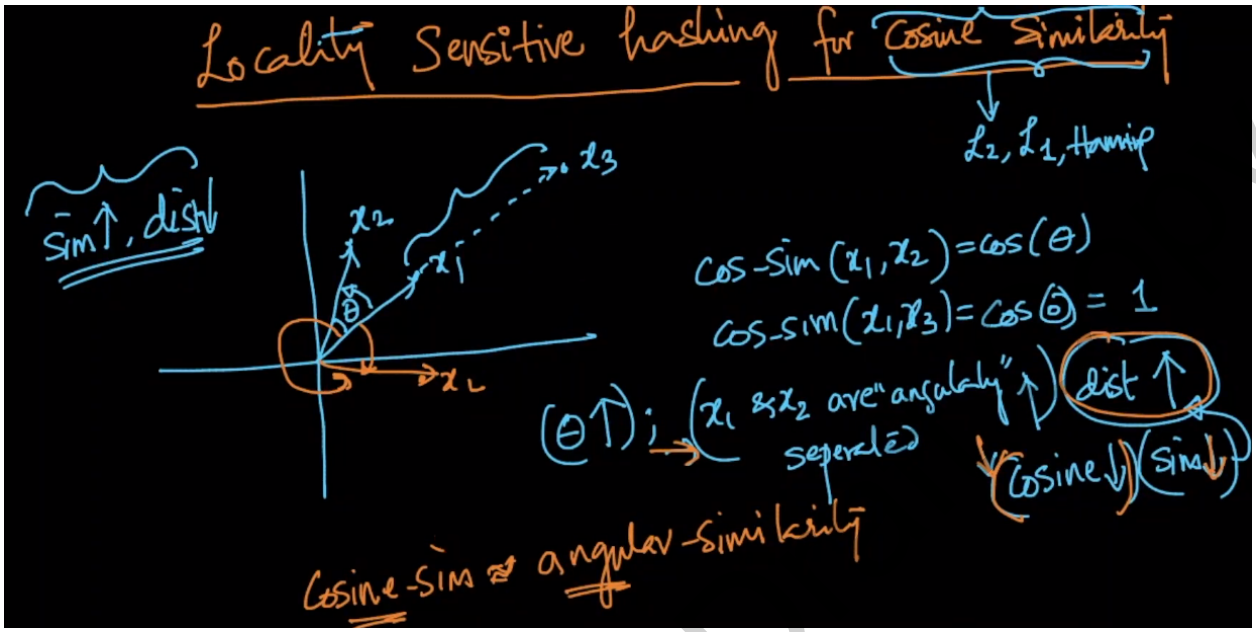
29.27 LSH for Cosine Similarity



We have 3 vectors 'x_1', 'x_2' and 'x_3'.The vectors 'x_1' and 'x_3' are in the same direction and the angle between them is 0^0. Whereas the vectors 'x_1' and 'x_2' are separated by an angle 'θ'. So we can say the vectors 'x_1' and 'x_2' are angularly separated. If the angle 'θ' increases, the cosine similarity decreases, and the cosine distance increases.

cosine_similarity(x_1, x_2) = cosθ

cosine_similarity(x_1, x_3) = cos(0^0) = 1

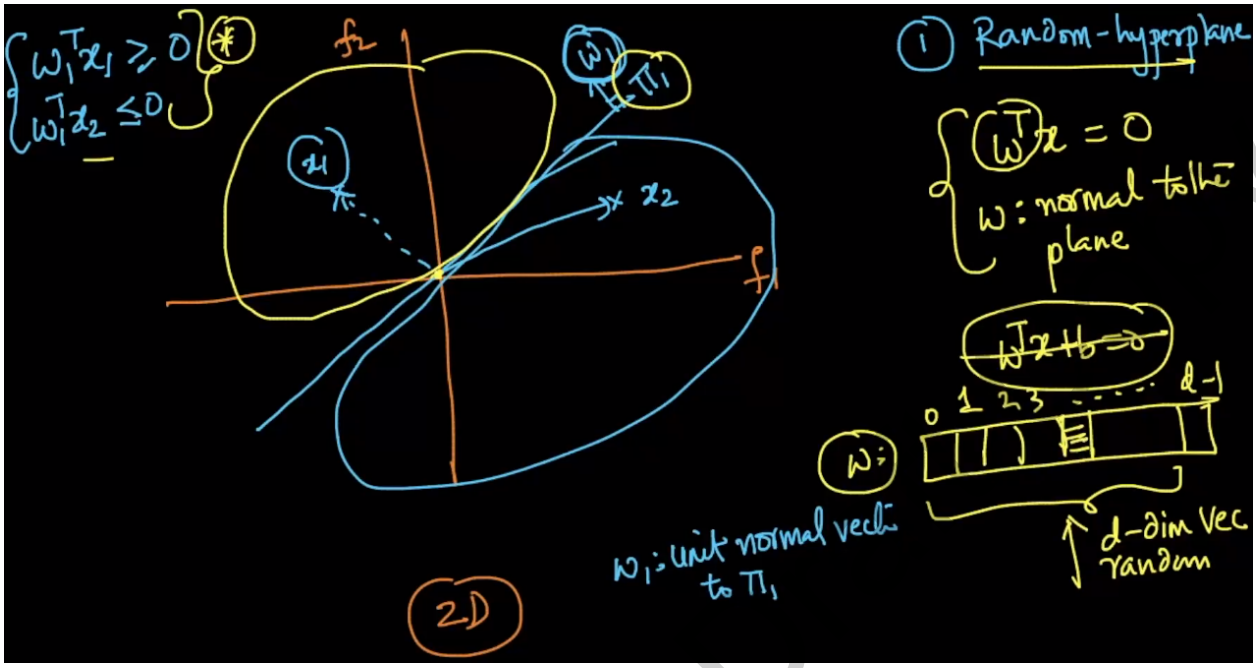
As 'θ' increases, 'x_1' and 'x_2' are more angularly separated. Cosine Similarity is all about angular similarity, but not the geometrical distance. So in LSH, all the points that are more similar/close should go to the same bucket.

So if 'x_1' and 'x_2' are similar, then the hash function associated with these two points will become the key.

Key = h(x_1) = h(x_2)

LSH is a randomized algorithm. LSH says it always doesn't give the correct answer. But it always says it will give the correct answer with high probability. Every time LSH might not yield the nearest neighbors accurately, but everytime whatever results are obtained, are given with high probabilities.

Let us consider a 2-D plane and divide it into two parts using a line/plane/hyperplane(here it is a random plane), as shown below.

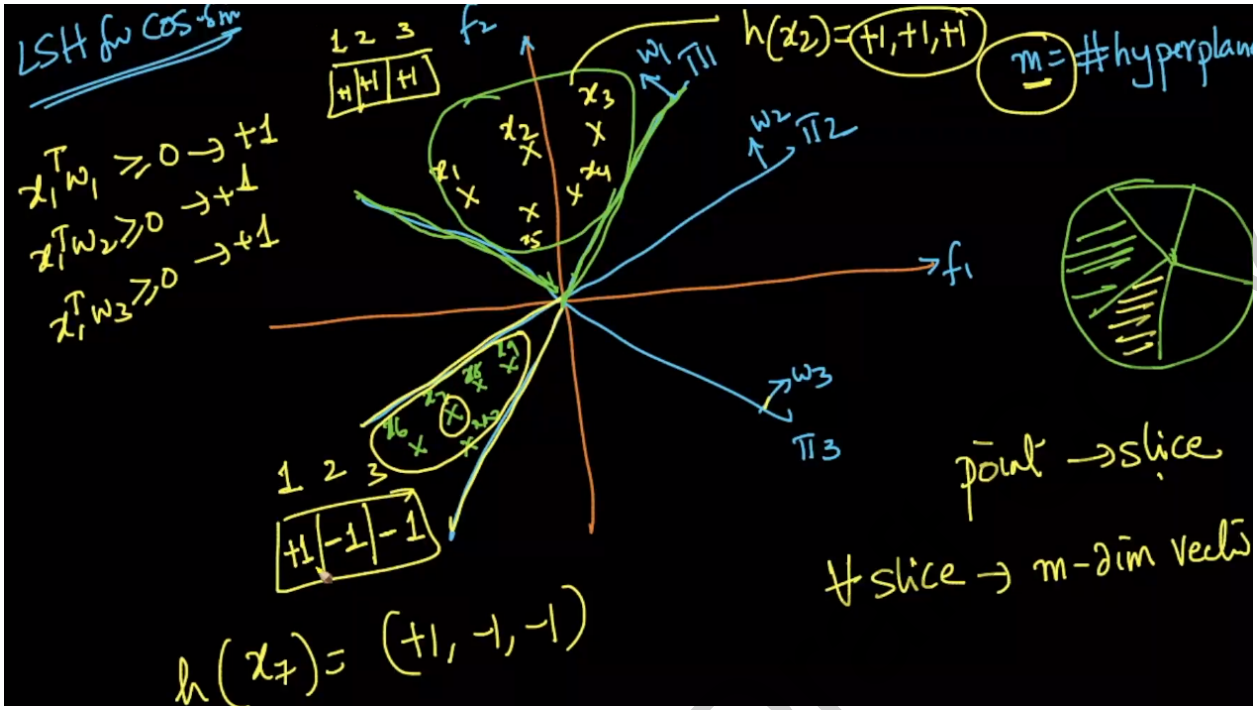


This plane is obtained by randomly generating the values for the unit normal vector 'w_1'. Since 'x_1' is present on the same side as 'w_1', $\mathbf{w_1^T x_1 >= 0}$.

But the point 'x_2' is present on the opposite side of the vector 'w_1'. So $\mathbf{w_1^T x_2 <= 0}$. Here 'w_1' is a d-dimensional random vector. If we are able to determine 'w_1', then we can easily find out the equation of the hyperplane 'π_1'.

If a hyperplane passes through the origin, then the equation of the hyperplane is given as $w^T.x = 0$. If the hyperplane doesn't pass through the origin, then the equation of the hyperplane is $w^T.x+b=0$. (Here 'w' is a unit vector normal to the hyperplane and 'b' is the intercept)

In 'w_1', every component is a random number sampled from Normal distribution. With the help of 'w_1', we can generate a random number hyperplane. Let us take a random hyperplane and understand how it works.



Let us consider 3 random hyperplanes and perform Locality Sensitive Hashing using Cosine Similarity.

If we multiply any point 'x_i' in slice 1 with 'w_1', then $w_1^T.x_i>0$.

So we shall compute the hash key for the points in slice 1.

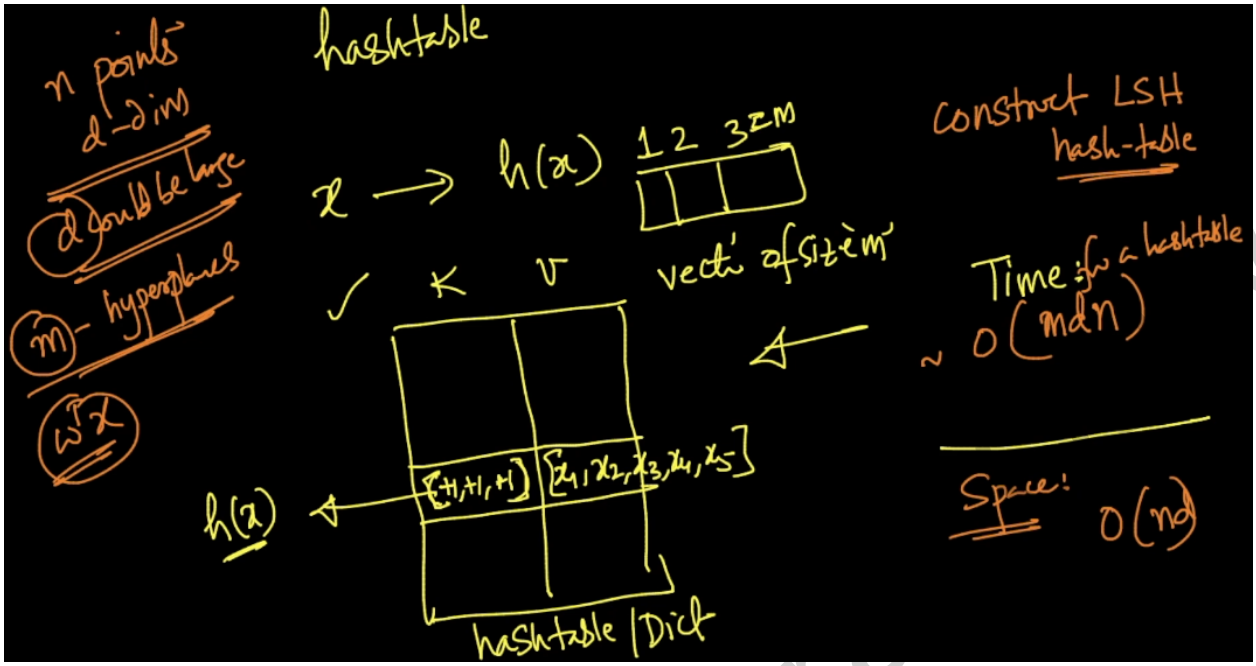
hash(x_1) = {+1, +1, +1} (These 3 values denote the signs of $w_1^T.x_1$, $w_2^T.x_1$ and $w_3^T.x_1$ respectively)

hash(x_2) = hash(x_3) = hash(x_4) = hash(x_5) = hash(x_6) = {+1, +1, +1}

As we got the keys using the hash function, we now have to construct the hash table.

Key (Hash Function Value)	Value
. . . {+1, +1, +1} . . .	. . . [x_1, x_2, x_3, x_4, x_5, x_6] . . .

Here we are taking the vectors as the key, and the points with these vectors, as the values. We have 'n' data points and each data point is 'd' dimensional.

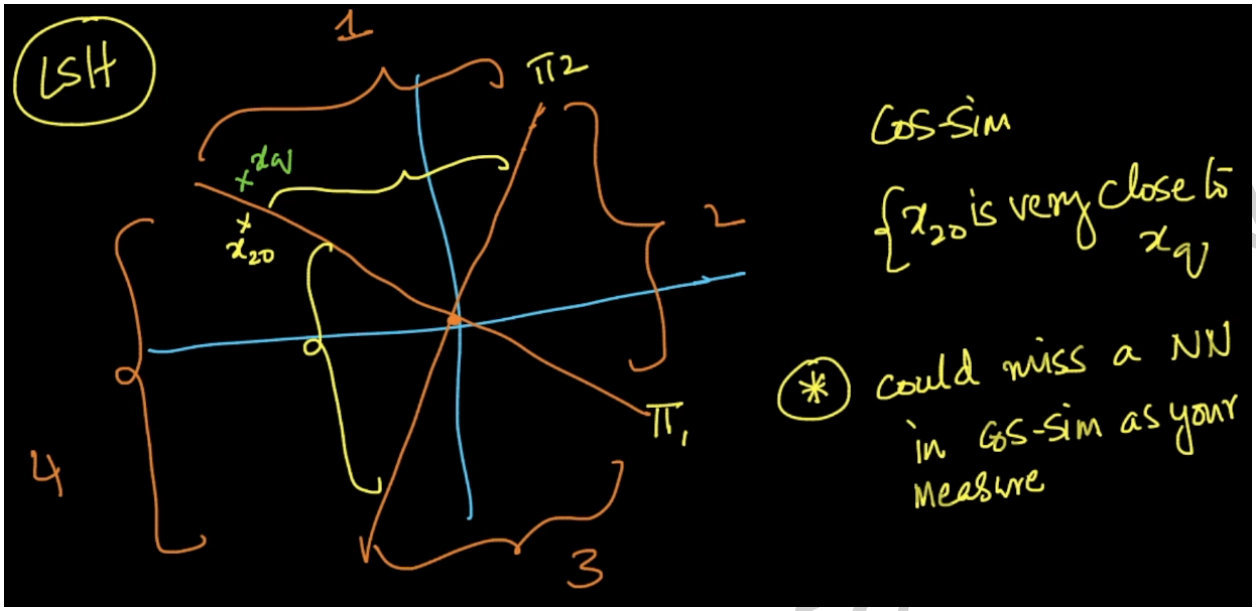


So space complexity → O(nd) (To compute a hashtable)
Let us assume we have 'm' hyperplanes.
Time Complexity to compute one key-value pair = O(m*d)
Time Complexity to compute all the key-value pairs = O(m*d*n)

So in order to find out the nearest neighbors for the point 'x_q', we need to compute the hash function for the point 'x_q', and pull out the value associated with this hash function. In case if we want 'K' nearest neighbors, then after pulling out the list of the values from the hashtable, we have to compute the cosine similarity scores of 'x_q' with all the points extracted from the hashtable. Those 'K' points with the top Cosine Similarity scores are chosen.

Time Complexity for querying a point = O(m*d)
In case, if there are n' elements in that bucket, then the total time complexity = O(md + n'd)
The ideal value for 'm' = log(n)
So the time complexity = O(d*log(n))

But here we come across a special case. So far, we have seen the points lying in the same bucket. But let us look at the figure below.



We have the point 'x_{20}' in a different bucket/slice when compared to 'x_q'. So here even if 'x_{20}' is a nearest neighbor of the query point 'x_q', it will not be considered in the same bucket. So we would definitely miss this point, if we use cosine distance as the metric.

In order to get such points under the same bucket, we need to

a) Take 'm' hyperplanes and create a hash function. Let it be $h_1(x_q)$.
b) Take another 'm' hyperplanes and create another hash function. Let it be $h_2(x_q)$.
In this way, we have to keep doing it. At one hash function value, we get both these points in the same bucket.

So now, for every data point 'x_i', we have to find out the hash value by passing it through all those hash functions, obtaining the nearest neighbors from every hash function, and performing a set union operation. Out of all the nearest neighbors, we have to pick our 'K' nearest neighbors, using the cosine distance.

So the total time complexity to query 'L' hash tables = O(m*d*L)

As the number of hyperplanes increases, the number of slices/buckets increase,and the number of points in each slice decreases.

As the number of hyperplanes decreases, the number of slices/buckets decreases, and the number of points in each slice increases, thereby the time complexity becomes worse.