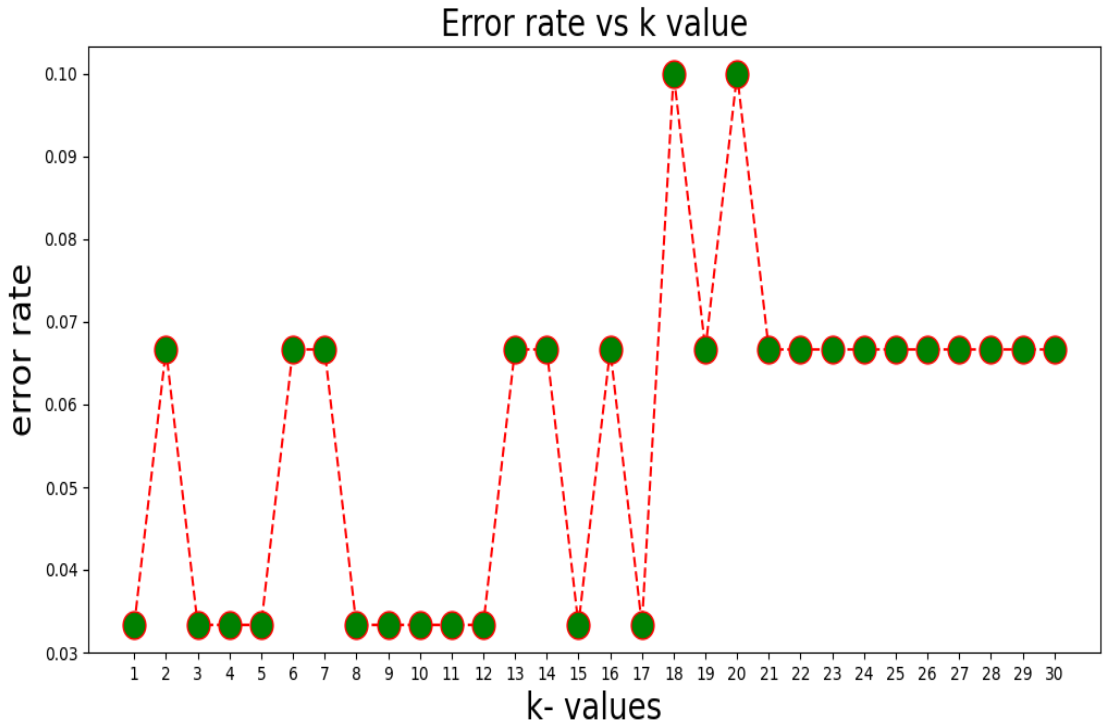


Fig: Trend between error rate and k-values

7. Code

- We have two classes KNN and KNN_test.

KNN:

```
import numpy as np
from collections import Counter


# euclidean distance
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))


# k_nearest_neighbors class

class KNN:

    # constructor for k nearest neighbors

    def __init__(self, k=3):
        self.y_train = None
        self.X_train = None
        self.k = k

    # fit method for machine learning

    def fit(self, X, y):
        """
        X: training samples
        y: training lables --target
        """
        self.X_train = X
        self.y_train = y

    # predict method
    def predict(self, X):
        predicted_labels = [self._helper(x) for x in X]
        return np.array(predicted_labels)
```

```
    # helper method
    def _helper(self, x):
        # compute distances
        distances = [euclidean_distance(x, x_train) for
                     x_train in self.X_train]

        # get k nearest samples,labels
        k_indices = np.argsort(distances)[:self.k]
        k_nearest_labels = [self.y_train[i] for i in
                            k_indices]

        # majority vote, most common class label
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]
```

KNN_TEST:

```
import numpy as np
import matplotlib.pyplot as plt
import KNN
from matplotlib.colors import ListedColormap
import plotly.express as px
import seaborn as sns
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

# Create color maps
cmap = ListedColormap(['#ff0000', '#00ff00', '#0000ff'])
cmap_light = ListedColormap(["orange", "cyan", "cornflowerblue"])
cmap_bold = ["darkorange", "c", "darkblue"]

iris = datasets.load_iris()
X, y = iris.data, iris.target
# print(y)
Z = iris.get('feature_names')
# print(Z)
A = iris.get('target_names')
```

```
# print(A)

"""
X_train:training samples
X_test: test samples
y_train: training lables
y_test : test lables
"""
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)

# shape of input--rows & columns - samples & lables
print(X_train.shape)
# print(X_train[0])  # features of first row

# shape of output-only one colum
print(y_train.shape)
# print(y_train)

# shape of input--rows & columns - samples & lables
print(X_test.shape)
# print(X_test[0])  # features of first row

# shape of output-only one colum
print(y_test.shape)
# print("Test", y_test)
plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap, edgecolor='k', s=20)
plt.title("Trend between Sepal Width and Sepal Length and Target")
plt.xlabel("Sepal Width")
plt.ylabel("Sepal Length")
plt.show()

# classifier clf
clf = KNN.KNN(3)

# fit method training data
clf.fit(X_train, y_train)

#  predict test sample
predictions = clf.predict(X_test)

# test accuracy
```

```
accuracy = np.sum(predictions == y_test) / len(y_test)
print(f"My KNN Model Accuracy is :  {accuracy:.3}")

#  sklearn KNN
s = KNeighborsClassifier(n_neighbors=3)
s.fit(X_train, y_train)
predict = s.predict(X_test)
print(f"Scikit learn KNN classifier accuracy:
{accuracy_score(y_test, predict):.3}")


def tune_parameter(X_train, y_train, X_test, y_test, k_num):
    accuracy = []
    # y_test = y_testtt
    for i in range(1, k_num):
        ms = KNeighborsClassifier(n_neighbors=i)
        # model_sklearn.fit(X, y)
        ms.fit(X_train, y_train)
        y_pred = ms.predict(X_test)
        accuracy.append(accuracy_score(y_test, y_pred))
    return np.array(accuracy)


accuracy_1 = tune_parameter(X_train, y_train, X_test, y_test, 100)
# print("ACC",accuracy_1)

x = np.arange(1, 100)
y = accuracy_1
plt.xlabel("Number of neighbors")
plt.ylabel("Accuracy")
plt.plot(x, y, color="blue")
plt.show()

fig = px.scatter(y_test, color=predictions)
fig.update_traces(marker_size=12, marker_line_width=1.5)
fig.update_layout(legend_orientation='h')
fig.show()

M = iris.data[:, :2]
N = iris.target
msk = KNeighborsClassifier(n_neighbors=3)
msk.fit(M, N)
```

```
_, ax = plt.subplots()
DecisionBoundaryDisplay.from_estimator(
    msk,
    M,
    cmap=cmap_light,
    ax=ax,
    response_method="predict",
    plot_method="pcolormesh",
    xlabel=iris.feature_names[0],
    ylabel=iris.feature_names[1],
    shading="auto",
)

sns.scatterplot(
    x=M[:, 0],
    y=M[:, 1],
    hue=iris.target_names[N],
    palette=cmap_bold,
    alpha=1.0,
    edgecolor="black",
)

plt.show()

error_rate = []  # list that will store the average error rate value
of k
for i in range(1, 31):  # Took the range of k from 1 to 50
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(X_train, y_train)
    predict_i = clf.predict(X_test)
    error_rate.append(np.mean(predict_i != y_test))
error_rate

# plotting the error rate vs k graph
plt.figure(figsize=(12, 6))
plt.plot(range(1, 31), error_rate, marker="o",
markerfacecolor="green",
         linestyle="dashed", color="red", markersize=15)
plt.title("Error rate vs k value", fontsize=20)
plt.xlabel("k- values", fontsize=20)
plt.ylabel("error rate", fontsize=20)
plt.xticks(range(1, 31))
plt.show()
```

```
error = [1 - x for x in error_rate]
optimal_n = error.index(min(error))
print("The optimal K value from the graph is :", optimal_n)

m = KNeighborsClassifier(n_neighbors=optimal_n)
m.fit(X_train, y_train)
y_pred1 = m.predict(X_test)
accuracy_score(y_test, y_pred1)
acc = accuracy_score(y_test, y_pred1) * 100
print(f"Scikit learn KNN classifier accuracy:
{accuracy_score(y_test, y_pred1):.3}")
print("The accuracy for optimal k = {0} is {1}".format(optimal_n,
acc))
```