COMP 441 — Large Scale Machine Learning	Jan 20, 2016

Lecture 5: Minwise Hashing

Lecturer: Anshumali Shrivastava *Scribe By: Junyan Guo*

1 Sub-Linear Search with Hashing

A popular technique for efficient approximate near-neighbor search, uses the underlying theory of *Locality Sensitive Hashing* (LSH). LSH is a family of functions, with the property that similar input objects in the domain of these functions have a higher probability of colliding in the range space than non-similar ones. Consider $\mathcal{H}$ a family of hash functions mapping $\mathbb{R}^D$ to a discrete set $[0, R-1]$.

Definition 1. Locality Sensitive Hashing (LSH) Family *A family $\mathcal{H}$ is called (S_0, cS_0, p_1, p_2)-sensitive if for any two point $x, y \in \mathbb{R}^d$ and h chosen uniformly from $\mathcal{H}$ satisfies the following:*

- *if $Sim(x, y) \geq S_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \geq p_1$*
- *if $Sim(x, y) \leq cS_0$ then $Pr_{\mathcal{H}}(h(x) = h(y)) \leq p_2$*

Sub-linear Search with (K, L) LSH Algorithm: To be able to answer approximate near neighbor queries in sub-linear time, the idea is to create hash tables, (see Figure 1). Given the collection $\mathcal{C}$ which we are interested in querying for the near-neighbor, the hash tables are generated using the locality sensitive hash family. We assume that we have an access to the appropriate locality sensitive family $\mathcal{H}$ for the similarity of interest. In the classical (K, L) parameterized LSH algorithm, we generate L different meta-hash functions given by $B_j(x) = [h_{j1}(x); h_{j2}(x); ...; h_{jK}(x)]$. Here $h_{ij}, i \in \{1, 2, ..., K\}$ and $j \in \{1, 2, ..., K\}$, are KL different evaluations of the appropriate locality sensitive hash function. Each of these meta-hash functions is formed by concatenating K sampled hash values from $\mathcal{H}$.

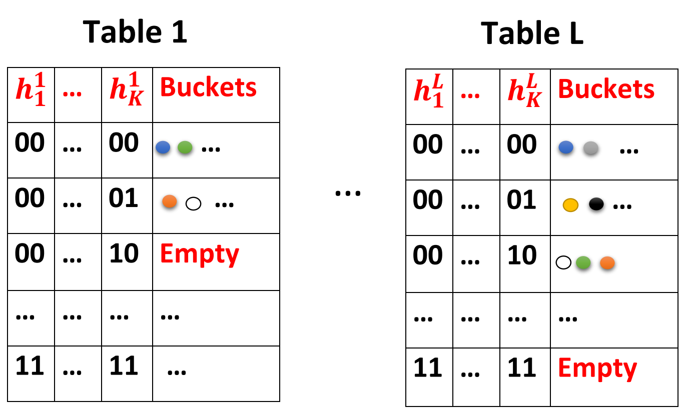

Figure 1: Hash Table Illustration

The overall algorithm works in two phases:

1) **Preprocessing Phase:** We construct L hash tables from the data by storing all elements $x \in \mathcal{C}$, at location $B_j(x)$ in hash-table j (See Figure 1 for an illustration). We only store

pointers to the vector in the hash tables, as storing data vectors will be very inefficient from the memory perspective.

2) **Query Phase:** Given a query Q whose neighbors we want to search for. We report the union of all the points in the buckets $B_j(Q)\ \forall j \in \{1, 2, ..., L\}$, where the union is over L hash tables. Note, we do not scan all the elements in $\mathcal{C}$, we only probe L different buckets, one from each hash tables. Thus, it is sub-linear search.

It was shown that having an LSH family for a given similarity measure and with appropriate choice of $K = O(\log n)$ and $L = O(n^\rho)$ the above algorithm is provably efficient. In particular,

Theorem 1. *(***Sub-linear Search***) For a (S_0, cS_0, p_1, p_2) -sensitive hash functions, then with $K = O(\log n)$ and $L = O(n^\rho)$, the LSH algorithm answers approximate near neighbor queries with $O(n^\rho \log_{1/p_2} n))$ query time, with high probability, where $\rho = \frac{\log p_1}{\log p_2} < 1$.*

2 Shingle Based Representation

Document is represented as a set of tokens over a vocabulary Ω. For example: Document "This is Rice University" for $k = 2$ can be represented by 2-shingles set {This is, is Rice, Rice University}

Most of web data is sparse and (near) binary. Modern "Big data" systems use only binary sparse data matrix.

3 Resemblance (Jaccard) Similarity

Definition: The resemblance (Jaccard) similarity between two sets $X, Y \subset \Omega$ is defined as

$$R = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a}$$

where $a = |X \cap Y|, f_x = |X|, f_y = |Y|$, and $|.|$ denotes the cardinality.

Binary vector representation: A set can be represented by an $|\Omega|$ dimensional binary vector. Each number (0 or 1) in the vector indicates whether or not a particular element exists in the set. Let x and y be the binary vector representations of sets X and Y, respectively. Then, to use the formula given in the definition, let

$$a = |X \cap Y| = x^T y, f_x = nonzeros(x), f_y = nonzeros(y)$$

4 Minwise Hashing

Algorithm: Sample a random permutation π over Ω, i.e.

$$\pi : \Omega \longrightarrow \Omega, \text{ where } \Omega = \{0, 1, ..., D - 1\}$$

The MinHash is given by

$$h_\pi(x) = min(\pi(x))$$

Proof: We want to prove that $\forall$ sets S_1 and S_2,

$$Pr(h_\pi(S_1) = h_\pi(S_2)) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

Proof. Let t be the element in $S_1 \cup S_2$ that has the smallest hash value. i.e.

$$t = \arg\min_{i \in S_1 \cup S_2} h_\pi(i)$$

Then, S_1 and S_2 have the same hash value if and only if $t \in S_1 \cap S_2$. Since π is a random permutation, every element in $S_1 \cup S_2$ has an equal probability to be t. Therefore,

$$Pr(h_\pi(S_1) = h_\pi(S_2)) = Pr(t \in S_1 \cap S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

□

Example: Let $D = 5, S_1 = \{0, 3, 4\}, S_2 = \{1, 2, 3\}$. Let permutation $\pi = [3, 2, 0, 4, 1]$ i.e. π maps $0, 1, 2, 3, 4$ to $3, 2, 0, 4, 1$, respectively. Then,

$$\pi(S_1) = \{3, 4, 1\} \text{ and } h_\pi(S_1) = min(\pi(S_1)) = 1$$

$$\pi(S_2) = \{2, 0, 4\} \text{ and } h_\pi(S_2) = min(\pi(S_2)) = 0$$

Binary vector view: Sample random permutation $\pi : [0, D] \mapsto [0, D]$ over attributes and shuffle the vector under π. Then, the hash value of the vector is the smallest index that is not zero.

Fingerprint: MinHash values can be used as fingerprint of data vectors because the hash of one vector is independent of other vectors and MinHash values can be used to estimate similarity.

5 A Real Problem : Counting Killings in Syria

Problem: Given a dataset of around 250,000 Syrian death records, estimate how many people died. Notice that each death record is a short noisy text description, and that multiple records may correspond to one individual.

Analysis: There are $\binom{250000}{2} \cong 3.1 \times 10^{10}$ pairs of records, so comparing every pair of records is costly.

Solution: Compute the MinHash for every record (use multiple hash functions/permutations and concatenate their values). Generate hash tables and buckets, i.e., the preprocessing step show in Section 1. This is near linear time $O(n)$. Now instead of all possible pairs only compare pairs of records that fall into the same bucket. If buckets are sparse (control using K) then the total number of pairs will be small and manageable.

References

Shrivastava, Anshumali. COMP 441 Lecture5.pdf. 2017
Chapter 3 of Mining Massive Datasets Book (http://infolab.stanford.edu/~ullman/mmds/book.pdf)