Distance metrics are a key part of several [machine learning algorithms](https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/?utm_source=blog&utm_medium=4-types-of-distance-metrics-in-machine-learning). These distance metrics are used in both supervised and unsupervised learning, generally to calculate the similarity between data points.

The notion of distance is the most important basis for classification.

Standard distances often do not lead to appropriate results.

For every individual problem the adequate distance is to be decided upon.

The right choice of the distance measure is one of the most decisive steps for the determination of cluster properties.

*An effective distance metric improves the performance of our machine learning model, whether that’s for classification tasks or clustering.*

**4 Types of Distance Metrics in Machine Learning**

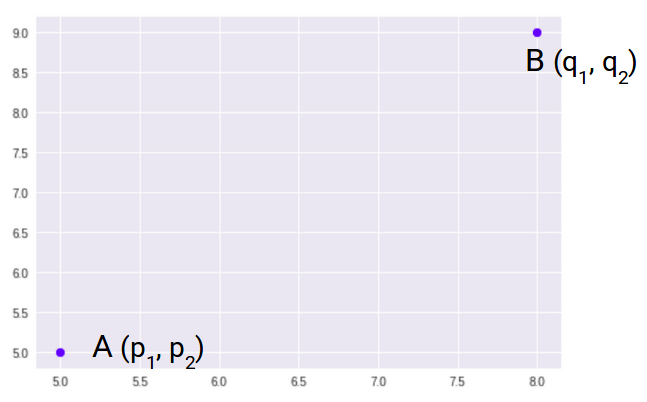
1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance
4. Hamming Distance

Let’s start with the most commonly used distance metric – Euclidean Distance.

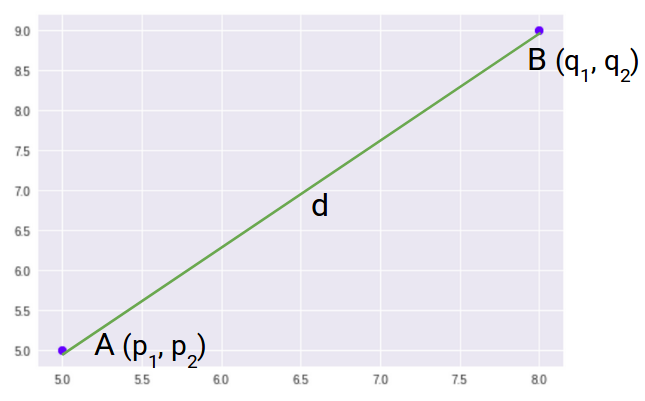
**1. Euclidean Distance**

*Euclidean Distance represents the shortest distance between two points.*

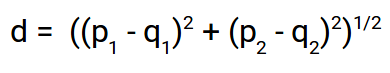
Most machine learning algorithms including K-Means use this distance metric to measure the similarity between observations. Let’s say we have two points as shown below:



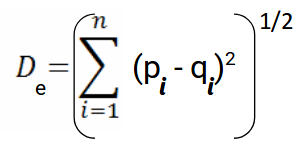
So, the Euclidean Distance between these two points A and B will be:



Here’s the formula for Euclidean Distance:



We use this formula when we are dealing with 2 dimensions. We can generalize this for an n-dimensional space as:



Where,

* n = number of dimensions
* pi, qi = data points

Let’s code Euclidean Distance in [Python](https://courses.analyticsvidhya.com/courses/introduction-to-data-science?utm_source=blog&utm_medium=4-types-of-distance-metrics-in-machine-learning). This will give you a better understanding of how this distance metric works.

We will first import the required libraries. I will be using the SciPy library that contains pre-written codes for most of the distance functions used in Python:

# importing the library

from scipy.spatial import distance

# defining the points

point\_1 = (1, 2, 3)

point\_2 = (4, 5, 6)

point\_1, point\_2

((1, 2, 3), (4, 5, 6))

# computing the euclidean distance

euclidean\_distance = distance.euclidean(point\_1, point\_2)

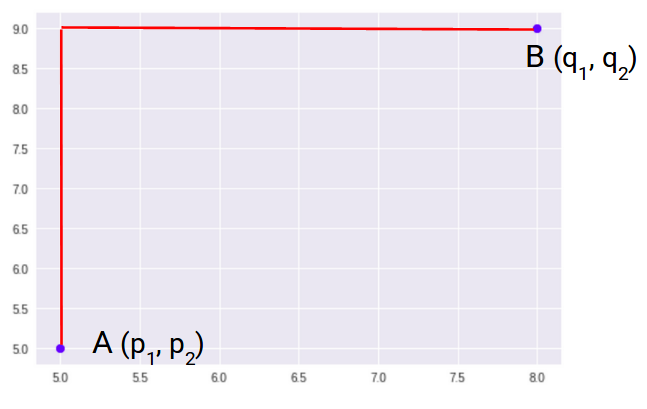
print('Euclidean Distance b/w', point\_1, 'and', point\_2, 'is: ', euclidean\_distance)

Euclidean Distance b/w (1, 2, 3) and (4, 5, 6) is: 5.196152422706632

## 2. Manhattan Distance

*Manhattan Distance is the sum of absolute differences between points across all the dimensions.*

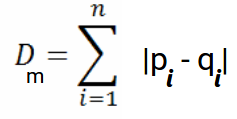
We can represent Manhattan Distance as:



Since the above representation is 2 dimensional, to calculate Manhattan Distance, we will take the sum of absolute distances in both the x and y directions. So, the Manhattan distance in a 2-dimensional space is given as:

manhattan distance formula

And the generalized formula for an n-dimensional space is given as:



Where,

* n = number of dimensions
* pi, qi = data points

Now, we will calculate the Manhattan Distance between the two points:

# computing the manhattan distance

manhattan\_distance = distance.cityblock(point\_1, point\_2)

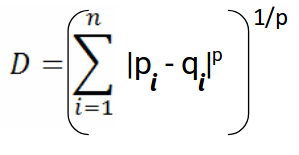
print('Manhattan Distance b/w', point\_1, 'and', point\_2, 'is: ', manhattan\_distance)

Manhattan Distance b/w (1, 2, 3) and (4, 5, 6) is: 9

## 3. Minkowski Distance

*Minkowski Distance is the generalized form of Euclidean and Manhattan Distance.*

The formula for Minkowski Distance is given as:



Here, p represents the order of the norm. Let’s calculate the Minkowski Distance of the order 3:

# computing the minkowski distance

minkowski\_distance = distance.minkowski(point\_1, point\_2, p=3)

print('Minkowski Distance b/w', point\_1, 'and', point\_2, 'is: ', minkowski\_distance)

Minkowski Distance b/w (1, 2, 3) and (4, 5, 6) is: 4.3267487109222245

The p parameter of the Minkowski Distance metric of SciPy represents the order of the norm. When the order(p) is 1, it will represent Manhattan Distance and when the order in the above formula is 2, it will represent Euclidean Distance.

Let’s verify that in Python:

# minkowski and manhattan distance

minkowski\_distance\_order\_1 = distance.minkowski(point\_1, point\_2, p=1)

print('Minkowski Distance of order 1:',minkowski\_distance\_order\_1, '\nManhattan Distance: ',manhattan\_distance)

Minkowski Distance of order 1: 9.0

Manhattan Distance: 9

Here, you can see that when the order is 1, both Minkowski and Manhattan Distance are the same. Let’s verify the Euclidean Distance as well:

# minkowski and euclidean distance

minkowski\_distance\_order\_2 = distance.minkowski(point\_1, point\_2, p=2)

print('Minkowski Distance of order 2:',minkowski\_distance\_order\_2, '\nEuclidean Distance: ',euclidean\_distance)

Minkowski Distance of order 2: 5.196152422706632

Euclidean Distance: 5.196152422706632

So far, we have covered the distance metrics that are used when we are dealing with continuous or numerical variables. But **what if we have categorical variables?** How can we decide the similarity between categorical variables? This is where we can make use of another distance metric called Hamming Distance.

## 4. Hamming Distance

*Hamming Distance measures the similarity between two strings of the same length. The Hamming Distance between two strings of the same length is the number of positions at which the corresponding characters are different.*

Let’s understand the concept using an example. Let’s say we have two strings:

**“euclidean”**and **“manhattan”**

Since the length of these strings is equal, we can calculate the Hamming Distance. We will go character by character and match the strings. The first character of both the strings (e and m respectively) is different. Similarly, the second character of both the strings (u and a) is different. and so on.

Look carefully – seven characters are different whereas two characters (the last two characters) are similar:

distance metrics

Hence, the Hamming Distance here will be 7. Note that larger the Hamming Distance between two strings, more dissimilar will be those strings (and vice versa).

Let’s see how we can compute the Hamming Distance of two strings in Python. First, we’ll define two strings that we will be using:

# defining two strings

string\_1 = 'euclidean'

string\_2 = 'manhattan'

These are the two strings “euclidean” and “manhattan” which we have seen in the example as well. Let’s now calculate the Hamming distance between these two strings:

# computing the hamming distance

hamming\_distance = distance.hamming(list(string\_1), list(string\_2))\*len(string\_1)

print('Hamming Distance b/w', string\_1, 'and', string\_2, 'is: ', hamming\_distance)

Hamming Distance b/w euclidean and manhattan is: 7.0

As we saw in the example above, the Hamming Distance between “euclidean” and “manhattan” is 7. We also saw that Hamming Distance only works when we have strings of the same length.

Let’s see what happens when we have strings of different lengths:

# strings of different shapes

new\_string\_1 = 'data'

new\_string\_2 = 'science'

len(new\_string\_1), len(new\_string\_2)

(4, 7)

You can see that the lengths of both the strings are different. Let’s see what will happen when we try to calculate the Hamming Distance between these two strings:

# computing the hamming distance

hamming\_distance = distance.hamming(list(new\_string\_1), list(new\_string\_2))

**ValueError**: The 1d arrays must have equal lengths.