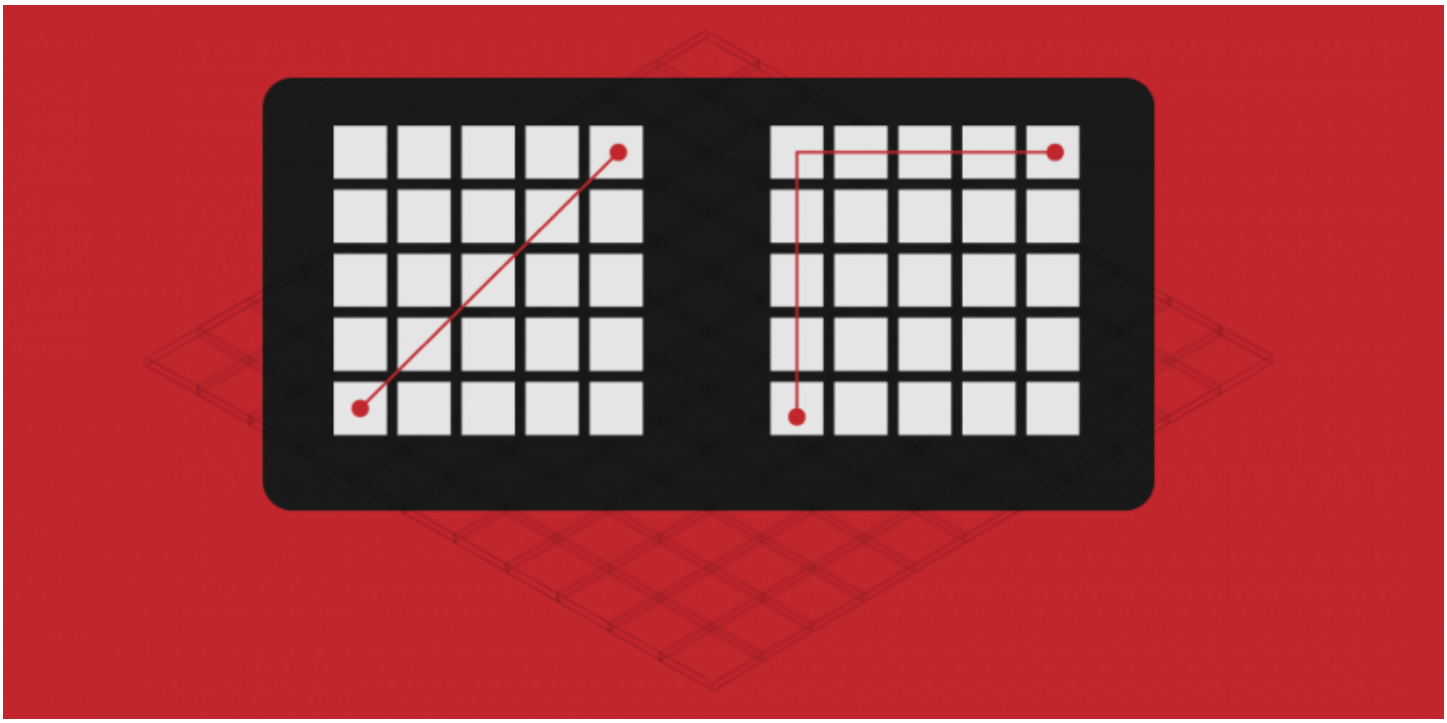
4 Types of Distance Metrics in Machine Learning

BEGINNER LISTICLE MACHINE LEARNING PYTHON

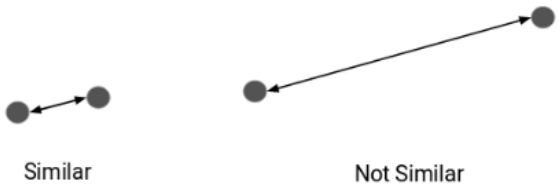
Distance metrics are a key part of several machine learning algorithms. These distance metrics are used in both supervised and unsupervised learning, generally to calculate the similarity between data points.

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



Let's say we want to create clusters using the K-Means Clustering or k-Nearest Neighbour algorithm to solve a classification or regression problem. How will you define the similarity between different observations here? How can we say that two points are similar to each other?

This will happen if their features are similar, right? When we plot these points, they will be closer to each other in distance.



Hence, we can calculate the distance between points and then define the similarity between them. Here's the million-dollar question – how do we calculate this distance and what are the different distance metrics in machine learning?

That's what we aim to answer in this article. We will walk through 4 types of distance metrics in machine learning and understand how they work in Python.

4 Types of Distance Metrics in Machine Learning

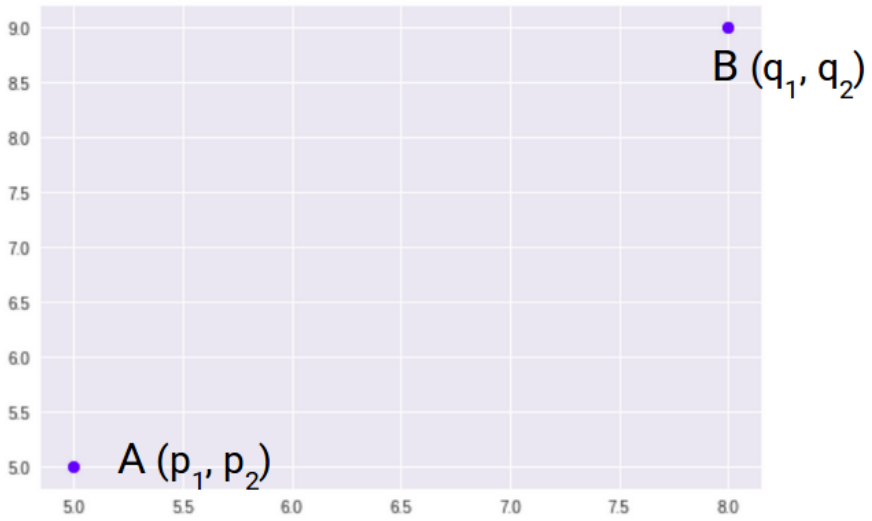
1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance
4. Hamming Distance

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

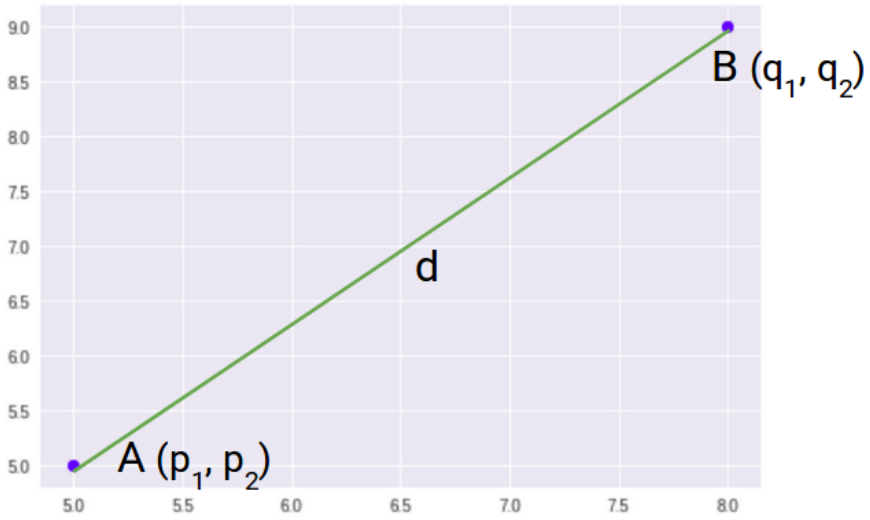
1. Euclidean Distance

> 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:

$$d = ((p_1 - q_1)^2 + (p_2 - q_2)^2)^{1/2}$$

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

$$D_e = \left(\sum_{i=1}^{n} (p_i - q_i)^2 \right)^{1/2}$$

Where,

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

Let's code Euclidean Distance in Python. 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
```

view raw

library_and_dataset.py hosted with ❤ by **GitHub**

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

These are the two sample points which we will be using to calculate the different distance functions. Let's now calculate the Euclidean Distance between these two points:

```
# 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)
```

view raw

euclidean_distance.py hosted with ❤ by **GitHub**

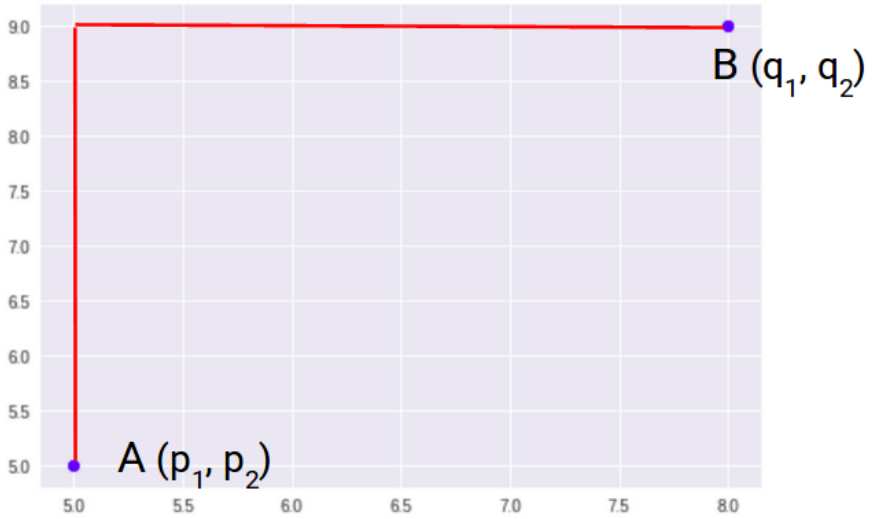
```
Euclidean Distance b/w (1, 2, 3) and (4, 5, 6) is:  5.196152422706632
```

This is how we can calculate the Euclidean Distance between two points in Python. Let's now understand the second distance metric, Manhattan Distance.

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:

$$d = |p_1 - q_1| + |p_2 - q_2|$$

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

$$D_m = \sum_{i=1}^{n} |p_i - q_i|$$

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)
```

view raw

manhattan_distance.py hosted with ❤ by GitHub

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

Note that **Manhattan Distance is also known as city block distance.** SciPy has a function called *cityblock* that returns the Manhattan Distance between two points.

Let's now look at the next distance metric – Minkowski Distance.

3. Minkowski Distance

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

The formula for Minkowski Distance is given as:

$$D = \left(\sum_{i=1}^{n} |p_i - q_i|^p \right)^{1/p}$$

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)
```

view raw

minkowski_distance.py hosted with ❤ by GitHub

```
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)
```

view raw

minkowski_distance_vs_manhattan_distance.py hosted with ❤ by GitHub

```
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)
```

view raw

minkowski_distance_vs_euclidean_distance.py hosted with ❤ by GitHub

```
Minkowski Distance of order 2: 5.196152422706632
Euclidean Distance:  5.196152422706632
```

When the order is 2, we can see that Minkowski and Euclidean distances are the same.

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

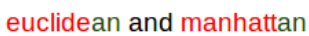
> Hamming Distance measures the similarity between two strings of the same length. The Hamming Distance between two strings of the same length is the number of positions at which the corresponding characters are different.

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:



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'
```

view raw

data_hamming.py hosted with ❤ by GitHub

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)
```

view raw

hamming_distance.py hosted with ❤ by GitHub

```
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)
```

view raw

data_hamming_2.py hosted with ❤ by GitHub

```
(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))
```

view raw

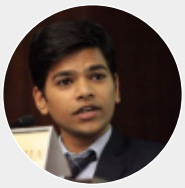
hamming_different_shape.py hosted with ❤ by GitHub

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-51d01805d328> in <module>()
      1 # computing the hamming distance
----> 2 hamming_distance = distance.hamming(list(new_string_1), list(new_string_2))

~/anaconda3/lib/python3.6/site-packages/scipy/spatial/distance.py in hamming(u, v, w)
    792     v = _validate_vector(v)
    793     if u.shape != v.shape:
--> 794         raise ValueError('The 1d arrays must have equal lengths.')
    795     u_ne_v = u != v
    796     if w is not None:

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

This throws an error saying that the lengths of the arrays must be the same. Hence, **Hamming distance only works when we have strings or arrays of the same length.**

These are some of the similarity measures or the distance matrices that are generally used in Machine Learning.

Article Url - https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/



Pulkit Sharma

My research interests lies in the field of Machine Learning and Deep Learning. Possess an enthusiasm for learning new skills and technologies.