

# K-Nearest Neighbours

The K-Nearest Neighbours (KNN) algorithm is a simple and intuitive way to make predictions based on the data around a particular point. This algorithm is based on Supervised Learning.

## Basic Idea:

Imagine you want to figure out what type of fruit a new object is, based on a bunch of fruits you've already classified. KNN looks at the closest "neighbours" to your new object and uses their characteristics to make a guess.

## How It Works:

1. **Choose K:** First, you decide how many neighbours (K) you want to consider. For example, if  $K=3$ , you'll look at the three closest data points.
2. **Distance Measurement:** For the new object, you measure the distance to all the other objects in your dataset. This could be how similar they are in terms of size, colour, or other features.
3. **Find Neighbours:** Once you have all the distances, you find the K nearest neighbours (the ones closest to your new object).
4. **Vote:** Finally, you look at what type of fruits those K neighbours are. If most of them are apples, you would classify your new object as an apple too.

## Example:

Suppose you have a dataset of different fruits:

- Apples (red, round)
- Oranges (orange, round)
- Bananas (yellow, long)

You receive a new fruit that's yellow and somewhat curved.

1. **K:** You decide  $K=3$ .
2. **Distances:** You measure how close this fruit is to the apples, oranges, and bananas based on colour and shape.
3. **Neighbours:** You find that the three closest fruits are two bananas and one apple.
4. **Vote:** Since two out of three are bananas, you classify the new fruit as a banana.

## Summary:

KNN is like asking your friends for their opinions on what something is based on similar things they know. The more "friends" (neighbours) you ask, the better your guess will likely be!

## Use Cases:

1. **Image Recognition:** KNN can classify images based on pixel similarity, helping to identify objects, faces, or handwritten digits.
2. **Recommendation Systems:** It can suggest products by finding similar users or items based on preferences or past behaviour.
3. **Medical Diagnosis:** KNN can be used to classify patient symptoms or medical records to predict diseases based on historical data.
4. **Credit Scoring:** Financial institutions can use KNN to assess the creditworthiness of applicants by comparing them to previous clients.
5. **Anomaly Detection:** KNN can help identify outliers in data, useful in fraud detection or network security.

#### **Benefits:**

1. **Simplicity:** KNN is easy to understand and implement, making it a great choice for beginners in machine learning.
2. **No Training Phase:** KNN doesn't require a training phase, as it works directly with the dataset, which can save time.
3. **Flexibility:** It can be used for both classification and regression tasks.
4. **Effective for Small Datasets:** KNN can perform well with smaller datasets where patterns are easy to identify.

#### **Disadvantages:**

1. **Computationally Intensive:** KNN can be slow, especially with large datasets, since it calculates distances to all points for every prediction.
2. **Sensitive to Noise:** The presence of outliers or irrelevant features can affect the accuracy of the predictions.
3. **Choice of K:** Selecting the right value for K can be tricky; too small may lead to noise, while too large can smooth out the distinctions.
4. **Curse of Dimensionality:** As the number of features increases, the distance between points becomes less meaningful, making it harder for KNN to find nearest neighbours.

#### **Summary:**

KNN is a versatile and user-friendly algorithm, but its performance can suffer with larger datasets or when noise is present. Balancing its simplicity with potential computational challenges is key when considering its application.



#### **To begin with the Lab:**

1. For this lab, we are going to use the same dataset Social Network Ads. You need to create a new Python Kernel.

jupyter Quit Logout

Files Running Clusters

Select items to perform actions on them. Upload New ↺

	Name	Last Modified	File size
0	/ Modelling		
	..	seconds ago	
<input type="checkbox"/>	Decision Tree Algorithm.ipynb	2 days ago	1.07 MB
<input type="checkbox"/>	KNN Classifiers.ipynb	Running seconds ago	72 B
<input type="checkbox"/>	Linear-Regression.ipynb	3 days ago	109 kB
<input type="checkbox"/>	Logistic Regression.ipynb	2 days ago	227 kB
<input type="checkbox"/>	Advertising.csv	3 days ago	4.76 kB
<input type="checkbox"/>	banking.csv	2 days ago	4.88 MB
<input type="checkbox"/>	Social_Network_Ads.csv	2 days ago	4.9 kB

- Here, we imported some important libraries and then loaded our dataset as a data frame. Then displayed some entries from our dataset.

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
In [2]: data = pd.read_csv("Social_Network_Ads.csv")
data.head()
```

Out[2]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0

- Preparing the Data:** The code starts by separating the features (X) from the target variable (y) in the dataset. The target variable is whether someone made a purchase or not.
- Splitting the Data:** It then splits the data into two parts: one for training the model (80%) and one for testing it (20%). This helps in evaluating how well the model performs on unseen data.
- Scaling the Features:** The features are standardized using a method called StandardScaler. This means that the values are adjusted to have a mean of 0 and a standard deviation of 1, which helps improve the model's performance.

6. **KNN Model Training and Error Calculation:** The code sets up a loop to train a K-Nearest Neighbors (KNN) model with different values of K (from 1 to 14). For each value of K, it:
  7. Trains the model using the training data.
  8. Makes predictions on both the training and testing data.
  9. Calculates the error rate (how often the model makes mistakes) for both the training and testing sets.
10. **Plotting the Results:** Finally, it plots the error rates for both the training and testing data against the different K values. This helps visualize how the model's performance changes with different numbers of neighbors.
11. Overall, the code helps determine the best K value for the KNN model by analyzing its performance based on error rates.

```
In [3]: x = data.drop(columns='Purchased')
        y = data['Purchased']

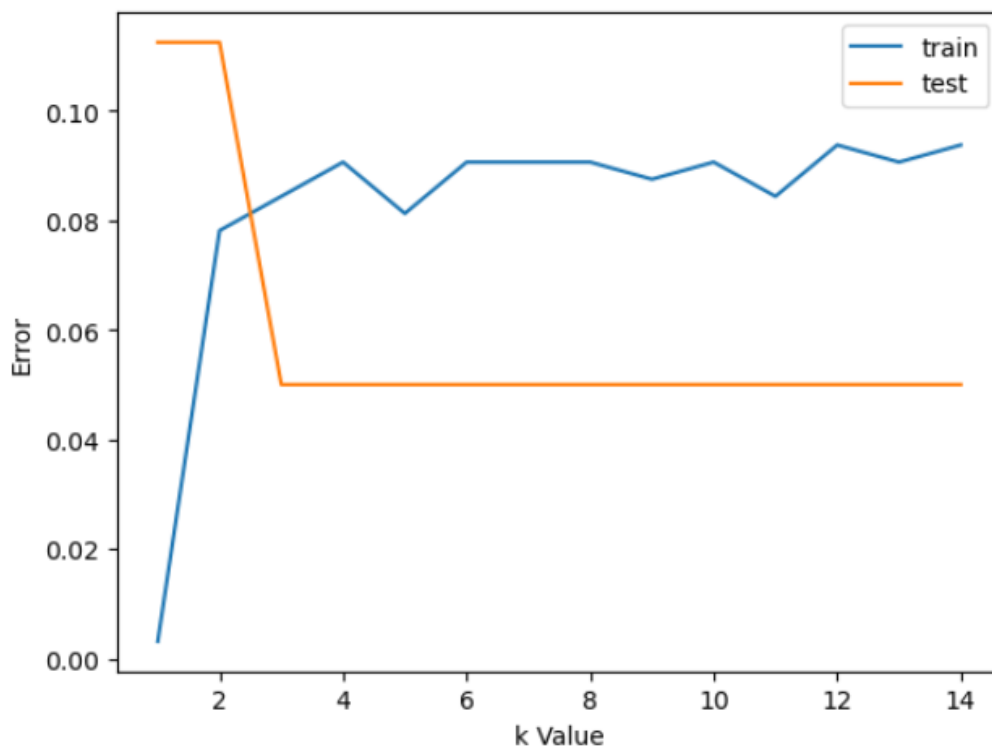
        from sklearn.model_selection import train_test_split
        X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2, random_state=0)

        from sklearn.preprocessing import StandardScaler
        ss = StandardScaler()
        X_train_transform = ss.fit_transform(X_train)
        X_test_transform = ss.transform(X_test)
```

```
In [4]: from sklearn.neighbors import KNeighborsClassifier

train_error= []
test_error= []
for k in range(1,15):
    knn= KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_transform,y_train)
    y_pred_train= knn.predict(X_train_transform)
    train_error.append(np.mean(y_train!= y_pred_train))
    y_pred_test= knn.predict(X_test_transform)
    test_error.append(np.mean(y_test!= y_pred_test))
# plt.figure(figsize=(10,5))
plt.plot(range(1,15),train_error,label="train")
plt.plot(range(1,15),test_error,label="test")
plt.xlabel('k Value')
plt.ylabel('Error')
plt.legend()
```

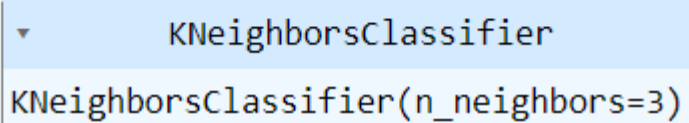
Out[4]: <matplotlib.legend.Legend at 0x23f30b7a5d0>



12. **Setting K Value:** It specifies that the K-Nearest Neighbors (KNN) model will use 3 neighbors (though the comment above suggests trying K=11, this line sets it to K=3).
13. **Training the Model:** The KNN model is trained using the transformed training data (X\_train\_transform) and the target variable (y\_train). This means the model learns how to classify data based on the training examples.
14. **Making Predictions:** The model then makes predictions on the transformed test data (X\_test\_transform).
15. **Calculating Accuracy:** Finally, it uses the accuracy\_score function to compare the predicted values (y\_pred) with the actual values (y\_test) from the test set. This tells you how many predictions the model got right.

16. Overall, this code helps you assess how well the KNN model (with 3 neighbors) performs on the test data by providing the accuracy of its predictions.

```
In [5]: # k = 11  
knn = KNeighborsClassifier(n_neighbors=3)  
knn.fit(X_train_transform,y_train)
```

```
Out[5]: A Jupyter Notebook output display for a KNeighborsClassifier object. It shows a dropdown arrow on the left, followed by the class name 'KNeighborsClassifier' on a light blue background. Below this, the full object representation 'KNeighborsClassifier(n_neighbors=3)' is displayed.
```

```
In [6]: from sklearn.metrics import accuracy_score  
y_pred = knn.predict(X_test_transform)  
accuracy_score(y_test,y_pred)
```

```
Out[6]: 0.95
```