**4 Pictures that Explain LSH - Locality Sensitive Hashing Tutorial**

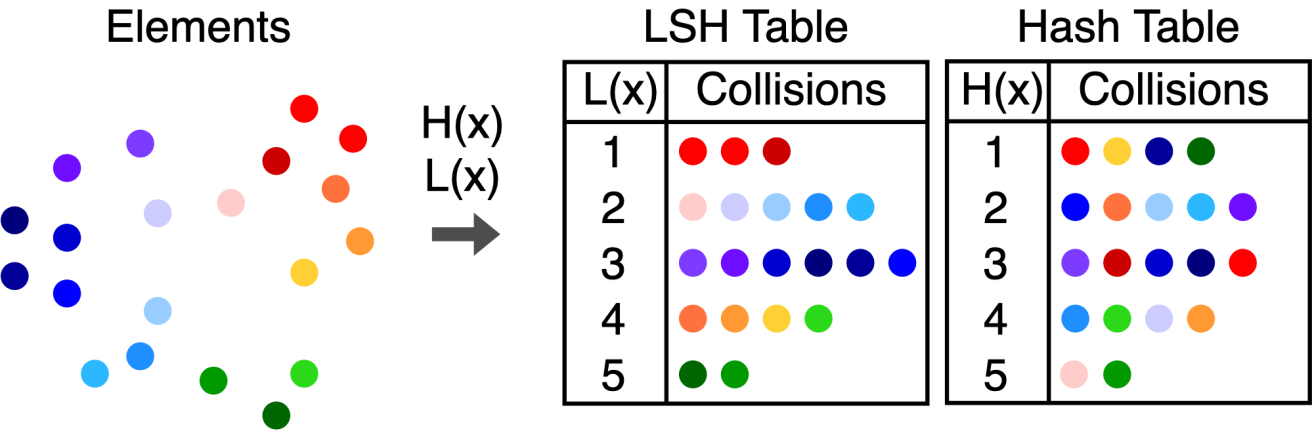
[**https://randorithms.com/2019/09/19/Visual-LSH.html**](https://randorithms.com/2019/09/19/Visual-LSH.html)

Locality sensitive hashing is a super useful trick. Most people use it for near-neighbor search but it’s also helpful for sketching algorithms and high-dimensional data analysis. This post will explain four common LSH functions.

**What is LSH?**

Hash functions map objects to numbers, or bins. A *locality sensitive* hash (LSH) function *L*(*x*) tries to map similar objects to the same hash bin and dissimilar objects to different bins. The picture below shows an example where we form two hash tables - one using an LSH function *L*(*x*) and the other using a normal hash function *H*(*x*).

*L*(*x*) preserves most of the clusters from the original dataset - most of the elements in each hash bin are similar. The key intuition behind LSH is that LSH functions try to group similar elements together into hash bins.



**LSH collision probabilities**

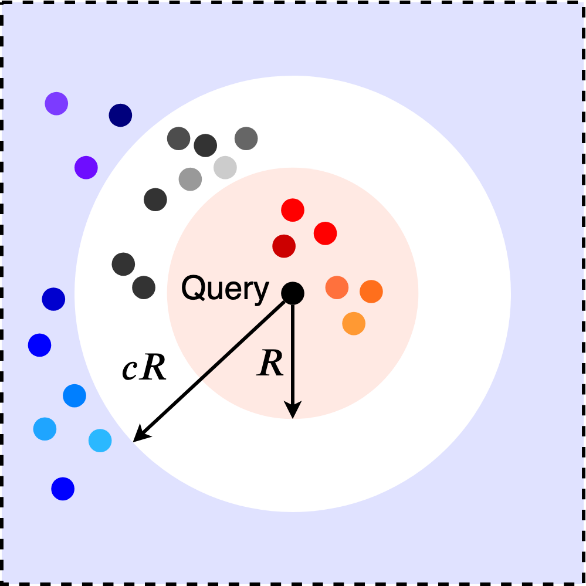
A hash collision occurs when two objects *x* and *y* have the same hash value. In our example, all of the red dots collided, but this was not guaranteed to happen. Under an LSH function, the chance that objects collide - called a collision probability - depends on how similar the objects are. LSH functions are defined based on their collision probabilities [1].

**Definition 1: Locality Sensitive Hash Family**  
We say that a **hash family H is (*R*,*cR*,*p*1,*p*2)-sensitive with respect to a distance function *d*(*x*,*y*)** if for any *h*∈H we have that:

**If *d*(*x*,*y*)≤*R* then PrH[*h*(*x*)=*h*(*y*)]≥*p*1**

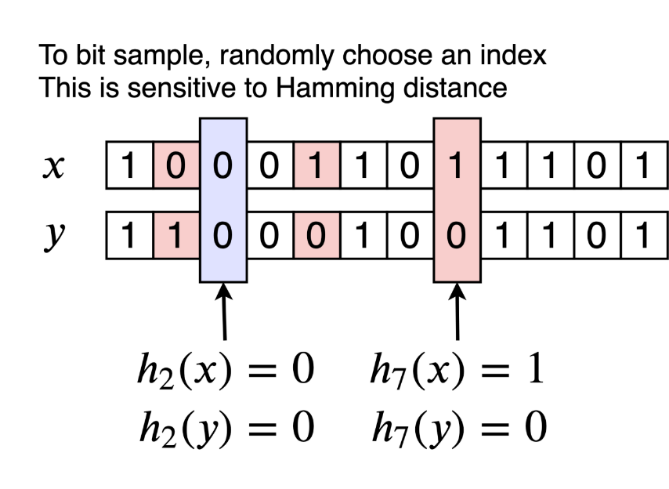
**If *d*(*x*,*y*)≥*cR* then PrH[*h*(*x*)=*h*(*y*)]≤*p*2**

The definition seems complicated, but it is really just a mathematical statement of our intuition. The picture below explains the definition. Definition 1 says that if we pick our LSH function from the hash family H, then all of the points in the red ball have a good chance (probability ≥*p*1) of colliding with the query. All of the points in the blue shaded region have a small chance (probability ≤*p*2) of colliding, and we don’t care what happens to the gray points in between.



Next, we’ll introduce LSH functions for some common distance measures. We will use *p*(*x*,*y*) to mean the collision probability.

**1. Bit Sampling LSH**



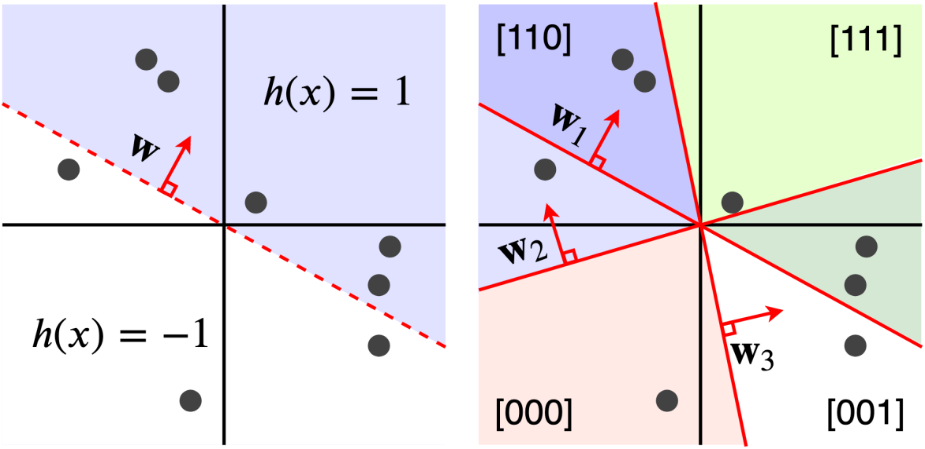
**Bit sampling** is one of the simplest and cheapest LSH functions. **It is associated with the Hamming distance**. Given two ***n*-length bit**-vectors **x** and **y**, the Hamming distance *d*(**x**,**y**) is the number of bits that are different between the two vectors. Bit sampling is an LSH family where *hi*(**x**)=*xi* ∈{0,1}

**We choose a random index *I* between 0 and *n*−1 and have the LSH function output the bit *xi*.**

To find the collision probability, observe that the number of indices for which *xi*=*yi* is *n*−*d*(**x**,**y**)=12-3=9. The **collision probability** is simply the chance that we picked one the indices where *xi*=*yi*.

*p*(**x**,**y**)=1−*d*(**x**,**y**)/*n=1-3/12=9/12*

**2. Signed Random Projections**



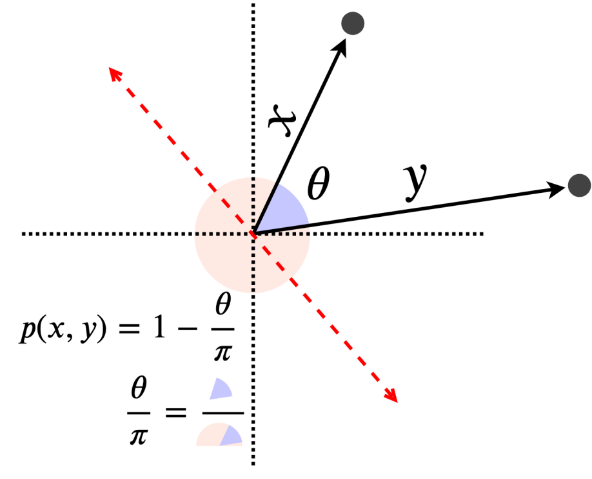
Signed random projections (SRP) also output binary values, but SRP is sensitive to the angular distance. To construct a SRP hash for a vector **x**, we generate a Gaussian distributed vector **w** with the same length as **x**. The hash function is

*h*(**x**)=sign(**w**⊤**x**)

The picture shows the hash bins for a single SRP (left) and for three SRP hashes (right). This function essentially cuts the space in half and assigns points in one half to 1 and the other half to -1. The decision boundary is based on the projection vector **w**, which is orthogonal to the boundary. This function is sensitive to the angle between two vectors because if we draw a random hyperplane in R*n*, the probability that the plane divides two points **x** and **y** is related to the angle *θ*(**x**,**y**).

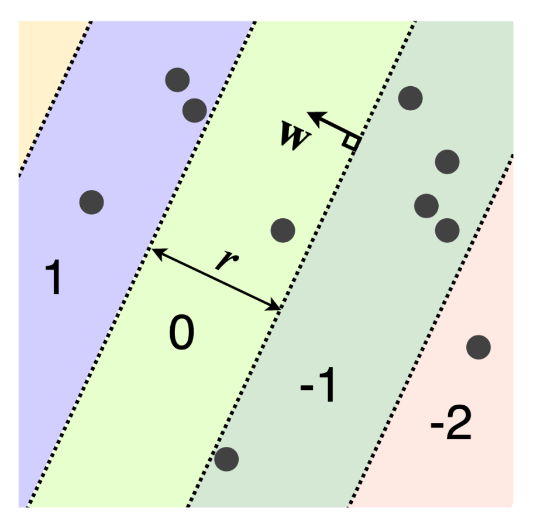
*p*(**x**,**y**)=1−*θ*(**x**,**y**)/*π*

The picture below provides intuition for why SRP is sensitive to angles. The probability that *h*(**x**)≠*h*(**y**) is the probability that a randomly chosen line falls between **x** and **y**. If we randomly pick an angle, there is a *θπ* chance that it falls between the two points and separates them.



This hash function can be very fast, especially when we use sparse random projections. For more information, see [2,3].

**3. Euclidean and Manhattan LSH**

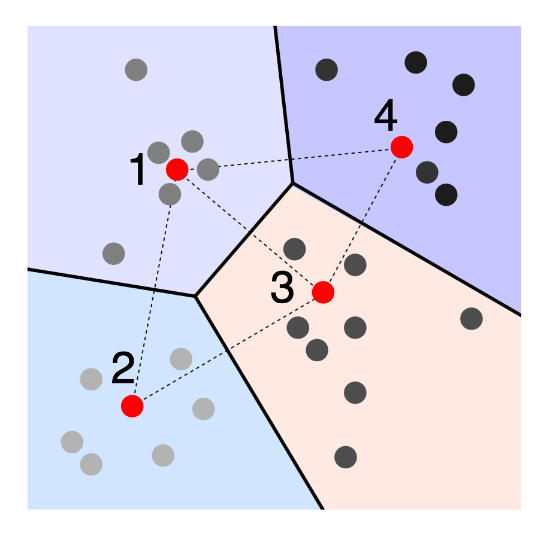


The LSH functions for the Euclidean (L2) and Manhattan (L1) distances are also based on random projections that cut R*n* into pieces [4]. Each piece becomes a hash bin. The hash function is identical to SRP, but we round instead of taking the sign of the projection. This produces hash bins that repeat in the direction of **w** instead of a single decision boundary as with SRP.

*h*(**x**)=⌊(**w**⊤**x**+*b)/r*⌋

The user-defined parameter *r* determines the width of each hash bin, and *b* is a random number in the range [0,*r*]. If we generate **w** with a Gaussian distribution, we get an LSH function for the Euclidean distance, while a Cauchy distribution results in an LSH for the Manhattan distance. Unlike the previous LSH functions, this hash function maps to the entire set of integers. The analytic expression for the collision probability is complicated to write down but a closed-form expression is available in [4].

**4. Clustering LSH**



Clustering LSH is an example of a learned or data-dependent LSH function. The idea is to find a set of cluster centers C that do a good job of approximating the dataset, shown in red. These centers might be found using k-means clustering, convex clustering, or any other clustering method. We can also select cluster centers randomly or on a grid, but we might need a larger number for good performance. Once we have the cluster centers, we assign an integer value to each one. The hash bins for *h*(*x*) are the same as the Voronoi cells for the set of centroids. Given an object to hash, we find the nearest center and output its index.

*h*(*x*)=argmin 1≤*i*≤|C| *d*(*x*,*ci*)

It is easy to show that this function is locality-sensitive but much harder to specify the collision probability, since collisions depend on the set of learned cluster centers C.

**References**

[1] Indyk, P., & Motwani, R. (1998, May). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (pp. 604-613). ACM.

[2] Li, P., Hastie, T. J., & Church, K. W. (2006, August). Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 287-296). ACM.

[3] Achlioptas, D. (2001, May). Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (pp. 274-281). ACM.

[4] Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004, June). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry* (pp. 253-262). ACM.