Document Diversity

**Problem statement:**

Let’s say we have N (Large number) documents, we have to find very similar documents among them and remove them so that our resources are used optimally.

**Possible Approaches:**

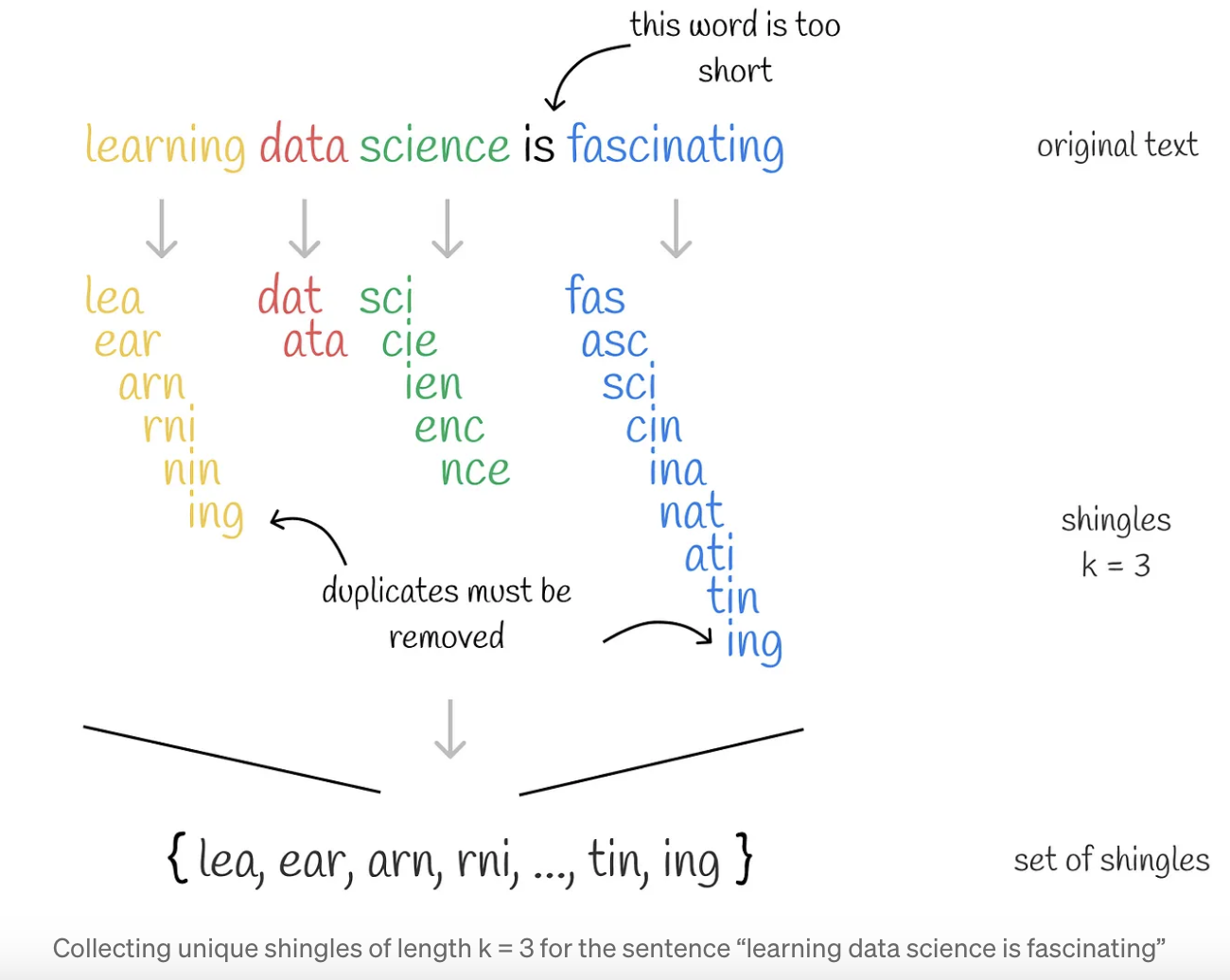
**Naive Approach:**

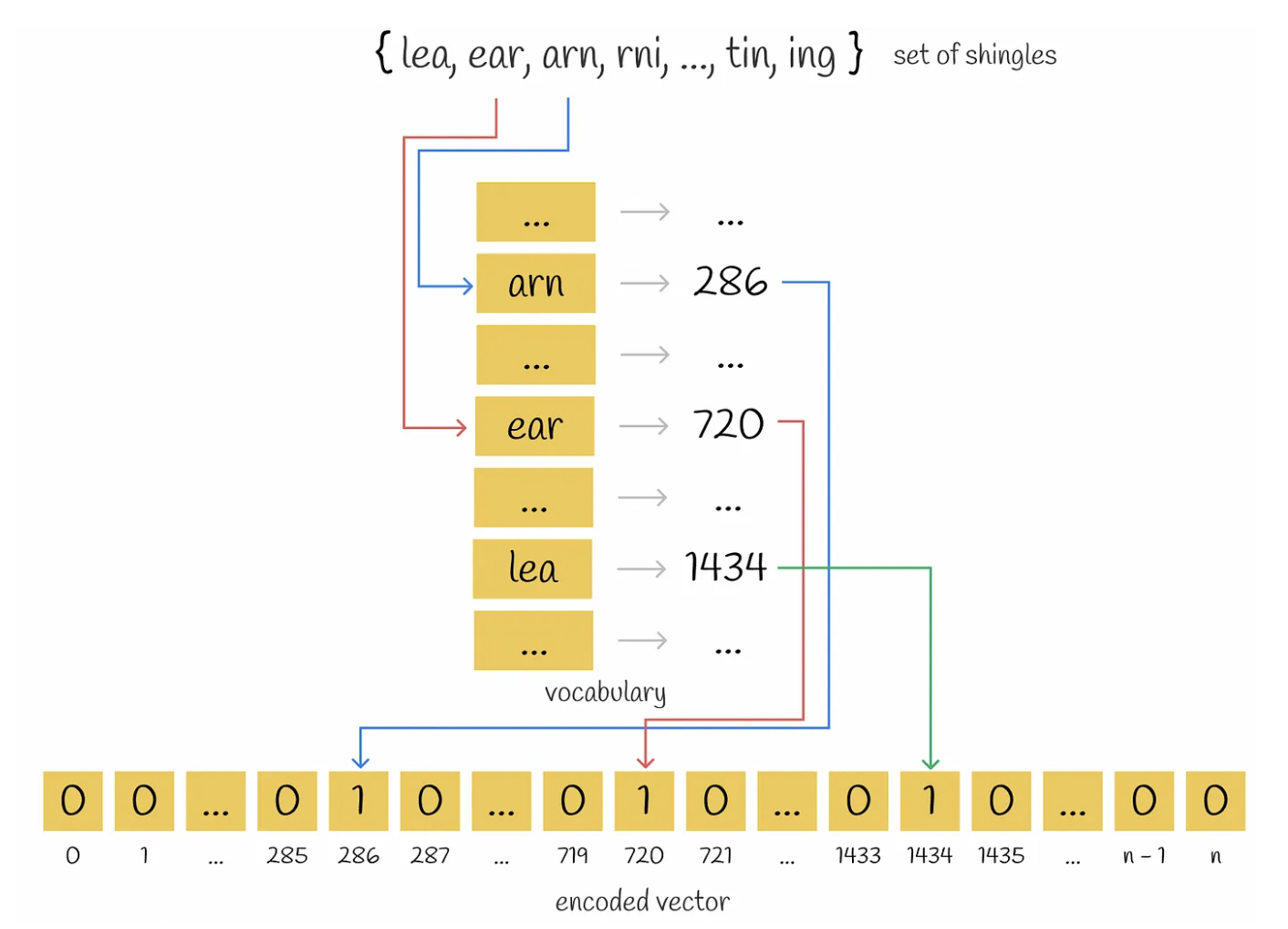
Compare every document with respect to other documents to find which documents are similar. This will have a total of nC2 comparisons.

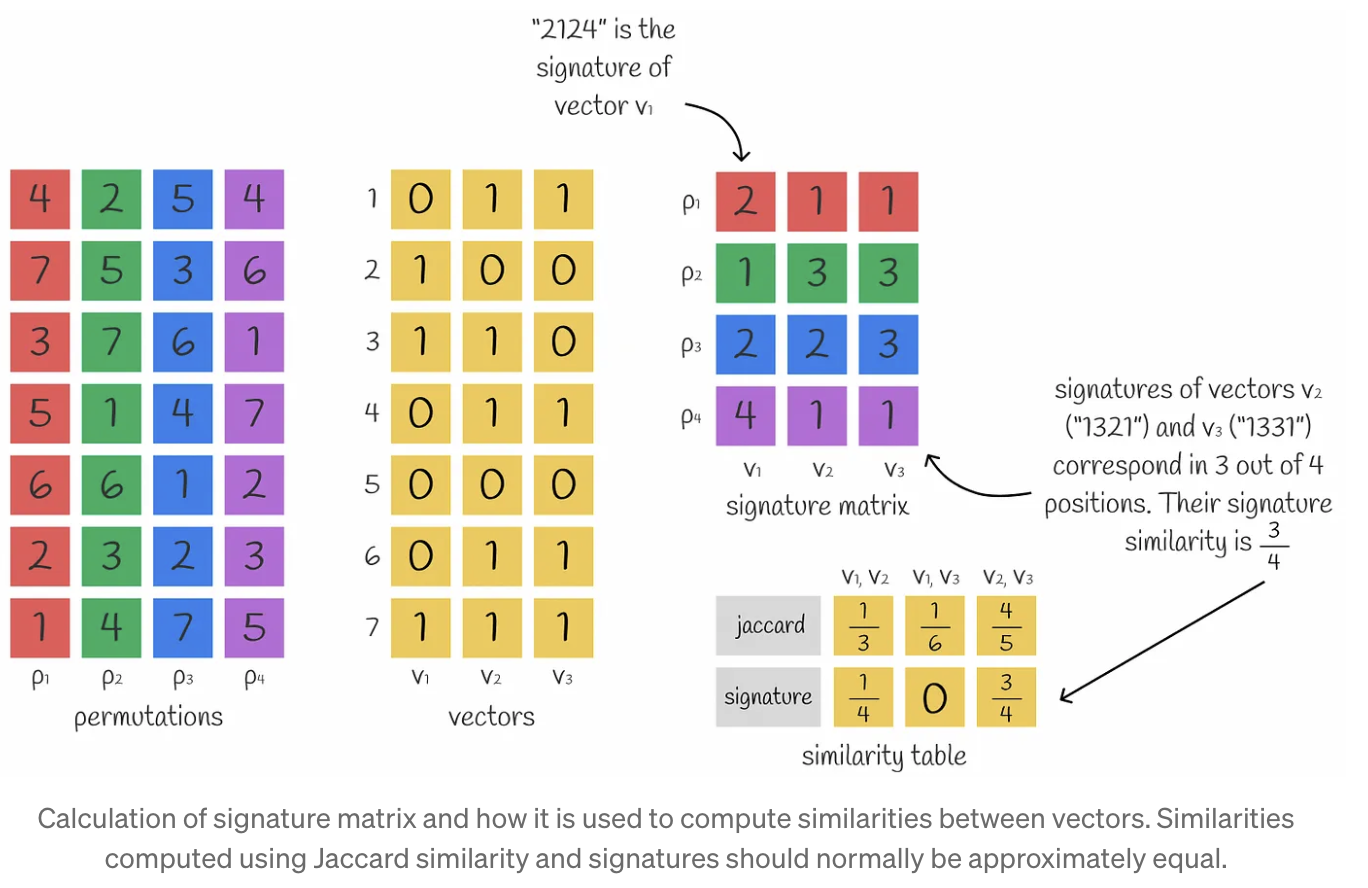
This Algorithm will have time complexity of O(N^2). This is computationally expensive and time consuming assuming our N is large. So we would like to explore some approximation algorithms for fast similarity searching.

**Locality Sensitive Hashing:**

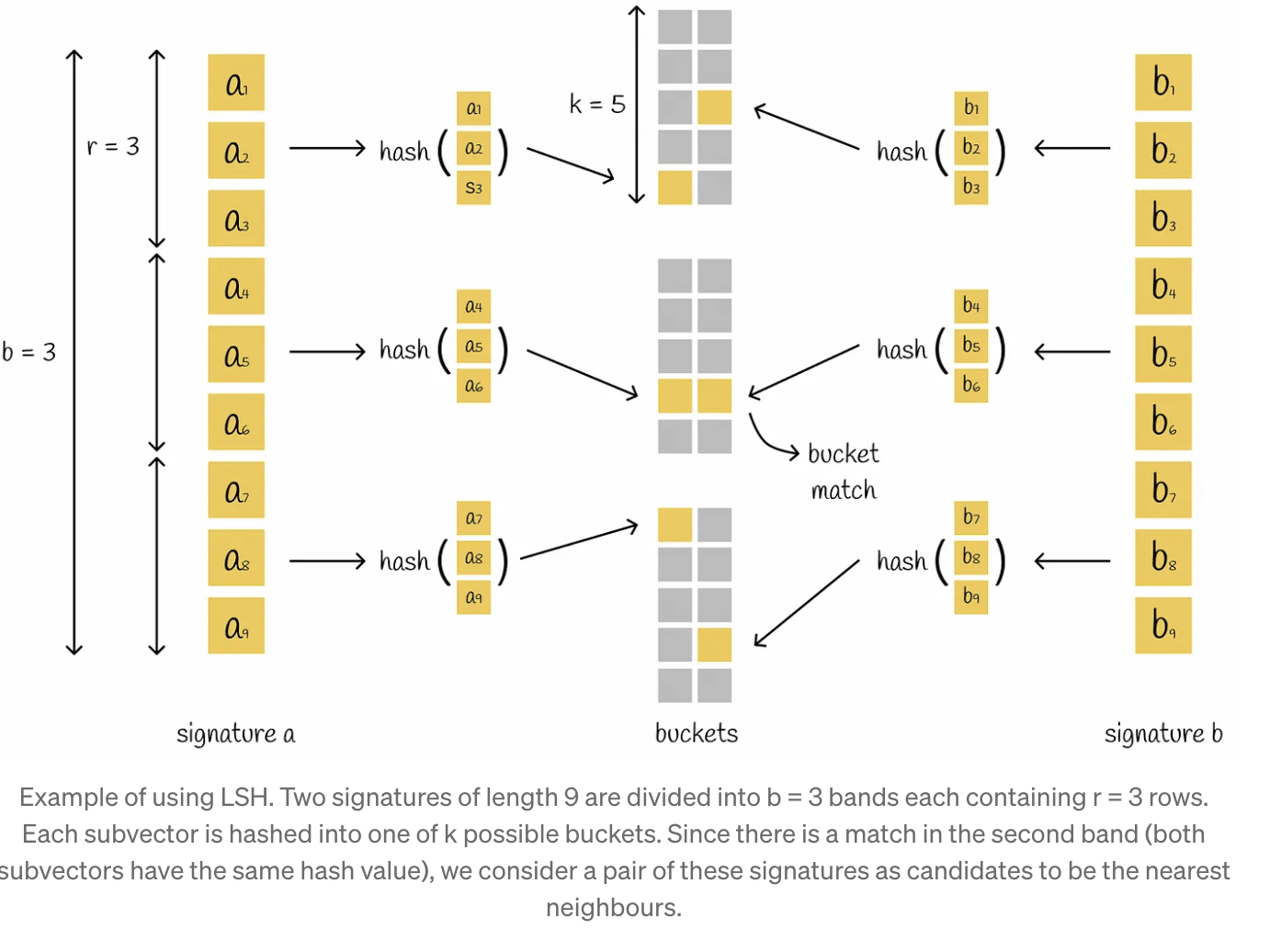
This algorithm as 3 steps:

**1) Shingling**: encoding original texts into vectors.



2) **MinHashing**: transforming vectors into a special representation called **signature** which can be used to compare similarity between them.

**3)LSH function**: hashing signature blocks into different buckets. If the signatures of a pair of vectors fall into the same bucket at least once, they are considered candidates.

LSH significantly optimizes search speed by using lower dimensional signature representations and a fast hashing mechanism to reduce the candidates’ search scope. At the same time, this comes at the cost of search accuracy but in practice, the difference is usually insignificant.

However, LSH is vulnerable to high dimensional data: more dimensions require longer signature lengths and more computations to maintain a good search quality.

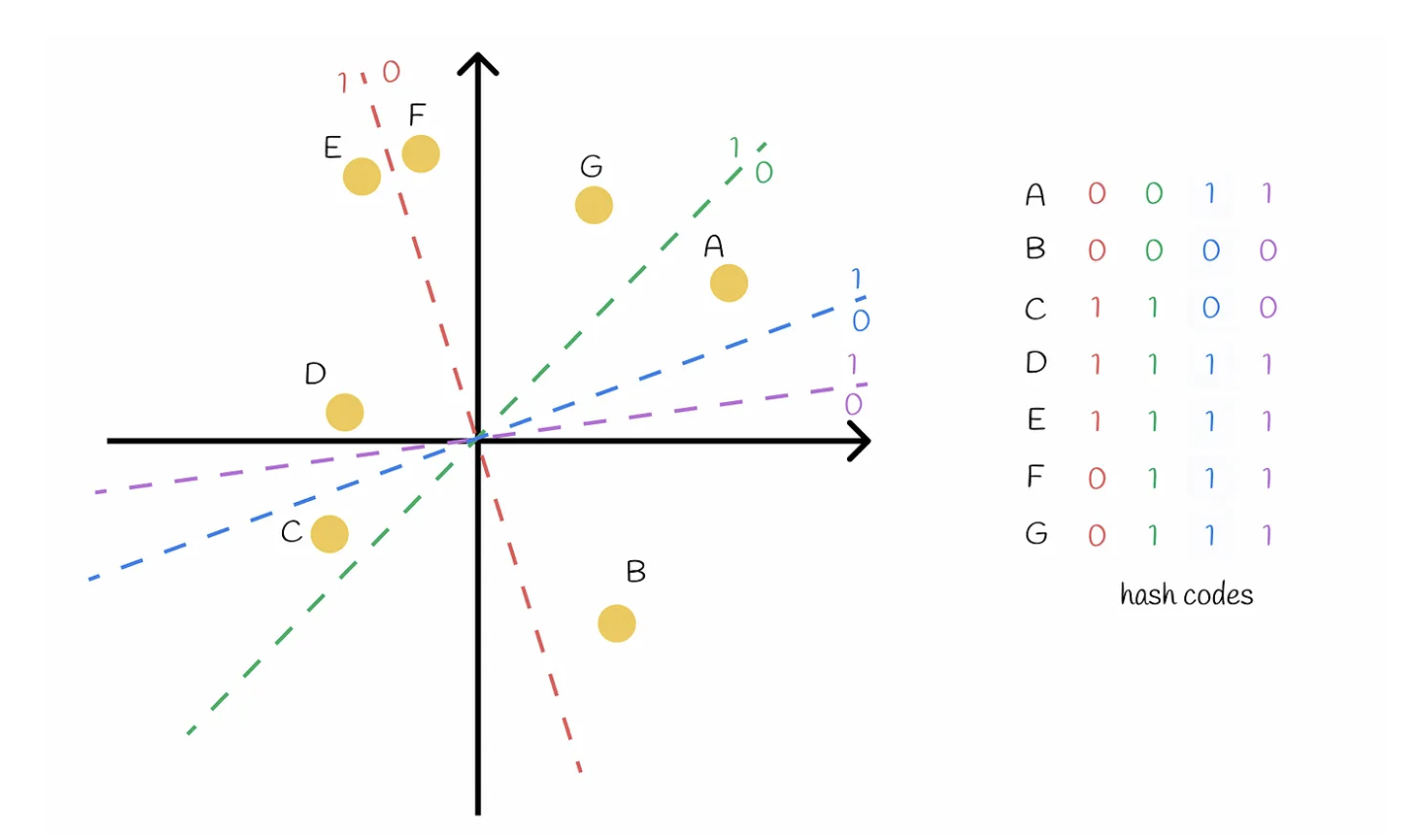
**Random Projections with LSH Forest:**

<https://towardsdatascience.com/similarity-search-part-6-random-projections-with-lsh-forest-f2e9b31dcc47>

In this algorithm to find the similarity between documents we construct randomized hyperplanes to split the data vectors

every dataset vector can be separated into one of two sides of a hyperplane. We take k hyper planes.

every vector can be encoded with that many values of 0 and 1 based on its relative position to a specific hyperplane. If two vectors have absolutely the same binary code, it indicates that none of the constructed hyperplanes could have separated them into different regions. Thus, they are likely to be very close to each other in reality.

Hyperplanes are constructed randomly. This may result in a scenario when they poorly separate dataset points which is shown in the figure below.

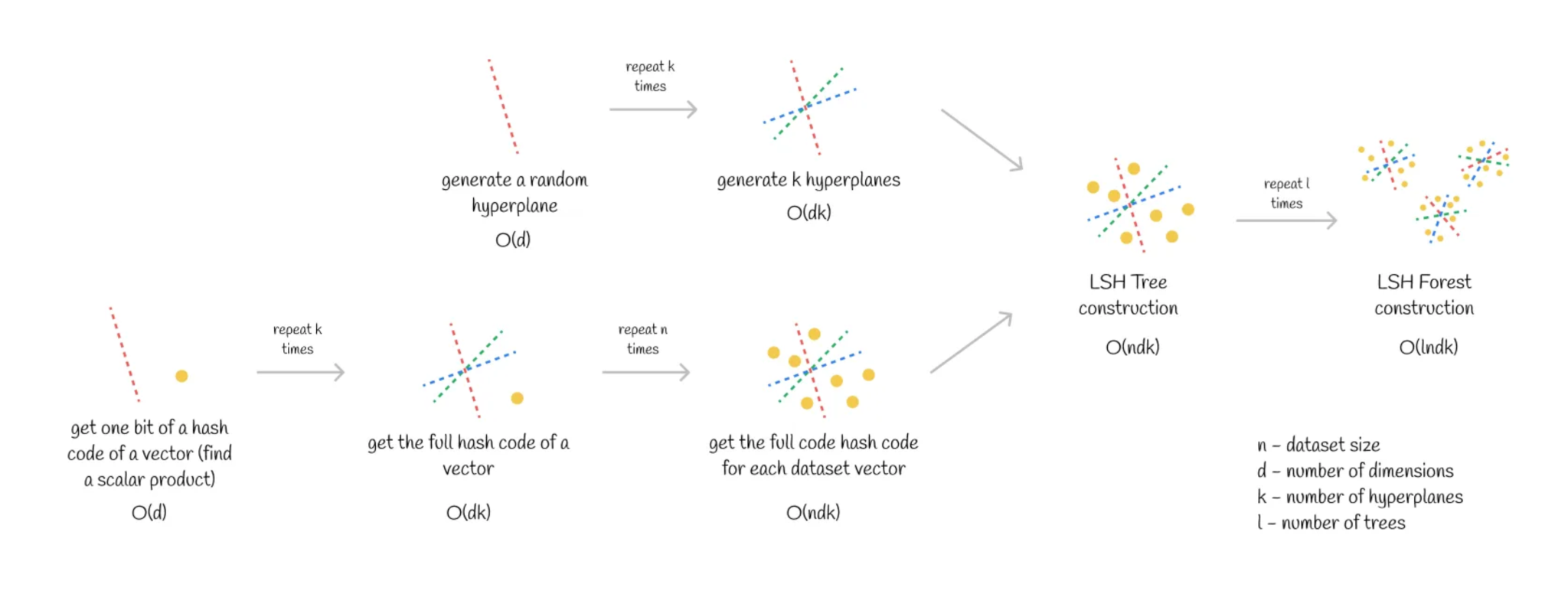
it is not a big deal when two points have the same hash code but are far from each other. In the next step of the algorithm, these points are taken as candidates and are fully compared — this way the algorithm can eliminate false positive cases. The situation is more complicated with false negatives:

If one estimator commits an error, other estimators can produce better predictions and alleviate the final prediction error. Using this idea, the process of building random hyperplanes can be independently repeated.

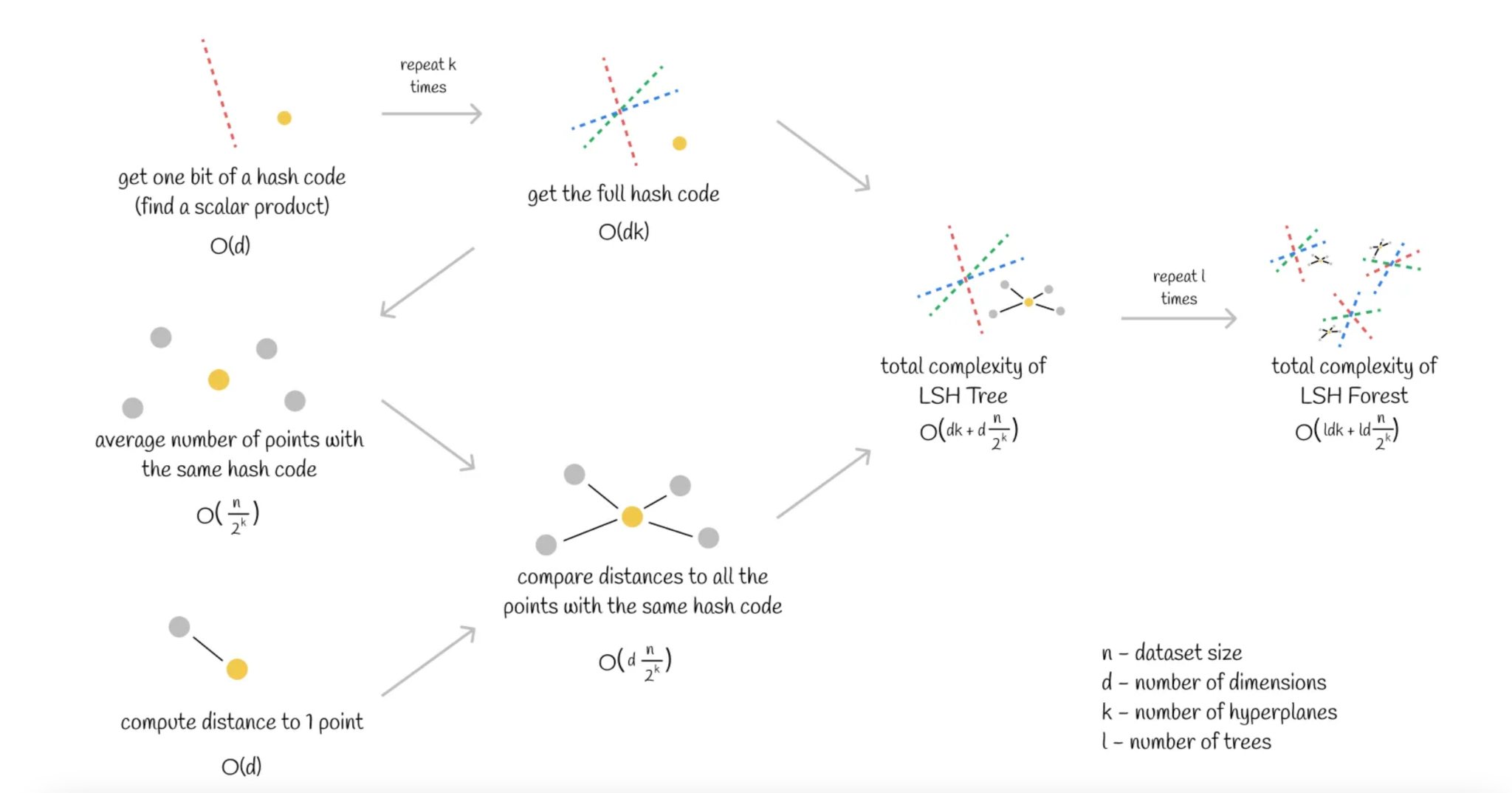
If a document has the same hash code at least once with another vector, then they are considered candidates.

Complexity:

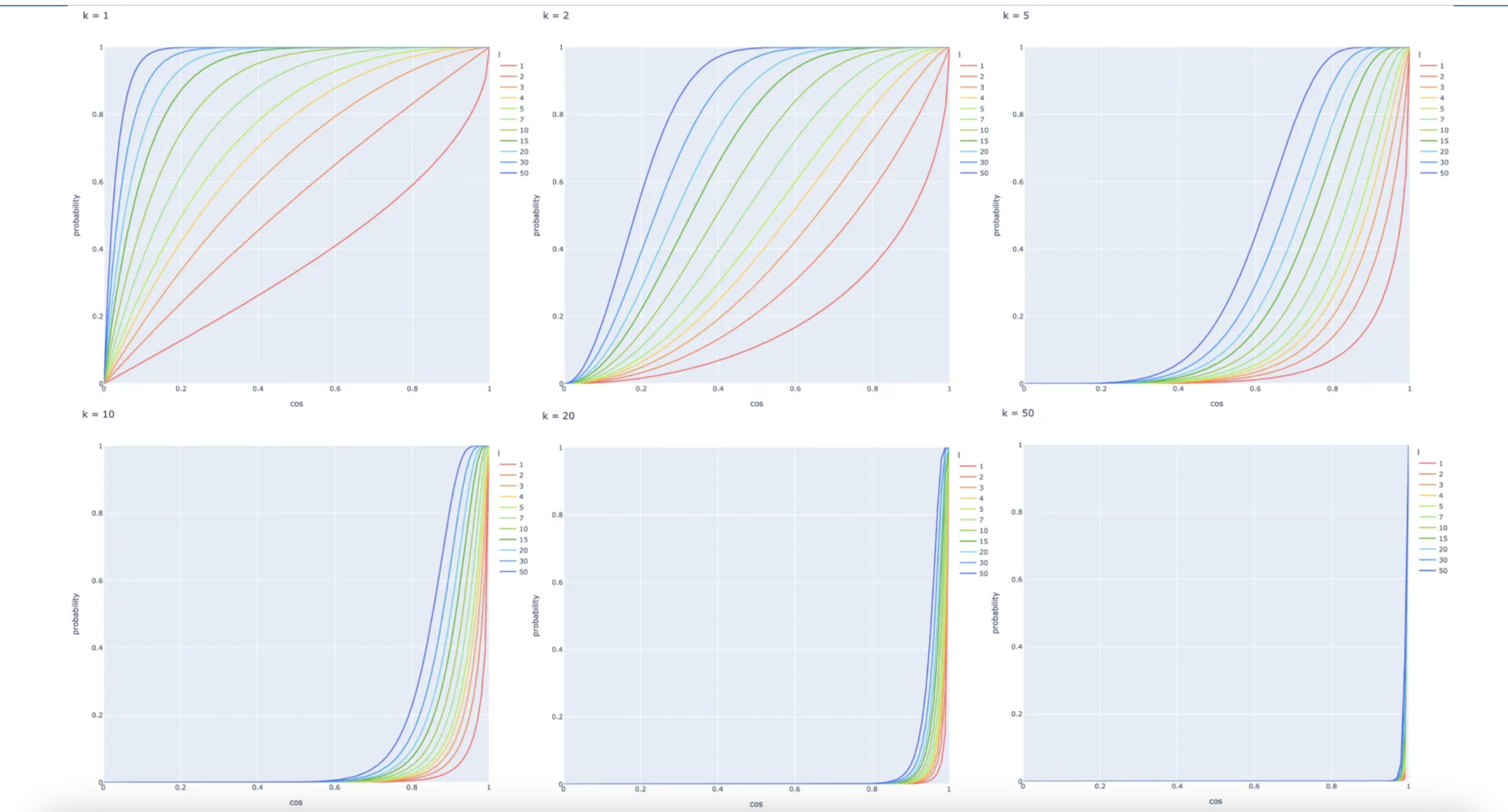
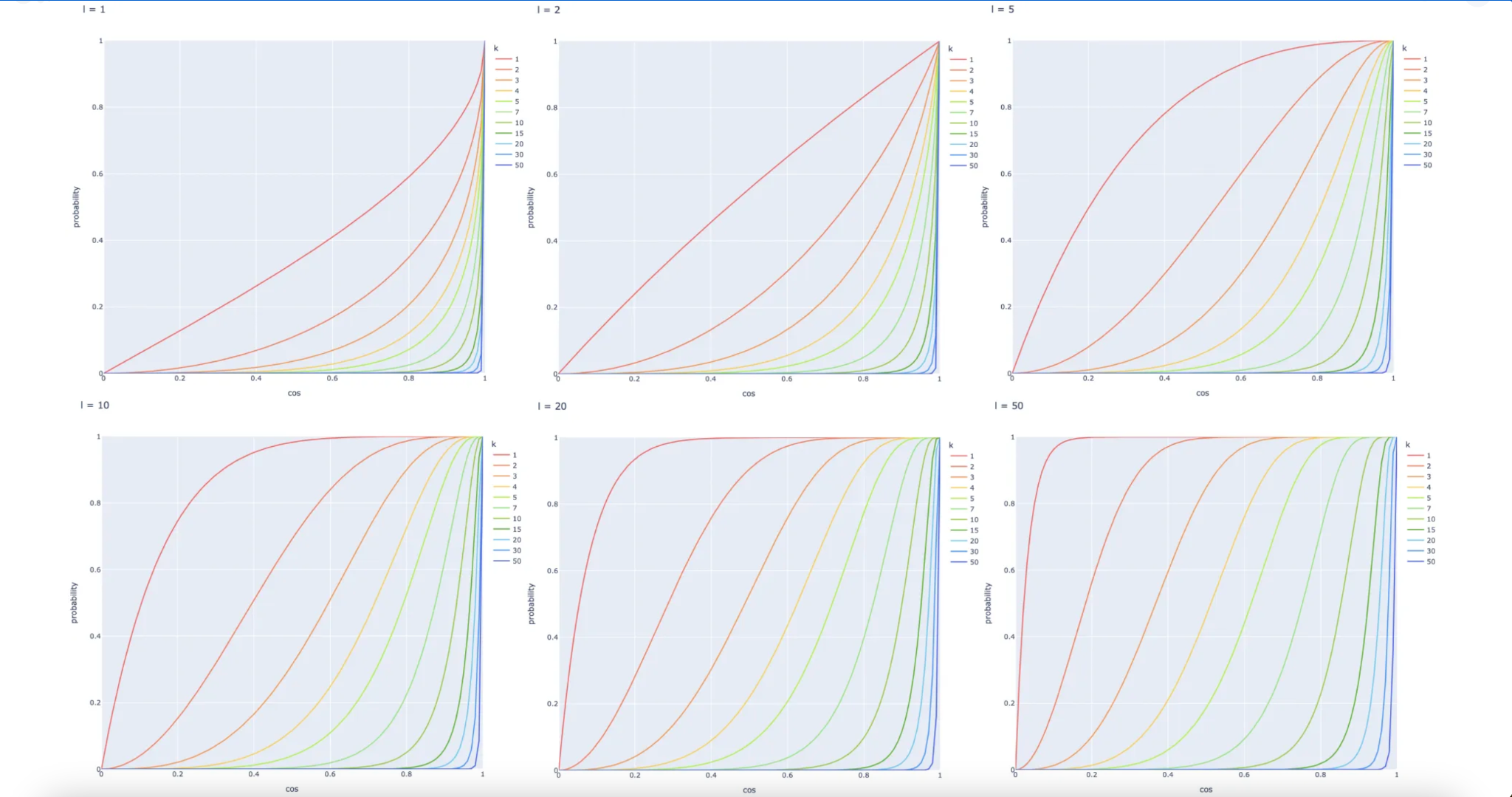
Training:



Inference :



When the number of hyperplanes k is chosen in such a way that n ~ 2ᵏ (which is possible in most cases), then the total inference complexity becomes O(ldk) (l is the number of trees). Basically, this means that the computational time does not depend on the dataset size! This subtlety results in efficient scalability of similarity search for millions or even billions of vectors.

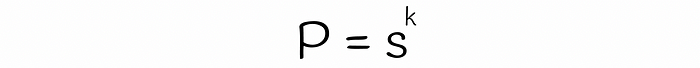


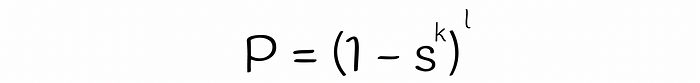
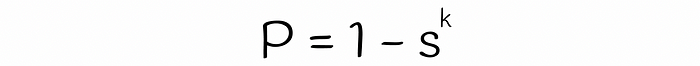
It is possible to adjust different values k and l based on a given problem and acquire the probability curve that satisfies the problem’s requirements.

Probability curve equation:

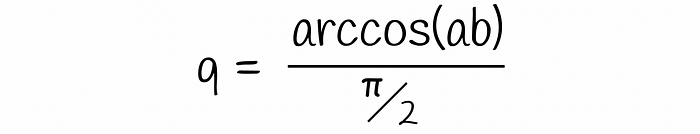
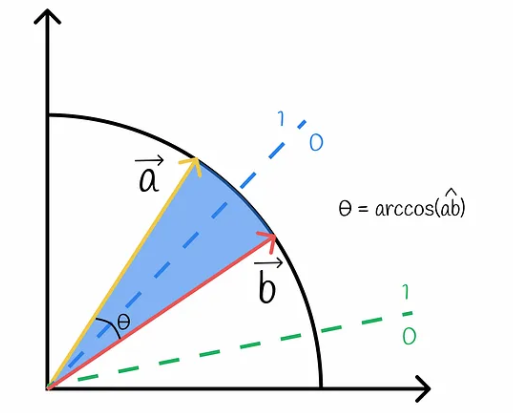
Let s be the probability that two vectors have the same bit at the same position of their hash values

The probability that hash codes of length k of two vectors are equal

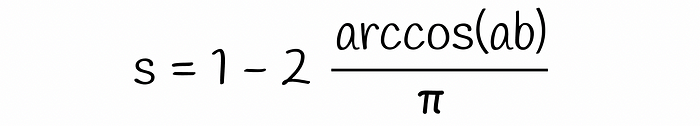
The probability that hash codes of length k of two vectors are different (or at least one bit is different)

The probability that all l hash codes (for l hyperplanes) of two vectors are different

The probability that at least one of l hash codes of two vectors is equal, so the vectors will become candidates



The probability that a random hyperplane separates two vectors (i.e. so they have different bits)



The probability that a random hyperplane does not separate two vectors ( ie. so they have same bit)

>A pair of vectors with the cosine similarity of 1 always become candidates.

>A pair of vectors with the cosine similarity of 0 never become candidates.

>The probability P of two vectors being candidates increases (i.e. more false positives) when the number of hyperplanes k decreases or the number of LSH trees l increases. The inverse statement is true.

**Implementation of Random Projections with LSH Forest:**

Used the confluence data for documents.

The data set had a total of 13618 documents.

Filtered out only the data which had token length below 500 for embedding.

The new filtered dataset had a total length of 7643.

Used snowflake-arctic-embed-l model for embedding the document content

Ran the Random Projections with LSH forest algorithm, with the following parameters:

l = 10 (no of times the random trees are generated)

d = 1024 (dimension of embedded vector)

k = 13 (no of planes (or) length of hash codes of document) (2^13= 8192)

Documents similarity threshold =0.9 (cosine similarity)

The algorithm took around **10 seconds** to find similar pairs of documents.

Total 887 different sets of similar documents were returned.

Now tried to run the normal O(N^2) algorithm:

The algorithm took **2 mins 20 seconds** to find similar pairs of documents.

Total 888 different sets of similar documents were returned.

The LSH random projections algorithm is **14x faster** compared to the general algorithm with the dataset size of 9900. The time difference is going to exponentially increase with the increase in size of the dataset.

From the total number of returned similar sets 813 sets were common in both the algorithms

We create similars sets based on the following concepts:

Let a, b ,c be documents

similarity(a,b) > threshold similarity

similarity (b,c) > threshold similarity

Similar\_set= { a, b, c}

In this process there is no guarantee if the similarity(a,c) is greater than the threshold similarity.

In our initial case we found around 887 similar document sets with similarity (cosine) threshold >0.9

So Out of these 887 sets we wanted to find the number of sets which had similarity (cosine) greater than 0.9 of documents with respect to all other documents in the set.

Here, 790 sets are good with all pairwise similarity above 0.9 (threshold).’

97 sets have some documents in it whose pairwise similarity is below 0.9.

Now tried with an initial similarity threshold of 0.95 to form sets.There were 864 similar sets formed.

And then checked pairwise similarity of documents in these 864 sets with threshold of 0.9.

Now 855 sets are good with all pairwise similarity above 0.9 (threshold).’

Only 9 sets have some documents in it whose pairwise similarity is below 0.9.

Similarity of documents subsets

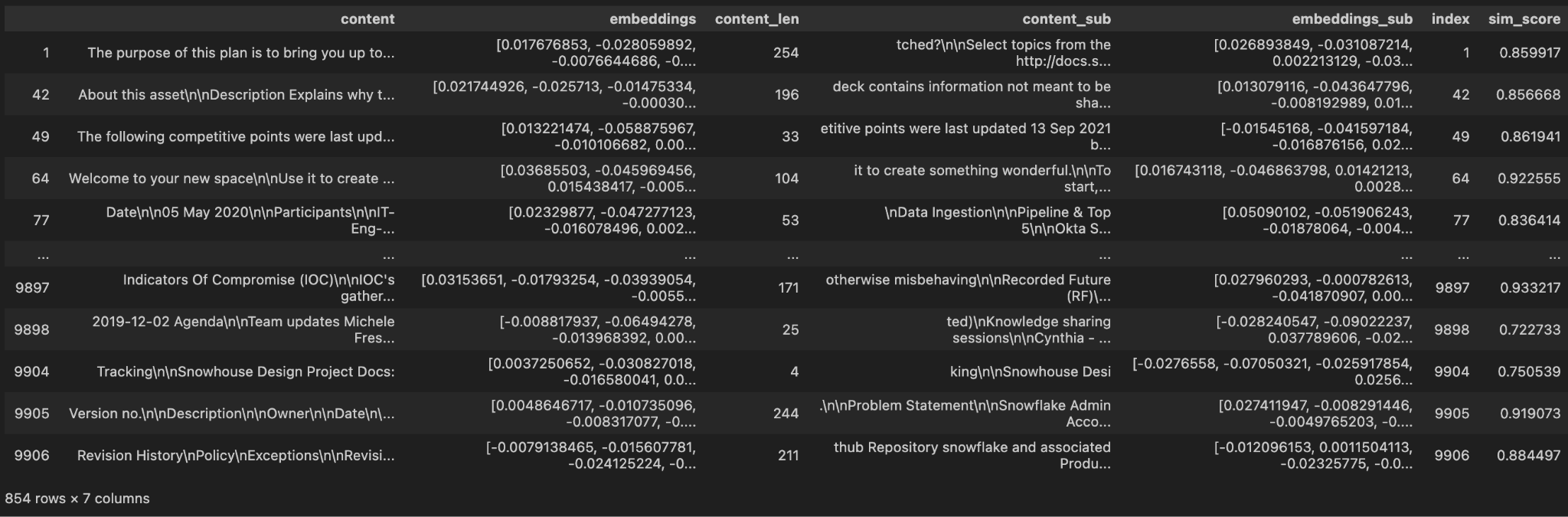
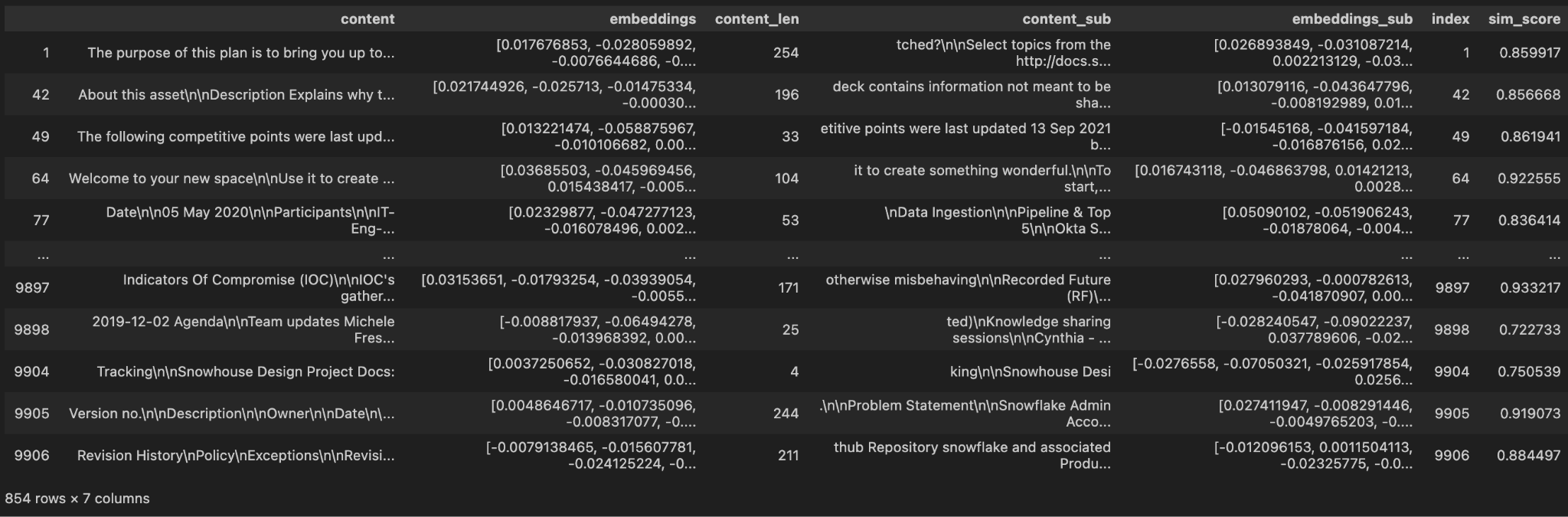
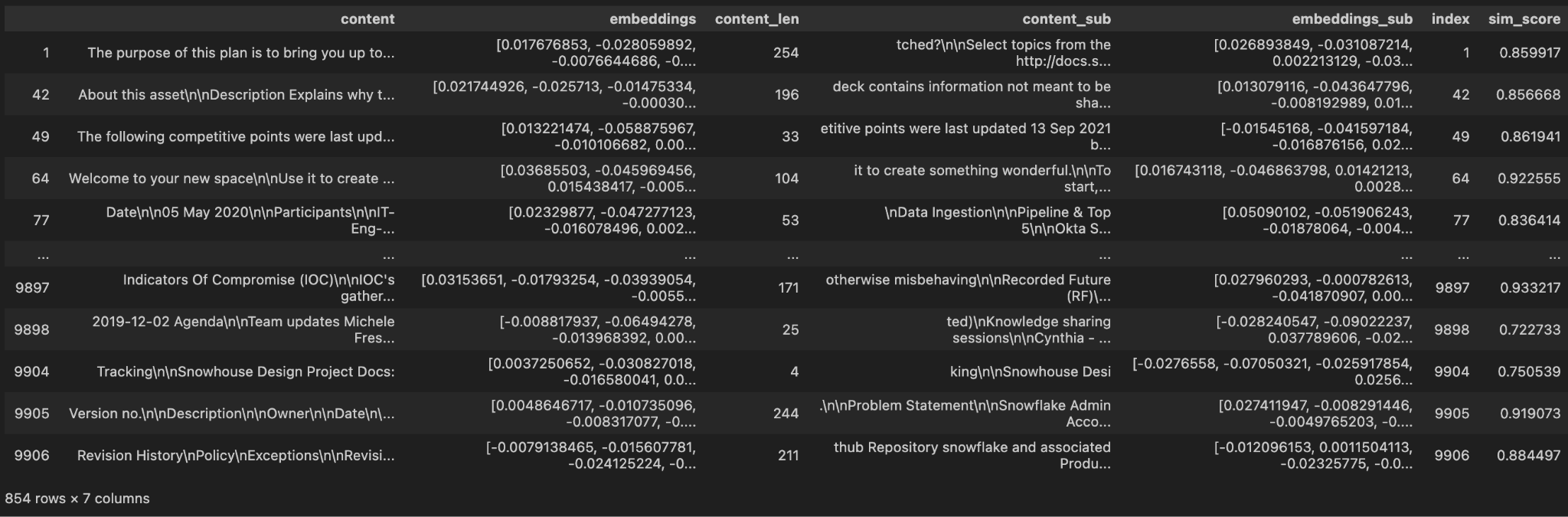
We want to know if the document is a subset of a different document then how is the similarity between these documents and how does our algorithm perform.

**Checking implementation:**

Took 854 unique documents and then randomly sampled a subset of size len(document)/2 from the document.

Then computed the similarity between the document and its subset.

Out of 854 only 204 had a similarity score above 0.9.



This suggests that while having subsets the semantic context of the document might differ if some part of the document is not available.

Embedding for Larger Document

Summarization method

The Idea is to summarize large documents below 500 tokens and then embed them.

The prompt for the summarization step is very important because we want the summary to be as similar to the original document so that their similarity scores are high.

The prompt used for summarization:

"Summarize in between 400 to 500 words. ONLY give a summary with the same language and tone. \n SUMMARIZE: {DOCUMENT} "

We specify the summary length to be between 400 to 500 words because more words can lead to higher similarity scores.

**Implementation**

I have taken a sample of 10 documents whose token length is between 500 and 1000.

Used 3 different LLMs for summarization:

> Llama3 70b

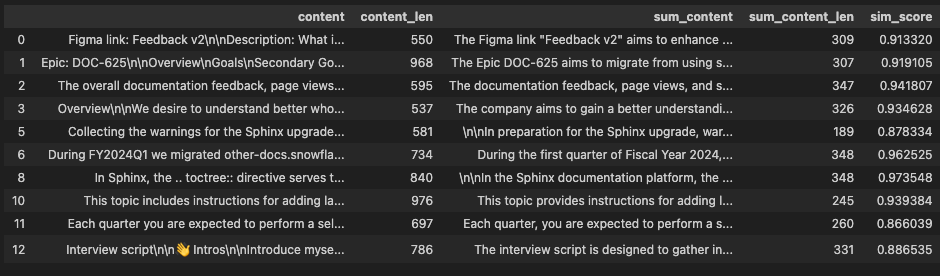
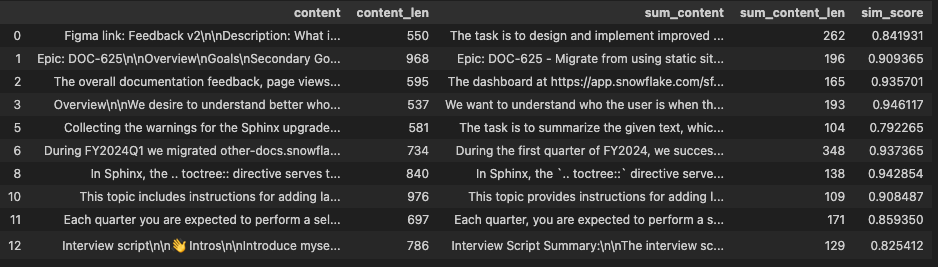
> Mistral Large

> Snowflake arctic

Used **gte-large-en-v1.5** model for embedding the document. This model has an input token size of 8192. So can use this for embedding the original document as well as summarized document

LLama 70b and Mistral large did comparatively better than snowflake arctic in the summarization process.

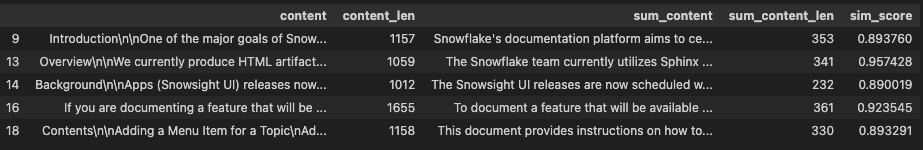
Llama3 70b

Mistral Large 

Snowflake Arctic

In Llama3 70b and mistral large all documents had similarity scores above 0.85 and 7 had similarity above 0.9. Snowflake arctic has similarity scores even below 0.8 and only 6 above 0.9.

Mistral for document size 1000-2000



From the above results we can conclude that this method can be used to embed larger documents and for similarity checks can use a threshold of 0.9.

Chunking method

The idea is to chunk the large documents into smaller documents for embedding.

Chunk the document in sizes below 512 tokens and then embed them.

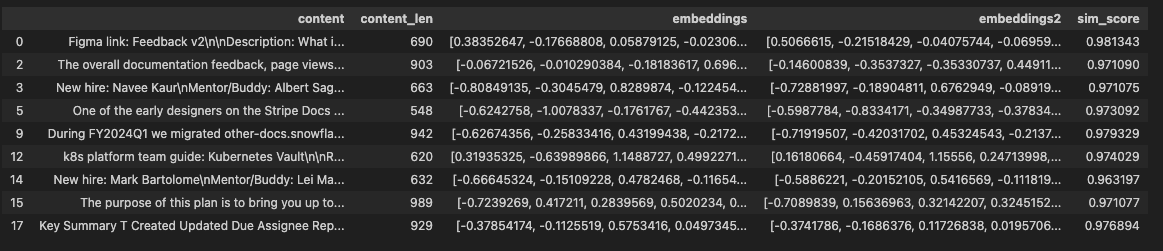
Return the mean of embedding of all chunks of document as document embedding.

Chunking: while chunking make sure all the chunks of document are of same token length so that the mean of chunks would be equally weighted of all chunks.

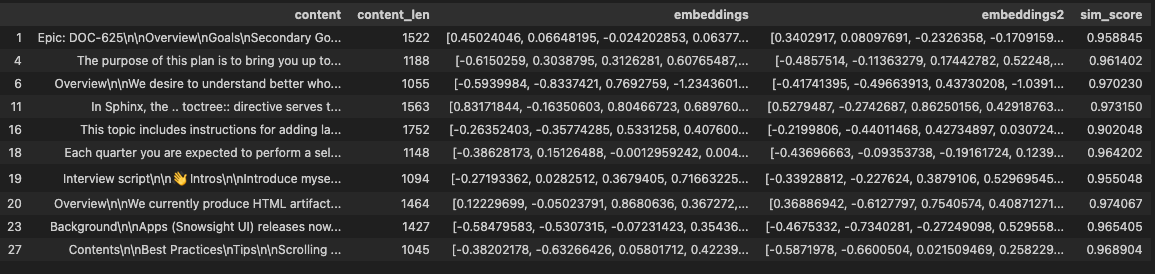
**Implementation**

I have taken a sample of 10 documents whose token length is between 500 and 1000.

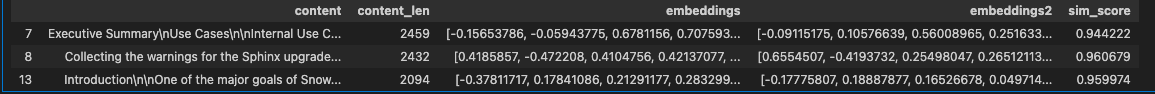
Used **gte-large-en-v1.5** model for embedding the document.



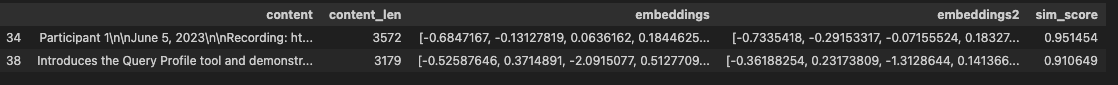
For document size 1000-2000



For document size 2000-3000



For document size >3000



The chunking method is performing better than the summarisation method and we use it for embedding larger models.

Also to check this method further we took only the documents of length 5000-6000 and used LSH algorithm to find similarity document matches and the grouped documents were very similar.

Also took a few large texts from wikipedia and paraphrased it and removed a few lines and embedded them using this chunking strategy and their similarity scores were in range of 0.97-0.99.

**Final Implementation**

Took all **13618** documents.

Used Snowflake\_arctic embedding

Used chunking method for embedding larger documents

Ran the Random Projections with LSH forest algorithm, with the following parameters:

l = 10 (no of times the random trees are generated)

d = 1024 (dimension of embedded vector)

k = 14 (no of planes (or) length of hash codes of document) (2^14= 16384)

Threshold for document similarity 0.95 (cosine)

The algorithm took around **25 seconds** to run .

There were in total **1406** sets of similar documents with a total of **5416** documents

With **1387** sets with all pairwise document similarity **above 0.9**.

And only **19** sets didn’t have all documents pairwise similarity above 0.9

Largest set has length of **861** documents and the documents are just empty ‘ ‘

The next largest set has length of **107** **documents**

The **median** set has length of **2**

The **mean** length of set is **3.85**

For complete dataset O(N^2) algorithm took around **8 mins 41 seconds** to run

Our algorithm is around **20x** faster for this length of 13000.

Found out there is a function named cdist in sklearn with in built function for distance measures with for O(n2) similarity

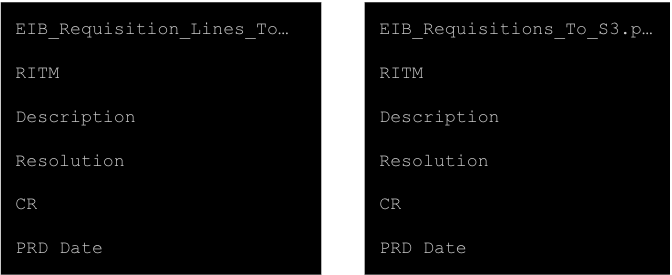
This cdist algorithm ran in around **3mins 22 sec** on whole dataset.

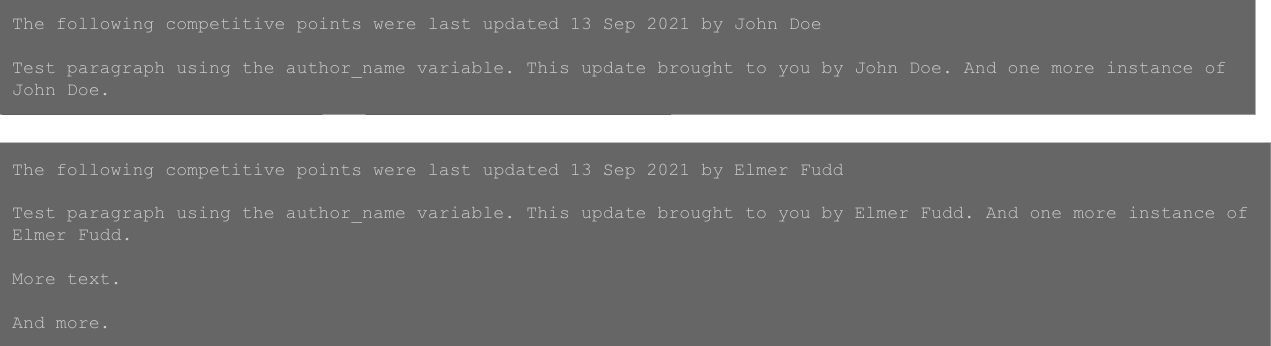
Found out that Cdist function is written in C so the runtime is a bit faster.

Wrote O(N2) algorithm in C to check its runtime, this time the code runtime is reduced to **5 mins 30 seconds from** 8 mins 40 seconds.

This suggests that writing the total Random projection LSH function in C and calling it can further reduce the runtime.

**Example of similar documents:**

**​​​​​​​​​​​​​​​​**



**What next and problems:**

Suppose we have a subset of a document in another document then how to identify that and remove the subset. There is no guarantee that the subset has a similarity score above 0.95/0.9. This is shown in the experimentation.