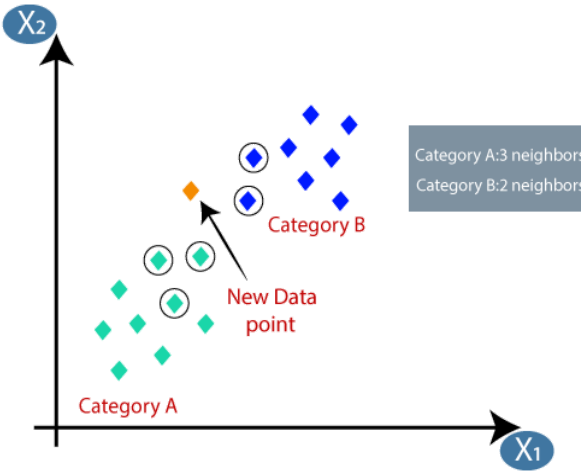
K-Nearest Neighbors - (KNN)

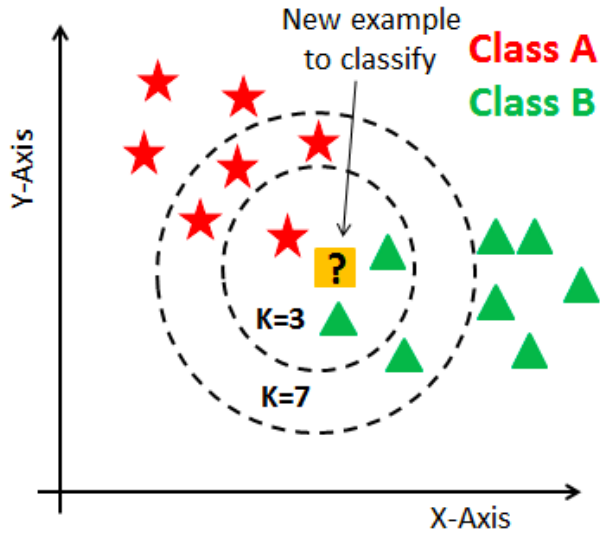
Definition,



K-Nearest Neighbors (**KNN**) is one of the simplest **algorithms** used in **Machine Learning** for regression and classification problem. **KNN algorithms** use data and classify new data points based on similarity measures (e.g. distance function). Classification is done by a majority vote to its neighbors.The non-parametric method refers to a method that does not assume any distribution. Therefore, KNN does not have to find any parameter for the distribution. While in the parametric method, the model finds new parameters, which in turn will be used for the prediction purpose. The only hyperparameter (provided by the user to the model) KNN has is K, which is the number of points that needs to be considered for comparison purpose.

Working of KNN

In the training phase, the model will store the data points. In the testing phase, the distance from the query point to the points from the training phase is calculated to classify each point in the test dataset. Various distances can be calculated, but the most popular one is the Euclidean distance (for smaller dimension data).



In the above image, yellow is the query point, and we want to know which class it belongs to (red or green)

With K=3, the 3 nearest neighbors of the yellow point are considered, and the class is assigned to the query point based on the majority (e.g., 2 green and 1 red — then it is of green class). Similarly, for K=5, 5 nearest neighbors are considered for the comparison, and the majority will decide which class the query point belongs to. One thing to notice here, if the value of K is even, it might create problems when taking a majority vote because the data has an even number of classes (i.e., 2). Therefore, choose K as an odd number when the data has an even number of classes and even number when the data has an odd number of classes.

Euclidean distance between a query point (q) and a training data point (p) is defined as

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Let's learn it with an example:
We have 500 N-dimensional points, with 300 being class 0 and 200 being class 1.
The procedure for calculating the class of query point is:
1, The distance of all the 500 points is calculated from the query point.
2, Based on the value of K, K nearest neighbors are used for the comparison purpose.
3, Let's say K=7, 4 out of 7 points are of class 0, and 3 are of class 1.
Then based on the majority, the query point p is assigned as class 0.

Real-world application of KNN

1, KNN can be used for Recommendation Systems. Although in the real world, more sophisticated algorithms are used for the recommendation system. KNN is not suitable for high dimensional data, but KNN is an excellent baseline approach for the systems. Many companies make a personalized recommendation for its consumers, such as Netflix, Amazon, YouTube, and many more.

2, KNN can search for semantically similar documents. Each document is considered as a vector. If documents are close to each other, that means the documents contain identical topics.

Advantages of KNN algorithms

1, The algorithm is simple and easy to implement.

2, There's no need to build a model, tune several parameters, or make additional assumptions.

3, The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).

Disadvantages of KNN algorithms

1, The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

Steps to implement KNN in python

```
#imported the algorithms from library
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
# to train the model you have to use the function of "fit()"
# while train if we only pass the 80 percent of our data
classifier.fit(X_train, y_train) # X_train = features #y_train= lable
# now we have to take prediction on testing data
y_pred = classifier.predict(X_test) #here we only pass the features
```