

Locality Sensitive Hashing

Lecture Notes for Big Data Analytics

Nimrah Mustafa

March 2019

Contents

1	LSH: Definition	2
1.1	A Motivating Example: LSH for Plagiarism Detection	3
1.2	Applying LSH for Nearest Neighbor Problem	5
2	Designing LSH Family for Hamming Distance	6
2.1	Hamming Distance (Hamming Similarity)	6
2.1.1	Bit-Sampling: LSH for Hamming Distance	7
3	LSH: Probability Amplification	7
3.1	Construction of new LSH families from old ones	8
3.1.1	The AND Construction:	8
3.1.2	The OR Construction:	9
3.2	Generic LSH scheme	10
3.2.1	AND-OR Construction	10
3.2.2	OR-AND Construction	11
3.2.3	The S -curve	12
4	LSH for other distances	13
4.1	Non-LSHable and Not Yet Known to be LSHable Measures	14
4.2	Jaccard Distance	15
4.2.1	Minhashing	15
4.2.2	Approximate Minhashing	16
4.3	Cosine Distance	17
4.3.1	simHashing: LSH for Cosine Distance	17
4.3.2	Computation of LSH for Cosine Function	19
4.4	Euclidean Distance	20
4.4.1	LSH for Euclidean distance	20
5	Computational Issues	22

Recall the two proximity computation problems underlying nearly all data analytics: the distance matrix and k -nearest neighbors computation. Given a set X of m -dim vectors, with $|X| = n$, the runtime of the brute force algorithms for computing the distance matrix and k -nearest neighbors is $O(n^2 \times m)$ and $O(n \times m)$, respectively. The runtimes grows linearly with dimensionality m and quadratically or linearly with number of points n . In dimensionality reduction we dealt with the factor of m , and now we deal with the factor n .

Recall that the Dictionary ADT can be implemented with *hash functions* and the concept of collisions and chaining.

1 LSH: Definition

Hashing is perfect for duplicate detection, since the same items will always hash to the same bucket, but it does not help for near duplicate detection as similar items, i.e. within the st threshold for near-duplicates, would not necessarily hash to the same bucket. For near-duplicate detection, we need hash functions where meaningful collisions are desired, i.e. we want similar objects hash to same buckets, and that is what Locality Sensitive Hashing (LSH) is essentially meant for.

A family $\mathcal{H} = \{h_1, h_2, \dots, \}$ is a (d_1, d_2, p_1, p_2) -family of LSH functions, if for a randomly chosen function h from \mathcal{H} , for objects x and y

1. **If $d(x, y) \leq d_1$, then $Pr[h(x) = h(y)] \geq p_1$** , i.e. $h(x) = h(y)$ with probability at least p_1
2. **If $d(x, y) \geq d_2$, then $Pr[h(x) = h(y)] \leq p_2$** , i.e. $h(x) = h(y)$ with probability at most p_2

Consider the above statements. The first one bounds the false negatives, which are highly similar pairs that do not hash to the same bucket. The second one bounds the false positives which are *not* highly similar pairs that do hash to the same bucket. The problem with only the first one is that it does not talk about false positives. The problem with only the second is that it does not talk about false negatives.

Keeping in view the near duplicates detection task, we would like p_1 to be very high (close to 1) to reduce false negatives. Similarly, we would like p_2 to be very low (close to 0) to reduce false positives. Furthermore, since we have no guarantee about pairs whose distance is

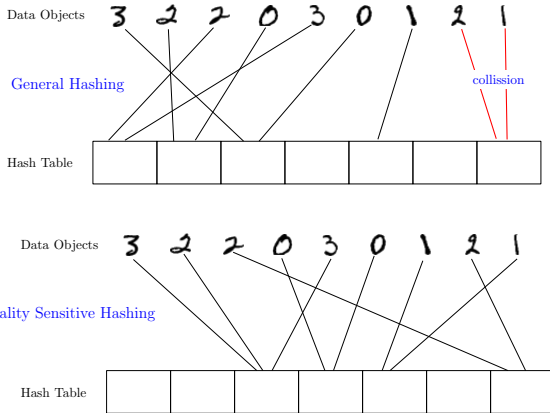


Figure 1 Uniform vs. locality sensitive hashing

between d_1 and d_2 , we would also like d_1 and d_2 both to be close to t (near duplicate threshold) to reduce the range of distances with no guarantees. Figure 2 shows what is guaranteed by a random function in the LSH family \mathcal{H} and Figure 3 depicts the ideal case we would like to have.

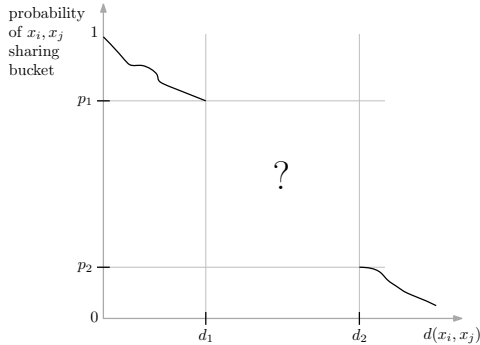


Figure 2 Guaranteed that, if $d(x, y) \leq d_1$ then the items will go to same bucket with probability at least p_1 , and if the $d(x, y) \geq d_2$ then the items will go to same bucket with probability at most p_2

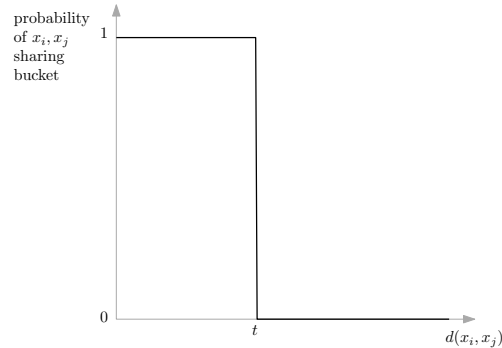


Figure 3 Ideal case in terms of distance

Sometimes we use the following equivalent definition of LSH functions in terms of similarity.

A family $\mathcal{H} = \{h_1, h_2, \dots\}$ is a (s_1, s_2, p_1, p_2) -family of LSH functions, if for a randomly chosen function h from \mathcal{H} , for objects x and y

- If $sim(x, y) \geq s_1$, then $Pr[h(x) = h(y)] \geq p_1$
- If $sim(x, y) \leq s_2$, then $Pr[h(x) = h(y)] \leq p_2$

Figure 4 shows what is guaranteed by a random function in the LSH family \mathcal{H} in terms of similarity and Figure 5 depicts the ideal case we would like to have, in terms of similarity.

1.1 A Motivating Example: LSH for Plagiarism Detection

Before we go into the details of constructing LSH families and working with them for different distance measures, let's see how LSH can be applied for the plagiarism detection task as a motivating example.

Suppose we have $1M$ documents, each of length 2000 (e.g. TF-IDF). Our goal is to find near duplicates, i.e. those pairs of documents whose similarity is more than $90\% = .9$ (distance is less than $10\% = .1$).

The naive (brute-force) approach is to compute all pairwise distances and output those that are less than $.1$. There are a total of means $\binom{1M}{2} \approx 1T = 10^{12}$ pairs and distance computation between a pair will take at least 2000 steps (e.g. Euclidean or cosine distance). So total number of operations is 2×10^{15} .

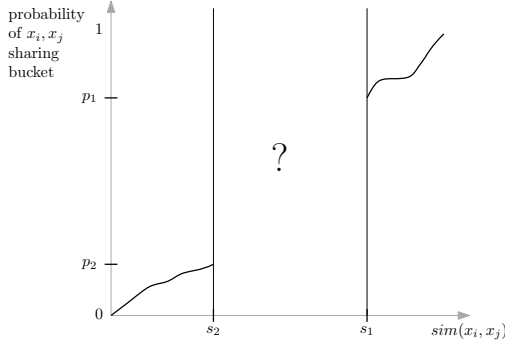


Figure 4 Guaranteed by a random function in the LSH family \mathcal{H} in terms of similarity, i.e. if $sim(x, y) \geq s_1$, then the items will go to the same bucket with probability at least p_1 and if the $sim(x, y) \leq s_2$, then the items will go to the same bucket with probability at most p_2

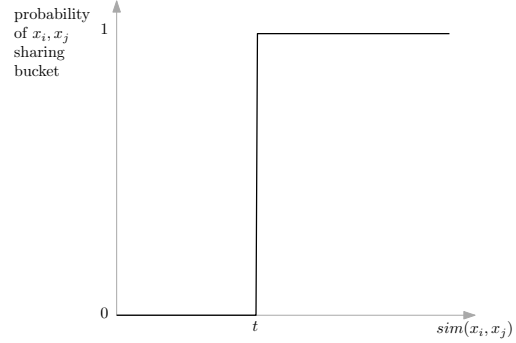
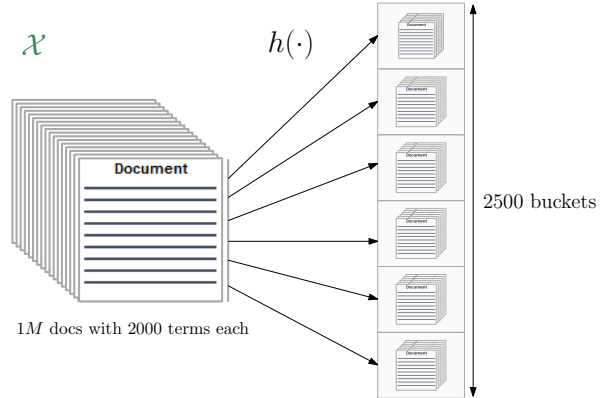


Figure 5 Ideal case in terms of similarity

Now suppose we use a random function h from a family \mathcal{H} of $(.15, .4, .8, .2)$ LSH functions. Elaborating the properties of these functions, this means that if we randomly choose a function h from \mathcal{H} , then for any two documents x and y

- **If $d(x, y) \leq .15$, then $Pr[h(x) = h(y)] \geq .8$**
- **If $d(x, y) \geq .40$, then $Pr[h(x) = h(y)] \leq .2$**

Assume that all function in \mathcal{H} are of the form $h : \mathbb{R}^n \rightarrow [2500]$, i.e each function maps documents to the set $\{1, \dots, 2500\}$ of bucket IDs. Furthermore, we assume that these LSH functions have one more good property, i.e. they map the document to any of these 2500 buckets almost uniformly, as in Figure 6. (This assumption is not realistic, as LSH does not offer any such guarantee, but we use it to make the concepts clear.)



The algorithm now is as follows. Choose a function h at random from \mathcal{H} and hash all documents with h , i.e. compute $h(x)$ for each document x . For each $i \in [2500]$, compute exact distance between all pairs in bucket i and if this distance is less $.1$, output the pair.

Each bucket on average has $1M/2500 = 400$ documents. The total pairwise distance computations are $2500 \binom{400}{2} \approx 200M$ and each distance computation takes 2000 steps, so total

number of operations is $200 \times 2000 \approx 4 \times 10^{11}$ operations. This is about 2500 times faster than the naive approach.

Though the runtime has reduced by a significant factor, we need to account for the quality of the algorithm (or specified family of LSH functions). The false positives are those pairs whose distance is more than 0.1, but we still computed their distances. Note, however, that we don't output them, so this is not a qualitative mistake but only a waste of time. The false negatives are those pairs whose distance is less than 0.1 but we didn't compare them, and hence did not output as well. This is actually a mistake in the output.

Although we don't know how many are actually false positives and false negatives, but we can bound their sizes by looking at even relaxed definition of false negatives and false positives.

From our LSH specification:

- $E(FN) < E(|\{\mathbf{x}, \mathbf{y} : d(\mathbf{x}, \mathbf{y}) \leq .15 \wedge h(\mathbf{x}) \neq h(\mathbf{y})\}|) \leq 20\%$, i.e. expected number of pairs of documents whose distance is less than .15 and going to different buckets is at most 20%. This includes all the false negatives. Hence, our average error is bounded by 20% (of $\binom{n}{2}$).
- $E(|\{\mathbf{x}, \mathbf{y} : d(\mathbf{x}, \mathbf{y}) \geq .4 \wedge h(\mathbf{x}) = h(\mathbf{y})\}|) \leq 20\%$, i.e. expected number of pairs of documents whose distance is more than .4, and going to the same buckets is most 20%. This also includes all our false positives. Hence, in expectation there will be small wasted computation

1.2 Applying LSH for Nearest Neighbor Problem

Next we use a LSH for the nearest neighbor search problem. Suppose we have a fixed radius nearest neighbor search problem at hand.

Suppose we have a dataset X of $1M$ documents each of length 1000 (e.g. TF-IDF vectors). For a query document \mathbf{q} (also a 1000 length TF-IDF vector) we want to find documents in X such that $d(\bullet, \mathbf{q}) \leq .1$ (i.e. the given radius $r = .1$).

The naive approach of computing distance of q with every document in X will take $1M$ distance computation which amounts to $\sim 10^9$ arithmetic operations.

Now suppose we use a random function h from a family \mathcal{H} of $(.15, .4, .8, .2)$ LSH functions. Elaborating the properties of these functions, this means that if we randomly choose a function h from \mathcal{H} , then for any two documents x and y

- **If $d(\mathbf{x}, \mathbf{y}) \leq .15$, then $Pr[h(\mathbf{x}) = h(\mathbf{y})] \geq .8$**
- **If $d(\mathbf{x}, \mathbf{y}) \geq .40$, then $Pr[h(\mathbf{x}) = h(\mathbf{y})] \leq .2$**

The schema of the algorithm is as follows:

We use a random h from \mathcal{H} of $(.15, .4, .8, .2)$ -LSH family and run the naive approach only on the subset of documents with $h(\cdot) = h(q)$.

We call a document x a false positive if $d(x, q) > 0.1$, yet $h(x) = h(q)$. These documents result in wasted computation, because by the algorithm prescription we unnecessary compute

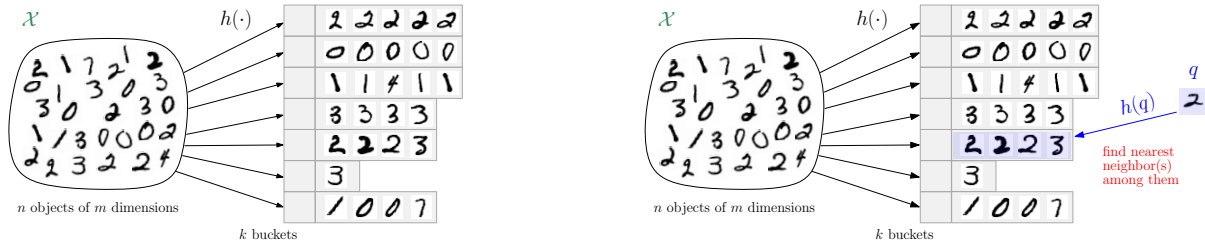


Figure 7 Find k -NN of q in a dataset \mathcal{X} of n objects of m dimensions

their distance with q , but will never output them. A document x is called false negative if $d(x, q) < 0.1$ and $h(x) \neq h(q)$. Again false negatives result in qualitative error, i.e. missed near neighbor, we were supposed to output it, yet the algorithm did not even compare it with q .

We summarize the analysis of runtime efficiency of this algorithm and it's quality

- $E(FN) < E(|\{(x, y) : d(x, y) \leq .15 \wedge h(x) \neq h(y)\}|) \leq 20\%$
- $E(|\{(x, y) : d(x, y) \geq .4 \wedge h(x) = h(y)\}|) \leq 20\%$
- On average $\leq 20\%$ missed near nbrs and hopefully small wasted computation

2 Designing LSH Family for Hamming Distance

LSH families are designed for specific distances, we first discuss in detail a LSH family for Hamming distance and elaborate all aspects of it. Then we discuss basic ideas of LSH functions for other distance measures. We also mention some distance for which it is either known that no LSH family exists or it is open problem to design one.

2.1 Hamming Distance (Hamming Similarity)

Hamming distance is used for character vectors of fixed length, i.e vectors whose coordinates take values from a finite (usually small) set called **alphabet**. The Hamming distance $d_H(\mathbf{x}, \mathbf{y})$ between two vectors x and y of length n is the number of coordinates in which they differ. Note that this is just the count of different coordinates. Hence, $0 \leq d_H(\mathbf{x}, \mathbf{y}) \leq n$.

It is easy to see that if two vectors are exactly the same, then $d_H(\mathbf{x}, \mathbf{y}) = 0$, and if they are totally different, i.e. they don't agree at any coordinate, then $d_H(\mathbf{x}, \mathbf{y}) = n$, the length of vectors. Hamming distance is gnerally used for binary vectors, in which case, it is the count of locations, where the bit of \mathbf{x} is 1 and that of \mathbf{y} is 0 or vice-versa.

It is easy to verify that Hamming distance is indeed a distance metric, i.e. it satisfies all the axioms. Hamming similarity is defined as $s_H = n - d_H(\mathbf{x}, \mathbf{y})$. This is the generally used definition of Hamming distance and Hamming similarity.

Since we have restricted ourselves to n -dimensional vectors, for technical reasons, that will be clear later, we will use

$$d_H(x, y) = \frac{\text{number of coordinates different in } x \text{ and } y}{n \text{ (the total number of bits in each vector } x \text{ and } y)}$$

In this setting, similarity is defined as $s_H(x, y) = 1 - d_H(x, y)$. When it is clear from the context that we are talking about Hamming distance and similarity, we will drop the subscript H .

2.1.1 Bit-Sampling: LSH for Hamming Distance

We give a family of locality sensitive hash functions for Hamming distance between binary strings (bit strings) of length n [4]. Each hash function in this family takes as input a bit string of length n (b_1, b_2, \dots, b_n), where each $b_i \in \{0, 1\}$ and outputs one bit, 0 or 1. In other words, each hash function h is given as $h : \{0, 1\}^n \mapsto \{0, 1\}$. Let $\mathcal{H} = \{h_i : 1 \leq i \leq n\}$. There are n hash functions in \mathcal{H} and the hash function h_i is defined as $h_i(x) := h_i(b_1, b_2, \dots, b_n) = b_i$, i.e. h_i maps the n -bits string (b_1, b_2, \dots, b_n) to the value of its i th bit. For example: $h_1(10101011) = 1$, $h_1(00110011) = 0$, $h_2(10101011) = 0$, and $h_3(10101011) = 1$. This seemingly simple family is indeed a locality sensitive hashing family.

\mathbf{x}	1	2	3	4	5	6	7	8	9
	1	0	1	1	0	1	1	0	0
	$h_1(\mathbf{x}) = 1$				$h_5(\mathbf{x}) = 0$				$h_8(\mathbf{x}) = 0$
	$h_1(\mathbf{y}) = 0$				$h_5(\mathbf{y}) = 0$				$h_8(\mathbf{y}) = 1$

\mathbf{y}	1	2	3	4	5	6	7	8	9
	0	1	1	0	0	1	0	1	0

Figure 8 LSH functions for Hamming distance between bit strings (x, y) of length 9

Lemma 1. \mathcal{H} is a $(r_1, r_2, 1 - r_1, 1 - r_2)$ -locality sensitive hash family

Proof. Choosing a random function h_i from \mathcal{H} corresponds to choosing a random index i from the n indices.

Suppose that two elements (bit vectors in this case) x and y have distance at most r_1 , i.e. $d(x, y) \leq r_1$. Then, since in total each of x and y has n bits, there are at least $(1 - r_1)n$ bits that x and y agree on. Hence, the probability we choose one of those bits (i.e. we choose h_i where i is the index of a same bit) is at least $\frac{(1-r_1)n}{n} = 1 - r_1$. Similarly, for the other side, observe that if $d(x, y) \geq r_2$, i.e. there are at most $n - r_2n$ bits same in x and y , then the probability of choosing a same bit index is at most $\frac{n-r_2n}{n} = 1 - r_2$. \square

3 LSH: Probability Amplification

There is no notion of absolute locality sensitive hashing - it is parametric. We can make statements of sensitivity to locality in terms of the four parameters s_1, s_2, p_1, p_2 . The parameters of a given LSH family may not be good enough for the given application. Hence, we use the general approach of probability amplification, the magic of independent trials, to bring the parameters to the desired range. Revisit Figures 2 and 3, the one guaranteed by the LSH specifications and the one we would ideally like with like $p_1 = 1$ and $p_2 = 0$, respectively.

For easier terminology, let us call two data items that hash to the same buckets as a **candidate pair** (for near duplicate problem). A LSH function takes as input a pair x and y and outputs *Yes* if x and y are a candidate pair and *No* otherwise, i.e. $h(x) = h(y)$ implies that h declares x and y a candidate pair. We do not need to go into the detail of how h computes the value. In fact, the values of $h(x)$ and $h(y)$, which are bucket IDs, are irrelevant. We only need to check for equality. Recall that two objects that are indeed near duplicates ($d(x, y) \leq t$) but hash to different buckets ($h(x) \neq h(y)$) is referred to as false negative whereas two objects that are not near duplicates ($d(x, y) \geq t$) that hash to the same bucket ($h(x) = h(y)$) are called false positive.

We will manipulate existing LSH function to bound the number of false positives and false negatives in a range that works for us.

Dealing with False Positives: We use several independent hash functions from \mathcal{H} and consider pairs that are declared candidate by **all** of them. This can be viewed as an **AND** operation on the output of the hash functions. We hash to the same buckets by the **AND** of (the output of) these hash functions. If two data items have large distance, then they are increasingly less likely to hash to the same bucket as the number of hash functions increase (because of the AND construction). Hence, dissimilar vectors are less likely to become candidate pair and there are fewer false positives.

Dealing with False Negatives: In order to reduce the false negatives, we give such pairs more chances to become candidate pairs. This is achieved by an **OR** construction. Again, use several independent hash functions from \mathcal{H} and consider pairs that are declared candidate by **any** of them. This can be viewed as an **OR** operation on the output of the hash functions. Similar vectors are now more likely to become candidate pairs.

3.1 Construction of new LSH families from old ones

A LSH scheme refers to constructing new families of LSH functions from existing ones with different parameters. We give two constructions to make new families of LSH families.

3.1.1 The AND Construction:

Let \mathcal{H} be a (s_1, s_2, p_1, p_2) -locality sensitive hash family. First, note that there is nothing special about similarity thresholds s_1 and s_2 , they could be $2/3, 1/3$ or $5/6, 1/8$ etc. See the proof above. The AND construction can be applied to \mathcal{H} as follows.

Each function h' in the new family \mathcal{H}' consists of r functions $h_{i1}, h_{i2}, \dots, h_{ir}$ from \mathcal{H} . \mathcal{H}' consists of $\binom{n}{r}$ functions of the form $h'_i = \{h_{i1}, h_{i2}, \dots, h_{ir}\} \in \mathcal{H}'$ operates such that $h'_i(x) = h'_i(y)$ if and

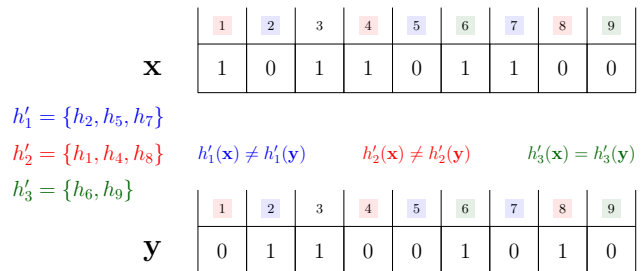


Figure 9 Applying the AND construction to (s_1, s_2, p_1, p_2) -LSH family \mathcal{H}

only if $\forall j h_{ij}(x) = h_{ij}(y)$. In other words, $h' \in \mathcal{H}'$ only declares a candidate pair if all r functions from \mathcal{H} do, as shown in Figure 9.

$$h'_i(x) = h'_i(y) \Leftrightarrow h_{i1}(x) = h_{i1}(y) \wedge h_{i2}(x) = h_{i2}(y) \wedge \dots \wedge h_{ir}(x) = h_{ir}(y)$$

It is easy to see that the functions in \mathcal{H}' are more restrictive than those in \mathcal{H} as it only declares a pair as a candidate pair if each of the r constituent functions from \mathcal{H} declare it as candidate pair. We establish the following property about function in \mathcal{H}' .

Lemma 2. \mathcal{H}' is a (s_1, s_2, p_1^r, p_2^r) -family of LSH functions

Proof. Its proof follows from the fact that if we randomly choose a function $h'_i = h_{i1}, h_{i2}, \dots, h_{ir}$ in \mathcal{H}' , it corresponds to choosing r independent random hash functions $\{h_{i1}, h_{i2}, \dots, h_{ir}\}$ from \mathcal{H} . For a pair of points x and y :

- If $\text{sim}(x, y) \geq s_1$, then $\Pr[h_{ij}(x) = h_{ij}(y)] \geq p_1$ for $h_{ij} \in \mathcal{H}$. Therefore,

$$\Pr[h'_i(x) = h'_i(y)] = \prod_{j=1}^r \Pr[h_{ij}(x) = h_{ij}(y)] \geq p_1^r$$

- Similarly, if $\text{sim}(x, y) \leq s_2$, then $\Pr[h_{ij}(x) = h_{ij}(y)] \leq p_2$, for $h_{ij} \in \mathcal{H}$. Therefore,

$$\Pr[h'_i(x) = h'_i(y)] = \prod_{j=1}^r \Pr[h_{ij}(x) = h_{ij}(y)] \leq p_2^r$$

□

Note that using an r -wise AND construction on \mathcal{H} results in \mathcal{H}' , which is a (s_1, s_2, p_1^r, p_2^r) -family of LSH functions and has both probabilities smaller than \mathcal{H} whereas our goal was to make only p_2 smaller. So we choose r such that p_2^r becomes very small (close to 0) but p_1^r is not very small.

3.1.2 The OR Construction:

Similar to the AND construction, let \mathcal{H} is a (s_1, s_2, p_1, p_2) -locality sensitive hash family. Each function $h'' = \{h_{i1}, h_{i2}, \dots, h_{ib}\}$ in the new family \mathcal{H}'' of $\binom{n}{b}$ functions consists of b functions $h_{i1}, h_{i2}, \dots, h_{ib}$ from \mathcal{H} and operates such that $h''_i(x) = h''_i(y)$ if and only if $\exists j h_{ij}(x) = h_{ij}(y)$. In other words, $h'' \in \mathcal{H}''$ declares a candidate pair if any of the r functions from \mathcal{H} do (see Figure 10).

$$h''_i(x) = h''_i(y) \Leftrightarrow h_{i1}(x) = h_{i1}(y) \vee h_{i2}(x) = h_{i2}(y) \vee \dots \vee h_{ir}(x) = h_{ib}(y)$$

\mathbf{x}	1	2	3	4	5	6	7	8	9
	1	0	1	1	0	1	1	0	0
$h'_1 = \{h_2, h_5, h_7\}$									
$h'_2 = \{h_1, h_4, h_8\}$	$h'_1(\mathbf{x}) = h'_1(\mathbf{y})$	$h'_2(\mathbf{x}) \neq h'_2(\mathbf{y})$							
$h'_3 = \{h_6, h_9\}$									
\mathbf{y}	0	1	1	0	0	1	0	1	0

Figure 10 Applying the OR construction to (s_1, s_2, p_1, p_2) -LSH family \mathcal{H}

It is easy to see that the functions in \mathcal{H}'' are less restrictive than those in \mathcal{H} as it declares a pair as a candidate pair if any of the b constituent functions from \mathcal{H} declares it as candidate pair. We establish the following property about function in $\text{cal}\mathcal{H}''$

Lemma 3. \mathcal{H}'' is a $(s_1, s_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)$ -family of LSH functions

Proof. Its proof follows from the fact that if we randomly choose a function $h_i'' = h_{i1}, h_{i2}, \dots, h_{ib}$ in \mathcal{H}'' , it corresponds to choosing b independent random hash functions $\{h_{i1}, h_{i2}, \dots, h_{ib}\}$ from \mathcal{H} . Note that the event that $h_i''(x) = h_i''(y)$ means that $\exists j \in [1, b], h_{ij}(x) = h_{ij}(y)$, which is equal to the complement of the event that $\forall j \in [1, b], h_{ij}(x) \neq h_{ij}(y)$. For a pair of points x and y :

- If $\text{sim}(x, y) \geq s_1$, then $\Pr[h_{ij}(x) = h_{ij}(y)] \geq p_1$, for any $h_{ij} \in \mathcal{H}$. Therefore,

$$\Pr[h_i''(x) = h_i''(y)] = 1 - \prod_{j=1}^b \Pr[h_{ij}(x) \neq h_{ij}(y)] \geq 1 - (1 - p_1)^b$$

- Similarly, if $\text{sim}(x, y) \leq s_2$, then $\Pr[h_{ij}(x) = h_{ij}(y)] \leq p_2$ for $h_{ij} \in \mathcal{H}$. Therefore,

$$\Pr[h_i''(x) = h_i''(y)] = 1 - \prod_{j=1}^b \Pr[h_{ij}(x) \neq h_{ij}(y)] \leq 1 - (1 - p_2)^b$$

□

Note that using b -wise OR construction on \mathcal{H} results in \mathcal{H}'' , which is a $(s_1, s_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)$ -family of LSH functions and has both probabilities larger than \mathcal{H} whereas our goal was to make only p_1 larger. So, we choose b such that $1 - (1 - p_1)^b$ becomes very large (close to 1) but $1 - (1 - p_2)^b$ doesn't grow too much.

3.2 Generic LSH scheme

The generic guiding principle of a LSH scheme is to compose the AND and OR construction so as to keep the number of false positives and false negatives in a bounded range. We combine both construction in a cascade to get the good of both worlds.

3.2.1 AND-OR Construction

Let \mathcal{H} be a (s_1, s_2, p_1, p_2) -LSH family. Apply the r -wise AND construction on \mathcal{H} to make a LSH family \mathcal{H}' that is a (s_1, s_2, p_1^r, p_2^r) family of LSH functions. Then, apply the b -wise OR construction on \mathcal{H}' to get a LSH family \mathcal{H}'' that is a $(s_1, s_2, 1 - (1 - p_1^r)^b, 1 - (1 - p_2^r)^b)$ family of LSH functions.

$$\mathcal{H} : (s_1, s_2, p_1, p_2)\text{-family} \xrightarrow{r\text{-wise AND}} \mathcal{H}' : (s_1, s_2, p_1^r, p_2^r)\text{-family}$$

$$\mathcal{H}' : (s_1, s_2, p_1^r, p_2^r)\text{-family} \xrightarrow{b\text{-wise OR}} \mathcal{H}'' : (s_1, s_2, 1 - (1 - p_1^r)^b, 1 - (1 - p_2^r)^b)\text{-family}$$

In other words, choose b collections of r independent random hash functions from \mathcal{H} , i.e. choose b meta hash functions f_1, f_2, \dots, f_b , where each $f_i \in \mathcal{H}'$ is the AND (concatenation) of r hash functions from \mathcal{H} . More formally, $f_i(x) = h_{i1}(x), h_{i2}(x), \dots, h_{ir}(x)$.

Then, a pair of item x and y are considered a **candidate pair** if

$$(f_1(x) = f_1(y)) \text{ OR } (f_2(x) = f_2(y)) \text{ OR } \dots \text{ OR } (f_b(x) = f_b(y))$$

You can visualize this as bands of $b \times r$ signature matrix. The final guarantees on the specification of \mathcal{H}'' follow from Lemma 2 and Lemma 3 applied to appropriate base LSH families.

This is called r -way AND construction followed by b -way OR construction and is denoted by (r, b) **AND-OR Construction**.

Theorem 4. *Let \mathcal{H} be a (s_1, s_2, p_1, p_2) -family of LSH functions. and let \mathcal{H}'' be the hash families constructed by (r, b) -AND-OR construction. Then, \mathcal{H}'' is $(s_1, s_2, 1 - (1 - p_1^r)^b, 1 - (1 - p_2^r)^b)$ family of LSH functions.*

Proof. The proof follows from successive applications of Lemma 2 and Lemma 3 above. We apply the OR construction on \mathcal{H}' , the output of the AND construction. Similarly, we apply result of Lemma 3 on the result of Lemma 2. \square

3.2.2 OR-AND Construction

We can also do the above composition in reverse order, i.e. b -way OR construction followed by r -way AND construction. This is denoted by (b, r) **OR-AND construction**.

$$\mathcal{H} : (s_1, s_2, p_1, p_2)\text{-family} \xrightarrow{b\text{-wise OR}} \mathcal{H}' : (s_1, s_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)\text{-family}$$

$$\mathcal{H}' : (s_1, s_2, 1 - (1 - p_1)^b, 1 - (1 - p_2)^b)\text{-family} \xrightarrow{r\text{-wise AND}} \mathcal{H}'' : (s_1, s_2, (1 - (1 - p_1)^b)^r, (1 - (1 - p_2)^b)^r)\text{-family}$$

Theorem 5. *Let \mathcal{H} be a (s_1, s_2, p_1, p_2) -family of LSH functions. and let \mathcal{H}'' be the hash families constructed by (b, r) -OR-AND construction. Then \mathcal{H}'' is $(s_1, s_2, (1 - (1 - p_1)^b)^r, (1 - (1 - p_2)^b)^r)$ family of LSH functions.*

Proof. The proof follows from successive applications of Lemma 2 and Theorem 1 above. We apply the AND construction on \mathcal{H}' the output of the OR construction. Similarly we apply result of Lemma 2 on the result of Lemma 3. \square

Multiple compositions one after the other can also be done.

3.2.3 The S -curve

We represent LSH families by 4 parameters s_1 and s_2 and the respective probabilities of candidacy when a pair has some given similarity. Regardless of b and r if we plot the function $(1 - (1 - p^r)^b)$, we get a S -shaped curve, i.e. for any value of b and r , we clearly see that this function stays low until a certain value of p , then rises (with reasonably steep slope) and then stays high for larger value of p . We try to understand how the values of b and r and the order of composition affect the S -curve with the help of examples.

Example of AND-OR Composition: Consider the AND-OR compositions for various values of p, r and b in Table 1.

	$(r, b) = (4, 4)$	$(r, b) = (4, 6)$	$(r, b) = (6, 4)$
p	$1 - (1 - p^r)^b$	$1 - (1 - p^r)^b$	$1 - (1 - p^r)^b$
0.1	0.0004	0.0006	0
0.2	0.00638	0.00956	0.00026
0.3	0.03201	0.04763	0.00291
0.4	0.09853	0.1441	0.01628
0.5	0.22752	0.32107	0.06105
0.6	0.42605	0.56518	0.17396
0.7	0.66655	0.80745	0.39387
0.8	0.8785	0.95765	0.70359
0.9	0.98601	0.99835	0.9518

Table 1 Effect of construction and values of b and r of steepness of the S -curve, AND-OR composition

Figure 11 plots $1 - (1 - p^r)^b$ from Table 1.

Note that a $(s_1, s_2, .2, .8)$ family is converted by

- $(r, b) = (4, 4)$ AND-OR construction into a $(s_1, s_2, 0.00638, 0.8785)$
- $(r, b) = (4, 6)$ AND-OR construction into a $(s_1, s_2, 0.00956, 0.95765)$
- $(r, b) = (6, 4)$ AND-OR construction into a $(s_1, s_2, 0.00026, 0.70359)$

There is a small range out of which the probability sharply decrease (for small values of p) or increase (for larger values of p). This is exactly what we want - recall our goal of the step function in Figure 5. We would choose b and r (for AND-OR construction) such that the higher probability p_2 is in this “right interval”, and the lower probability p_1 is on the left portion of the curve. Indeed any S -curve has a fixed-point, i.e. there is a p satisfying $p = 1 - (1 - p^r)^b$. Above this value of p , the probability of candidacy $(1 - (1 - p^r)^b)$ increases and vice-versa.

Example of OR-AND Composition: Consider the OR-AND compositions for various values of p, r and b in Table 2, the plot for which is shown in Figure 12.

Note that a $(s_1, s_2, .2, .8)$ family is converted by

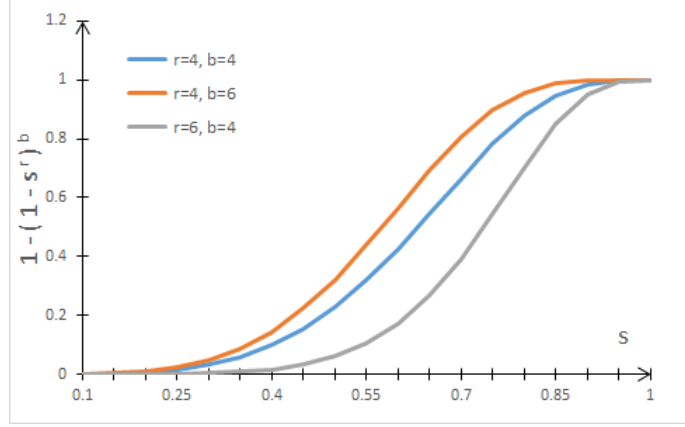


Figure 11 Plot of $1 - (1 - p^r)^b$ is an S -shaped curve for every b and r , AND-OR composition

	$(b, r) = (4, 4)$	$(b, r) = (4, 6)$	$(b, r) = (6, 4)$
p	$(1 - (1 - p)^b)^r$	$(1 - (1 - p)^b)^r$	$(1 - (1 - p)^b)^r$
0.1	0.01399	0.0482	0.00165
0.2	0.1215	0.29641	0.04235
0.3	0.33345	0.60613	0.19255
0.4	0.57395	0.82604	0.43482
0.5	0.77248	0.93895	0.67893
0.6	0.90147	0.98372	0.8559
0.7	0.96799	0.99709	0.95237
0.8	0.99362	0.99974	0.99044
0.9	0.9996	1	0.9994

Table 2 Effect of construction and values of b and r of steepness of the S -curve, OR-AND composition

- $(b, r) = (4, 4)$ OR-AND construction into $(s_1, s_2, 0.1215, 0.99362)$
- $(b, r) = (4, 6)$ OR-AND construction into $(s_1, s_2, 0.29641, 0.99974)$
- $(b, r) = (6, 4)$ OR-AND construction into $(s_1, s_2, 0.04235, 0.99044)$

Similar to the AND-OR construction, the S -curve for the OR-AND construction also has a fixed-point, i.e. there is a p satisfying $p = (1 - (1 - p)^b)^r$. Above this value of p , the probability of candidacy $((1 - (1 - p)^b)^r)$ increases and vice-versa.

4 LSH for other distances

We have given a LSH family for the Hamming distance. Next, we will construct basic LSH families for Jaccard, Cosine and Euclidean similarities.

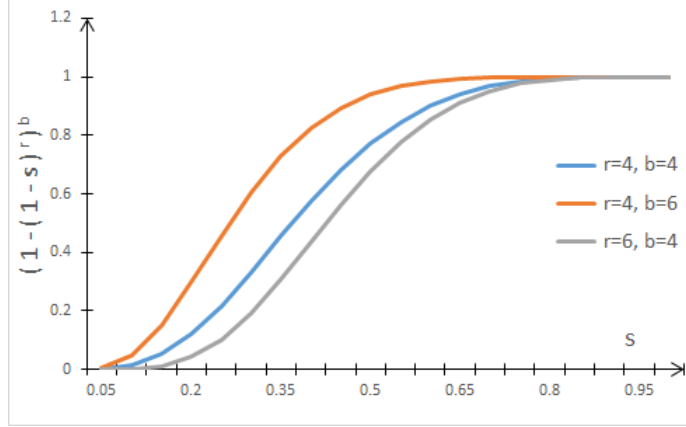


Figure 12 Plot of $1 - (1 - p^r)^b$ is an S -shaped curve for every b and r , OR-AND composition

We need is to construct a basic family \mathcal{H} of hash functions such that a randomly chosen function h from \mathcal{H} has the property of locality sensitivity i.e. it is (d_1, d_2, p_1, p_2) -locality sensitive. In this case the d_1 and d_2 will be measuring distance with respect to other (than Hamming) distances. In other words, for a random $h \in \mathcal{H}$, we want the property if $sim(x, y)$ is high, then with high probability $h(x) = h(y)$ and if $sim(x, y)$ is low, then with high probability $h(x) \neq h(y)$. With the amplification technique, we can adjust the parameters. Clearly, such hash functions will depend on the particular similarity but we know that not all similarities have such suitable hash functions.

4.1 Non-LSHable and Not Yet Known to be LSHable Measures

There are certain measures for which it is known that no LSH scheme is possible.

1. **Sørensen-Dice:** This is a similarity measure between sets. For two sets X and Y it is defined as

$$sim_{sd}(X, Y) = \frac{2|X \cap Y|}{|X| + |Y|}$$

Verify that for $X = \{a\}$, $Y = \{b\}$, and $Z = \{a, b\}$, we have $sim_{sd}(X, Y) = 0$, $sim_{sd}(X, Z) = 2/3$, and $sim_{sd}(Y, Z) = 2/3$

2. **Overlap Similarity:** This also is a similarity between sets defined as

$$sim_{ov}(X, Y) = \frac{|X \cap Y|}{\min\{|X|, |Y|\}}$$

Verify that for $X = \{a\}$, $Y = \{b\}$, and $Z = \{a, b\}$, we have $sim_{ov}(X, Y) = 0$, $sim_{ov}(X, Z) = 1$, and $sim_{ov}(Y, Z) = 1$.

In both cases, distance is defined to be $1 - sim_*(\cdot, \cdot)$.

For other distance measures, designing an LSH scheme is an open question and it is not yet known whether an LSH scheme can be made.

1. **Anderberg:** This is a similarity measure between sets. For two sets X and Y it is defined as

$$sim_{an}(X, Y) = \frac{|X \cap Y|}{|X \cap Y| + 2|X \oplus Y|}$$

Compute this similarity for pairs of $X = \{a\}, Y = \{b\}$, and $Z = \{a, b\}$

2. **Rogers-Tanimoto** This is a similarity measure between sets. For two sets X and Y it is defined as

$$sim_{rt}(X, Y) = \frac{|X \cap Y| + |X \cup Y|}{|X \cap Y| + |X \cup Y| + 2|X \oplus Y|}$$

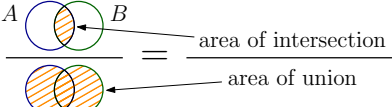
Compute this similarity for pairs of $X = \{a\}, Y = \{b\}$, and $Z = \{a, b\}$

4.2 Jaccard Distance

Recall that the Jaccard distance is defined on sets. For two sets S_1 and S_2 , their Jaccard similarity is defined as

$$J(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

The Jaccard distance is $1 - J(S_1, S_2)$.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{area of intersection}}{\text{area of union}}$$


It is not very difficult to prove that Jaccard distance is a distance metric.

4.2.1 Minhashing

LSH family for Jaccard distance are called **Minhashes** or Minimum-wise hashing [1].

Let U be the universal set, of which all sets are a subset. For example, if sets are documents, then U could be the English lexicon. Let \mathcal{H} be the set of all permutations of elements in U . We will show that \mathcal{H} is a family of LSH function. For a permutation π of elements in U the hash function h_π has the following properties.

- h_π is of the form $h_\pi : 2^U \mapsto U$ (takes as input a subset of U and returns an element of U)
- h_π maps a set $S \subseteq U$ such that $h_\pi(S)$ is the first element of S in the order of π

Note that $|\mathcal{H}| = |U|!$

Example: Let $U = \{w_0, w_1, w_2, w_3, w_4\}$ and the permutation $\pi = (w_1, w_4, w_0, w_3, w_2)$. Let the four sets S_1, S_2, S_3, S_4 be described by the characteristic vectors as in the following table and reordered according to π .

Elem.ID	S_1	S_2	S_3	S_4
w_0	1	0	0	1
w_1	0	0	1	0
w_2	0	1	0	1
w_3	1	0	1	1
w_4	0	0	1	0

Elem.ID	S_1	S_2	S_3	S_4
w_1	0	0	1	0
w_4	0	0	1	0
w_0	1	0	0	1
w_3	1	0	1	1
w_2	0	1	0	1

Table 3 Given sets (left) and sets reordered according to π (right)

Then, $h_\pi(S_1) = w_0$, $h_\pi(S_2) = w_2$, $h_\pi(S_3) = w_1$, $h_\pi(S_4) = w_3$.

After reordering, $h_\pi(S)$ is the index of row (the element ID) with first 1 in the permuted order π . The scheme is called min-wise hashing or minhashing because of this first index (minimum index).

Let \mathcal{H} be a family of functions corresponding to all permutations of elements in U . Recall we assumed data is m -dimensional, i.e. $|U| = m$, so there are $m!$ functions in \mathcal{H} . Then,

Theorem 6. \mathcal{H} is a $(d_1, d_2, (1 - d_1), (1 - d_2))$ -family of LSH functions

Proof. Let h_π be chosen at random from \mathcal{H} , this correspond to choosing a random permutation π of U . Let S and T be two arbitrary subsets of U . Suppose $d(S, T) \leq d_1$. Note that the event that $h_\pi(S) = h_\pi(T)$ is the event that the first element in the order of π is the same in both S and T . In other words, if we picture S and T as two columns with rows (elements of U ordered by π), then this is the event that we get a [1 1] row before any [1 ; 0] and [0 1] row (as we ignore any [0 0]). Since π is a random permutation, the probability of this happening is

$$\frac{\text{No. of [1 1] rows}}{\text{No. of [1 1], [1 ; 0], [0 1] rows}},$$

which is the just the Jaccard similarity between S and T or $1 - d_J(S, T)$. Thus

$$Pr[h_\pi(S) = h_\pi(T)] \geq 1 - d_1$$

Thus, $Pr[h_\pi(S) = h_\pi(T)] \geq 1 - d_1$. The other bound for d_2 is analogous. \square

Remark 1. *It is not very easy to pick a random permutation and even after picking a permutation, processing the sets and finding their minhash values is computationally expensive as it needs sorting by π and finding the first 1. This becomes even more problematic when U is very large, as then every column generally would have a lot of 0's (sparse matrix). Therefore, we can use an approximate minhashing scheme.*

4.2.2 Approximate Minhashing

An approximate way is to use universal hash functions (the one we discussed to implement the Dictionary ADT) as an alternative to random permutation. Recall that permutation is of the form $\pi : [m] \mapsto [m]$ (it is a bijective function with no collisions). If we take a universal hash function $h : [m] \mapsto [m]$ or even better $[m] \mapsto [2m]$, there will be collisions, but very few. This

is a reasonably good approximation to a permutation as the order of two elements $w_i, w_j \in U$ is determined based on whether or not $h(w_i) < h(w_j)$. By the randomness of h we get that either order is equally likely. The (approximate) minhash value is then computed as follows:

$$\text{minhash}(S) = \arg \min_{w \in S} h(w)$$

The good thing about simulating a permutation with a hash function is that now, we only need to compute the minimum of elements that are actually in S and ignore the 0 rows in the column of S .

4.3 Cosine Distance

Cosine distance is good for discrete versions of Euclidean Space. For example, the ratings of two movies by three users. Here, we ignore the magnitude of the vector and only consider its direction, i.e. a vector is the same as a unit vector in that direction.

Cosine distance is the angle between two vectors in the range $[0^\circ - 180^\circ]$ and is calculated by first computing the cosine of the angle between the two vectors and then computing the arc-cosine to get the angle.

$$\cos \theta = \frac{u \cdot v}{\|u\| \|v\|} = \frac{\sum_{i=1}^m u_i v_i}{\|u\| \|v\|} = \left(\frac{u}{\|u\|} \right)^T \left(\frac{v}{\|v\|} \right)$$

Cosine distance is a distance metric, as we consider a vertex and its scalar multiples the same (i.e. we only consider unit vectors).

Vectors could be in any number of dimensions but they always define a plane and the angle between them is measured in this plane. This angle between vectors ranges from 1° to 180° .

Usually vectors take positive values as each coordinate takes positive values. In that case, the similarity will range between 0° and 90° , i.e. $d_{\cos} \in [0^\circ - 90^\circ]$. For example, when the coordinates are word frequencies (bag of word model) or TF-IDF features.

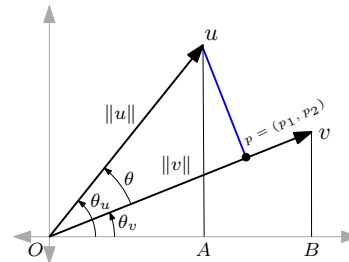


Figure 13 Cosine distance example

4.3.1 simHashing: LSH for Cosine Distance

A LSH family \mathcal{H} for cosine distance for points in \mathbb{R}^m can be made as follows [2].

Choose a **hyperplane** h in \mathbb{R}^m . A hyperplane is just a line in $2d$, a plane in $3d$, and is $d-1$ dimensional subspace of \mathbb{R}^d . Every hyperplane divides the space in two half-spaces (that we refer to as upper or positive and lower or negative half-spaces). The function f_h with respect to a hyperplane h assigns every vector in the upper half-space to bucket $+$ and every vector in the lower half-space to bucket $-$ as shown in the Figure 14.

For a hyperplane h , we say that two vectors u and v share a hash bucket (becomes a candidate pairs) if the corresponding hash function f_h maps them to the same bucket i.e. $f_h(u) = f_h(v)$ otherwise they do not become candidate pair.

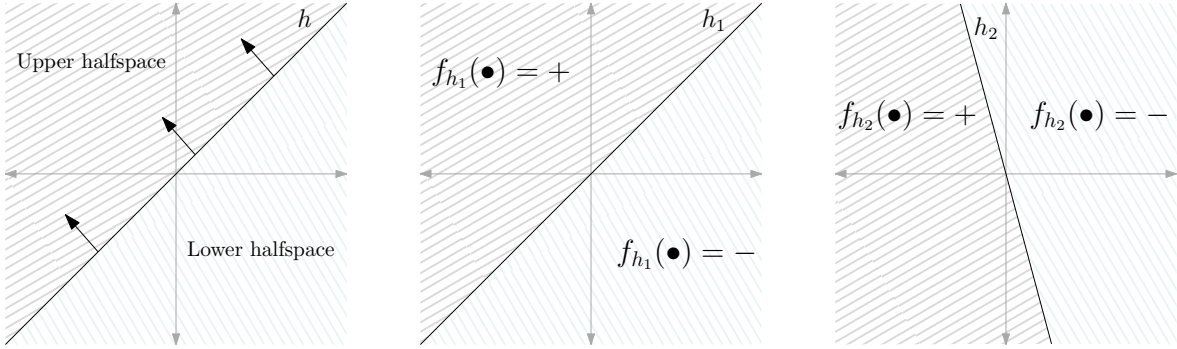


Figure 14 Division of upper and lower half spaces, \mathbf{u} and \mathbf{v} is a candidate pair if $f_h(\mathbf{u}) = f_h(\mathbf{v})$ otherwise they do not become candidate pair

The same concept applies to higher dimensions. In Figure 15 a hyperplane (a $2d$ plane) splits the $3d$ space into two half spaces. We show only a sphere, as without loss of generality we may consider only unit vectors and work with the unit ball in \mathbb{R}^d only, as our concern is the angle between vectors. Here too, any vector in the upper half-space is mapped to $+$ and vectors in the lower half-space are mapped to $-$ by the function corresponding to the given hyperplane h .

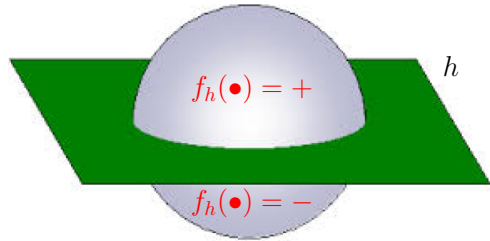


Figure 15 A higher dimension example: A hyperplane (a $2d$ plane) splits the $3d$ space into two half spaces

Let \mathbf{x} and \mathbf{y} be two vectors with angle θ_{xy} between them, as in Figure 16. The probability that a random hyperplane h goes between the two vectors is exactly $\theta_{xy}/180^\circ$.

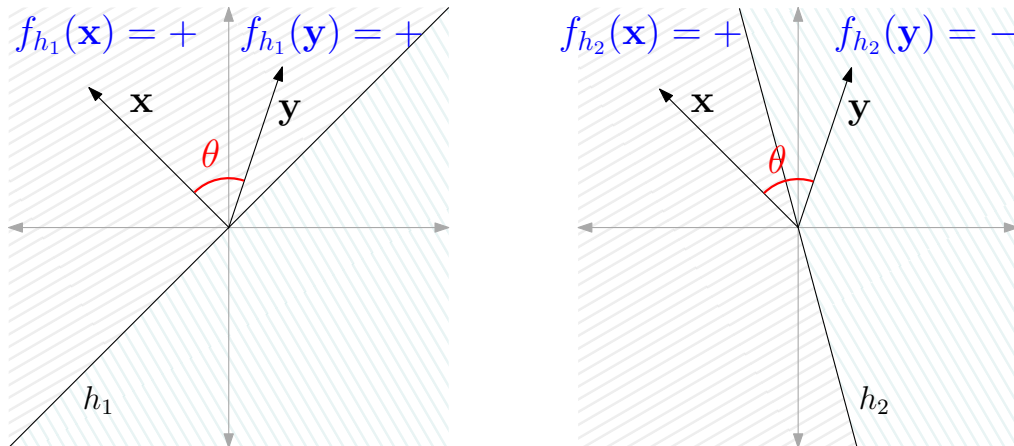


Figure 16 Vector \mathbf{x} and \mathbf{y} is a candidate pair under f_{h_1} (left) but not under f_{h_2} (right)

Let f_{h_1} and f_{h_2} in \mathcal{H} correspond to hyperplanes h_1 and h_2 . Since $f_{h_1}(\mathbf{x}) = f_{h_1}(\mathbf{y})$, \mathbf{x} and \mathbf{y} is a candidate pair under f_{h_1} but not under f_{h_2} .

Let \mathcal{H}_{\cos} be the family of functions corresponding to $m - 1$ -dimensional hyperplanes (passing through origin of \mathbb{R}^m). Note that \mathcal{H}_{\cos} has infinitely many functions. Then,

Theorem 7. \mathcal{H}_{\cos} is a $(d_1, d_2, \binom{180-d_1}{180}, \binom{180-d_2}{180})$ -family of LSH functions

Proof. Let f_h be a randomly chosen function from \mathcal{H}_{\cos} . This corresponds to choosing a random hyperplane h . Suppose that if

- Suppose $d_{\cos}(x, y) \leq d_1$, then there is at least $\binom{180-d_1}{180}$ chance that h does not separate \mathbf{x} and \mathbf{y} , i.e. $f_h(\mathbf{x}) = f_h(\mathbf{y})$
- On the other hand, suppose $d_{\cos}(x, y) \geq d_2$, then there is at most $\binom{180-d_2}{180}$ chance that h does not separate \mathbf{x} and \mathbf{y} , i.e. $f_h(x) = f_h(y)$

Combining the above two statements we get the statement of the theorem □

This family we can amplify as we wish. In particular, note that the base family has infinitely many functions unlike LSH for Hamming and Jaccard similarity which have a limited of n and $n!$ functions, respectively, in the base family.

4.3.2 Computation of LSH for Cosine Function

Computationally, we accomplish the hashing by not picking a random hyperplanes, as it is not easy to find the half-space where vector x lies, but instead picking a random unit vector v and consider the hyperplane to which the vector v is normal. This unit vector v “uniquely” represent the hyperplane, but makes computation very easy. Technically, there are infinitely many vectors that are normal to a hyperplane, i.e. all scalings of v . But we consider unit vectors only, and there are only two of them, v and $-1v$, that are normal to the same hyperplane.

Recall that the hyperplane to which v is normal is exactly the family of vectors (the $m - 1$ dimensional subspace) whose dot-product with v is 0. And it is not very hard to picture (at least in $2d$) that the vectors that are in the upper half-space are those whose dot-product with v is positive (> 0) and those in the lower half-space have dot-product with v negative (< 0).

Thus, $f_h(x)$ is compute as follows. Let v be a normal to the hyperplane h , then

$$f_h(x) = \text{sign}(v \cdot x), \quad \text{where} \quad \text{sign}(a) = \begin{cases} + & \text{if } a \geq 0 \\ - & \text{otherwise} \end{cases}$$

Remark 2. *Choosing a random unit vector v in \mathbb{R}^d , is just choosing a random direction in \mathbb{R}^d , which is a challenging task in random number generation. Recall how we choose approximately random directions for dimensionality reduction.*

4.4 Euclidean Distance

Euclidean distance is the most well-known and common distance measure. It's general applicability, the nice geometric interpretation (shortest straight line distance between two points), and the fact that we can do all kind of geometric and algebraic operations on these vectors make it very useful. As we have earlier discussed, one has to be really careful about the scale of each coordinate; all coordinates should be on a common scale and in the same units (or unitless).

4.4.1 LSH for Euclidean distance

The overall idea of LSH for Euclidean distance is that if two points (in m -dimensional Euclidean space) are “close” together and if we project them onto some other vector, then they should remain “close” to each other [3].

Suppose points in X are vectors in \mathbb{R}^m . Let ℓ be a line in \mathbb{R}^m passing through the origin, i.e. v is a unit vector in the direction of ℓ and the line is the span of this v . Let $a > 0$ be a fixed constant that is the length of segments into which the line ℓ is divided. Each of these segments is a bucket for the hash function corresponding to ℓ (Figure 17). The function $h_v = h_\ell$ (corresponding to the line ℓ or the unit vector v) maps a vector \mathbf{x} to the bucket ID (segment of ℓ) where the projection of x on ℓ lies, i.e.

$$h_v(x) = \left\lfloor \frac{\langle \mathbf{x}, \mathbf{v} \rangle}{a} \right\rfloor$$

Essentially, h_v projects \mathbf{x} onto v and then discretize the projection into a multiple of a .

Now the LSH for family $\mathcal{H}_{Euc} = \mathcal{H}$ is composed of the infinitely many functions corresponding to unit vectors in \mathbb{R}^m . Next, we analyze the locality sensitivity of these functions. Intuitively, as we discussed above, if two vector \mathbf{x} and \mathbf{y} are close to each other, then it is likely that they will fall into the same bucket and vice-versa. Indeed, the inverse of this statement is a little tricky, i.e. far vectors are less likely to fall into the same bucket.

Let d be the distance between two points x and y . Depending on how large or small d compared to a , we can calculate the chances of whether or not x and y will fall into the same bucket, i.e. $Pr[h_v(\mathbf{x}) = h_v(\mathbf{y})] \propto d(\mathbf{x}, \mathbf{y})$. It also depends on the angle between the line ℓ and line segment joining the two points x and y . Let's discuss few scenarios before we establish probabilistic guarantees.

If $d(\mathbf{x}, \mathbf{y})$ is small compared to a , then it is likely that \mathbf{x} and \mathbf{y} will fall in the same bucket, though it is not necessary. \mathbf{x} and \mathbf{y} may fall close to the boundary of two adjacent buckets,

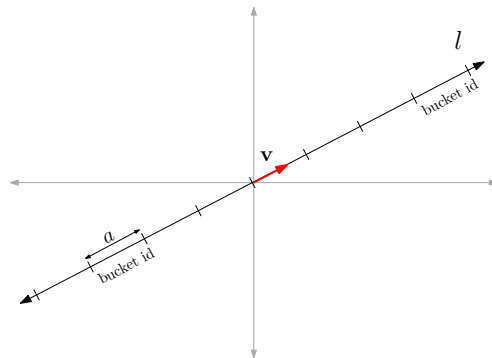


Figure 17 A unit vector v in the direction of line ℓ (passing through the origin), $a > 0$ is the fixed constant dividing the line into equal segments

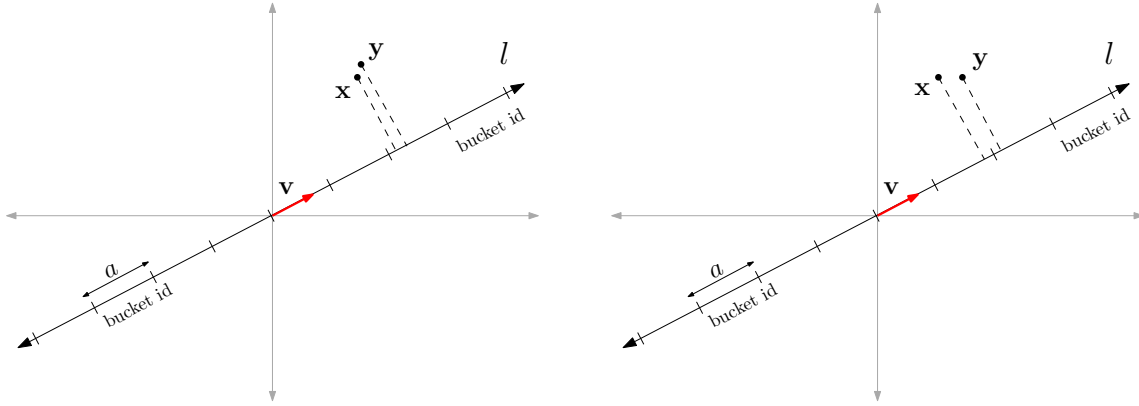


Figure 18 Vector \mathbf{x} and \mathbf{y} will fall in the same bucket as the $d(\mathbf{x}, \mathbf{y})$ is small compared to a (left), though it is not necessary, like the vector \mathbf{x} and \mathbf{y} may fall close to the boundary of two adjacent buckets (right)

as in Figure 18. Even if say \mathbf{x} is really at the boundary, there is still a good chance that they will go to the same bucket (the event that \mathbf{y} is on the “left side” of \mathbf{x} as in Figure 18).

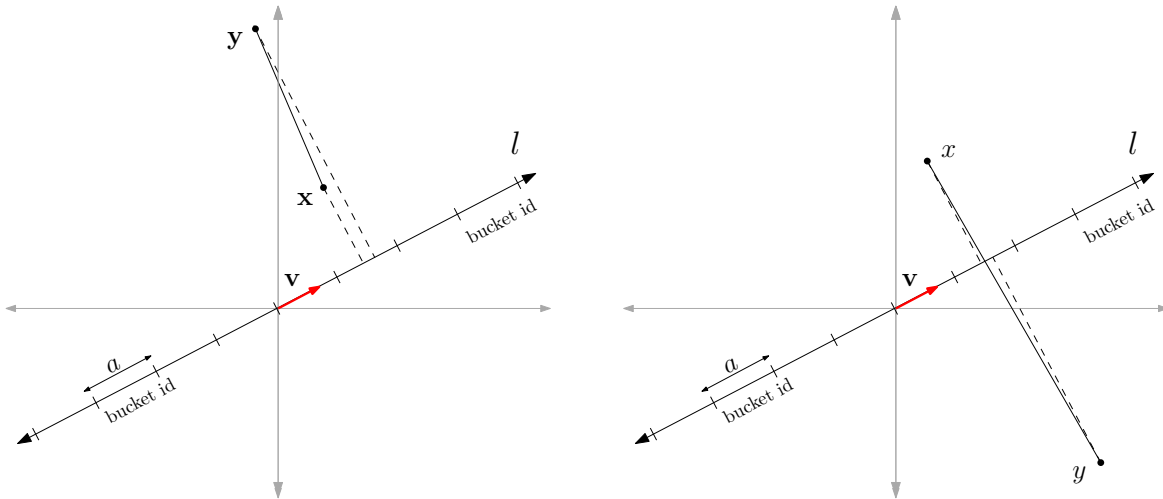


Figure 19 Vector \mathbf{x} and \mathbf{y} unlikely to fall in one bucket, if $d(\mathbf{x}, \mathbf{y})$ is large compared to a (left), if line segment joining \mathbf{x} and \mathbf{y} is almost perpendicular to ℓ , then it is likely that they fall in same bucket (right)

If $d(x, y)$ is large compared to a , x and y unlikely to fall in one bucket, though it is not necessary as in Figure 19. If d is large but line segment joining x and y is almost perpendicular to ℓ , then it is still likely that they fall in same bucket. In other words, if d is large, the angle has to be close to 90° for them to go to the same bucket.

The precise dependence of x and y going to the same bucket, i.e. $h_v(x) = h_v(y)$ and the angle between the \overline{xy} segment and ℓ is as follows. If x and y go to the same bucket, then $d \cos \theta < a$ as in Figure 20. Note that this is only a necessary condition, not sufficient, i.e. even $d \cos \theta \ll a$, the two points x and y may still go to different buckets.

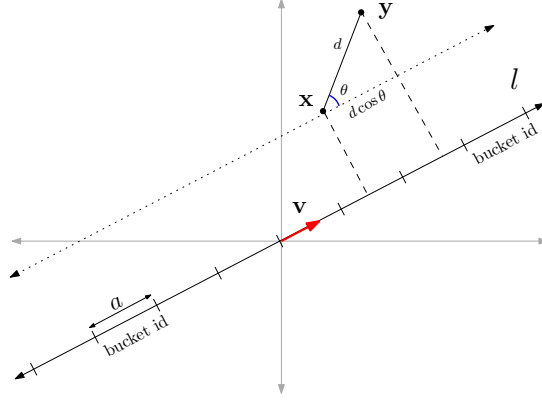


Figure 20 If $d \cos \theta < a$ then x and y will go to the same bucket

Theorem 8. \mathcal{H}_{Euc} is a $(\frac{a}{2}, 2a, \frac{1}{2}, \frac{1}{3})$ -family of LSH functions

Proof. Choose h at random from \mathcal{H}_{Euc} , which corresponds to choosing a random line ℓ in \mathbb{R}^m . Note that the angle θ between ℓ and the line through x and y is random.

- Let $d(x, y) < \frac{a}{2}$. Since the distance is very small, there is at least half chance that the projection of x on l is smaller than that of y . This implies that even if one of the points is really at the border of the bucket there is at least half a chance that the other one is close to middle of the same bucket. Thus, $Pr[h_\ell(x) = h_\ell(y)] \geq \frac{1}{2}$
- Let $d(x, y) = d > 2a$. Let d' be the distance between the projections of x and y on l . From Figure 20, it is clear that $d' = d \cos \theta$. If x and y go to the same bucket, then we must have $d' < a \implies d \cos \theta < a \implies 2a \cos \theta < a \implies \cos \theta < \frac{1}{2} \implies \theta \in [60^\circ, 90^\circ]$. Since θ is random, the probability that $\theta \in [60^\circ, 90^\circ]$ (rather than $\theta \in [0^\circ, 60^\circ]$) is $\frac{1}{3}$.

Combining the above two bounds, we get the statement of the theorem, but note that these bounds are very loose. \square

Finally, note the difference between this LSH-family \mathcal{H}_{euc} and those for other distance measures. The other ones are very strong, where we got that for any d_1 and d_2 , the probabilities were $(1-d_1)$ and $(1-d_2)$. Here for ell_2 distances, for any distance $d_1 < d_2$, all we get is $p_1 > p_2$. With amplification techniques applied to more number of functions, these probabilities can be brought to desired values, for which we have infinitely many functions available.

5 Computational Issues

The memory requirements for LSH can be reduced using an implementation trick. Given that the resulting hash tables have at-most n non-zero entries, one can reduce the amount of memory used per each hash table to $O(n)$ using standard (universal) hash functions.

6 Data Dependent LSH

All LSH schemes we discussed are sensitive to specific distance measure, but they are all data oblivious (they do not look at the data). A data dependent LSH scheme is Clustering LSH, which cluster datasets into k clusters, as in Figure 21 (using some method and proximity measure) and each clusters serves as a hash bucket. Bucket ID of each point is it's cluster id.

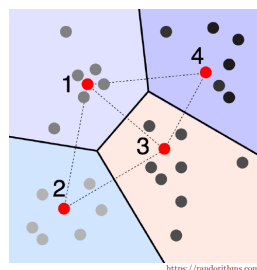


Figure 21 Clustering LSH

References

- [1] A. Broder. On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of SEQUENCES*, pages 21–29, 1997.
- [2] M Charikar. Similarity estimation techniques from rounding algorithms. In *ACM symposium on Theory of computing (STOC '02)*, page 380–388, 2002.
- [3] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational geometry (SCG '04)*, page 253–262, 2004.
- [4] P. Indyk and R. Motwani. Approximate nearest neighbor: towards removing the curse of dimensionality. In *Proceedings of the Symposium on Theory of Computing (STOC 1998)*, 1998.