**K-NEAREST NEIGHBORS ALGORITHM.**

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

It uses proximity to make classifications or predictions about the grouping of an individual data point.

The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point. K is the number of nearest neighbors to use.

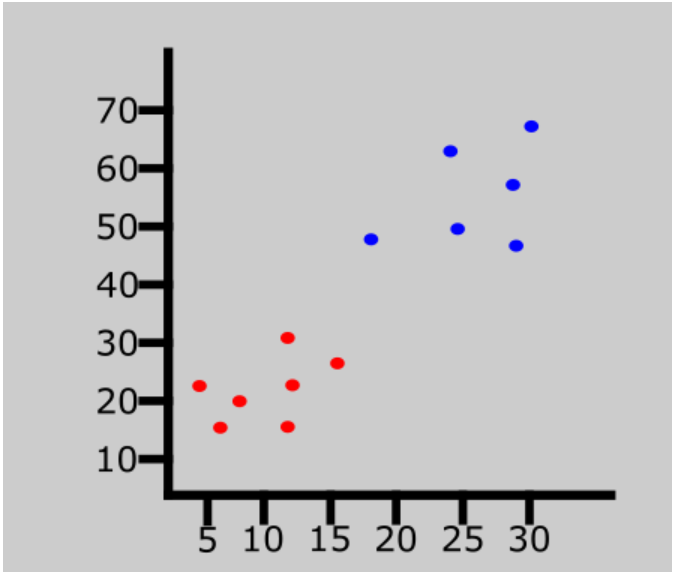
The K-NN algorithm compares a new data entry to the values in a given data set (with different classes or categories).

Based on its closeness or similarities in a given range (K) of neighbors, the algorithm assigns the new data to a class or category in the data set (training data).

Procedure:

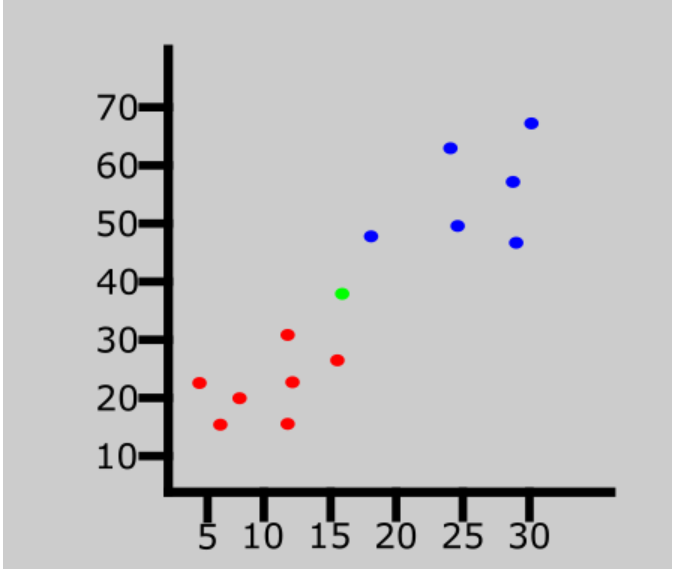
1. Assign a value to K.
2. Calculate the distance between the new data entry and all other existing data entries (you'll learn how to do this shortly). Arrange them in ascending order.
3. Find the K nearest neighbors to the new entry based on the calculated distances.
4. Assign the new data entry to the majority class in the nearest neighbors.

The graph represents a data set consisting of two classes — red and blue.



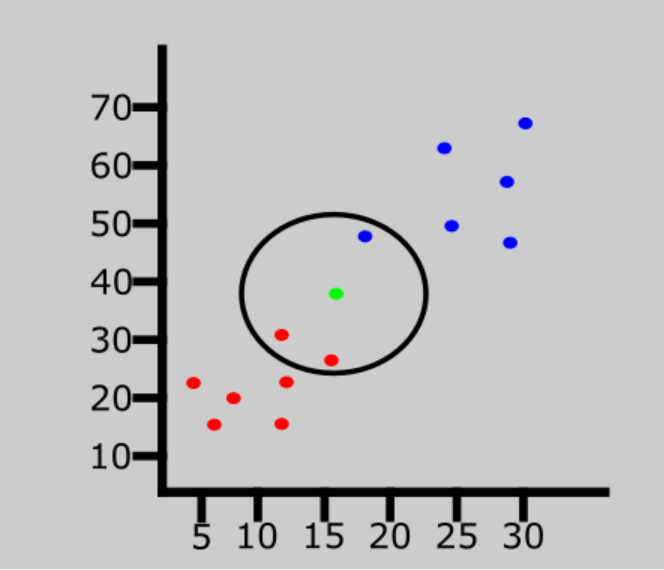
A new data entry has been introduced to the data set. This is represented by the green point in the graph below.

Next assign a value to K which denotes the number of neighbors to consider before classifying the new data entry. Assume the value of K is 3.

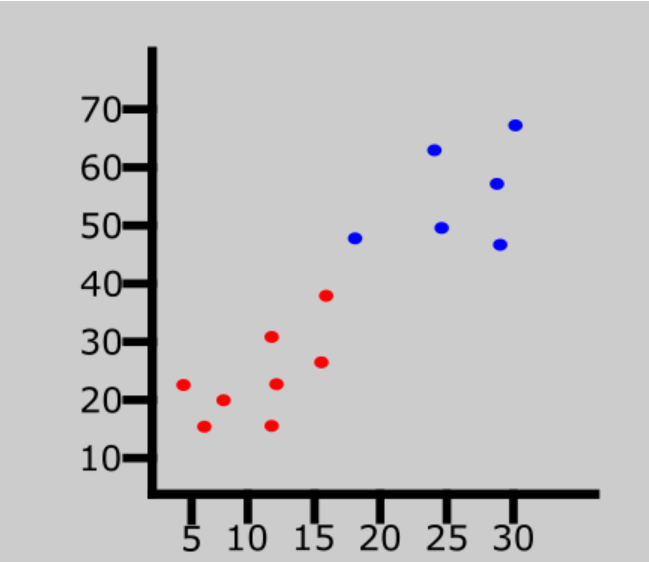


Since the value of K is 3, the algorithm will only consider the 3 nearest neighbors to the green point (new entry). This is represented in the graph below.

Out of the 3 nearest neighbors in the diagram below, the majority class is red so the new entry will be assigned to that class.



The last data entry has been classified as red.



DETERMINING THE DISTANCE METRICS BETWEEN DATA POINTS.

In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated.

These distance metrics help to form decision boundaries, which partitions query points into different regions.

Some of these metrics include:

1. **Euclidean distance** - this is the most commonly used distance measure, and it is limited to real-valued vectors. It measures a straight line between the query point and the other point being measured ie. the length of the straight line that joins the two points which are into consideration.
2. **Manhattan distance** - this is also another popular distance metric, which measures the absolute value between two points. This is the distance between real vectors using the sum of their absolute difference.
3. **Minkowski distance** - this distance measure is the generalized form of Euclidean and Manhattan distance metrics.
4. **Hamming distance** - this technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. It is used for categorical variables.

**EXAMPLE CODE.**

The code below uses the iris dataset which has been attached with this file.

1. **Importing Libraries.**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

import warnings

warnings.filterwarnings('ignore')

- Imports necessary libraries, including NumPy, pandas, matplotlib, seaborn, and sets up inline plotting. It also ignores any warnings that might be displayed.

2. **Loading Iris Dataset.**

from sklearn.datasets import load\_iris

data = load\_iris()

- Loads the Iris dataset from scikit-learn's sample datasets.

3. **Exploring Dataset.**

data

data.target\_names

- Displays information about the dataset and its target names.

4. **Creating DataFrame.**

df = pd.DataFrame(data.data)

df.columns = data.feature\_names

df['Species'] = data.target

- Creates a DataFrame from the dataset, sets column names, and adds a 'Species' column with target values.

5. **Data Splitting.**

X = df.drop('Species',axis=1)

y = df['Species']

- Splits the data into features (`X`) and labels (`y`).

6. **Train-Test Split.**

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=43)

- Splits the data into training and testing sets.

7. **K-Nearest Neighbors (KNN) Classifier.**

from sklearn.neighbors import KNeighborsClassifier

- Imports the KNeighborsClassifier from scikit-learn for the KNN algorithm.

8. **Finding the Optimal K.**

k\_range = range(1,26)

scores = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors = k)

knn.fit(X\_train,y\_train)

y\_pred = knn.predict(X\_test)

scores.append(accuracy\_score(y\_test,y\_pred))

- Iterates over a range of values for K, fits a KNN model for each K, and appends the accuracy scores to a list.

9. **Plotting Accuracy vs. K.**

plt.xlabel("Value of K in KNN")

plt.ylabel("Testing Accuracy")

plt.plot(k\_range,scores)

- Plots the testing accuracy against different values of K to find the optimal K.

10. **Training a KNN Model with Optimal K.**

knn = KNeighborsClassifier(n\_neighbors = 5)

knn.fit(X\_train,y\_train)

- Chooses an optimal K (in this case, 5) based on the accuracy plot and trains a KNN model with that K.

11. **Evaluating Model Performance.**

y\_pred = knn.predict(X\_test)

print("Accuracy Score:{:.2f}%".format(accuracy\_score(y\_test,y\_pred)\*100))

- Predicts labels on the test set and prints the accuracy score of the model.

12. **F1 Score Calculation.**

from sklearn.metrics import f1\_score

print("f1 score:",f1\_score(y\_test,y\_pred,average="weighted"))

- Calculates and prints the F1 score, a metric that considers both precision and recall.

13. **Making a Prediction.**

df = df.replace({0:"setosa",1:'versicolor',2:'virginica'})

test = df.sample(1).values

- Replaces numerical target values with corresponding species names.

- Selects a random sample from the DataFrame for testing.

14. **Predicting and Displaying Result.**

if knn.predict(test[:,0:4]) == 0:

print("Species : setosa")

elif knn.predict(test[:,0:4]) == 1:

print("Species : versicolor")

else:

print("Species : virginica")

- Predicts the species using the trained KNN model and prints the result.