Design and Analysis of Algorithms

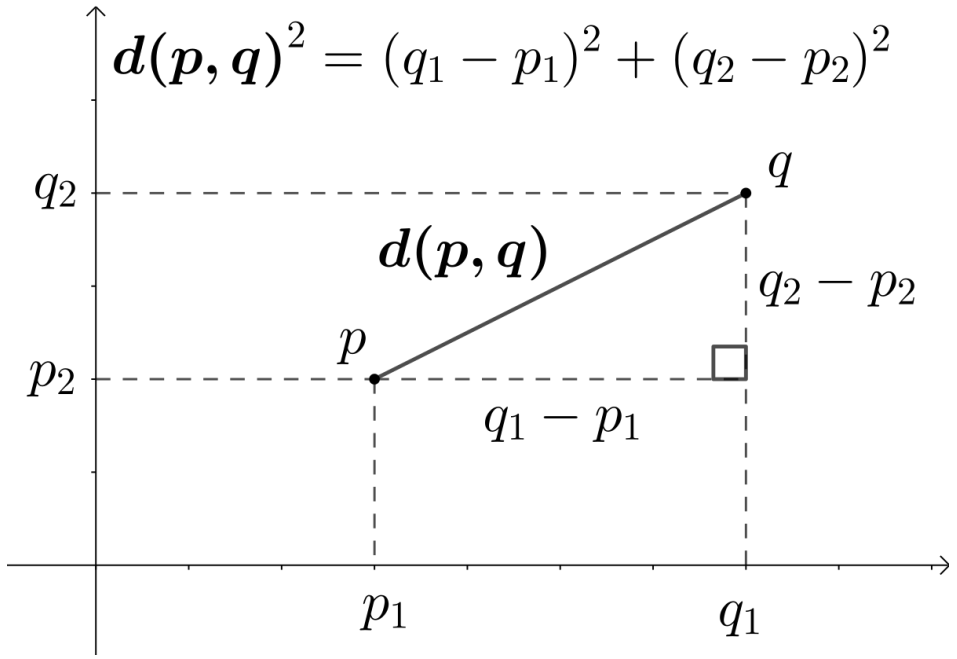
Lecture 04 – Working with Distances

Distances

The distance between two data instances (ie. points) is a very important concept.

Due to our shared background in high school mathematics, when we are asked about a distance we immediately think of points in space, a Cartesian coordinate system and a Euclidean distance d(p,q).

Taking a step back. Why do we need distances?



Distances

Taking a step back. Why do we need distances?

Distances help us analyze and solve problems about being near o far away.

Assume you are at a point near a rectangle. You want to reach the rectangle. What point on the rectangle is nearest to you? Why?

How can you conceptualize distance so that you can use it to solve problems?

Different problems need different conceptualizations of distances.

The common Euclidean distance is just one way of looking at this.

In problem solving we can choose any topology that correctly represent our problem.

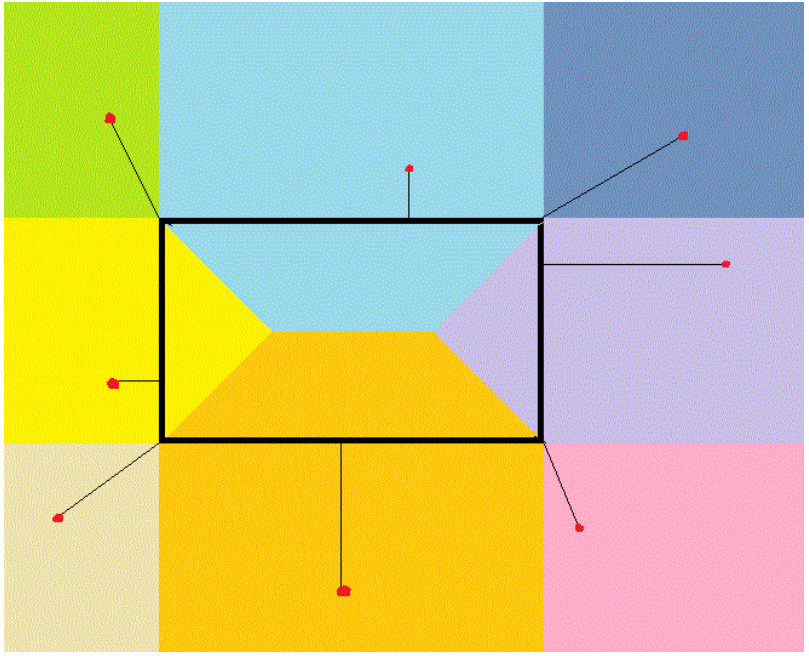
And then we try to define the best distance metric to represent our concept of a distance.

In mathematics, a topology is concerned with the properties of a geometric object that are preserved under continuous deformations, such as stretching, twisting, crumpling, and bending; that is, without closing holes, opening holes, tearing, gluing, or passing through itself.

These deformations are usually represented by mathematical transformations.

We all know about the Laplace transformation. It conveniently converts (transforms) linear differential equations into algebraic equations which are easier easier to solve.

Based on our problem space, and our data types we may need to use such transformations.



Distances

In problem solving we can choose any topology that correctly represent our problem.

And then we try to define the best distance metric to represent our concept of a distance.

Example. Find the nearest rectangle to “our” rectangle. How to we define the problem? Nearest means:

(1) Smallest distance between the “centers of mass” of two rectangles?

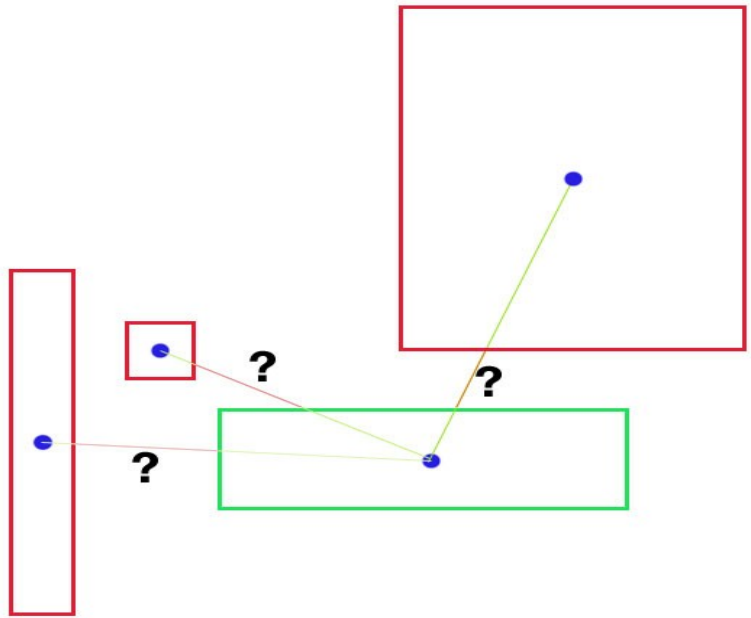
(2) Smallest distance between the edges?

In mathematics, a topology is concerned with the properties of a geometric object that are preserved under continuous deformations, such as stretching, twisting, crumpling, and bending; that is, without closing holes, opening holes, tearing, gluing, or passing through itself.

These deformations are usually represented by mathematical transformations.

We all know about the Laplace transformation. It conveniently converts (transforms) linear differential equations into algebraic equations which are easier easier to solve.

Based on our problem space, and our data types we may need to use such transformations.



Distances

What if our shapes were not necessarily rectangles, but the important part was the “center of mass.”

So that our problems about shapes could be about centers of mass.

The image to the right could well be subway stations in a city. Traveling from one district to the other would be “ride the subway train first, then take a bus.”

Then the shapes represents area served by buses taking you to a particular subway station.

Solving any travel problem could require you to conceptualize many if not all **permutations** of travel routes (not combinations; the order is important) because you could exchange modes of travel at some points.

However, you would still need to evaluate the distances between these points.

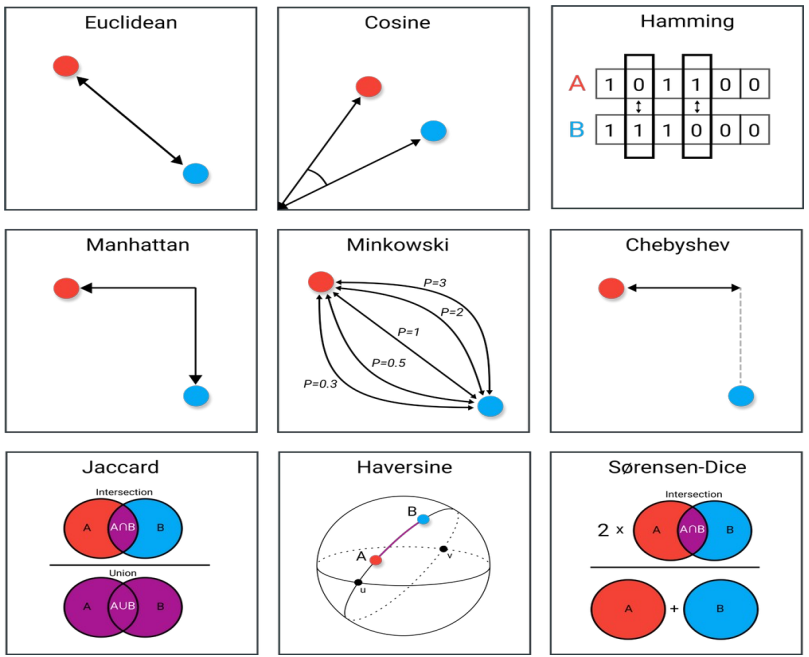
Regarding algorithms (and data structures) how we define our problem and our problem data is very important.

Distances

There are many ways to show distances.

We will discuss some ways to represent a distance.

For each method we will also discuss practical applications.



Distances

The Euclidean distance metric assumes a Cartesian coordinate system.

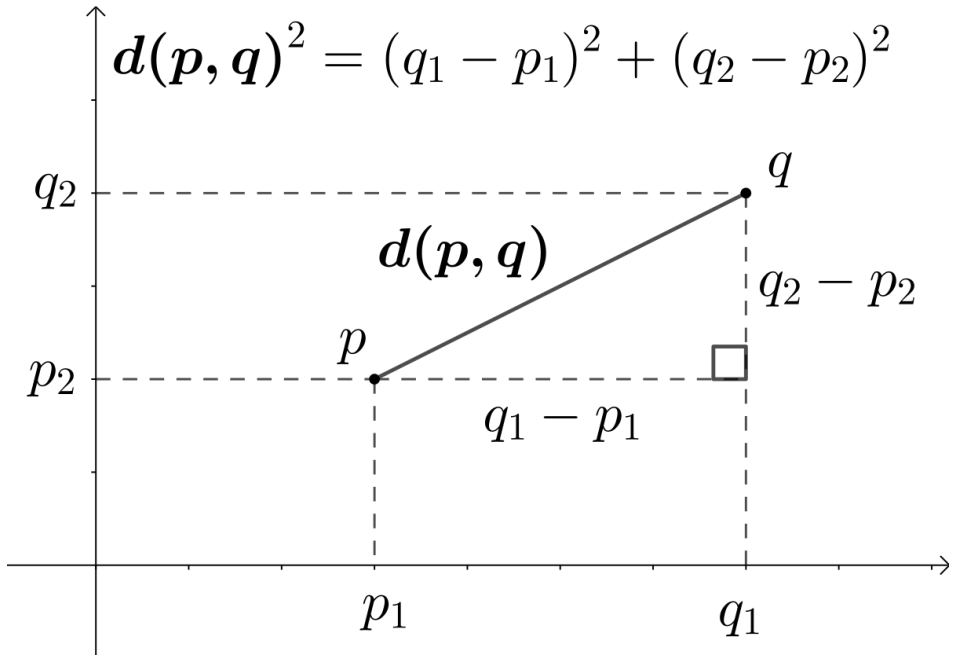
The distance is calculated from the Cartesian coordinates of the points using the Pythagorean theorem.

The advantage is that it is very simple to conceptualize. It also has some physical meaning.

However, there is an underlying assumption that the units of the coordinates are the same. If your data is multi-dimensional and each dimension has a different unit, then the distance is going to be **skewed**. To solve this disadvantage you need to scale your data points in a per-unit fashion.

There are multiple ways to scale dimensional data. Linear scaling or any particular non-linear scaling method may be better for your use case.

Moreover, Euclidean distance also gives us an implicit understanding that



Distances

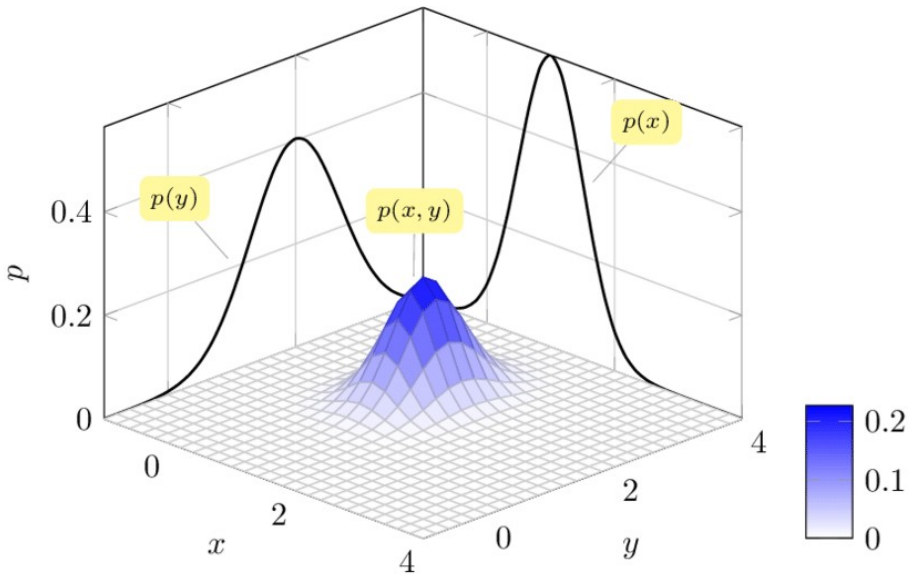
The Euclidean distance metric assumes a Cartesian coordinate system.

This assumption also gives us an implicit understanding that data in every dimension behaves similarly to what we have covered in every single variable concept in every “introduction to” course.

This is particularly tricky about distribution of data in each dimension. Even with numerical data, their distributions need not be similar, or even of the same type.

After normalization we could have two dimensions with normal distribution but their variances could be very different.

After normalization we could have two dimensions one of which is normally distributed and the other non-normal distributed.



Distances

Euclidean metrics are also hard to use when we have missing data.

The way you discard data points with missing data or impute your missing data could change your results significantly.

How can you impute income in the figure?

So Euclidean distance metrics are useful, but not necessarily best for all types of numeric data.

For non-numerical data points and even numerical data points, it is in many cases pointless to define a Euclidean distance metric.

How can you define distance between categorical data points? **Example.** Distance between male and female. Is coding Male=0, Female=1 OK?

Example. What is the meaning between distances of income? Is the Euclidean distance of incomes between 10.000 and 20.000 the same as the distance between 500.000 and 510.000?

Index	Age	Sex	Income
1	NA	M	NA
2	39	NA	75000
3	NA	NA	NA
4	28	F	50000
...	...	...	...
10000	18	F	NA

Distances

Calculation of the Euclidean metric itself requires computation.

Luckily most, if not all, computer systems have a direct instruction to calculate a floating point square root, so the real cost of a sqrt operation is **O(1)**.

Therefore, calculating a single distance of dimension d is **O(d+1) ~ O(d)** due to the d subtractions/multiplications/additions and the single square root. For n points, to calculate distances to each other, this becomes **O(d x n^2).**

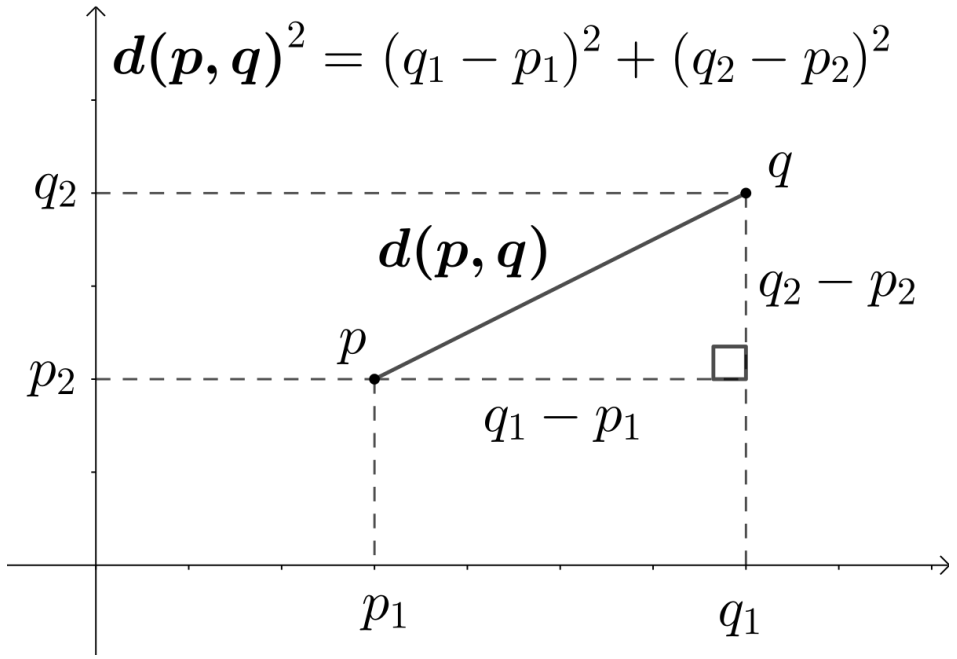
So if you have a large number of dimensions, and a large data set then choosing the Euclidean distance may become problematic in just calculation of the distances.

In particular **if d~n then O(dxn^2) ~ O(n^3)**.

You will still have the complexity of the algorithm that used the distance values in addition to that.

If your data set is dynamic, adding and removing items all the time, this cost could be incremental. You could calculate the distance of a new data point to all existing points (ie. a distances vector) with **O(dxn)**, and then update the distance values data-set (ie. add a column to the distances matrix).

However, based on your algorithm you might need to re-calculate something else as well.



Distances

The cosine distance metric is simply defined as the cosine of the angle between the vectors for two points.

It is a metric about **similarity of directions, not exact values or sizes of distances**.

Example.

Points x1(1,1,0), y1(1,0,1), and z1(0,1,1); all with magnitude sqrt(2).

D(x1,y1) = D(x1,z1) = D(y1,z1) = ½.

The differences in their directions are the same.

Points x2(2,2,0), y2(2,0,2), and z2(0,2,2); all with magnitude sqrt(8)=2sqrt(2).

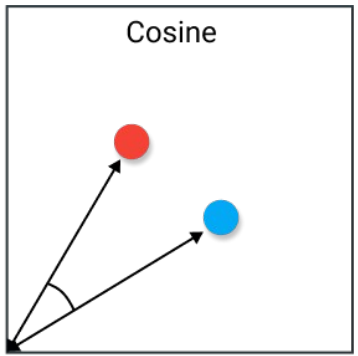
They are in the same direction but with different sizes.

D(x2,y2) = D(x2,z2) = D(y2,z2)= ½ as well.

Because their directions did not change, the difference in their directions are also the same.

How about D(x1,y2), etc? D(x1,y2) = ½ again.

If you want to play with some numbers, you can use – https://onlinemschool.com/math/assistance/vector/angl/



$$D(x,y) = cos(\theta) = \frac{x \cdot y}{\|x\| \, \|y\|}$$

Distances

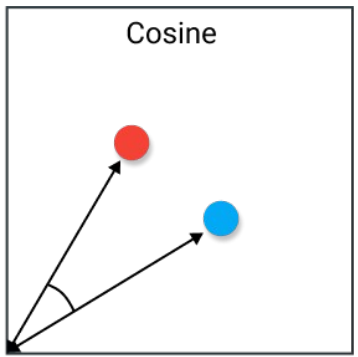
Note that this metric is very different from the Euclidean metric.

It does not suffer much from high-dimensionality or difference in distributions of dimensions.

However it requires the calculation of **three** dot products. ||x|| and ||y|| are vector magnitudes, dot products onto themselves. This is therefore **O(3d) ~ O(d)**.

However, if you need to process n data points, **you can reuse the magnitudes**. Incremental calculation of a set of distances will be O(dxn) at a step as well.

From a big-oh point of view, Euclidean and Cosine distances are equally computation intensive.



$$D(x,y) = cos(\theta) = \frac{x \cdot y}{\|x\| \, \|y\|}$$

Distances

The Hamming distance metric was invented in telecommunications area.

In 1950, Hamming published a paper that would serve as the basis for modern coding theory. He postulated that it was possible to not only detect, but correct errors in bit strings by **calculating the number of bits disparate between valid codes and the erroneous code.** This came to be known as Hamming Distance.

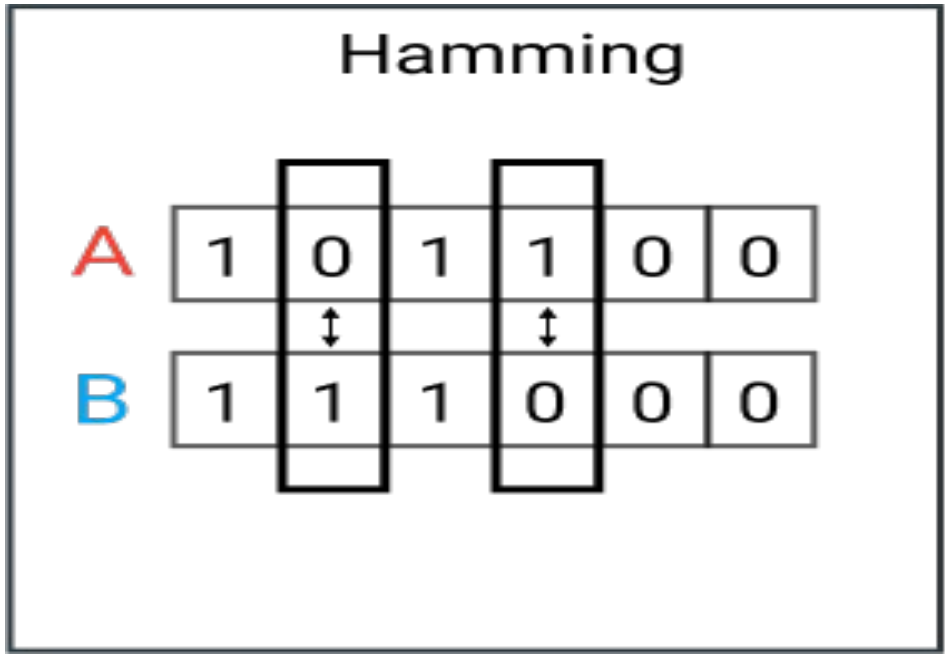
We can extend this to any categorical variable in a data point.

Example. Consider data points x(gender="male", position=" undergrad student", age="21 to 30"), y(gender="male", position="grad student", age="21 to 30") and z(gender="female", position="grad student", age="21 to 30") in a marketing study.

D(x,y) = 1 (due to difference in position field)

D(x,z) = 2 (due to difference in all gender and position fields).

D(y,z) = 1 (due to difference in gender field)



Distances

The Hamming distance is very easy to compute.

For dimension d, all it requires is d comparisons, so it is in the worst case **O(d)**.

Why not exactly O(d)?

For bitwise operations this can be done **very very fast.**

If you implement your categorical data as bit-fields (or enumeration data types which are compiled as such) then you will have to consider how many CPU-words your bitfields fit in.

Example. x(gender,position,age). Gender has two options = 1 bit. Suppose position has 5 options = 3 bits. Age groups are 10 in total = 4 bits. Total of 8 bits.

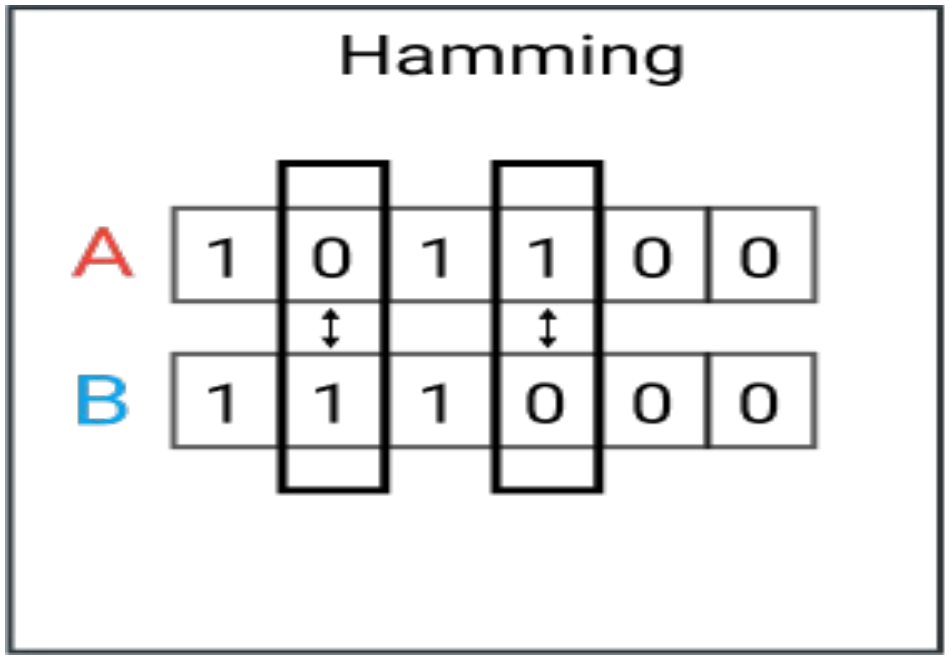
Your typical integer type is 64 bits. So **you could bitwise encode 8 data points in an integer.**

An easy way to compute this is to XOR the two strings and count the number of 1 bits in the results. That can be done, on recent computers, with a 5 instruction loop per 64 bits.

So for this particular case, you have **O(5/8) ~ O(1)**.

Also consider the I/O requirements. All this is done in memory, usually in the CPU cache, so with modern CPUs you can do maybe 60 billion bit comparisons per second.

So the Hamming distance is not a concern for computational complexity if done using bit fields.

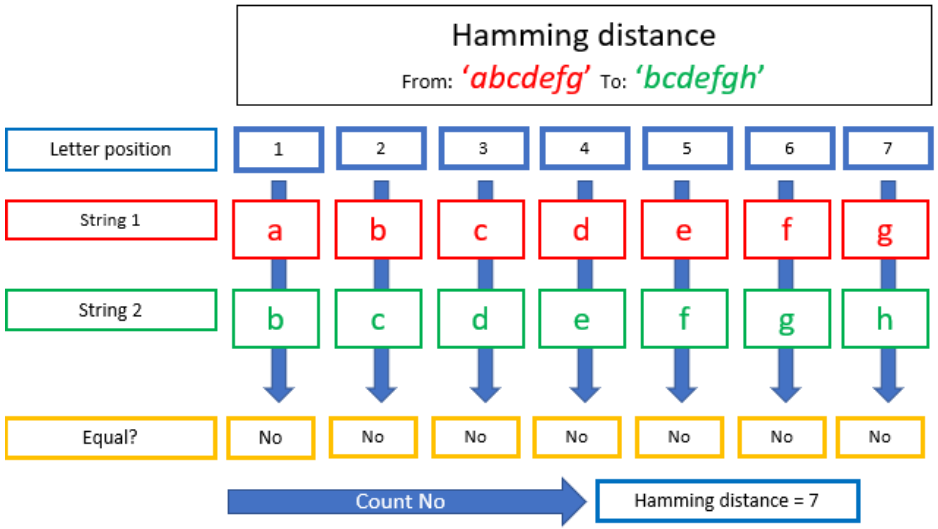


Distances

Hamming distance is also used to compare textual data.

- However it is not always the best method, based on your intention.
- The 7-character strings “abcdefg” and “bcdefgh” are very similar, but their Hamming distance is 7.
- We will also discuss some other textual distances.

Also note that the metric is by definition working on same size data points.



Distances

An alternate to Hamming distance is the Levenshtein distance.

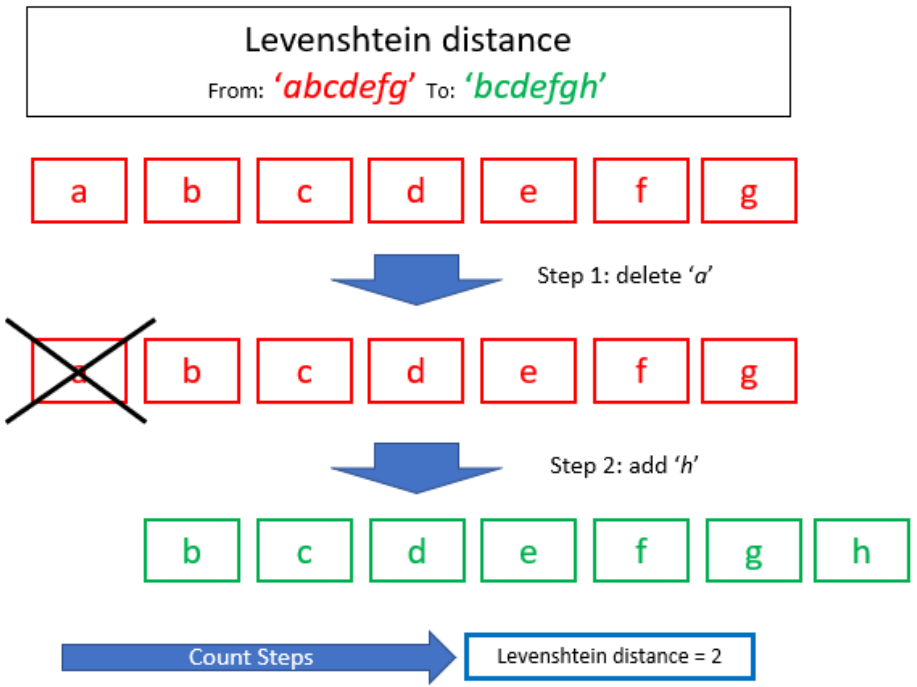
It is a measure of the number of operations needed to transform one string (of letters, bits, categorical values, etc) into the other.

The types of operations are (1) removing/deleting, and (2) adding as well as (3) transforming/substituting.

This is a much more meaningful distance for many applications, however calculation is not straightforward.

For strings of sizes n1 and n2, the algorithm has complexity of O (n1 x n2). If n1 ~ n2 ~ n, then we could say it is **O (n^2)**.

Compare this to O(1) for Hamming distance!



Distances

Another alternative to Hamming distance is to transform the a discrete and finite feature set into a coordinate system and use the Cosine transform.

Step 1. Enumerate the discrete and finite state-space for the features.

Step 2. Use each possible value of this state-space as a coordinate axis. The values for these axes could be designed as 0-1 (does not have, has) or as a count (0 and above, strictly integers).

Step 3. Calculate the cosine distance as a similarity measure.

Steps 1 and 2 are not necessarily easy.

But they are to be done once and for all.

Step 1 creating a with-duplicate list of the state-space. This is based on the type of discrete features.

Step 2 is removing the duplicates to have a no-duplicate list of the state-space. This is expected to be done in O(n).

Discussion. How can you implement Step 2, and with what type of data structure? Would the choice of data structure change the worst case performance.

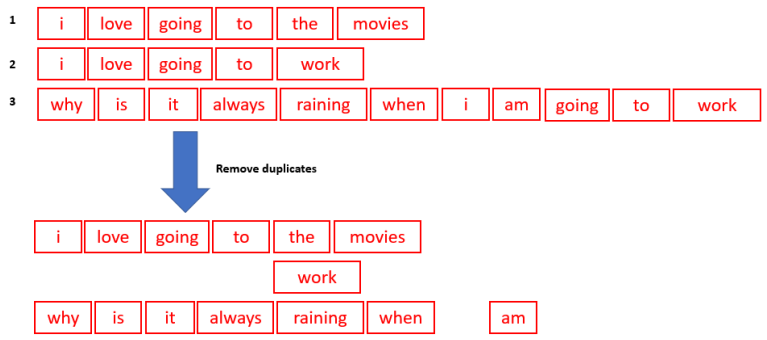
Step 3 is very easy for a binary design

A bit-wise dot product x.y tells you how many 1 bits x has in common with y.

This is basically the logical AND operation. Hence its computational complexity is similar to the calculation of Hamming distance.

Note that the variance on the distance distribution can be large for this metric.

Distances



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

	1	2	3
i	1	1	1
love	1	1	0
going	1	1	1
to	1	1	1
the	1	0	0
movies	1	0	0
work	0	1	1
why	0	0	1
is	0	0	1
it	0	0	1
always	0	0	1
raining	0	0	1
when	0	0	1
am	0	0	1

Distances

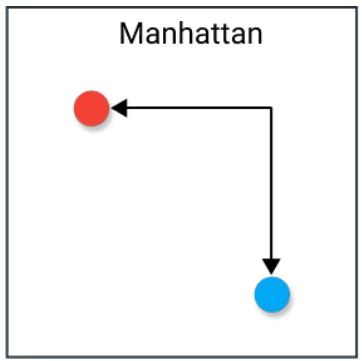
An alternative to Euclidean distance is the Manhattan Distance.

This is inspired by the distance on has to walk in a city block, because you have to use the perpendicular roads. Such geometric interpretation is due to the mathematician Hermann Minkowski who studied non-Euclidean geometries in the 19th century.

In fact Minkowski studied distances in detail. Manhattan distance is better defined as the 1st order, and Euclidean distance as the 2nd order Minkowski distances.

From a daily point of view, Euclidean distance is for designed for flight, Manhattan distance is for urban transport.

This metric is useful when mixing normalized numerical distances with categorical/discrete attributes.



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

Distances

For Manhattan distance defined for d dimensions (ie. d=2 in the taxicab analogy).

There are several (but finite) paths between two points.

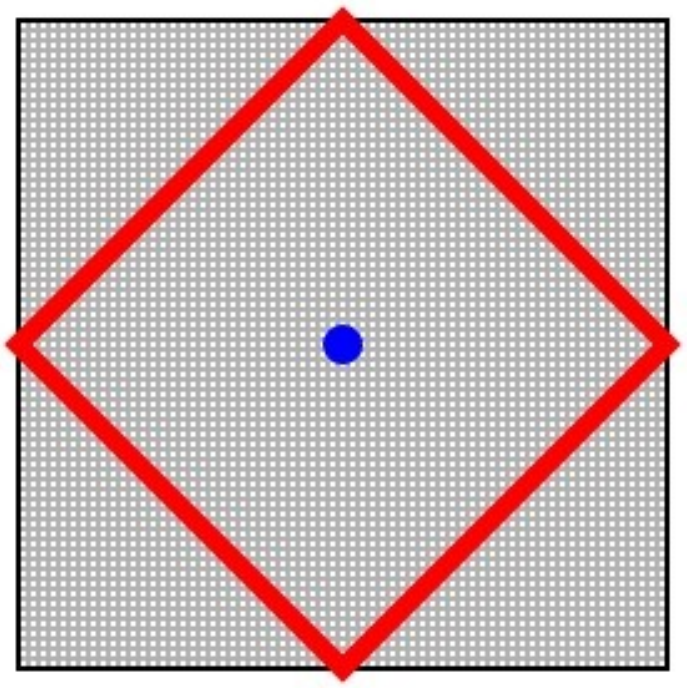
A straight path with length equal to Manhattan distance has d types of permitted moves. For d=2, vertical (one direction), horizontal (one direction)

For a given point, the other point at a given Manhattan distance lies in a square around the given point.

Example.

Consider the chess board and a pawn. The permitted chess moves are (1) one step advance, (2) two step advance at the beginning (3) ordinary diagonal capture, and (4) "en passant" capture as response to two step advance.

The Manhattan distances covered in these chess moves are 1, 2, 2, and 2 respectively.



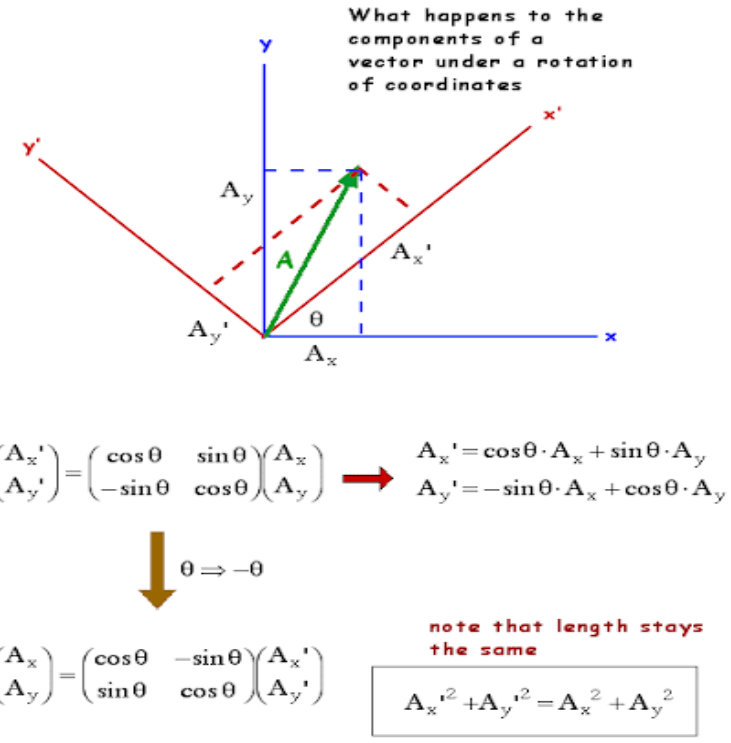
Distances

When we rotate a coordinate system…

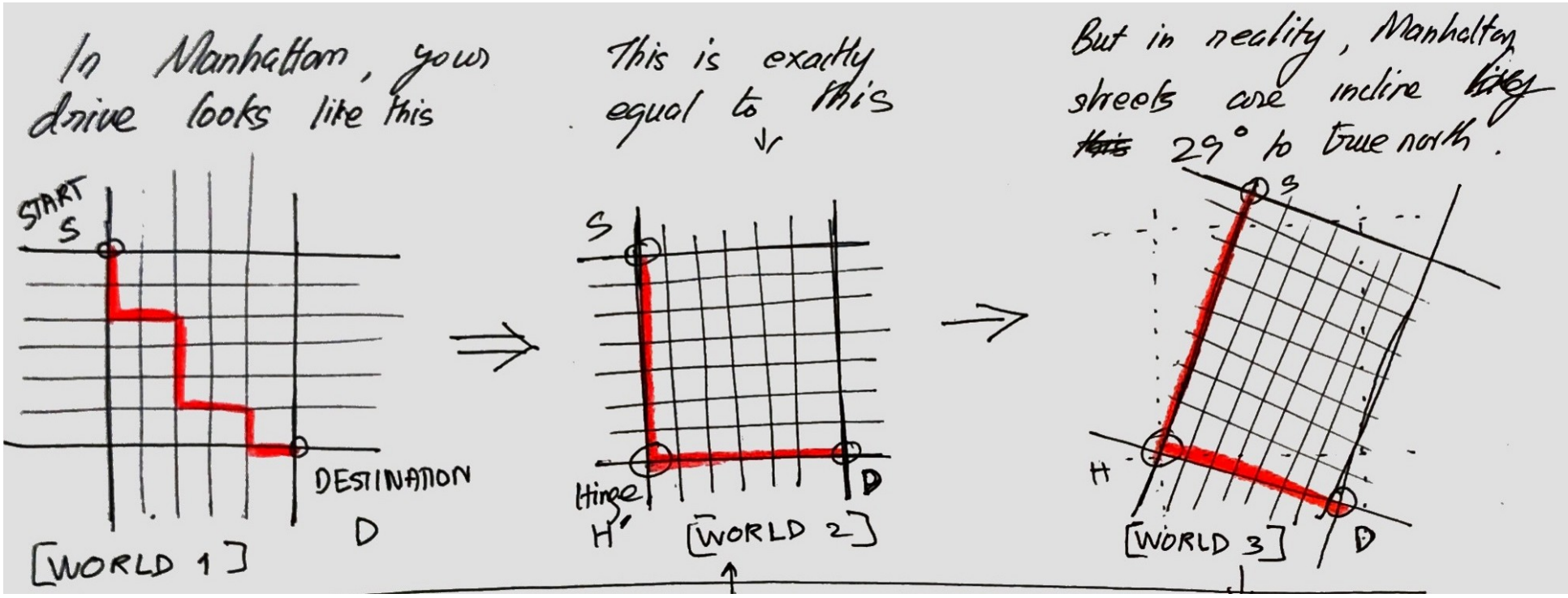
- Euclidean distances remain the same.
- Manhattan distances change, so they have to be re-calculated.

In fact, Manhattan distances do not apply to Manhattan itself.

- See – https://bit.ly/2ZCU6gW



Distances


In Manhattan, your drive looks like this
START
S
DESTINATION
D
[WORLD 1]
This is exactly equal to this
S
Hinge
H'
D
[WORLD 2]
But in reality, Manhattan streets are incline ~~kinky~~ ~~Hats~~ 29° to true north.
S
H
D
[WORLD 3]

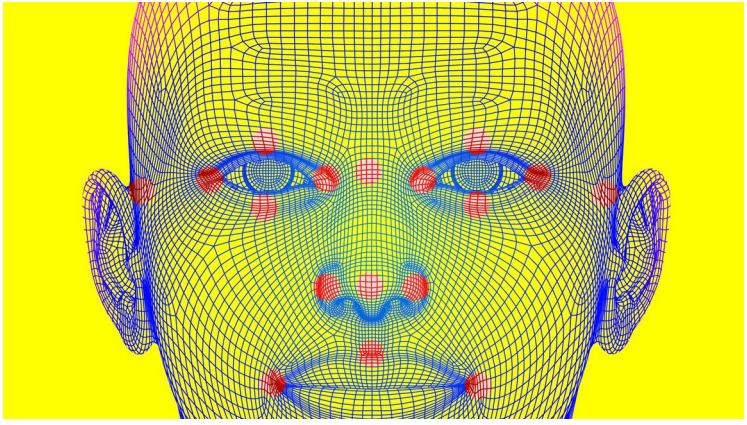
Distances

This metric is useful when we need to calculate distances within categorical/discrete attributes.

Many multi-criteria decision making problems have finite options for each category. So we end up with a spatial search.

Basic face recognition algorithms convert each an image into a set of shape points, and then calculate the Manhattan distances between selected particular points (landmark points).

Using these distances, one can classify images (ie. smiling/non-smiling) or compare with reference image (ie. Bora, non-Bora).



Distances

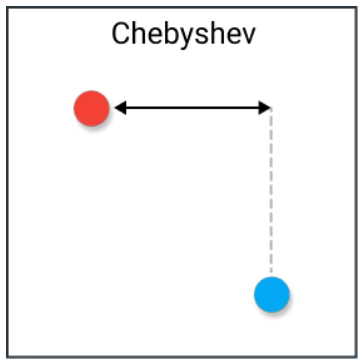
Another alternative to Euclidean distance metric is the Chebyshev Distance.

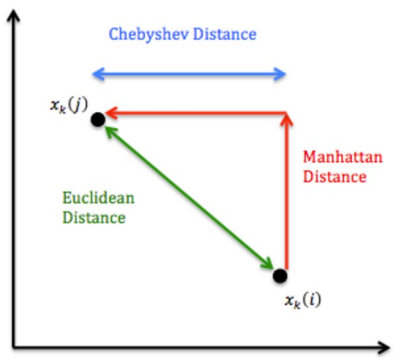
Chebyshev distance is defined as the greatest of difference between two vectors along any coordinate dimension. In other words, it is simply the maximum distance along one axis.

Chebyshev is typically used in very specific use-cases.

Most of these use cases involve the number of movements (ie. steps) needed for some operation.

Examples include control of robotic arms, warehouse operations, stops to make in distribution logistics, movements in computer games.





Distances

The Minkowski distance is the generalization of Manhattan, Euclideann and Chebyshev distances.

The Minkowski distance is a generalization of the formula for these distances. They all involve a power p and a root at power p.

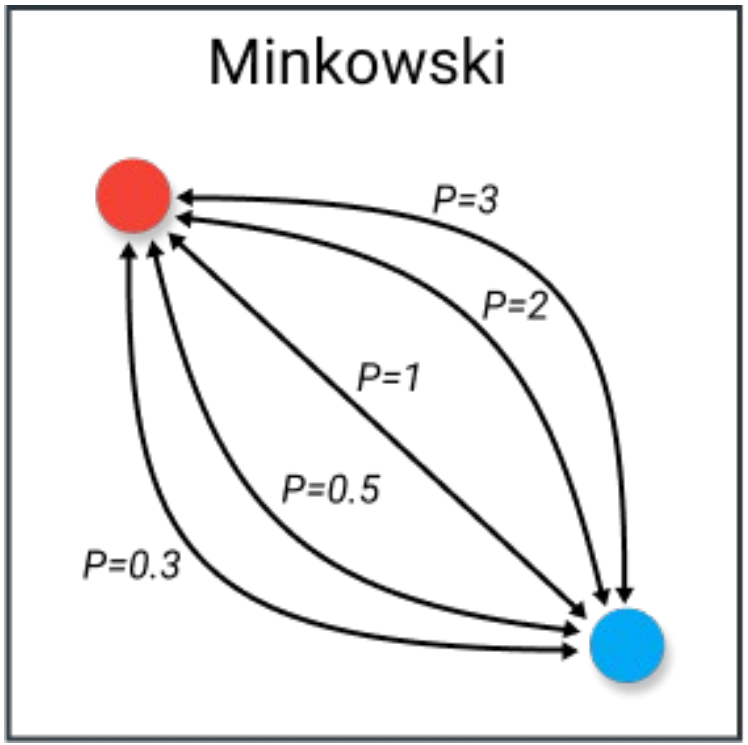
For p=1 we get Manhattan, for p=2 we get Euclidean, and p=infinity we get Chebyshev (ie. the largest valued dimension dominates the others in the root taking).

How about the same for any p?

Why? In **practical applications (not mathematical theory!)**, all these metrics are trying to fit to the real-world. If the fitness is better at a particular power p, we **choose** to use that power.

The idea of Minkowski distance is to calibrate our algorithm's success using different values of p and deciding on the better/best value of p.

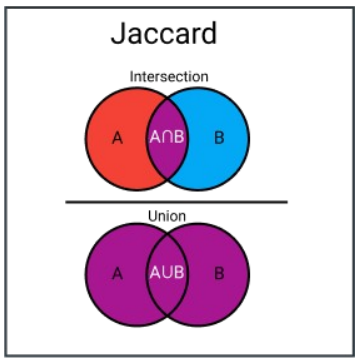
However such calibration is **not an easy feat**.



Distances

The Jaccard index is a distance metric designed to measure the similarity of two sets.

- It is the size of the intersection divided by the size of the union of the sample sets.
- The index can be 1.0 at most and 0.0 at minimum.
- The distance is simply 1 minus the index.

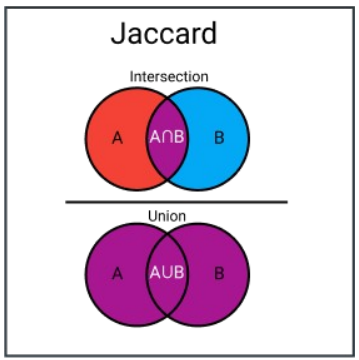


$$D(x, y) = 1 - \frac{|x \cap y|}{|y \cup x|}$$

Distances

It is used to compare data points with large and possibly different number of features for similarity.

- Compare images of automobiles to decide which is similar to which, and also identify outliers.
- Compare company balance sheets to decide which has similar performance to which, and also identify outliers.
- Compare two homework assignments and decide if one is a derivative of the other.

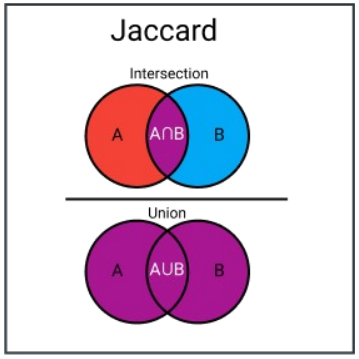


$$D(x, y) = 1 - \frac{|x \cap y|}{|y \cup x|}$$

Distances

However, because it is used with large feature spaces, the size of the union can be very large compared to the size of the intersection.

- This can end up the index values being very close to zero, hence the distances being very close to 1.
- So the statistical distribution of the distance can be heavily skewed. You might consider some transformation on the distance metric before using.
- Checking a graphical representation for the frequency distribution might be useful for this decision.

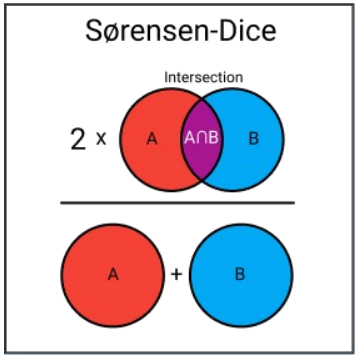


$$D(x, y) = 1 - \frac{|x \cap y|}{|y \cup x|}$$

Distances

The Sorensen-Dice index is similar to the Jaccard index.

- It is again a percentage of overlap between two sets.
- However we need not compute the union set. Instead we just use the sizes of all sets.
- The distance is again from 0 to 1.



$$D(x, y) = \frac{2\,|x \cap y|}{|x| + |y|}$$

Distances

The Havershine distance is the distance between two points on the surface of a sphere.

- This is a curved Euclidean distance.
- It is often used in navigation problems where the curvature of Earth has to be taken into account.
- Many decisions using positions on Earth such as site selection or long distance logistics planning also prefer this distance metric.
- See this post with code examples (in Go) – https://bit.ly/3GOsR46