Distance Measure

A set of points is called a space. A space is necessary to define any distance measure. Let x and y be two points in the space, then a distance measure is defined as a function which takes the two points x and y as input, and produces the distance between the two points x and y as output. The distance function is denoted as : d(x, y)

– The output produced by the function d is a real number which satisfies the following axioms:

1. Non-negativity : The distance between any two points can never be negative.

$d(x, y) \geq 0$

2. Zero distance : The distance between a point and itself is zero.

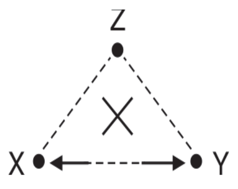
$d(x, y) = 0$ iff $x = y$

3. Symmetry : The distance from x to y is same as the distance from y to x.

$d(x, y) = d(y, x)$

4. Triangle inequality : The direct distance between x and y is always smaller than or equal to the distance between
x and y via another point z. In other words, distance measure is the length of the shortest path between two
points x and y.

$\mathbf{d(x, y) \leq d(x, z) + d(z, y)}$



The Euclidean distance is the most popular out of all the different distance measures.

– The Euclidean distance is measured on the Euclidean space. If we consider an n-dimensional Euclidean space then
each point in that space is a vector of n real numbers. For example, if we consider the two-dimensional Euclidean
space then each point in the space is represented by (x_1, x_2) where x_1 and x_2 are real numbers.

– The most familiar Euclidean distance measure is known as the L_2– norm which in the n-dimensional space is
defined as :

$d([x1, x2,\ldots xn], [y1, y2,\ldots yn]) = \sum n$
$i = 1$
$(xi – yi)2$

– For the two-dimensional space the L_2– norm will be :
$d([x1, x2], [y1, y2]) = (x1 – y1)2 + (x2 – y2)2$

– We can easily verify all the distance axioms on the Euclidean distance :

1. Non-negativity : The Euclidean distance can never be negative as all the sum terms ($x_i – y_i$) are squared and the
square of any number whether positive or negative is always positive. So the final result will either be zero or a
positive number.

2. Zero distance : In case of the Euclidean distance from a point to itself all the x_i's will be equal to the y_i's. This in
turn will make all the sum terms ($x_i – y_i$) equal to zero. So the final result will also be zero.

3. Symmetry : $(x_i – y_i)_2$will always be equal to $(y_i– x_i)_2$. So the Euclidean distance is always symmetric.

4. Triangle inequality : In Euclidean space, the length of the side of a triangle is always less than or equal to the sum
of the lengths of the other two sides.

– Some other distance measures that are used on the Euclidean space are :

1. L_r– norm where r is a constant :

$$d\,([x_1, x_2,\ldots x_n], [y_1, y_2,\ldots y_n]) = \left(\sum_{i=1}^{n} |x_i - y_i|^r\right)^{\frac{1}{r}}$$

2. L_1– norm which is commonly known as Manhattan distance :

$$d\,([x_1, x_2,\ldots x_n], [y_1, y_2,\ldots y_n]) = \sum_{i=1}^{n} |x_i - y_i|$$

3. L_∞– norm which is defined as :

$$d\,([x_1, x_2,\ldots x_n], [y_1, y_2,\ldots y_n]) = \max\,(|x_i - y_i|)\ \forall\ i$$

Q . Consider the two points (10, 4) and (6, 7) in the two-dimensional Euclidean space. Find the Euclidean distance
between them.
Soln. :
(1) L2– norm = (10 – 6)2 + (4 – 7)2
= 42 + 32
= 16 + 9
= 25
= 5
(2) L1 – norm = |10 – 6| + |4 – 7|
= 4 + 3
= 7
(3) L∞– norm = max (|10 – 6|, |4 – 7|)
= max (4, 3)
= 4

Jaccard Distance
Jaccard distance is measured in the space of sets. Jaccard distance between two sets is defined as :
d(x, y) = 1 – SIM(x, y)
SIM(x, y) is the Jaccard similarity which measures the closeness of two sets. Jaccard similarity is given by the ratio of
the size of the intersection and the size of the union of the sets x and y.
We can verify the distance axioms on the Jaccard distance :
1. Non-negativity : The size of the intersection of two sets can never be more than the size of the union. This means
the ratio SIM(x, y) will always be a value less than or equal to 1. Thus d(x, y) will never be negative.
2. Zero distance : If x = y, then x u x = x n x = x. In this case SIM(x, y) = x/x = 1. Hence, d(x, y) = 1 – 1 = 0. In other
words the Jaccard distance between the same set and itself is zero.

3. Symmetry : As both union as well as intersection are symmetric x u y = y u x and x n y = y n x, hence Jaccard
distance is also symmetric d(x, y) = d(y, x).
4. Triangle inequality : Jaccard distance can also be considered as the probability that a random minhash function
does not map both the sets x and y to the same value.
P[h(x) ≠ h(y)] <=P[h(x) ≠ h(z)] + P[h(z) ≠ h(y)]
Where h is the random minhash function.

Cosine Distance
The cosine distance is measured in those spaces which have dimensions. Examples of such spaces are :
1. Euclidean spaces in which the vector components are real numbers, and
2. Discrete versions of Euclidean spaces in which the vector components are integers or Boolean (0 and 1).
Cosine distance is the angle made by the two vectors from the origin to the two points in the space. The range of this
angle is between 0 to 180 degrees.

The steps involved in calculating the cosine distance given two vectors x and y are :
1. Find the dot product x.y :

$$([x_1, x_2, \ldots x_n], [y_1, y_2, \ldots y_n]) = \sum_{i=1}^{n} x_i y_i$$

2. Find the L2– norms of both x and y,
3. Divide the dot product x.y by the L2– norms of x and y to get cos θ where θ is the angle between x and y.
4. Finally, to get θ use the cos-1 function.

The distance axioms on the Cosine distance may be verified as follows :
1. Non-negativity : The range of the possible values is from 0 to 180 degrees. So there is no question of negative
distance.
2. Zero distance : If two vectors are the same direction, only then the angle between them will be zero.

3. Symmetry : The angle between x and y will always be equal to the angle between y and x.

4. Triangle inequality : The sum of the rotations from x to z and then z to y can never be less than the direct rotation from x to y.

Ex: Consider the following two vectors in the Euclidean space :
x = [1, 2, – 1], and y = [2, 1, 1].
Calculate the cosine distance between x and y.
Soln. :

Given x = [1,2, – 1] ;y = [2, 1, 1]

(i) $x \cdot y = [1\times2] + [2\times1] + [(-1)\times1]$
$= 2 + 2 + (-1) = 2 + 2 - 1$
$= 4 - 1 = 3$

(ii) L_2 norm for x $= \sqrt{(1)^2 + (2)^2 + (-1)^2} = \sqrt{6}$
L_2 norm for y $= \sqrt{(2)^2 + (1)^2 + (1)^2} = \sqrt{6}$

(iii) $$\cos\theta = \frac{x,y}{(L_2 - \text{norm of x}), (L\text{- norm of y})} = \frac{3}{\sqrt{6},\sqrt{6}} = \frac{3}{6} = \frac{1}{2}$$

Now, $\text{Cos}^{-1} = 60°$
∴ The angle between the two vectors x and y is 60°.

Edit Distance

The points in the space for edit distance are strings.
There are two different methods for defining and calculating edit distances :
1. The classical method
2. The Longest Common Subsequence (LCS) method

(i) Classical method

The Edit distance between two strings x and y is the least number of edit operations required to convert x into y.

Only two edit operations are allowed:
1. Insertion of a single character, and
2. Deletion of a single character.
To illustrate let us take the following two strings :
x = JKLMN
y = JLOMNP
For calculating the Edit distance between x and y we have to convert string x into string y using the edit operations of insertion and deletion.
At first compare the character sequences in both the strings :

```
x = J K L M N
y = J L O M N P
    1  2 3  4  5  6
```

① ② ③ ④ ⑤ ⑥ Positions

Clearly, positions②,③and⑥are having different characters in x and y. So we need to make the necessary insertions and deletions at these three positions.

```
K  L  –
L  O  P
2  3  6
```

From position ② of string x, we have to delete the character K. The characters following K will be shifted one position to the left.

After the first edit operation (deletion)the status of the string x is :

```
x = J   L    M   N
     ① ② ③ ④
```

Now the character O has to be inserted at position 3 i.e. after the character L and before the character M.
After the second edit operation (insertion) the status of the string x is :

```
x = J  L   O   M   N
    ① ② ③ ④ ⑤
```

In the final step, the character P has to be inserted in the string x at position 6.
After the third and final edit operation (insertion) the status of the string x is :
X = J L O M N P
① ② ③ ④ ⑤ ⑥

So the Edit distance between x and y is :
d (x, y) = Number of deletions + Number of insertions
= 1 + 2 = 3

(ii) Longest Common Subsequence (LCS)
 The longest common subsequence of two strings x and y is a subsequence of maximum length which appears in
both x and y in the same relative order, but not necessarily contiguous.
 Let us illustrate the concept of finding the Edit distance using LCS method with the same set of strings as in the
previous method :
x = J K L M N
y = J L O M N P
The longest common subsequence in x and y = JLMN.
The formula of Edit distance using LCS is :
d(x, y) = length of string x + length of string y – 2 X (length of LCS)
Here,
length of string x = 5,
length of string y = 6,
length of LCS = 4,
So,
d(x, y) = 5 + 6 – 2 X (4)
= 11 – 8
= 3

The distance axioms on the Edit distance may be verified as follows :

1. Non-negativity : To convert one string into another string at least zero or more insertions and/or deletions are necessary. So, the Edit distance can never be negative.
2. Zero distance : Only in the case of two identical strings, the Edit distance will be zero.
3. Symmetry : The edit distance for converting string x into string y will be the same for converting string y into string x as the sequence of insertions and deletions can be reversed.
4. Triangle inequality : The sum of the number of edit operations required for converting string x into string z and then string z into string y can never be less than the number of edit operations required for converting the string x directly into the string y.

Hamming Distance

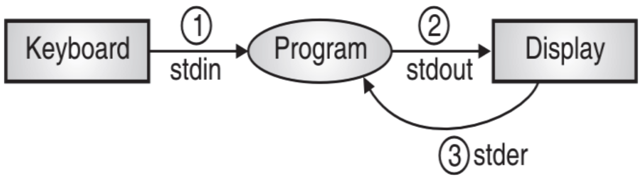
Hamming distance is applicable in the space of vectors. Hamming distance between two vectors is the number of components in which they differ from each other.
– For example, let us consider the following two vectors :

$$\begin{array}{lcccccc} x = & 1 & 0 & 0 & 0 & 1 & 1 \\ y = & 1 & 1 & 1 & 0 & 1 & 0 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & \textcircled{1} & \textcircled{2} & \textcircled{3} & \textcircled{4} & \textcircled{5} & \textcircled{6} \end{array} \rightarrow \text{Positions}$$

– The Hamming distance between the above two vectors is 3 because components at positions 2, 3 and 6 are different.
– The distance axioms on the Hamming distance may be verified as follows :
1. Non-negativity : Any two vectors will differ in at least zero or more component positions. So, the Hamming
distance can never be negative.
2. Zero distance : Only in the case of two identical vectors, the Hamming distance will be zero.
3. Symmetry : The Hamming distance will be the same whether x is compared with y or y is compared with x.
4. Triangle inequality : The number of differences between x and z, plus the number of differences between z and y can never be less than the number of differences between x and y.

Stream Computing

– Stream computing is useful in real time system like count of items placed on a conveyor belt.
– IBM announced stream computing system in 2007, which runs 800 microprocessors and it enables to software
applications to get split to task and rearrange data into answer.
– AT1 technologies derives stream computing with Graphical Processors (GPUs) working with high performance with
low latency CPU to resolve computational issues.
– AT1 preferred stream computing to run application on GPU instead of CPU.



A Stream - Clustering Algorithm

– BDMO Algorithm has complex structures and it is designed in approach to give guaranteed performance even in worst
case.
– BDMO designed by B. Bahcock, M. Datar, R. Motwani and L. OCallaghan.

Details of BDMO algorithm

(i) Stream of data are initially partitioned and later summarized with help of bucket size and bucket is a power of two.
(ii) Bucket size has few restrictions size of buckets are one or two of each size within a limit. Required bucket may start
with sized or twice to previous for example bucket size required are 3, 6, 12, 24, 48 and so on.
(iii) Bucket size are restrained in some scenario, buckets mostly O (log N).
(iv) Bucket consists with contents like size, timestamp, number of points in cluster, centriod etc.
Few well – known algorithm for data stream clustering are :
(a) Small – Spaces algorithm
(b) BIRCH
(c) COBWEB
(d) C2ICM

Initializing and Merging Buckets

A small size 'p' is chosen for bucket where p is power of 2. Timestamp of this bucket belongs to a timestamp of most
recent points of bucket.
– Clustering of these points done by specific strategy. Method preferred for clustering at initial stage provide the
centriod or clustroids, it becomes record for each cluster.
Let,
* 'p' be smallest bucket size.
* Every p point, creates a new bucket, where bucket is time stamped along with cluster points.
* Any bucket older than N is dropped
* If number of buckets are 3 of size p
p $\rightarrow$ merge oldest two
– Then propagated merge may be like (2_p, 4_p, …).
– While merging buckets a new bucked created by review of sequence of buckets.
– If any bucket with more timestamp than N time unit prior to current time, at such scenario nothing will be in window
of the bucket such bucket will be dropped.
– If we created p bucket then two of three oldest bucket will get merged. The newly merged bucket size nearly z_p, as we
needed to merge buckets with increasing sizes.
– To merge two consecutive buckets we need size of bucket twice than size of 2 buckets going to merge. Timestamp of
newly merged bucket is most recent timestamp from 2 consecutive buckets. By computing few parameters decision of
cluster merging is taken.
– Let, k-means Euclidean. A cluster represent with number of points (n) and centriod (c).
Put p = k, or larger – k-means clustering while creating bucket
To merge, $n = n_1 + n_2$, $c = n_1c_1 + n_2c_2$
$n_1 + n_2$
– Let, a non Euclidean, a cluster represented using clusteroid and CSD. To choose new clusteroid while merging, k-points
furthest are selected from clusteroids.

$$CSD_m(P) = CSD_1(P) + N_2(d_2(P, c_1) + d_2(c_1, c_2)) + CSD_2(c_2)$$

Answering Queries

– Given m, choose the smallest set of bucket such that it covers the most recent m points. At most 2m points.
– Bucket construction and solution generation are the two steps used for quarry rewriting in a shared – variable bucket
algorithm, one of the efficient approaches for answering queries.