1.6. K-NN

The K-nearest neighbors (KNN) algorithm is one of the simplest, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.

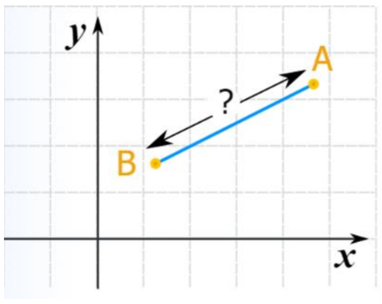
The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

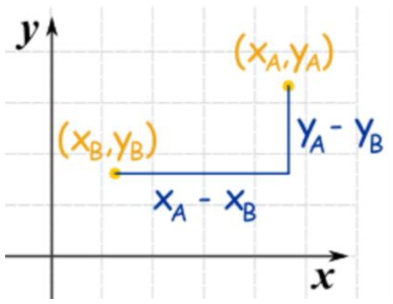
KNN is a non-probabilistic supervised learning algorithm i.e. it doesn't produce the probability of membership of any data point rather KNN classifies the data on hard assignment, e.g., the data point will either belong to 0 or 1.

It is a non-parametric machine learning method. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real world datasets do not follow mathematical theoretical assumptions.

KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph.

The straight-line distance (also called the Euclidean distance) is a popular and familiar choice for calculating distance between two data points. However, there are also other ways of calculating distance like Euclidean, Manhattan, Minkowski, Cosine Similarity and so on, and one way might be preferable depending on the problem we are solving.





Euclidean Distance between point A & B :

$$d(A,B) = \sqrt{(x_A - x_B)^2 + (y_A - y_B)}$$

Refer the following link to recall what's Euclidean Distance:

https://www.mathsisfun.com/algebra/distance-2-points.html

The KNN Algorithm

1. Load the data

2. Initialize K to your chosen number of neighbours

3. For each example in the data

    3.1 Calculate the distance between the query example and the current example from the data.

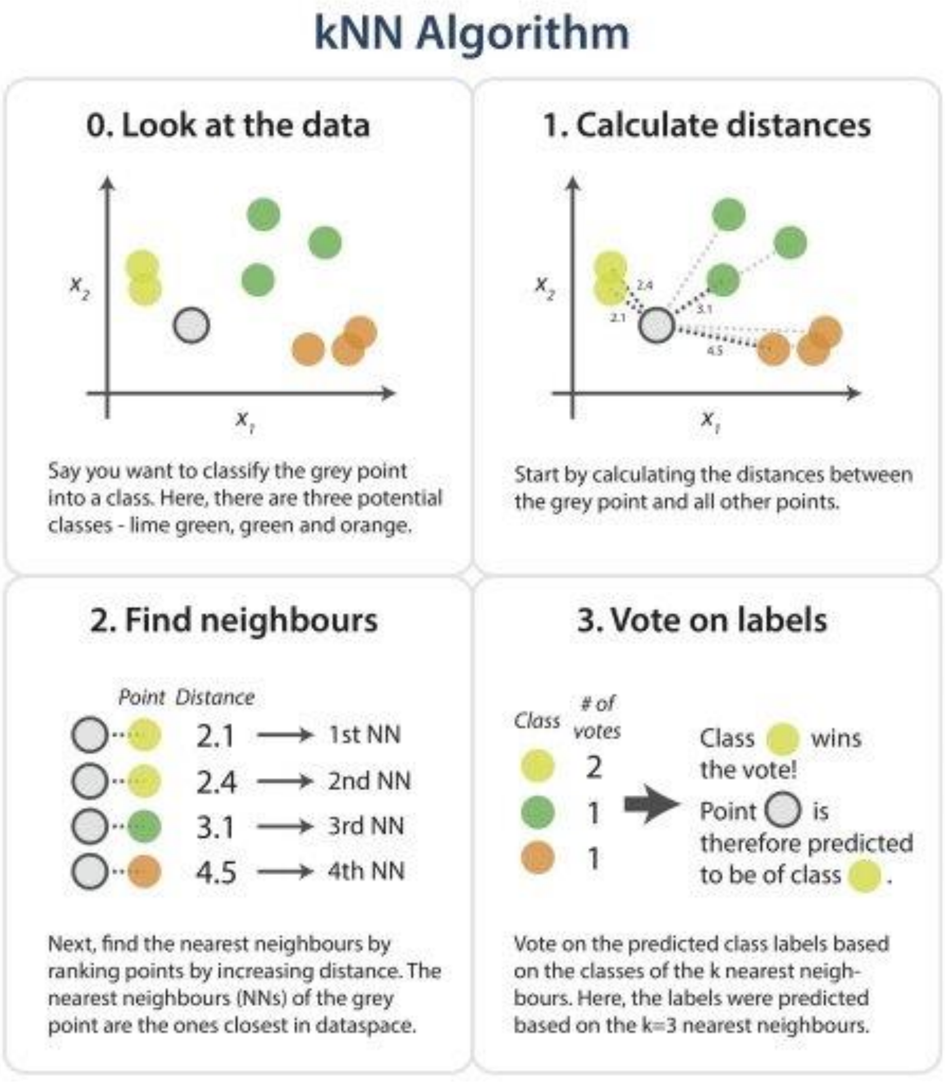
    3.2 Add the distance and the index of the example to an ordered collection

4. Sort the ordered collection of distances and indices from smallest to largest (in ascending order) by the distances

5. Pick the first K entries from the sorted collection

6. Get the labels or target values of the selected K entries

7. If regression, return the mean of the K labels. If classification, return the mode of the K labels


kNN Algorithm
0. Look at the data
x_2
x_1
Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.
1. Calculate distances
2.4
2.1
3.1
4.5
Start by calculating the distances between the grey point and all other points.
2. Find neighbours
Point Distance
2.1 → 1st NN
2.4 → 2nd NN
3.1 → 3rd NN
4.5 → 4th NN
Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.
3. Vote on labels
Class
# of votes
2
1
1
Class wins the vote!
Point is therefore predicted to be of class .
Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

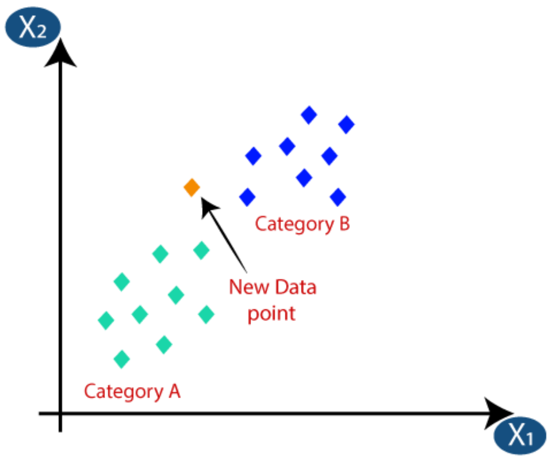
NOTE:

In K-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its K nearest neighbours (K is a positive integer, typically small). If K = 1, then the object is simply assigned to the class of that single nearest neighbour.

In K-NN regression, the output is the property value for the object. This value is the average of the values of k nearest neighbours.

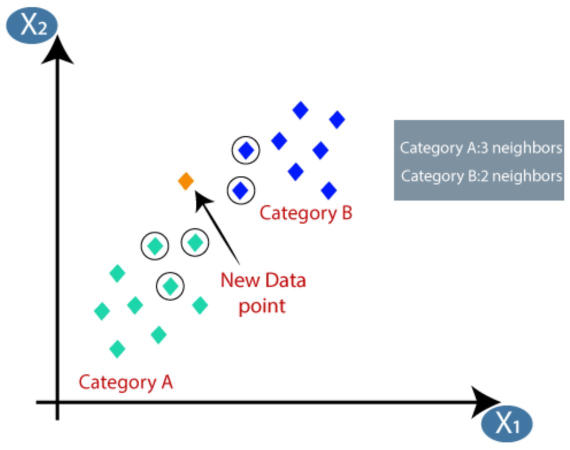
Example: KNN Classification

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



Firstly, we will choose the number of neighbors, so we will choose the k=5. Next, we will calculate the Euclidean distance between the data points.

By calculating the Euclidean distance, we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

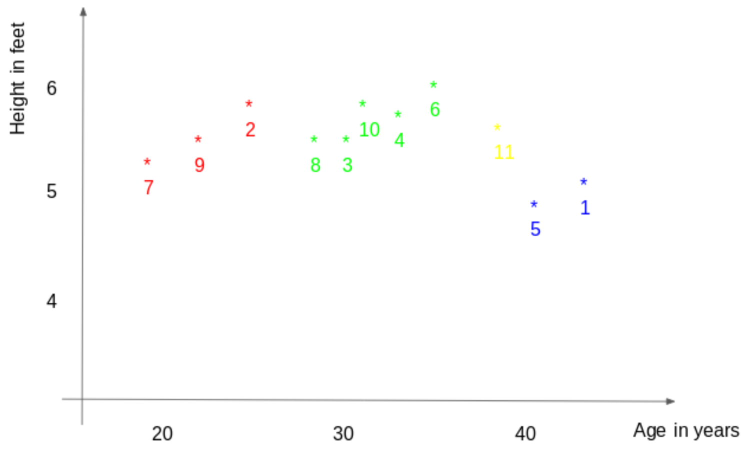


As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

Example: KNN Regression

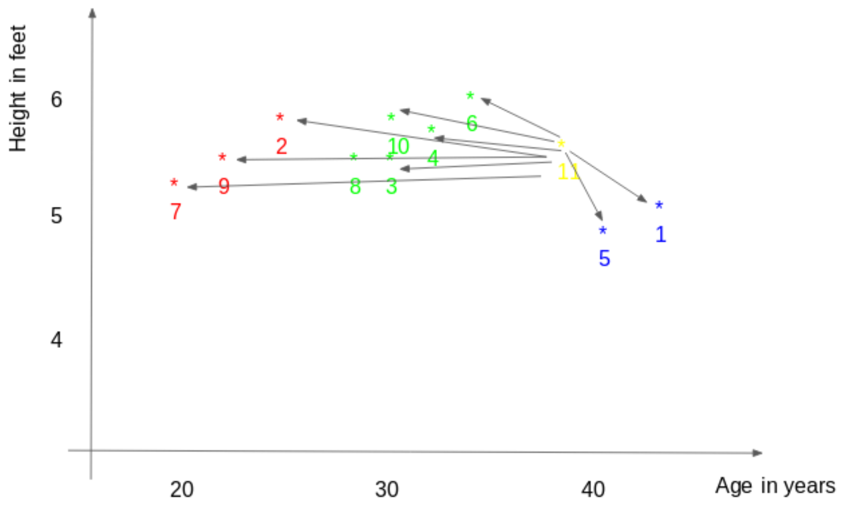
Let us start with a simple example. Consider the following table – it consists of the height, age and weight (target) value for 10 people. As you can see, the weight value of ID11 is missing. We need to predict the weight of this person based on their height and age.

ID	Height	Age	Weight
1	5	45	77
2	5.11	26	47
3	5.6	30	55
4	5.9	34	59
5	4.8	40	72
6	5.8	36	60
7	5.3	19	40
8	5.8	28	60
9	5.5	23	45
10	5.6	32	58
11	5.5	38	?

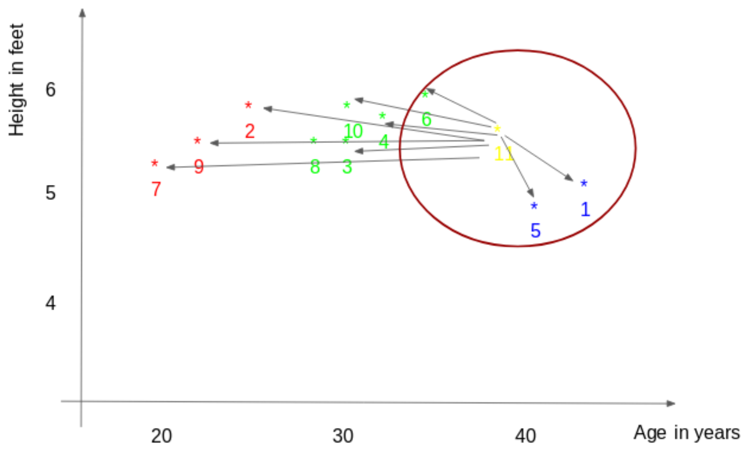


In the above graph, the y-axis represents the height of a person (in feet) and the x-axis represents the age (in years). The points are numbered according to the ID values. The yellow point (ID 11) is our test point.

1. First, the distance between the new point and each training point is calculated.



2. The closest k data points are selected (based on the distance). In this example, points 1, 5, 6 will be selected if value of k is 3.



3. The average of these data points is the final prediction for the new point. Here, we have weight of ID11 = (77+72+60)/3 = 69.66 kg.

Choosing the right value for K

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Here are some things to keep in mind:

1. As we decrease the value of K to 1, our predictions become less stable.

2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.

3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

<u>Advantages</u>

The algorithm is simple and easy to implement.

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

The algorithm is versatile. It can be used for classification, regression.

Disadvantages

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

Curse of Dimensionality

KNN performs better with a lower number of features than a large number of features. You can say that when the number of features increases than it requires more data. Increase in dimension also leads to the problem of overfitting.

KNN used in the variety of applications such as finance, healthcare, political science, handwriting detection, image recognition and video recognition. In Credit ratings, financial institutes will predict the credit rating of customers. In loan disbursement, banking institutes will predict whether the loan is safe or risky. In political science, classifying potential voters in two classes will vote or won't vote.

EXTRA READING

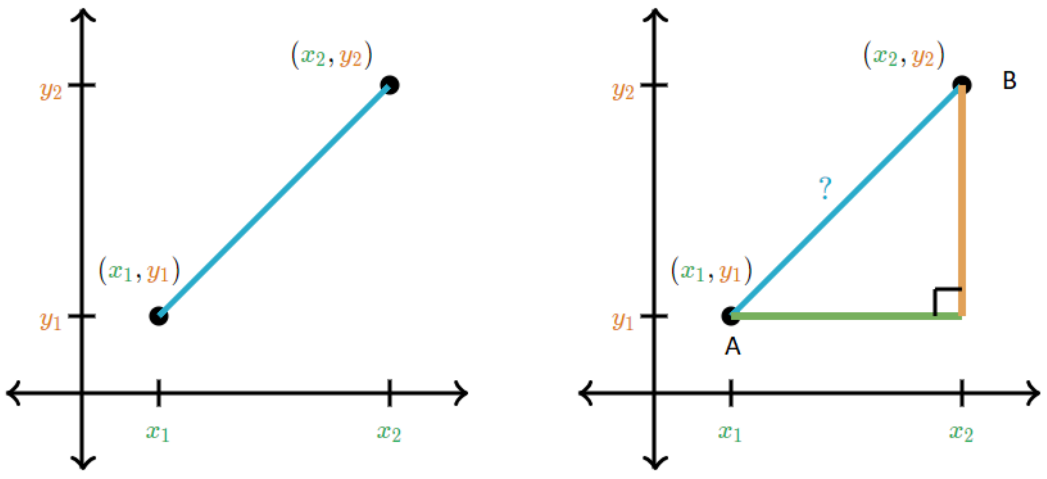
Distance metric uses distance function which provides a relationship metric between each elements in the dataset. As we can see, distance measures play an important role in machine learning. The four most commonly used Distance metrics in Machine learning are:

- Euclidean Distance
- Minkowski Distance
- Manhattan Distance
- Hamming Distance

This Reference Link from sklearn documentation will become very handy.

Euclidean Distance:

Euclidean distance is the straight line distance between 2 data points in a plane.



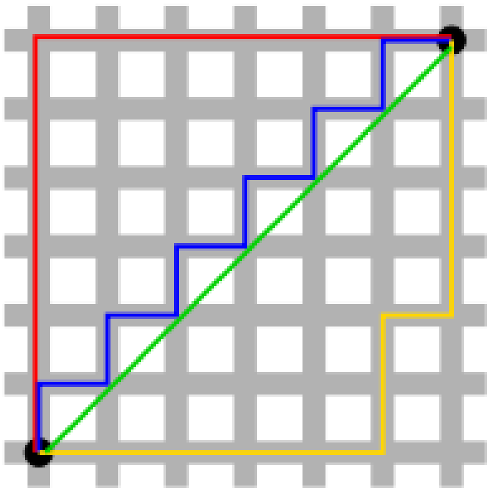
In order to calculate the distance between data points A and B Pythagorean theorem considers the length of x and y axis.

$$? = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

This formula is similar to the Pythagorean theorem formula; thus it is also known as the Pythagorean Theorem.

Manhattan Distance:

We use Manhattan distance if we need to calculate the distance between two data points in a grid-like path.



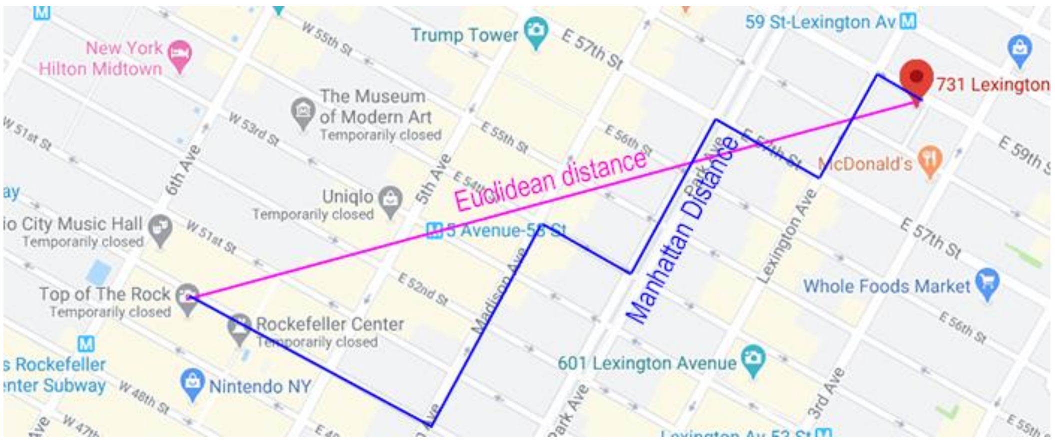
The above figure shows, Manhattan Distance versus Euclidean distance: the red, yellow, and blue paths all have the same shortest path length of 12. In Euclidean geometry, the green line has length $6\sqrt{2} \approx 8.49$ and is the unique shortest path.

The formula of Manhattan Distance is given by:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Manhattan Distance is preferred over the Euclidean distance metric as the dimension of the data increases. This occurs due to something known as the 'curse of dimensionality'.

The Manhattan distance is also known as the taxicab geometry.

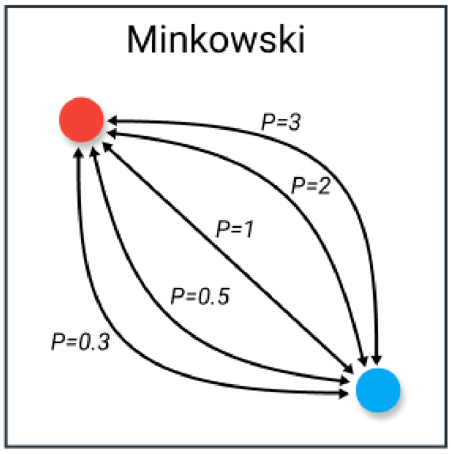


The Euclidean distance formula is good for measuring theoretical distances. However, in real life, for example, in a city, it is most times impossible to move from one point straight to another. Fences, buildings, streets will not allow doing this and you have to follow the streets, which are often arranged in a grid. In a city, the Manhattan distance formula is much more useful because it allows calculating the distance between two data points on a uniform grid, like city blocks or a chessboard, in which there can be many paths between the two points that are equal to the same Manhattan distance. They call it Manhattan because of the grid layout of most streets on Manhattan island except for Broadway, which preceded the grid plan. This causes the shortest path a car could take between two intersections in the borough to have length equal to the intersections' distance in taxicab geometry.

Minkowski Distance:

Minkowski distance is a generalized distance metric. The distance can be calculated using the below formula:

$$d(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$



We can manipulate the above formula by substituting ‘p’ to calculate the distance between two data points in different ways such as:

p = 1, Manhattan Distance

p = 2, Euclidean Distance

Intermediate values of p, for example, $p = 1.5$, provide a balance between the two measures.

Hamming Distance:

Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different.

Suppose there are two strings [0, 0, 0, 0, 0, 1] and [0, 0, 0, 0, 1, 0].

To calculate the Hamming distance between the two bitstrings, we can see that there are two differences between the strings, or 2 out of 6 bit positions different, which averaged (2/6) is about 1/3 or 0.333.

You are most likely going to encounter bitstrings when you one-hot encode categorical columns of data. Thus, Hamming distance is used to measure the distance between categorical variables.