Locality-Sensitive Hashing

Andrew Wylie

December 2, 2013

Chapter 1

Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) is a method which is used for determining which items in a given set are similar. Rather than using the naive approach of comparing all pairs of items within a set, items are hashed into buckets, such that similar items will be more likely to hash into the same buckets. As a result, the number of comparisons needed will be reduced; only the items within any one bucket will be compared. Locality-sensitive hashing is often used when there exist an extremely large amount of data items that must be compared. In these cases, it may also be that the data items themselves will be too large, and as such will have their dimensionality reduced by a feature extraction technique beforehand.

The main application of LSH is to provide a method for efficient approximate nearest neighbor search through probabilistic dimension reduction of high-dimensional data. This dimensional reduction is done through feature extraction realized through hashing (eg. minhash signatures), for which different schemes are used depending upon the data. LSH is used in fields such as data mining, pattern recognition, computer vision, computational geometry, and data compression. It also has direct applications in spell checking, plagiarism detection, and chemical similarity.

In this chapter, we will give an illustration of the technique using the problem of finding similar documents, while concurrently showing supporting theory. Following this will be core theory including the definition of locality-sensitive hash families, and examples of families for different distance measures. Throughout the chapter, examples are also given for the application of families to illustrate their usage.

1.1 Similarity of Documents

We'll begin to introduce LSH by an illustration of the technique using the problem of finding similar documents. This problem appears, for example, on the web when attempting to find similar, or even duplicate web pages. A search engine would use this technique to allow these similar documents to either be grouped or only shown once on a results page, so that other possible search matches could also be prominently displayed.

Problem 1.1.1 *Given a collection of web-pages, find the near duplicate web-pages.*

Clearly the content of the page is what matters, so for a preprocessing step any HTML tags are stripped away and main textual content is kept. Typically multiple white spaces are also replaced by a singe space during this preprocessing. As a result we're left with a document containing a string of text characters. Keeping in mind that we want to compare similarity of documents opposed to exact equality, the text is split up into a set of smaller strings by a process called shingling. This allows the documents to be represented as sets, where fragments of documents can match others. Shingle length is an important factor in this, though will not be explained in depth. Generally though, the shingle length k should be large enough so that the probability of any given k-shingle appearing in any given document is relatively low. For our purposes, it is sufficient to know that choosing $k = 5$ works well for electronic mail, and $k = 9$ is suitable for large text documents.

Definition 1.1.2 k-shingle*: A k-shingle of a text document is defined to be any substring of length k which appears in the document.*

Example 1.1.3 *Let the document $D = \{adbdabadbcdab\}$, and $k = 2$. Then the possible k-shingles for D are: $\{ad, db, bd, da, ab, ba, bc, cd\}$.*

Using shingling, we have seen that a document can be represented as a set of k-shingles. However we can work instead with integers by using a hash function to map each k-shingle to an integer. This is also useful to reduce space requirements, as a regular 9-byte shingle can be converted to a 4-byte integer. Note however that even if we represent shingles this way, we still may be using 4 times more space than the original document. A solution to this is shown in the next section.

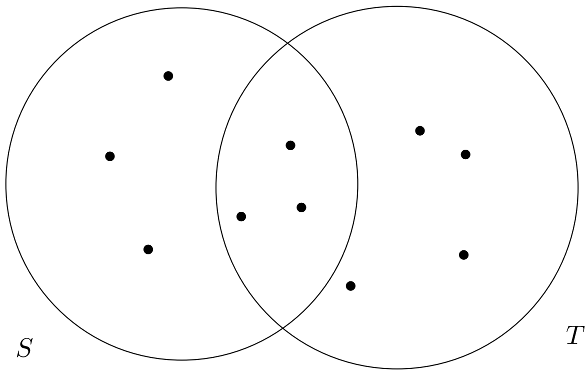
Example 1.1.4 *Let $k = 9$, and let $\mathcal{H}$ be a hash function that maps the set of characters to integers; $\mathcal{H} : |C|^9 \to \mathbb{Z}$. Let $|\mathbb{Z}| \leq 2^{32} - 1$.*

Now that we have documents mapped to sets of k-shingles, we can use a similarity measure called Jaccard Similarity to compare the two sets. The Jaccard Similarity is defined with respect to two sets S and T, and is the ratio of the size of the intersection of S and T to the size of their union. As a result of this, we can also redefine our problem statement.

Problem 1.1.5 *Given a collection of sets, find the pairs which have large intersections.*

Definition 1.1.6 Jaccard Similarity*: Let S and T be two sets. Define the Jaccard Similarity of the sets S and T as $\frac{|S \cap T|}{|S \cup T|}$.*

Example 1.1.7 *Shown below are two sets S and T. The intersection $|S \cap T| = 3$, with the union $|S \cup T| = 10$. Therefore their Jaccard similarity is $3/10$.*



1.2 Similarity-Preserving Summaries of Sets

As mentioned in the previous section, we'll now present a solution for the storage of sets of shingles using smaller representations called signatures. These signatures will also have the property that the similarity measure between any two will be approximately the same as the similarity between the two sets of shingles which they are derived from.

First, let's consider the natural representation of a set. We have a universe U from which elements of the set are drawn, which we'll assume are arbitrarily ordered. A set S with elements from U can be represented by a 0-1 vector of length $|U|$, where a 1 represents that the corresponding element from the universe is present in the set, and a 0 represents that element's absence from the set. Similarly for a collection of sets over the universe U, we can associate a *characteristic matrix*, where each column represents the vector corresponding to a set and each row corresponds to an element of U.

An example is given below in Table 1.1, where we have four sets (representing households in a neighborhood) and a universe U consisting of five elements (possible vacationing destinations). From this *characteristic matrix*, we can ascertain that set S_1 prefers cruise and safari, while S_2 just loves to visit resorts.

	S_1	S_2	S_3	S_4
Cruise	1	0	0	1
Ski	0	0	1	0
Resorts	0	1	0	1
Safari	1	0	1	1
Stay at Home	0	0	1	0

Table 1.1: A characteristic matrix for 4 sets with a universe consisting of 5 elements.

One effective way to compute the signature for a collection of sets is to use *minhashing*. For *minhashing*, the rows of the characteristic matrix are first randomly permuted. Then for each set (column in the characteristic matrix), its minhash value h is the number of the first row which is a 1. Using the previous example, suppose the permutation of rows results in Table 1.2 as shown below. The minhash values are then: $h(S_1) = 2$, $h(S_2) = 4$, $h(S_3) = 1$, and $h(S_4) = 2$.

	S_1	S_2	S_3	S_4
Ski	0	0	1	0
Safari	1	0	1	1
Stay at Home	0	0	1	0
Resorts	0	1	0	1
Cruise	1	0	0	1

Table 1.2: Characteristic matrix after the permutation of rows of Table 1.1.

In the following lemma we establish an important connection between the Jaccard similarity of two sets and the probability that their minhash values are the same after a random permutation of the rows of the characteristic matrix. In fact, we'll be able to shown that the Jaccard similarity is equal to the probability that these minhash values are the same.

Lemma 1.2.1 *For any two sets S_i and S_j, the probability that $h(S_i) = h(S_j)$ is equal to the Jaccard similarity of S_i and S_j.*

Proof. Focus on the columns representing the sets S_i and S_j in the characteristic matrix before the random permutation. For any row, the entries corresponding to these columns are either; (a) both 0, (b) both 1, or (c) one is 0 and the other is 1. Let X be the number of rows of type (b) and let Y be the number of rows of type (c). Observe that the Jaccard similarity of S_i and S_j is $\frac{|X|}{|X|+|Y|}$.

Now, what is the probability that when we scan the rows from top to bottom, after the random permutation, we meet a type (b) row before a type (c) row? This is exactly $\frac{|X|}{|X|+|Y|}$, which is also precisely when $h(S_i) = h(S_j)$. ∎

Next, we'll turn our attention to *minhash signatures*, which are smaller matrices created from repeatedly minhashing a characteristic matrix. Let M denote the characteristic matrix for a set system S with universe U. Pick a set of n random permutations of rows of the characteristic matrix. For each set in S compute its h-value with respect to each of the n-permutations. This results in a *signature matrix* — it consists of $|S|$ columns and n-rows, and the (i, j)-th entry corresponds to the signature of the j-th set with respect to the i-th hash function. Typically the signature matrix should be significantly smaller than M in size, as the size of universe U will be larger than n.

Example 1.2.2 *Shown below is the minhash signature created from Table 1.1 using $n = 2$ permutations (using functions $h_1 = x + 1 \bmod 5$ & $h_2 = 3x + 1 \bmod 5$).*

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Table 1.3: A signature matrix for 4 sets.

Computation of this signature matrix should be done in an efficient way, though following the explanation given above we'd be calculating n random permutations of a large characteristic matrix M; a very time-intensive task. Clearly we should attempt to avoid these permutations. To do this, we can use n random hash functions h to 'permute' the rows of M. We'll assume the mapping of $|U|$ elements onto themselves won't cause many collisions, however as there are certain to be some collisions $h(r)$ is not a 'perfect permutation'. So, given a row r of M, we can say that $h(r)$ denotes the new row number after a permutation. Shown below is an outline of the required steps to compute this signature matrix.

Step 1: Pick n random hash functions $h_1, \ldots, h_n$.

Step 2: Execute the following three steps for each row r of M.

1. Set Signature$(r, c) := \infty$, for $c = 1, \ldots, |S|$.
2. Apply each h-function to r for the new row numbers $h_1(r), \ldots, h_n(r)$ of r.
3. For $c = 1, \ldots, |S|$; if $M[r, c] = 1$ then Signature$(r, c) := \min(h_i(r), \text{Signature}(r, c))$.

Observe that for each non-zero entry of M, we perform $O(n)$ computations, and we do not need any additional memory except to store the description of n hash functions and the characteristic matrix M.

Although the signature matrix is fairly small compared to the characteristic matrix, its size could still be large. Let's look at the example from [18, Example 3.9], where there are signatures of length 250 for a set of one million documents. The total size of the signature matrix will be 1GB. Moreover, if documents are compared pairwise, the computation will be of the order of comparing a trillion documents!

The above example suggests that we should avoid the computation of pairwise similarity of documents as this requires $\binom{|S|}{2}$ comparison computations, which for a large number of documents any type of signatures will be prohibitively expensive. As an answer to this, we'll use Locality Sensitive Hashing (LSH) to quickly find any sets of documents which are likely to be similar, without using direct comparisons.

1.3 LSH for Minhash Signatures

The general idea, as expressed at the beginning of this chapter, is to hash documents in such a way that similar documents are more likely to be hashed into the same bucket as opposed to dissimilar documents. The documents will each be hashed multiple times, with the intent of altering the probability that similar items will become candidates while dissimilar items do not. The key point is that only the documents which fall into the same bucket are considered as potentially similar. If two documents do not map to the same bucket in any of the hashings then they are never considered as similar in this technique. Obviously, we want to minimize the number of dissimilar documents falling into the same bucket. At the same time, we hope that similar documents at least hash once into the same bucket. The good thing is that we have a methodology to achieve these objectives.

Let us assume that we have computed the minhash signature matrix for the set of documents of the previous section. We are going to partition the rows of this matrix into $b = n/r$ bands, where each band is made of r-consecutive rows. We'll assume that r divides n. For each band we define a hash function $h : \mathbb{R}^r \to \mathbb{Z}$, which takes a column vector of length r and maps it to an integer (i.e. a bucket). If we want we can even choose the same hash function for all the bands, but the buckets are kept distinct for each band. Now if two vectors of length r in any one of the bands hash to the same bucket, we declare that the corresponding documents are potentially similar. That's it — that's the technique. Let's do some analysis.

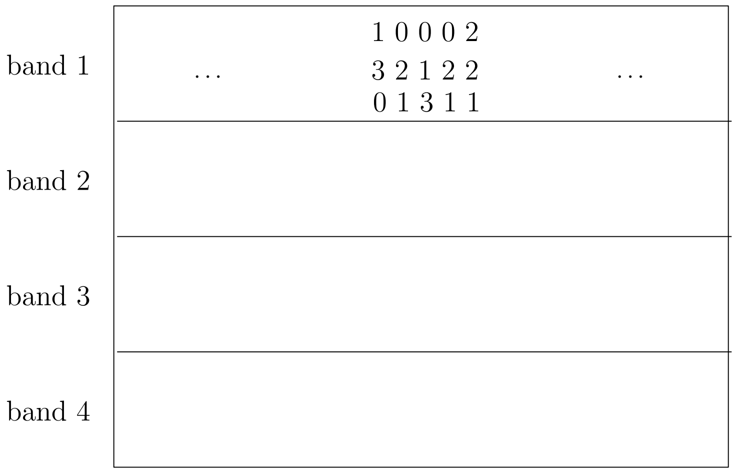


Figure 1.1: A signature matrix which has been separated into four bands of three rows each.

Lemma 1.3.1 *Let s be the Jaccard similarity of two documents. The probability that the minhash signature matrix agrees in all the rows of at least one band for these two documents is $1-(1-s^r)^b$. Note that this is also the probability that they will become a similar pair after the hashing and banding.*

Proof. Recall from Lemma 1.2.1, the probability that the minhash signature for these two documents are the same in any particular row of the signature matrix is s. Therefore, the probability that the signatures agree in all the rows in one particular band is s^r. The probability that the signature will not agree in at least one of the rows in this band is $1 - s^r$.

Next let us calculate the probability that each band of the signature will not agree in at least one of its rows. This will be $(1 - s^r)^b$ as there are b bands. Therefore, the probability that the signatures will agree in at least one of the bands is $1 - (1 - s^r)^b$. ∎

The function $1-(1-s^r)^b$ can be plotted with the x-axis representing values of s and the y-axis representing the output values of the function, with the values of b & r fixed. The curve created by this function has the form of an S-shaped curve. In fact, the curve will have this shape regardless of the values of b and r. From inspecting this curve, we see that as the similarity s of documents approaches 1, the value of this function — the probability of the input elements being mapped to the same bucket — also increases towards 1. One important aspect of this curve however, is that the steepest slope occurs at the value of s which is around $t = (1/b)^{(1/r)}$. In other words, if the Jaccard similarity of the two documents is above the threshold t, then the probability that they will be found potentially similar using LSH is very high. What this technique has done is to give us some idea in terms of which documents are very likely to be similar. If required, we can now actually compare the documents (or their minhash signatures) to find out whether they are actually similar.

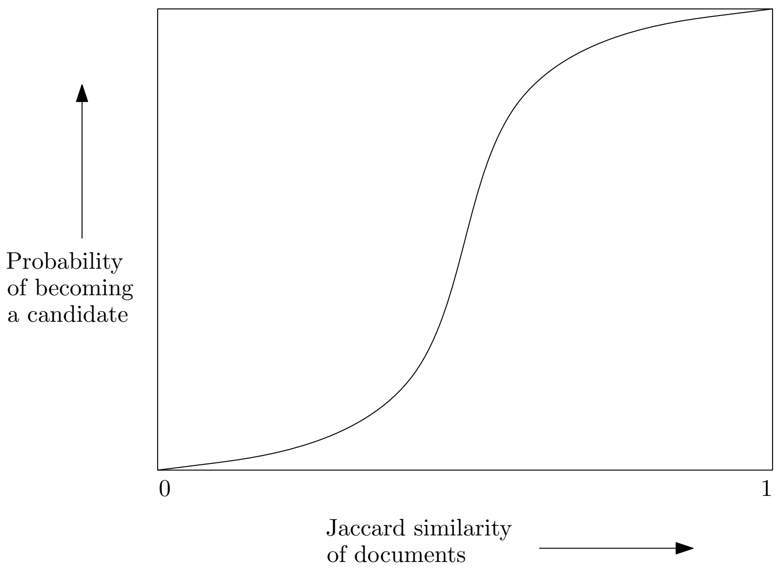


Figure 1.2: The S-curve.

At an abstract level what we have done here is to use a family of functions (the minhash functions) and the banding technique to distinguish between pairs which are at a low distance (similar) to the pairs which are at a large distance (dissimilar). The steepness of the S curve suggests a threshold where this technique can be effective. In the next section, we will see that there are other families of functions that can be considered to separate the pairs which are at a low distance from the pairs which are at a higher distance.

1.4 Theory of Locality Sensitive Functions

In this section we will consider a family of functions $\mathcal{F}$. The families are typically hash functions, and we say that $f(x) = f(y)$ if the items x and y are hashed to the same bucket for a function $f \in \mathcal{F}$. For example, the minhash functions seen previously form a family of functions.

Recall that a distance measure satisfies certain natural properties: non-negativity, symmetry, and the triangle inequality. For example, the Euclidean distance between points in a space, edit distance between strings, Jaccard distance, and hamming distance all satisfy these properties.

Definition 1.4.1 *Let d be a distance measure, and let $d_1 < d_2$ be two distances in this measure. A family of functions $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-sensitive if for every $f \in \mathcal{F}$ the following two conditions hold;*

1. *If $d(x,y) \leq d_1$ then Probability$[f(x) = f(y)] \geq p_1$*
2. *If $d(x,y) \geq d_2$ then Probability$[f(x) = f(y)] \leq p_2$*

Basically, if the two items are close to each other with respect to the distance function, then the probability that they hash to the same bucket will be high. Conversely, if the two items are far from each other, then the probability that they hash to the same bucket will be low.

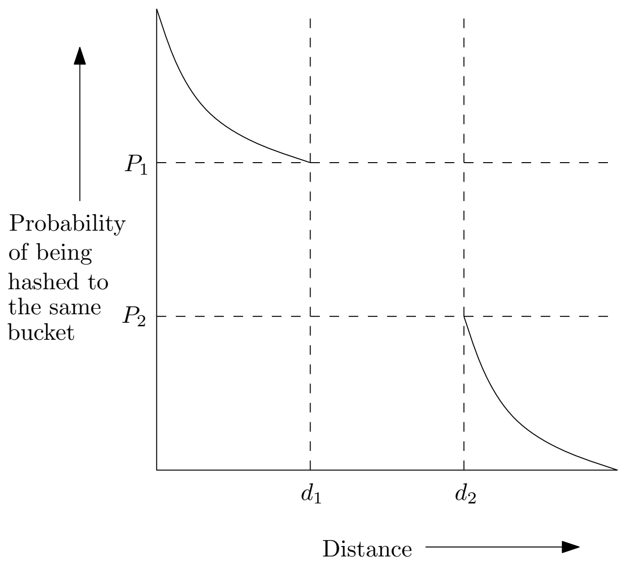


Figure 1.3: A smaller distance between items corresponds to a higher probability of similarity.

For example, consider the Jaccard distance, which is defined as $1-$ Jaccard similarity of two sets. Let $0 \leq d_1 < d_2 \leq 1$. Note that the family of minhash-signatures is $(d_1, d_2, 1-d_1, 1-d_2)$-sensitive. Suppose that the Jaccard Similarity between two documents is at least s. Then their distance is at most $d_1 = 1-s$. By Lemma 1.2.1 the probability that they will be hashed to the same value by minhash signatures is $s = 1-d_1$. Suppose that the distance between two documents is $\geq d_2$. Hence their Jaccard similarity is at most $s = 1-d_2$ and the probability that the minhash signatures map them to the same value is at most $s = 1-d_2$.

Suppose we have a (d_1, d_2, p_1, p_2)-sensitive family $\mathcal{F}$. We can construct a new family $\mathcal{G}$ as follows. Each function $g \in \mathcal{G}$ is formed from a set of r independently chosen functions of $\mathcal{F}$, say $f_1, f_2, \ldots, f_r$ for some fixed value of r. Now, $g(x) = g(y)$ if and only if for all $i = 1, \ldots, r$, $f_i(x) = f_i(y)$. This is referred to as an *AND*-family.

Claim 1.4.2 *$\mathcal{G}$ is an $(d_1, d_2, {p_1}^r, {p_2}^r)$ family.*

Proof. Each function f_i is chosen independently, and this is the probability of all the r-events to occur simultaneously. ■

We can also construct $\mathcal{G}$ as an *OR*-family. Each member g in $\mathcal{G}$ is constructed by taking b independently chosen members $f_1, f_2, \ldots, f_b$ from $\mathcal{F}$. We say that $g(x) = g(y)$ if and only if $f_i(x) = f_i(y)$ for at least one of the members in $\{f_1, f_2, \ldots, f_b\}$.

Claim 1.4.3 *$\mathcal{G}$ is an $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^r)$ family.*

Proof. Each function f_i is chosen independently, and this is the probability of at least one of the b-events to occur. We can compute this by first finding the probability that none of the b events occur. ■

Let us play with some values of b and r to see the effect of ANDs and ORs. We will also combine ORs and ANDs through function composition. We first construct an AND-family for a certain value of r, and then construct an OR-family on top of it for a certain value of b. Similarly we do this for an OR-family followed by an AND-family. Let us look at the amplification of the probabilities in Table 1.4 for different values of p.

p	p^5 $(\mathcal{F}_1)$	$1-(1-p)^5$ $(\mathcal{F}_2)$	$1-(1-p^5)^5$ $(\mathcal{F}_3)$	$(1-(1-p^5))^5$ $(\mathcal{F}_4)$
0.2	0.00032	0.67232	0.00159	0.13703
0.4	0.01024	0.92224	0.05016	0.66714
0.6	0.07776	0.98976	0.33285	0.94983
0.8	0.32768	0.99968	0.86263	0.99840
0.9	0.59049	0.99999	0.98848	0.99995

Table 1.4: Illustration of four families obtained for different values of p. $\mathcal{F}_1$ is the AND family for $r = 5$. $\mathcal{F}_2$ is OR family for $b = 5$. $\mathcal{F}_3$ is the AND-OR family for $r = 5$ and $b = 5$. $\mathcal{F}_4$ is the OR-AND family for $r = 5$ and $b = 5$.

Let's try to understand the columns of Table 1.4. Let $\mathcal{F}$ be a $(0.2, 0.6, 0.8, 0.4)$-sensitive minhash function family. This means if the distance between two documents x and y is ≤ 0.2, it is likely with probability ≥ 0.8 that they will hash to the same bucket. Similarly if the distance between x and y is ≥ 0.6, it is unlikely that they will hash to the same bucket (probability of hashing to the same bucket is ≤ 0.4). Let us focus our attention on the two rows corresponding to $p_2 = 0.4$ and $p_1 = 0.8$. For the AND-family we see that p^5 is substantially lower than p, but still $p_2^5 \to 0$ and p_1^5 is away from 0. For the OR-family $1-(1-p)^5 \geq p$, but still the value corresponding to p_1 tends towards 1 and the value corresponding to p_2 is away from 1. More interesting are the last two columns. Notice that the AND-OR family ($\mathcal{F}_3$) corresponds to the LSH for minhash signature family of Section 1.3. In this case the value corresponding to p_1 tends towards 1 and the value corresponding to p_2 is closer to 0. Notice that the value of this function with respect to the values of p forms an S-curve. Its fixed-point $p = 1-(1-p^5)^5$ is around $p \approx 0.7548$ [1]. This implies that for values of p significantly less than 0.75, AND-OR family amplifies it towards 0. Similarly, values of p larger than 0.75 are amplified to 1. Notice that this technique amplifies the probabilities in the right direction, provided we can apply the function several times (e.g. 25 times in our example), and are away from the fixed-point.

1.5 LSH Families

In the previous section we saw LSH families for the Jaccard distance measure. In this section we will construct LSH-families for various other distance measures, including Hamming and Euclidean distances. Note that not every distance measure may have a corresponding LSH-family.

1.5.1 Hamming Distance

Consider two d-dimensional vectors x and y. The Hamming distance $h(x, y)$ is defined as the number of coordinates in which x and y differ. Without loss of generality, we can assume that they are Boolean vectors — that is, each of their coordinates are either 1 or 0.

[1]Thanks to WolframAlpha

Example 1.5.1 *Let $x = 110011$ and $y = 100111$. Then $h(x, y) = 2$, as they differ in exactly two coordinates.*

Applying the requirements for a distance measure, we see that distances are non-negative, symmetric, and that the distance between two identical vectors is 0. They are also transitive, as the Hamming distance between any three vectors x, y, and z, satisfy the triangle inequality; $h(x, y) + h(y, z) \geq h(x, z)$. Therefore, we can use Hamming distance as a metric over the d-dimensional vectors.

Next, we construct a locality-sensitive family for the d-dimensional Boolean vectors. Let $f_i(x)$ denote the i-th coordinate (bit) of x. Then the probability that $f_i(x) = f_i(y)$ for a randomly chosen i will equal the number of coordinate agreements out of the total number of coordinates. Since the vectors x and y disagree in $h(x, y)$ positions out of d positions, then they agree in $d - h(x, y)$ positions. Hence $Pr[f_i(x) = f_i(y)] = 1 - h(x, y)/d$.

Claim 1.5.2 *For any $d_1 < d_2$, $\mathcal{F} = \{f_1, f_2, \ldots, f_d\}$ is a $(d_1, d_2, 1 - d_1/d, 1 - d_2/d)$-sensitive family of hash functions.*

Proof. Recall Definition 1.4.1. Let $p_1 = 1 - d_1/d$, and $p_2 = 1 - d_2/d$. A family of functions $\mathcal{F}$ is said to be (d_1, d_2, p_1, p_2)-sensitive if for every $f_i \in \mathcal{F}$ the following two conditions hold;

1. If $h(x, y) \leq d_1$ then Probability$[f_i(x) = f_i(y)] \geq p_1$
2. If $h(x, y) \geq d_2$ then Probability$[f_i(x) = f_i(y)] \leq p_2$

∎

Note that the family $\mathcal{F}$ consists of only d-functions. This is in contrast to the minhash function family, where the size is not bounded.

1.5.2 Cosine Distance

As an alternative to Hamming distance, we can use Cosine distance to construct a locality-sensitive family of functions for d-dimensional vectors in a space. Consider two vectors in this space, x and y, along with a hyperplane through the origin with a normal vector v. These two vectors x and y have a cosine distance which is equal to the angle θ between them, and taken alone, this angle can be measured in the plane defined by the vectors. The hyperplane is randomly chosen via its normal vector v, and will then determine the sign of the dot products $v.x$ and $v.y$.

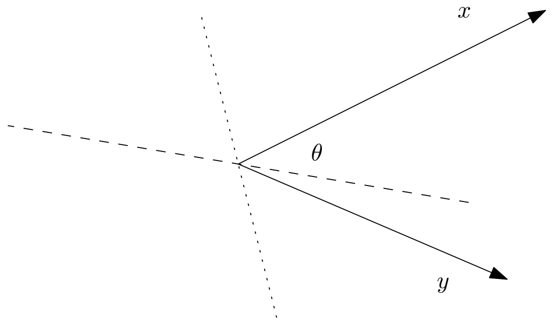


Figure 1.4: Two vectors x and y shown in a plane; two possible hyperplanes are also shown.

In Figure 1.4, the two vectors x and y are shown along with two possible hyperplanes. If the hyperplane is similar to the dashed line, then the dot products $v.x$ and $v.y$ will have different signs. However, if the hyperplane is similar to the dotted line, then $v.x$ and $v.y$ will have the same sign.

The probability that v will produce a hyperplane similar to the dotted line is $\theta/180$, as v would have to lie between x and y on the right (or on the left; $2\theta/360$). So, we create each hash function $f \in \mathcal{F}$ using a randomly chosen vector v, and can then say that $f(x) = f(y)$ if the dot products $v.x$ and $v.y$ have the same sign. Our resultant family $\mathcal{F}$ is a $(d_1, d_2, (180 - d_1)/180, (180 - d_2)/180)$-sensitive family of functions. Similar to other families, we can amplify probabilities by taking any combination of ANDs and ORs as before.

1.5.3 Euclidean Distance

To start with, let's consider points in a 2-dimensional space. Each hash function h will be represented by a line with random orientation in this space. We subdivide the line into intervals of an equal size a, with points landing in the same bucket if their orthogonal projection lies on the same interval. Using this method, if two points are close to each other then there are high chances that they will map to the same interval. Conversely, if the points are far away, it is unlikely they will fall into the same bucket (unless they are almost vertically aligned!).

Without loss of generality, we'll assume that the line is horizontal. Let p_i and p_j be two points in the plane. Let θ be the angle of the line passing through p_i and p_j with respect to the horizontal line. Observe that if the distance between the points is at most $a/2$ then the probability that the points map to the same interval is at least $1/2$. Moreover, if the distance between them is more than $2a$, then they will fall in the same interval with a probability of at most $1/3$. This is due to the fact that they have to be almost vertical; $|p_ip_j| \cos\theta \leq a$. For this to happen $\cos\theta \leq 1/2$, or the angle of the line p_ip_j forms with the horizontal needs to be between $60°$ and $90°$. Observe that there is only 1/3rd chance that the angle between the horizontal and p_ip_j is in that range.

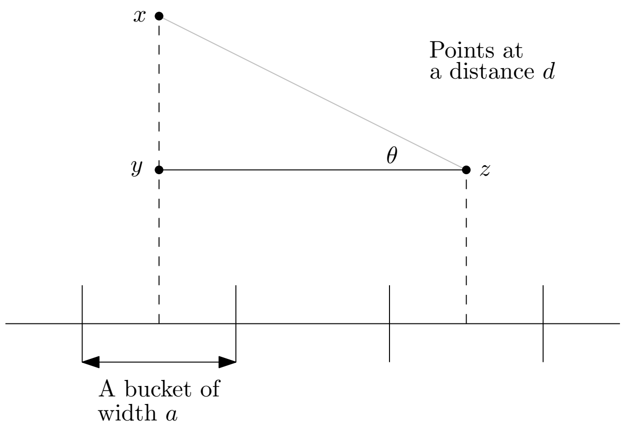


Figure 1.5: With a distance $d > 2a$, points x and z are in separate intervals unless θ is large enough.

Hence, the family with respect to the projection on a random line with intervals of size a is a $(a/2, 2a, 1/2, 1/3)$-sensitive family of hash functions. Again, we can easily amplify these probabilities by taking any combination of ANDs and ORs as before.

1.5.4 Fingerprint Matching

Fingerprint matching typically requires comparison of several features of the fingerprint pattern. These include ridge lines which form arches, loops, or circular patterns, along with minutia points & patterns which form ridges, and bifurcations. We can think of a fingerprint as a set of grid points, where matching any two fingerprints amounts to matching elements in corresponding grid cells.

Fingerprint matching falls into an application of Locality-Sensitive Hashing, and as a result of the data representation we are able to use its methods; mainly the family of minhash functions for set comparison. However in this section we will approach the problem in another manner, as the sets produced from fingerprints form a small grid of around 1000 points, and don't need to have more succinct signatures created.

Consider the probabilities of minutia appearances and matches between fingerprints. For this, let's say that the probability of a minutia existing in a random grid cell of any given fingerprint is 0.1. Also, assuming that we have two fingerprints of the same finger, let the probability of a minutia appearing in the same grid cell of both fingerprints given that one of them does have a minutia there be 0.9.

We'll define a locality-sensitive family of functions $\mathcal{F}$ as follows. Each function $f \in \mathcal{F}$ sends two fingerprints to the same bucket if they have minutia in each of three specific grid cells. Let us estimate what the chance is of ending up with two matching fingerprints. First, the probability that two arbitrary fingerprints will map to the same bucket with respect to function f is $0.1^6 = 0.000001$, which is a one in a million chance [2]. Assuming that we have two fingerprints which are from the same finger though, we can see that they are mapped to the same bucket by f with a probability of $0.1^3 \times 0.9^3 = 0.00729$. This is only about a 1 in 138 chance, though it can be amplified by using several hash functions (OR-family) and still not produce many false positives.

As an example, suppose we use 1000-way OR-functions. Then two fingerprints from different fingers will map to the same bucket with a probability of $1 - (1 - 0.000001)^{1000}) \approx 0.0001$. Along with this, two fingerprints from the same finger will map to the same bucket with a probability of $1-(1-0.00729)^{1000} \approx 0.9993$. This is significant, and hence finding the match in this case is highly likely. Also note that using this scheme, the chances of false negatives are less than 1 in 1000, and the chances of false positives are negligible.

For another example, let's now use 2000 functions which are partitioned into two groups of 1000 functions each. Assume also that we have constructed buckets for each of the two composite functions. Given a query fingerprint, we'll find the fingerprints which are potential matches using the above scheme in each group independently. Then we'll select only those fingerprints which are potential matches in both the groups (in the intersection). This produces a set of fingerprints which we'll actually compare to the query fingerprint. Note that in this scheme, actual comparisons of fingerprints happen with only a few fingerprints (those in the intersection). Here, the probability that we will detect a matching fingerprint is $(0.9993)^2 \approx 0.9986$. The number of false negatives are now about 1.4 in a 1000. The probability of false positives will be 0.0001^2, which is insignificant. Hence, we have to examine only 0.000001% of the database compared to 0.01% in the previous scheme.

[2]Wonder why you are always the unlucky one :(

1.5.5 Image Similarity

Image similarity can also be found using locality-sensitive hashing, as exemplified in earlier research on LSH. Often, images are processed using global descriptors such as color histograms to produce feature vectors which are then compared using different distance measures. One of the obvious applications of image similarity is in web search, as seen in use by TinEye, Google, and Bing. However most search engines do not use image similarity in search; instead they rely on information gathered from surrounding text on the web page along with any image meta-data. In this section, we will show how locality-sensitive hashing is used by Google in their VisualRank algorithm for visual image search, along with several locality-sensitive families which have been proposed for image similarity measures.

VisualRank

In the VisualRank algorithm, computer vision techniques are used along with locality-sensitive hashing. Consider a regular image search by a text query. An existing search technique may be relied on to retrieve the initial result candidates, which along with other images in the index are clustered according to their similarity (which may be precomputed). Centrality is then measured on the clustering, which will return the most canonical image(s) with respect to the query. The idea here is that agreement between users of the web about the image and its related concepts will result in those images being deemed more similar. VisualRank is defined iteratively by $VR = S^* \times VR$, where S^* is the image similarity matrix. As matrices are used, eigenvector centrality will be the measure applied, with repeated multiplication of VR and S^* producing the eigenvector we're looking for. Clearly, the image similarity measure is crucial to the performance of VisualRank since it determines the underlying graph structure.

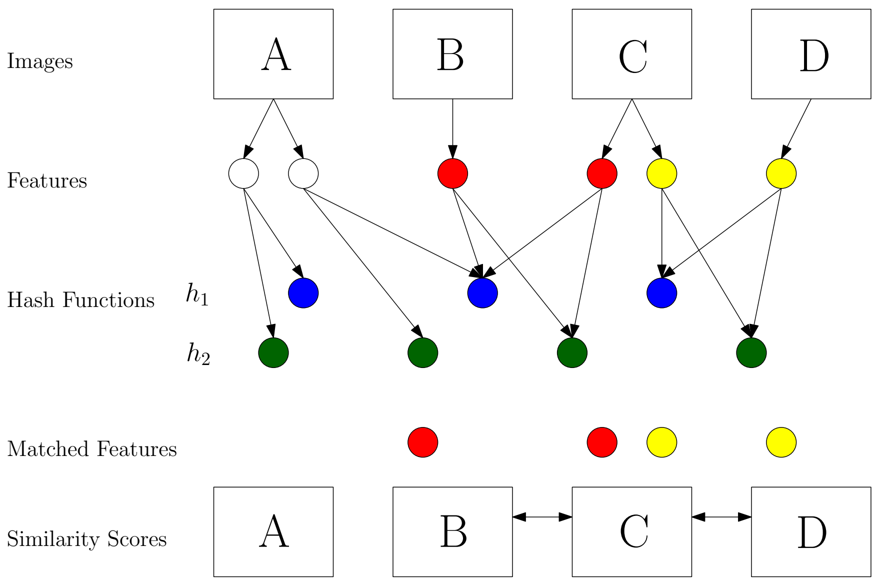


Figure 1.6: Images are considered similar if their local features are hashed into the same bin in multiple hash tables. In this example, the blue and green circles are buckets where features are hashed; B is similar to C, and C is similar to D.

The main VisualRank system begins with local feature vectors being extracted from images using scale-invariant feature transform (SIFT). Local feature descriptors are used instead of the previously mentioned color histograms as they allow similarity to be considered between images with potential rotation, scale, and perspective transformations. Locality-sensitive hashing is then applied to these feature vectors using the p-stable distribution scheme. As seen in previous sections, amplification using AND/OR constructions are applied. As part of the applied scheme, a Gaussian distribution is used under the l_2 norm.

p-Stable Distributions

This locality-sensitive hashing scheme uses stable distributions to apply a sketching technique to the high dimensional feature vectors. However, instead of using the sketch to estimate the vector norm, it is used to assign hash values to each vector. These values map each vector to a point on the real line, which is the split into equal width segments to represent buckets. If two vectors v and u are close, they will have a small difference between l_p norms $||v-u||_p$, and they should collide with a high probability.

Definition 1.5.3 *A distribution $\mathcal{D}$ over $\mathcal{R}$ is called* p-stable*, if there exists* $p \geq 0$ *such that for any n real numbers $v_1, \ldots, v_n$ and independent identically distributed variables $X_1, \ldots, X_n$ with distribution $\mathcal{D}$, the random variable $\sum_i v_i X_i$ has the same distribution as the variable $(\sum_i |v_i|^p)^{1/p} X$, where X is a random variable with distribution $\mathcal{D}$.*

Example 1.5.4 *Cauchy Distributions are 1-stable, and Gaussian Distributions are 2-stable.*

Note how the sum $\sum_i v_i X_i$ appears if we consider that v_i's are taken from a vector v. This operation then corresponds exactly to the dot product between vectors v and a, where a has entries taken from the stable distribution. From our definition then, we see that this random variable has the same distribution as the l_p norm when the latter is multiplied by a random variable from the same distribution. This is exactly the method of the sketching technique, where multiple $v.a$'s are computed to estimate $||v||_p$.

However as mentioned beforehand, in this case we'll use the $v.a$'s to assign a hash value to each v instead of estimating the l_p norm. Also, as the dot product $v.a$ projects each vector onto the real line, we'll separate the line into equal width segments of size r. We define each hash function of our family by

$$h_{a,b}(v) = \lfloor \frac{v.a+b}{r} \rfloor,$$

where a is a d-dimensional random vector with entries chosen independently from our stable distribution, and b is a real number chosen uniformly from the range $[0, r]$. Let $f_p(t)$ denote the probability density function of absolute value of our stable distribution, and let $c = ||v-u||_p$ for two vectors v, u. Since we have a random vector a from our stable distribution, $v.a - u.a$ is distributed as cX where X is a random variable from our stable distribution. Therefore our probability function is;

$$p(c) = Pr_{a,b}[h_{a,b}(v) = h_{a,b}(u)] = \int_0^r \frac{1}{c} f_p(\frac{t}{c})(1-\frac{t}{r})dt.$$

For a fixed r the probability of collision decreases monotonically with $c = ||v-u||_p$. Thus, our family of hash functions are (r_1, r_2, p_1, p_2)-sensitive for $p_1 = p(1)$, $p_2 = p(c)$, and $r_2/r_1 = c$. Note that for each c we attempt to choose a finite r to make p as small as possible.

χ^2 Distance

In the context of image retrieval & similarity search, the Euclidean metric (l_2) does not always provide as accurate or relevant results when compared to χ^2 distance. This is especially so when using global descriptors such as color histograms. In this section, we will show an adaptation of the p-stable distribution scheme for χ^2 distance.

Definition 1.5.5 *Given any two vectors v and u with d positive components, the χ^2 distance between them is*

$$\chi^2(v,u) = \sqrt{\sum_{i=1}^{d} \frac{(v_i - u_i)^2}{v_i + u_i}}.$$

To begin with, all points are mapped into a space of smaller dimension where they will be clustered in a way which preserves local sensitivity. This is done by projecting all points onto a line l_a using a random vector a whose entries are chosen independently from a Gaussian distribution. The line is then uniformly partitioned into intervals of size W with respect to χ^2 distance. Our hash functions are then defined in a similar manner to the previous section — as the dot product of the feature vector v with a random vector a with entries independently chosen from a Gaussian distribution;

$$h_{a,b}(v) = \lfloor yw(v.a) + b \rfloor.$$

In this case, b is a real number taken randomly from the uniform distribution $\mathcal{U}([0,1])$. The function yw is defined as

$$yw(x) = \frac{\sqrt{\frac{8x}{W^2} + 1} - 1}{2}$$

as a result of the non-uniform mapping from l_2 to χ^2 distance; points mapping to the same bucket in the χ^2 scheme may be mapped to different buckets in the l_2 scheme.

Following from the reasoning in p-stable distributions, we know that $v.a - u.a$ is distributed as cX, where $c = ||v - u||_p$, and X is a random variable drawn from the distribution. Also similar to the previous section, $f_p(t)$ denotes the probability density function of the absolute value of the 2-stable Gaussian distribution. Therefore our probability function is;

$$p(c) = Pr_{a,b}[h_{a,b}(v) = h_{a,b}(u)] = \int_0^{(n+1)W^2} \frac{1}{c} f_p(\frac{t}{c}) \left(1 - \frac{t}{(n+1)W^2}\right) dt.$$

We then define $c' = X^2(v,u)$. Since v, u have positive entries, the two distances will vary similarly when taking the dot product. Therefore, p decreases monotonically with respect to c, and p also decreases monotonically with respect to c'. If we set $p_1 = p(r_1)$ and $p_2 = p(r_2)$ (with $r_1 < r_2$), then this family of functions is (r_1, r_2, p_1, p_2)-sensitive.

1.6 Bibliographic Notes

Concepts relating to locality-sensitive hashing first started to form around 1994, when a method for finding similar files in a large file system was proposed [16]. This research introduced the idea behind shingling through 'fingerprinting', where several anchor strings were chosen to begin a substring used in the similarity search. Along with this, checksums were also used to represent these fingerprints. Shingling itself was then formalized a few years later, along with k-shingles of a document (using words as tokens), and resemblance (similarity) of two documents was defined [3]. The authors also introduced a method for computing a sketch (signature) of a document which could be used in comparisons; hence, preserving similarity.

After this research came a couple of papers which proposed similarity search through the use of hashing [7, 10]. Approximate nearest-neighbor search was first reduced to the problem of point location in equal balls, where locality-sensitive hashing was introduced for sublinear time queries. This algorithm was then improved in [10] with respect to query time. Along with this, the distance measures for hamming distance and the resemblance measure given in [3] were used as schemes. It's worth noting that the approximate search is deemed accurate enough for practical purposes, where the actual closest neighbor can still be found by checking all approximate near-neighbors [1].

Concurrent with this research, shingling was formally introduced along with min-wise independent families of permutations [2]. These concepts were developed as a result of the authors' work on the AltaVista web index algorithm which was used for detecting similar documents. Minwise-independent families also underpin the theory behind minhashing, as the resemblance of document-sets is shown to be equal to the probability that the min-wise permutation of two sets using random permutation functions are equal.

Next in the development of locality-sensitive hashing was a scheme based on p-stable distributions ($p \in (0, 2]$) under the l_p norm [5]. This scheme both improved on the l_2 norm distance running time found in [10], and works in Euclidean space without embeddings. These findings are then improved upon by [1], resulting in a near-optimal algorithm.

Locality-sensitive hashing is used in for video identification and retrieval. In this case, feature vectors are usually constructed from video frames using certain color histograms. In [14], the authors address two weaknesses of using LSH for this purpose. Responding to a non-uniform distribution of points in the space, they focus on using a hierarchical hash table to produce a more uniform distribution. In addition to this, they also attempt to partition dimensions more carefully in order to produce a more even hashing. A new scheme for video retrieval is then proposed in [9], where a color histogram is used which better handles the adverse effects of brightness & color variations. This is used in conjunction with an additional uniform distance shrinking projection which is applied to the produced feature vectors.

Locality-sensitive hashing is also used in image search. Similar to applications in video (for a single frame), images are often processed using color histograms to produce feature vectors which can be compared. This method was used in [7], with the histograms compared using the l_1 norm. Following this, techniques using chi^2 distance [8], p-stable distributions [12], and kernelized locality-sensitive hashing [15] have been proposed. Kernelized locality-sensitive hashing has also been used as a basis for search on text images in [17].

Along with these applications, LSH has recently been proposed for use in a wide range of other areas. One of these is the creation of hash values in P2P networks [6]. This would allow for a conceptual search, as data with similar ontologies would be located near each other. Here, data is defined by its extracted concept vectors, which are then hashed into buckets based on the cosine distance measure. Another, in [11], kernelized LSH is applied to an utterance model in order to identify speakers. In this case the hamming distance metric is used. Other areas include use for species diversity estimation by allowing ease of grouping similar DNA sequences [19], and incremental clustering for affinity groups in Hadoop in order to store related data closer together [13].

Recently, [21] also proposed a new scheme based on entropy for LSH. The argument is that an improvement can be made for [5] such that the distribution of mapped values will be approximately uniform. Note how this approach attempts the same result as [14] using different methods.

1.7 Exercises

1.1 *Given two documents $D_1 = \{$ "His brow trembled"$\}$ and $D_2 = \{$ "The brown emblem"$\}$, compute all k-shingles for each document with $k = 4$.*

1.2 *Compute the Jaccard Similarity of the two sets of k-shingles for D_1 and D_2.*

1.3 *Compute the signature matrix (minhash signature) of a given permutation of characteristic matrix in Table 1.5, using $n = 3$ different hash functions.*

Element	S_1	S_2	S_3	S_4
a	1	0	1	1
b	1	0	1	0
c	0	1	0	1
d	1	0	1	0
e	0	0	1	0

Table 1.5: A characteristic matrix.

1.4 *Explain the reasoning behind the proof of Lemma 1.2.1.*

1.5 *Explain the overall procedure used to calculate document similarity, where locality-sensitive hashing is used in the process, and why locality-sensitive hashing would be favorable to use.*

1.6 *Show that the minhash function family is $(d_1, d_2, 1 - d_1, 1 - d_2)$-sensitive.*

1.7 *Calculate hamming distance for the vectors $v = [5, 1, 3, 2, 4]$ and $u = [1, 1, 2, 2, 4]$.*

1.8 *Discuss the similarity between Jaccard distance and Hamming distance.*

1.9 *Show that the AND amplification construction is (d_1, d_2, p_1^r, p_2^r)-sensitive.*

1.10 *When applying amplification constructions to a locality-sensitive family of functions, which order of composition is 'better', and why? Explain when you would want to use different orders of construction.*

Bibliography

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS '06. 47th Annual IEEE Symposium on*, pages 459–468, 2006.

[2] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations. *Journal of Computer and System Sciences*, 60:327–336, 1998.

[3] A.Z. Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29, 1997.

[4] Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *In Proc. of 34th STOC*, pages 380–388. ACM, 2002.

[5] Mayur Datar and Piotr Indyk. Locality-sensitive hashing scheme based on p-stable distributions. In *In SCG 04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM Press, 2004.

[6] L.B. de Paula, R.S. Villaca, and M.F. Magalhaes. A locality sensitive hashing approach for conceptual classification. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 408–413, 2010.

[7] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. pages 518–529, 1997.

[8] D. Gorisse, M. Cord, and F. Precioso. Locality-sensitive hashing for chi2 distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):402–409, 2012.

[9] S. Hu. Efficient video retrieval by locality sensitive hashing. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 2, pages 449–452, 2005.

[10] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. pages 604–613, 1998.

[11] Woojay Jeon and Yan-Ming Cheng. Efficient speaker search over large populations using kernelized locality-sensitive hashing. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4261–4264, 2012.

[12] Yushi Jing and S. Baluja. Visualrank: Applying pagerank to large-scale image search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1877–1890, 2008.

[13] K.A. Kala and K. Chitharanjan. Locality sensitive hashing based incremental clustering for creating affinity groups in hadoop; hdfs - an infrastructure extension. In *Circuits, Power and Computing Technologies (ICCPCT), 2013 International Conference on*, pages 1243–1249, 2013.

[14] Zixiang Kang, Wei Tsang Ooi, and Qibin Sun. Hierarchical, non-uniform locality sensitive hashing and its application to video identification. In *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, volume 1, pages 743–746 Vol.1, 2004.

[15] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(6):1092–1104, 2012.

[16] Udi Manber. Finding similar files in a large file system. In *in Proceedings of the USENIX Winter 1994 Technical Conference*, pages 1–10, 1994.

[17] Tanmoy Mondal, Nicolas Ragot, Jean-Yves Ramel, and Umapada Pal. A fast word retrieval technique based on kernelized locality sensitive hashing. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1195–1199, 2013.

[18] J; Ullman J Rajaraman, A; Leskovec. *Mining of Massive Datasets.* 2010.

[19] Z. Rasheed, H. Rangwala, and D. Barbara. Lsh-div: Species diversity estimation using locality sensitive hashing. In *Bioinformatics and Biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–6, 2012.

[20] N. Raval, M.R. Pillutla, P. Bansal, K. Srinathan, and C.V. Jawahar. Privacy preserving outlier detection using locality sensitive hashing. In *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pages 674–681, 2011.

[21] Qiang Wang, Zhiyuan Guo, Gang Liu, and Jun Guo. Entropy based locality sensitive hashing. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 1045–1048, 2012.