## **Metric Space**

A mapping is said to be Metric if it satisfies these 4 conditions.

1. Distance (x,y) >= 0 ; for all x,y belongs to X (d is never negative)
2. Distance (x,y) = 0 if x == y ; for all x,y belongs to X (d will be 0 if both points are same)
3. Distance (x,y) == (y,x) ; for all x,y belongs to X (d is same both ways)
4. Distance (x,y) <= D(x,z) + D(z,y) ; for all x,y belongs to X (if we take z in between x,y, then sum of d(x,z) and d(z,y) > d(x,y)) [traingle inequality]

Example – On a number line, let’s take x = 2 and y = 10. And a new point z = 6

1. D(x,y) = 8 (non-negative)
2. If both x and y were the same then d would be = 0
3. D(2,10) == D(10,2) = 8
4. D(x,z) = 4 + D(z,y) = 4 =======> 8 and d(x,y) = 8 [ it is equal (<=)]

**Since x and y satisfy all 4 conditions, we can say the set that they belong to (real number or integer let's say) is Metric.**

## **Nearest Neighbors Motivation**

Today as users consume more and more information from the internet at a moment’s notice, there is an increasing need for efficient ways to do search. This is why “Nearest Neighbor” has become a hot research topic, in order to increase the chance of users to find the information they are looking for in reasonable time.

## **NMSLIB vs ANNOY**

## **Annoy**

In Annoy, in order to construct the index we create a forest (aka many trees) Each tree is constructed in the following way, we pick two points at random and split the space into two by their hyperplane, we keep splitting into the subspaces recursively until the points associated with a node is small enough.

In order to search the constructed index, the forest is traversed in order to obtain a set of candidate points from which the closest to the query point is returned.

### Annoy Pros

* Decouple index creation from loading them, so you can pass around indexes as files and map them into memory quickly.
* We can tune the parameters to change the accuracy/speed tradeoff.
* It has the ability to use static files as indexes, this means you can share indexes across processes.

### Annoy Cons

* The exact nearest neighbor might be across the boundary to one of the neighboring cells.
* No support for GPU processing.
* No support for batch processing, so in order to increase throughput “further hacking is required”.
* Cant incrementally add points to it ([annoy2](https://github.com/Houzz/annoy2) tries to fix this).

## 

## **Hierarchical Navigable Small World Graphs (NMSLIB)**

The intuition of this method is as follows, in order to reduce the search time on a graph we would want our graph to have an average path.

Many real-world graphs on average are highly clustered and tend to have nodes that are close to each other which are formally called small-world graph:

* highly transitive (community structure) it’s often hierarchical.
* small average distance ~log(N).

In order to search, we start at some entry point and iteratively traverse the graph. At each step of the traversal, the algorithm examines the distances from a query to the neighbors of a current base node and then selects as the next base node the adjacent node that minimizes the distance, while constantly keeping track of the best-discovered neighbors. The search is terminated when some stopping condition is met.

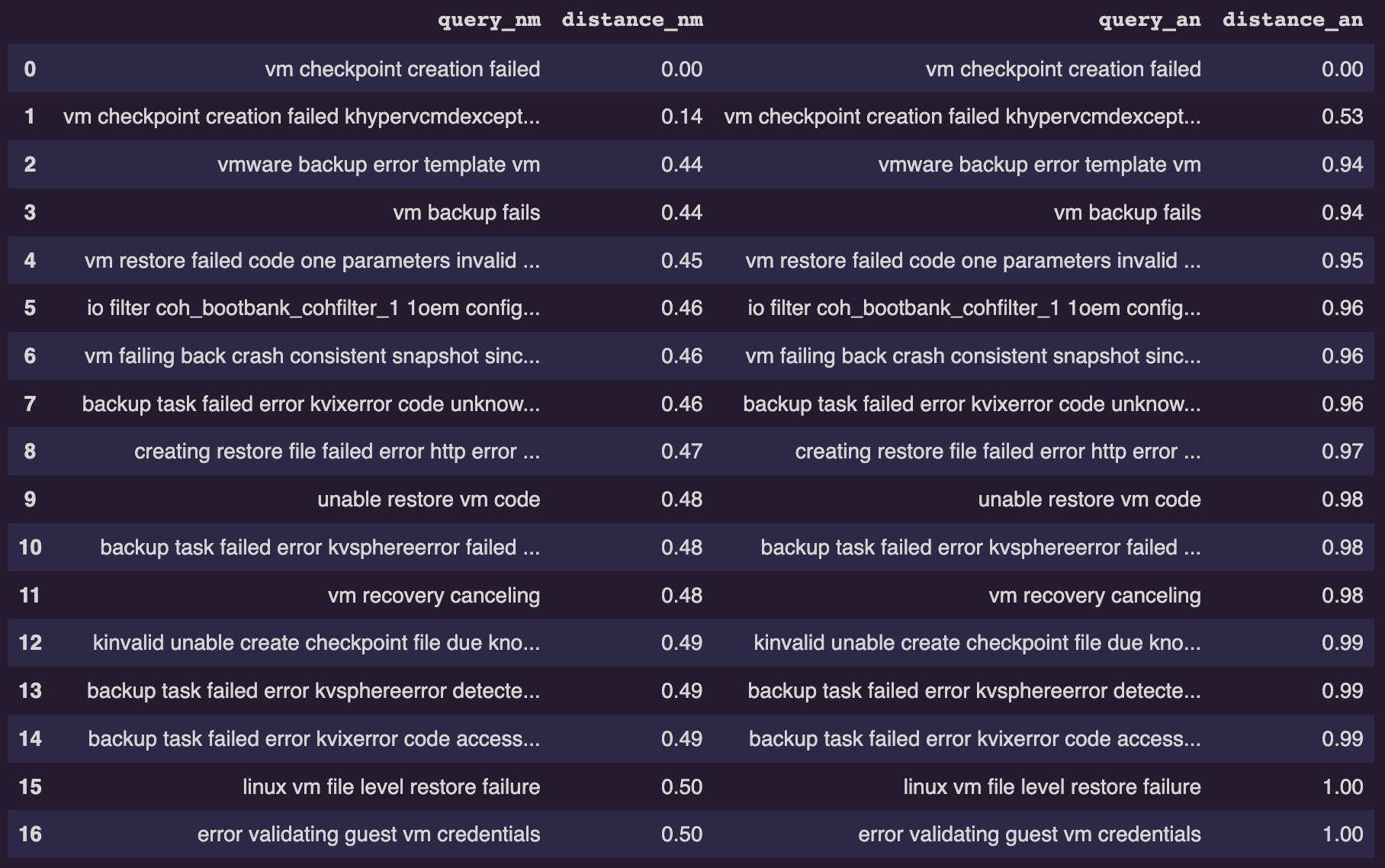
### Hierarchical Navigable Small World Graphs Pros

* We can tune the parameters to change the accuracy/speed tradeoff.
* Support batch queries.
* The NSW algorithm has polylogarithmic time complexity and can outperform rival algorithms on many real-world datasets.

### Hierarchical Navigable Small World Graphs Cons

* The exact nearest neighbor might be across the boundary to one of the neighboring cells.
* Cant incrementally add points to it.
* Require quite a lot of RAM.

## **Top 16 nearest neighbors for query (“vm checkpoint creation failed”) using both HNSW & ANNOY –**



## **Results**

There are no differences if we pick ~50 NNs. Differences come up only when we take higher number of NNs.

***Performance wise HNSW is faster than other algorithms including ANNOY in Querying the data.***

***Performance wise HNSW is slower than other algorithms including ANNOY in building the index.***

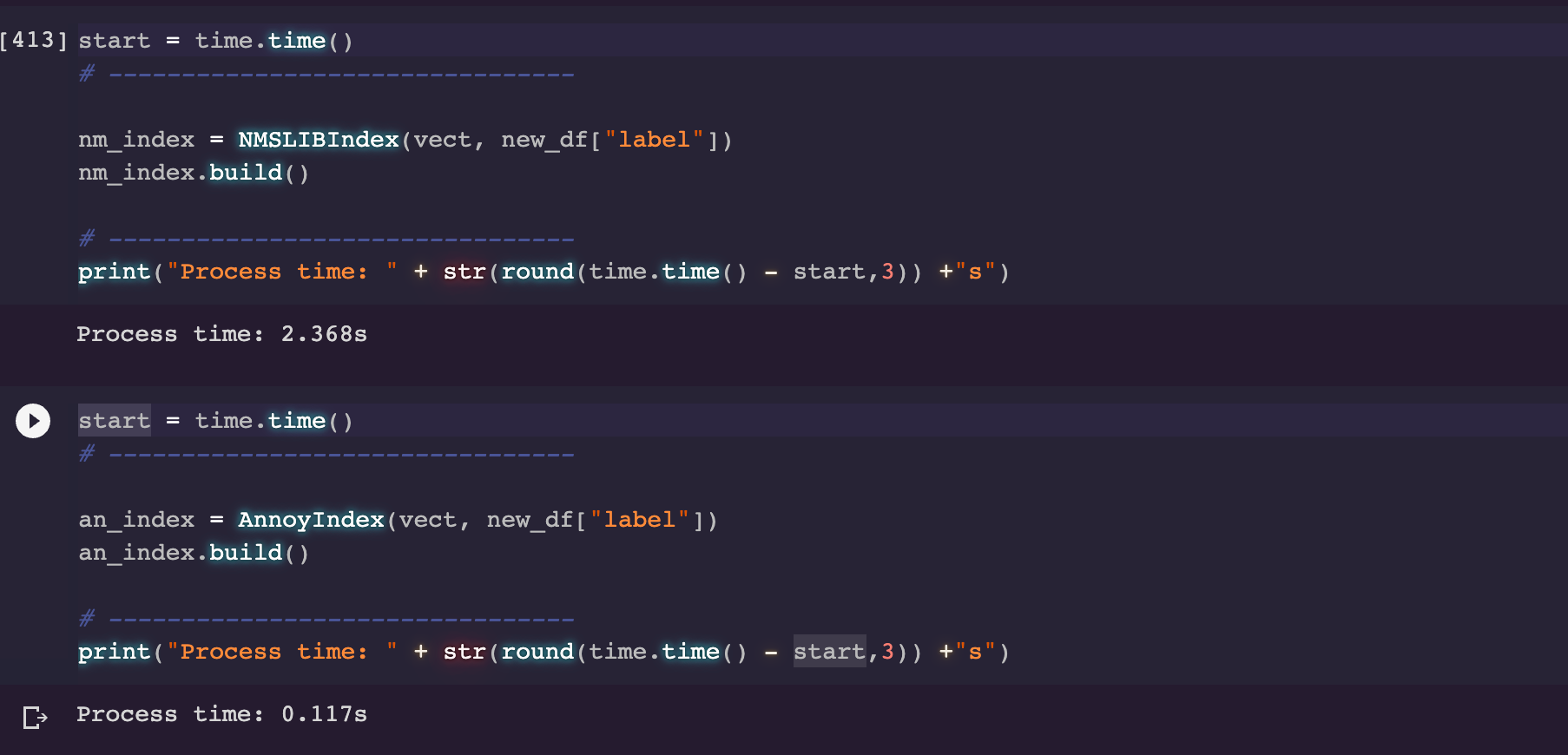
Here is the execution time for both indexes in getting the top 100 nearest neighbors for a single query and creating a dataframe as well –



NMSLIB – **0.003**s

ANNOY – **0.008**s

Here is the time for **building the index** for both techniques-



NMSLIB – **2.368**s

ANNOY – **0.117**s