Anna Watson
Brad McCoy

Locality Sensitive Hashing

1 Overview

Proposed by Piotr Indyk and Rajeev Motwani in 1988, Locality Sensitive Hashing (LSH) is a family of hash functions in which objects which are similar in nature have a high probability of colliding with one another in a given, limited range space. We follow the description given in [3]. More conventional uses for hashing rely on the fact that each output produced by the hash to be radically different. LSH relies on the notion that two similar objects when hashed will produce a similar hash. For example, when hashing two different documents which differ by one byte with MD5 we see that it produces a completely separate and unrelated hash. While conventional uses for hashing try to minimize collisions, we want to maximize them. Formally, a LSH function h is a function which maps every d $\in$ the dataset, assuming similarity in between d_i and d_j are set in a way which the probability of collision P is high for objects which are similar:

$$\begin{cases} Pr(h(\mathrm{d}_i) = h(d_j)) \geq p_1 & \text{if } \mathrm{Sim}(\mathrm{d}_i, d_j) \leq \theta \\ Pr(h(\mathrm{d}_i) = h(d_j)) \leq p_2 & \text{if } \mathrm{Sim}(\mathrm{d}_i, d_j) \geq \gamma\theta \end{cases}$$

where $0 < \gamma < 1$ and $0 \leq p_2 \leq p_1 \leq 1$. θ is the measure of similarity between two objects in our dataset.
We begin by hashing every d_i in our dataset and appending it to the resulting bucket. The idea is similar objects will hash to the same bucket at a high probability. As this algorithm is probabilistic in nature false positives and false negatives can occur. False positives occur when dissimilar objects are mapped to the same bucket. This can be eliminated by going through each bucket and comparing objects and rejecting those outside of the threshold θ. False negatives occur when two similar objects are mapped to two different buckets. Unfortunately, false negatives cannot be helped but can be minimized by building more than one hash table and using more than one hash function. By increasing our number of hash tables we increase the accuracy.
After successfully bucketing our data we may now find similar documents for each d_i. Given each d_i we check it against each document contained in it's bucket. If the similarity is within θ then we can append it to d_i's list of similar objects.
The performance of this algorithm depends on appropriate values for both the number of hash tables used for bucketing and θ for a given dataset. Poor choices for these parameters can lead to incorrect groupings or too many documents for the last step which slow the algorithm down.

2 Examples

In this section we consider two examples of LSH to near neighbor searching. First, in [1] the authors apply LSH to polygonal curves in $\mathbb{R}^d$. Second, in [2] the authors apply LSH to persistence diagrams. Both examples involve snapping features of the data to a rectangular grid, then hashing objects based on grid points.

2.1 Curves

In [1] the authors apply LSH to curves. A polygonal curve can be thought of as a sequence of m points in $\mathbb{R}^d, P = \{p_1, p_2, \ldots, p_m\}$. For example, consider the gps coordinates of a taxi cab as they drive a passenger. We might wish to find determine if the taxi is driving on a common route. To do this we can find the nearest common route to the current path.
The paper considers two distances for curves, the Frećhet distance and the dynamic time warping distance.
The general idea to to place a $d-$dimensional grid over the input space. Then snap each $p_i \in P$ to the closest intersection point in the grid.
The paper proves the following:
Let P, Q be two curves with m_1 and m_2 points, respectively, and let $m = \min\{m_1, m_2\}$. For any $\delta = 4dmr$ and $c = 4d2m$ it holds that:

- if $d_F(P, Q) < r$, then $\Pr(h_\delta(P) = h_\delta(Q)) > 0.5$
- if $d_F(P, Q) > cr$, then $\Pr(h_\delta(P) = h_\delta(Q)) = 0$

We see that δ is a function of m so our approximation factor is linear. The paper goes on to show that the approximation factor can be improved but this comes at a cost to query time.

2.2 Persistence diagrams

In [2] the authors apply LHS to persistence diagrams. The idea is similar to curves. Snap the points in the diagram to the intersection points of a grid. Persistence diagrams present an additional challenge in that points may be mapped to the diagonal as well as off diagonal points. The paper explains how to overcome this challenge and provides run time grantees.
The paper constructs a data structure that gives a six approximation of nearest neighbor searching in
$O((m \log n + m^2) \log \tau)$ time. Where m is the maximum number of points in a diagram and n is the number of diagrams in the database and τ is the number of levels in the data structure.
An interesting question is the following: What other data types that might benefit from locality sensitive hashing?

References

[1] Anne Driemel and Francesco Silvestri. Locality-sensitive hashing of curves. *CoRR*, abs/1703.04040, 2017.

[2] Brittany Terese Fasy, Xiaozhou He, Zhihui Liu, Samuel Micka, David L. Millman, and Binhai Zhu. Approximate nearest neighbors in the space of persistence diagrams. *CoRR*, abs/1812.11257, 2018.

[3] Andrii Gakhov. *Probabilistic data Structures and Algorithms for big data applicaitons*. Gakhav, 2019.