COMP 480/580 — Probabilistic Algorithms and Data Structure Feb 27, 2020

Lecture 12

Lecturer: Anshumali Shrivastava *Scribe By: Sharon Sun*

1 Parity of Minhash

In the last lecture, we discussed how to estimate the Jaccard Similarity between two documents through minwise hashing. It turns out that comparing the parities of minwise hashing values also gives us information about the similarity between two documents. We define the parity of a minhash value to be 0 if the value is even and 1 otherwise. For two documents S_1 and S_2, if $Minhash(S_1) = Minhash(S_2)$, the parities of $Minhash(S_1)$ and $Minhash(S_2)$ are guaranteed to be equal. As mentioned in the last lecture, the probability that $Minhash(S_1) = Minhash(S_2)$ is the Jaccard Similarity between S_1 and S_2, $\frac{|S_| \cap S_2|}{|S_1 \cup S_2|}$. If $Minhash(S_1) \neq Minhash(S_2)$, the probability that they have the same parity is 0.5. Therefore, we have a formula for the probability that a minwise hashing function generates the same parity for two documents S_1 and S_2:

$$\begin{aligned} Pr(Parity(Minhash(S_1)) = Parity(Minhash(S_2))) &= \frac{|S_| \cap S_2|}{|S_1 \cup S_2|} + (1 - \frac{|S_| \cap S_2|}{|S_1 \cup S_2|}) * 0.5 \\ &= 0.5(1 + \frac{|S_| \cap S_2|}{|S_1 \cup S_2|}). \end{aligned}$$

This probability can be used to estimate the similarity between S_1 and S_2.

1.1 Example

Suppose that we are given 50 parity of minhashes for document S_1 and S_2 respectively and we want to estimate Jaccard Similarity between S_1 and S_2.
We can store the data in less than 7 bytes (one bit for each parity). Let the 50 parity bits of S_1 be P_1 and those of S_2 be P_2. $Pr(Parity(Minhash(S_1)) \neq Parity(Minhash(S_2)))$ can be calculated by counting the number of 1s in $P_1 \hat{} P_2$.
When Jaccard Similarity is 0.8, the error is a little greater than 0.05.

2 Minwise Hashing in Similarity Search

2.1 Example

Suppose we have two minwise hash functions h_1 and h_2 and a data set in R^D. For any document x in the data set, we compute $h_1(x)$ and $h_2(x)$ and put x in bucket $h_1(x), h_2(x)$.
Given a query q, suppose $h_1(q) = 01$ and $h_2(q) = 11$. Instead of searching q in the whole data set, we only consider documents in bucket 0111. The reason is that hash collisions are indicators of similarity, so bucket 0111 contains documents that are likely to have more similarities with q. Therefore, we are able to shrink the search range with the cost of constant time.
We only have one table of buckets now. It is possible that we will miss documents very similar

with q but have different minhash values from q. Therefore, to decrease the probability of missing good matches of q, we can have L tables with independent hash functions, and consider the union of buckets in these tables when searching.
To increment the similarity between documents in the same bucket, we can increase K, the number of minwise hash functions we use in each table.

2.2 The LSH Algorithm

Create L independent hash tables and use K minwise hash functions in each table.

Algorithm 1: PreprocessDatabase

Input: A data set D. L hash tables $T_1, ..., T_L$. K minwise hash functions $h_1^i, ..., h_K^i$ for every table T_i.

---

for $x \in D$ **do**
  for $i \leftarrow 1$ **to** L **do**
    Put x in $T_i[h_1^i(x), h_2^i(x), ..., h_K^i(x)]$;

The time complexity is $O(|D|LK)$ because we compute $L \times K$ hash values for every element in the data set, which contains $|D|$ elements in total.

Algorithm 2: Query

Input: A document q. L hash tables $T_1, ..., T_L$. K minwise hash functions $h_1^i, ..., h_K^i$ for every table T_i.

$U \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** L **do**
  $U \leftarrow U \cup T_i[h_1^i(x), h_2^i(x), ..., h_K^i(x)]$;
Get the best elements in U base on similarity with q;

2.3 A bit of Analysis

For a document x in the data set and query q, let $p_x = Pr(h(x) = h(y))$ where h is any minhash function. We know that p_x equals the Jaccard Similarity between x and q.
Since there are K hash functions used in each bucket, p_x^K is the probability that x is mapped to q in a single bucket. It follows that $1 - p_x^K$ is the probability that x is not mapped to q in a bucket.
Because there are L buckets in total, the probability that x is not mapped to q in any of the buckets is $(1 - p_x^K)^L$. Then $1 - (1 - p_x^K)^L$ is the probability that x is retrieved. Using this information, we can solve K and L to have the probability of retrieval we desire.
For example, suppose $K = 5$ and $L = 10$. When $J(x, q) > 0.8$, the probability of retrieving x is greater than 0.98. When $J(x, q) < 0.5$, the probability of retrieving x is less than 0.2.

2.4 Choice of K and L

Let P_1 be the probability of hash collisions of query q and a document similar with q. Let P_2 be the probability of hash collisions of query q and a document not similar with q. Suppose that $P_1 > S_0$, $P_2 < cS_0$, and we have n documents in our data set.
The probability that a document is retrieved is $1 - (1 - P_i^k)^L$. This value is greater than $1 - (1 - S_0^k)^L$ for a document similar with q and less than $1 - (1 - c^k S_0^k)^L$ for a document not similar with q. Therefore, to bound P_1 and P_2, we can solve K and L for:

$$1 - (1 - S_0^k)^L > P$$
$$1 - (1 - c^k S_0^k)^L < \frac{m}{n}$$

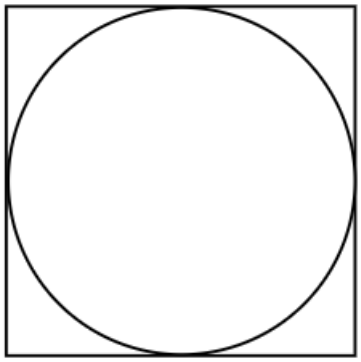
, where m is the number of documents not similar with q we expect to see in the bucket.

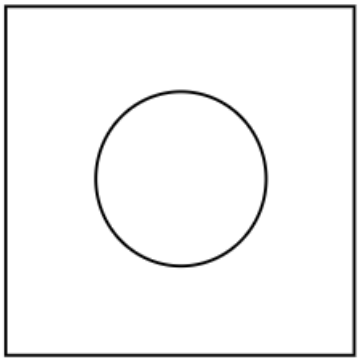
3 Advantage of Minwise Hashing

Instead of getting the minhash value of a document, we can pick a word from the document using random sampling. If two documents have the same random sampled word, we know that the word is in both of them and it is possible that they are similar to each other. If two documents have the same minhash value, we also know that there are contents contained in both of them. However, aside from what is contained in both of the documents, minwise hashing also tells us what is absent both of them. If documents S_1 and S_2 are both mapped to 128 by minwise hashing, we can conclude that no element in S_1 or S_2 has hash value less than 128. Therefore, minwise hashing gives us a lot more information than random sampling does.

3.1 Illustration

Suppose that we want to estimate π and we have two figures. Length of the radius of the circle is $\frac{1}{2}$ of the length of the side of the square in the left figure, and $\frac{1}{4}$ in the right figure.





We randomly sample points from the figure and calculate the probability that a point falls within the circle. In the left figure, this probability estimates $\frac{\pi}{4}$. In the right figure, this probability estimates $\frac{\pi}{16}$.
The left figure produces more accurate estimations of π because the variance goes up more for the right figure than the left as the number of points we sampled goes up. Therefore, to estimate a value, it is better to sample within a smaller limit.