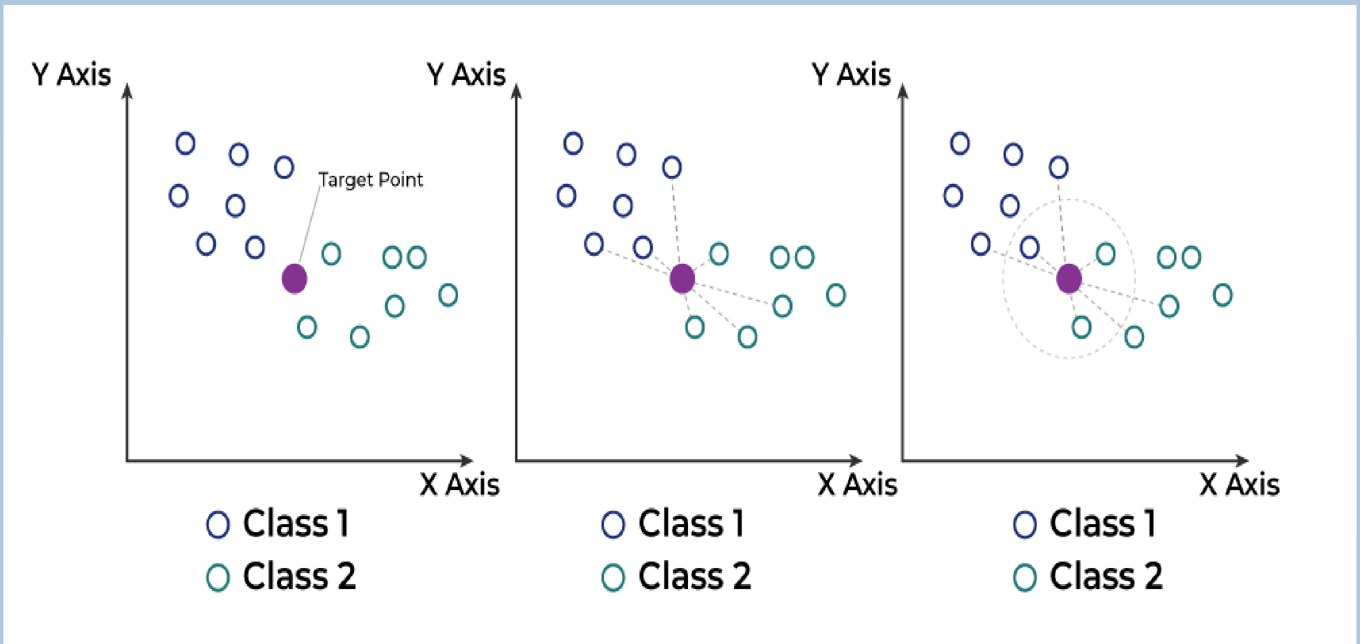
الجامعة الألمانية الأردنية
German Jordanian University

SEEIT

KNN Algorithm



Rana Husni

KNN is a supervised learning algorithm that can be used for both classification and regression problems. The main idea behind KNN is to find the k-nearest data points to a given test data point and use these nearest neighbors to make a prediction. The value of k is a hyperparameter that needs to be tuned, and it represents the number of neighbors to consider.

For classification problems, the KNN algorithm assigns the test data point to the class that appears most frequently among the k-nearest neighbors. In other words, the class with the highest number of neighbors is the predicted class. For regression problems, the KNN algorithm assigns the test data point the average of the k-nearest neighbors' values. The distance metric used to measure the similarity between two data points is an essential factor that affects the KNN algorithm's performance. The most commonly used distance metrics are Euclidean distance, Manhattan distance, and Minkowski distance.

The K-Nearest Neighbors (K-NN) algorithm is a popular Machine Learning algorithm used mostly for solving classification problems.

In this article, you'll learn how the K-NN algorithm works with practical examples.

We'll use diagrams, as well sample data to show how you can classify data using the K-NN algorithm. We'll also discuss the advantages and disadvantages of using the algorithm.

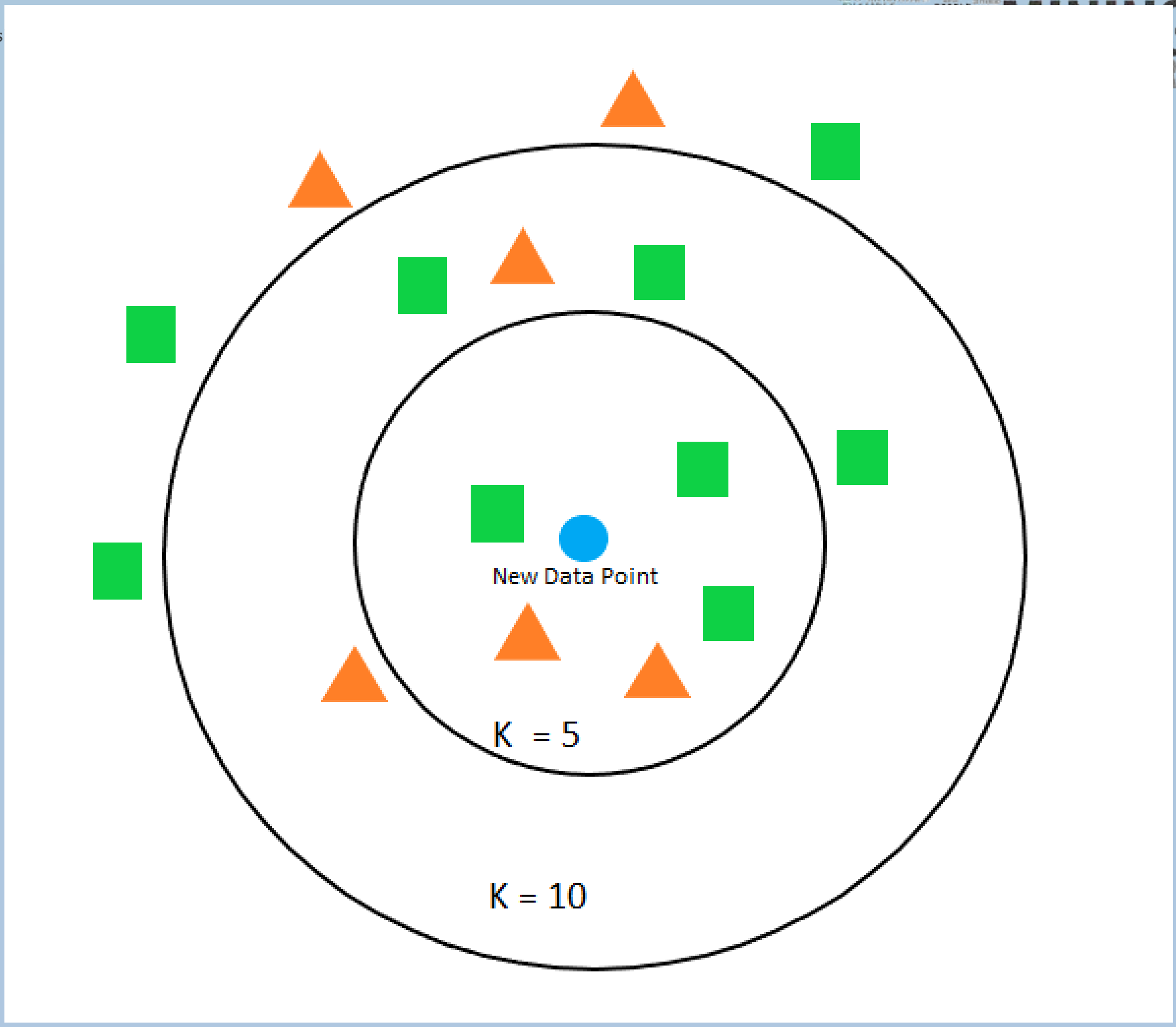
How Does the K-Nearest Neighbors Algorithm Work?

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** ,srobhgien fo (tes atad eht ni yrogetac ro ssalc a ot atad wen eht sngissa mhtirogla eht .(atad gniniart)
Let's break that down into steps:

Step # -1 Assign a value to **K**.
Step # -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.
Step # -3 Find the **K** nearest neighbors to the new entry based on the calculated distances.
Step # -4 Assign the new data entry to the majority class in the nearest neighbors.

Choosing K in KNN



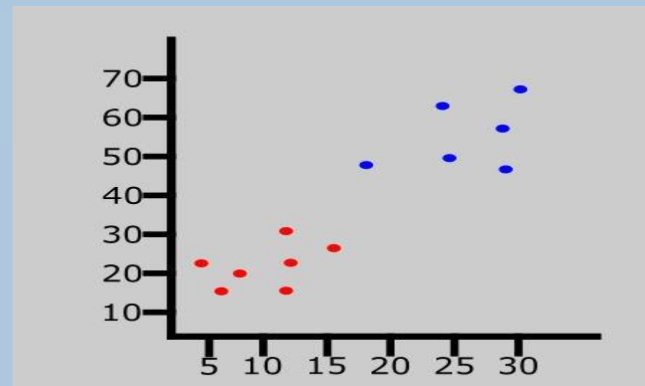
When using KNN, to classify a new data point, we need to specify K. K is the most important hyperparameter in the KNN algorithm. KNN makes predictions based on the K nearest instances in the training data.

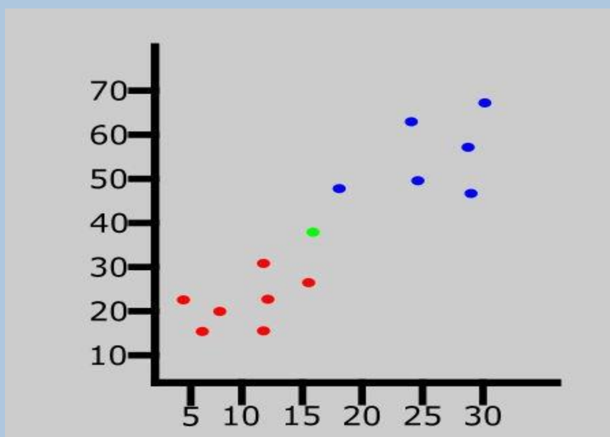
It is better to start with a small K value and increase the value gradually. With a smaller K value, the algorithms will form a smaller decision boundary. It will also go over less number of instances for comparison. In most cases, we need to find a large and optimal value of K which can form a large decision boundary.

Choosing K is always a question of bias-variance tradeoff. When K is small, it will lead to low bias but higher variance and vice-versa for larger K value.

Example

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

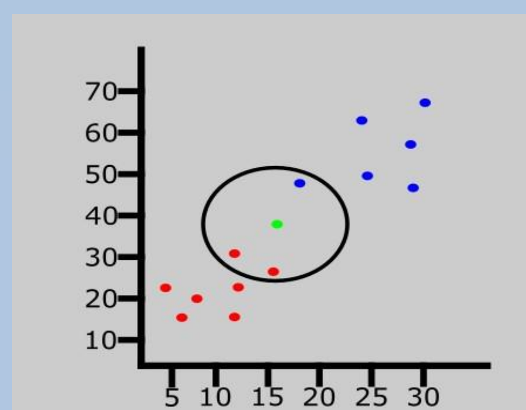


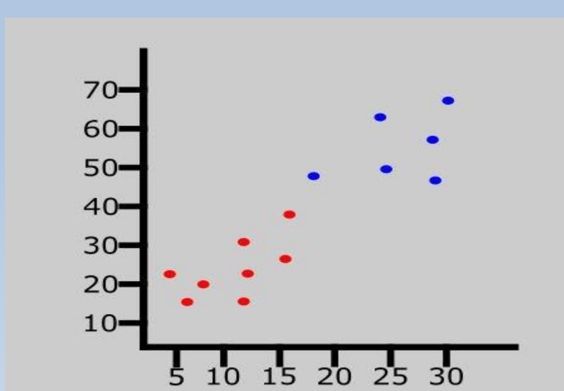


A new data entry has been introduced to the data set. This is represented by the green point in the graph above.

We'll then assign a value to **K** which denotes the number of neighbors to consider before classifying the new data entry. Let's assume the value of **K** is .3

Since the value of **K** is mhtirogla eht ,3 3 eht redisnoc ylno lliwnearest neighbors to the green point (new entry). This is represented in the graph above.
Out of the 3 nearest neighbors in the diagram above, the majority class is red so the new entry will be assigned to that class.





The last data entry has been classified as red.

K-Nearest Neighbors Classifiers and Model Example With Data Set

In the last section, we saw an example the K-NN algorithm using diagrams. But we didn't discuss how to know the distance between the new entry and other values in the data set.

In this section, we'll dive a bit deeper. Along with the steps followed in the last section, you'll learn how to calculate the distance between a new entry and other existing values using the Euclidean distance formula.

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue

The table above represents our data set. We have two columns — **Brightness** and **Saturation** rehtie fo ssalc a sah elbat eht ni wor hcaE . **Red** or **Blue**.
Before we introduce a new data entry, let's assume the value of **K** is .5

How to Calculate Euclidean Distance in the K-Nearest Neighbors Algorithm

Here's the new data entry:

BRIGHTNESS	SATURATION	CLASS
20	35	?

We have a new entry but it doesn't have a class yet. To know its class, we have to calculate the distance from the new entry to other entries in the data set using the Euclidean distance formula.

Here's the formula: $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

Where:

- X_2 = New entry's brightness (20).
- X_1= Existing entry's brightness.
- Y_2 = New entry's saturation (35).
- Y_1 = Existing entry's saturation.

Let's do the calculation together.

Distance #1

For the first row, d1:

$d1 = \sqrt{(20 - 40)^2 + (35 - 20)^2}$

$= \sqrt{400 + 225}$

$= \sqrt{625}$

$= 25$

Here's what the table will look like after all the distances have been calculated:

BRIGHTNESS	SATURATION	CLASS	DISTANCE
40	20	Red	25
50	50	Blue	33.54
60	90	Blue	68.01
10	25	Red	10
70	70	Blue	61.03
60	10	Red	47.17
25	80	Blue	45

Let's rearrange the distances in ascending order:

BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17
70	70	Blue	61.03
60	90	Blue	68.01

Since we chose 5 as the value of **K** evif tsrif eht redisnoc ylno ll'ew , :si tahT .swor

BRIGHTNESS	SATURATION	CLASS	DISTANCE
10	25	Red	10
40	20	Red	25
50	50	Blue	33.54
25	80	Blue	45
60	10	Red	47.17

As you can see above, the majority class within the 5 nearest neighbors to the new entry is **Red** wen eht yfissalc ll'ew ,erofereht .
sa yrtne**Red**.
Here's the updated table:

BRIGHTNESS	SATURATION	CLASS
40	20	Red
50	50	Blue
60	90	Blue
10	25	Red
70	70	Blue
60	10	Red
25	80	Blue
20	35	Red

How to Choose the Value of K in the K-NN Algorithm There is no particular way of choosing the value **K** snoitnevnoc nommoc emos era ereh tub , :dnim ni peek ot

- Choosing a very low value will most likely lead to inaccurate predictions.
- The commonly used value of **K** is .5
- Always use an odd number as the value of **K**.

Advantages of K-NN Algorithm

- It is simple to implement.
- No training is required before classification.

- **Disadvantages of K-NN Algorithm**

- Can be cost-intensive when working with a large data set.
- A lot of memory is required for processing large data sets.
- Choosing the right value of **K** can be tricky.

KNN in python

```
from sklearn.neighbors import KNeighborsClassifier
import numpy as np

# Data: [Brightness, Saturation]
X = np.array([[40, 20],[50, 50],[60, 90],[10, 25],[70, 70],[60, 10],[25, 80]])

# Labels: Class
y = np.array(['Red','Blue','Blue','Red','Blue','Red','Blue'])

# Create and fit the model
k = 5
model = KNeighborsClassifier(n_neighbors=k)
model.fit(X, y)

# Test point
test_point = [[20, 35]]

# Make a prediction
predicted_class = model.predict(test_point)

print(f"Predicted class for the test point {test_point} is:
{predicted_class[0]}")
```

Social_Network_Ads

An example problem for getting a clear intuition on the K -Nearest Neighbor classification. We are using the Social network ad dataset. The dataset contains the details of users in a social networking site to find whether a user buys a product by clicking the ad on the site based on their salary, age, and gender.

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0

GJU
الجامعة الألمانية الأردنية
German Jordanian University

SEEIT

KNN in python

Step 1: Import Libraries

Step 2: Importing a Dataset

Step 3: Encode LabelEncode

Step 4: Train Test and Split on Dataset

Step 5: Training KNN Model

Step 6: Performance of KNN Model (Accuracy, Precision, Recall,f-Measure)

Rana Husni

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Importing the dataset
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [1,2, 3]].values
y = dataset.iloc[:, -1].values
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
# Splitting the dataset into the Training set 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 = 0.20
, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Training the K-NN model on the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
fscore = f1_score(y_test, y_pred)
print('ac: ',ac)

print('Precision: ',precision)
print('Recall: ',recall)
print('fscore: ',fscore)
```