CS7280: Scalable Algorithms Spring 2025

Lecture Lecture 18 - March 19, 2025

Prof. Prashant Pandey *Scribe: Aidan Ih*

1 Overview

In the previous lecture, we studied locality sensitive hashing (LSH) using a variety of techniques. We established that using minhashing, the probability that two sets will hash to the same location is equal to the Jaccard Similarity between those sets. We expanded on this idea with parity based min-hashing, and 1-permutation based min hashing.

In this lecture, we explore how to use LSH tables to build data structures which enable fast approximate nearest neighbor queries

2 Nearest Neighbor Search

2.1 Precision/Recall

Before designing a data structure to allow NNS queries, we must understand the notions of precision and recall. When querying an approximate data structure, all values may be classified in one of four categories: True Positives, True Negatives, False Positives, and False Negatives

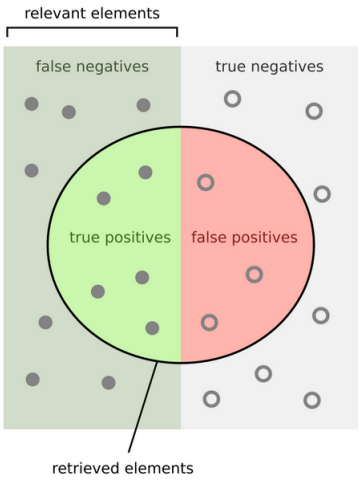

Fig 1: Precision and Recall [1]

We define Precision and Recall as follows:

$$Precision = \frac{T_p}{T_p + F_P} \qquad Recall = \frac{T_p}{T_p + F_n}$$

Put in english, precision is the ratio of retrieved elements that are correct to the total number of retrieved elements. Recall is the ratio of retrieved and relevant elements to the total number of relevant elements. In other words, precision is the probability that an element in the retrieved elements is also a relevant element. Similarly, recall is the probability that some relevant element was successfully retrieved.

When designing and querying NNS data structure, we must balance precision and recall. For instance, if we are interested in retrieving a high percentage of the relevant elements, we can increase our query size. However, doing this also increases the number of false positives, which may lead to issues down the pipeline.

2.2 A NNS data structure

Let's design a data structure that allows approximate NNS queries.

1. let there be L hash tables with K LSH hash functions each
2. For each table, each element is hashed using K hash functions. Each hash is concatenated into a single hash signature, then inserted into the table
3. To query NNS for some element x, compute the hash signature of x for each hash table, and return the union of all elements present at that hash location.

intuition: By using multiple (K) hash functions, we decrease the likelihood that distant elements hash to the same location. By using multiple (L) hash tables, we increase the likelihood that similar elements are hashed to the same location at least once.

Example:

Let us have a collection of sets S_1, S_2, S_3, S_4, S_5. We hash these into 4 hash tables:

	S_1
	S_5
	S_2
	S_3, S_4
	...

	S_4
	S_1, S_2
	S_3
	S_5
	...

	S_1
	S_2
	S_3
	S_4
	S_5

	S_5
	S_2, S_1
	S_3
	S_4
	...

We then query S_6 by taking the union of elements found at the location S_6 hashes to within each table, marked below with an X.

	S_1
X	S_5
	S_2
	S_3, S_4
	...

	S_4
	S_1, S_2
X	S_3
	S_5
	...

	S_1
	S_2
	S_3
	S_4
X	S_5

	S_5
	S_2, S_1
	S_3
	S_4
X	...

Thus, our NNS query returns S_5, S_3. To increase the recall, we could use a larger number L of tables to increase the number of returned elements. We could similarly increase the precision by increasing the number of of hash functions K to decrease the odds of distant elements hashing to the same bucket.

Let's analyze the probabilistic properties of this structure. Suppose we are using minhashing, and have L = 10 and K = 10. Let y be a set that we have hashed and inserted into the data structure. We will now query x for it's nearest neighbors in the structure, and J(x, y) = .9.

$$Pr(\text{x, y land in the same bucket in a specific table}) = J^k = .9^{10} \approx .35$$
$$Pr(\text{y is retrieved when querying x}) = 1 - (1 - J^k)^L = 1 - (1 - J^{10})^{10} \approx .986$$

So, we find that when the similarity is high, the likelihood that we identify it as an approximate nearest neighbor is also high.

3 Two Choices

Below is a quick sketch for the intuition behind Mitzenmacher's proof on the power of 2 choices in balls and bins. It is known that the maximum load after throwing n balls into n bins $O(\frac{log(N)}{loglog(N)})$. Using the two-choice strategy, where we "check" two bins and place the ball into the bin with lesser load, this bound can be reduced to $O(lglg(N))$.

Proof Sketch:

Let the height of a ball be the number of balls, including itself, in the bin that it is placed into. Similarly, let the height of a bin be the number of balls it contains. After throwing N balls, we have:

$$\text{Max \# bins with height at least 2} = \frac{N}{2}$$
$$P(\text{Selected bin has height of at least 2}) \leq \frac{N}{2} \cdot N = \frac{1}{2}$$

The above statement is true as any additional bins with at least 2 balls would require greater than N balls to have been thrown. We can use this information to bound the maximum number of bins with height of at least 3. In order for the height of a ball to be 3, it must have selected two bins with height of at least 2. The probability this occurs is $(\frac{1}{2})^2 = \frac{1}{4}$, so we have at most 1/4 of the bins have height 3. We can then bound the number of bins with height of at least 4 the same way, by choosing 2 bins with height at least 3: $(\frac{1}{4})^2 = \frac{1}{16}$.

Generally, we have the expected number of bins with height h is $N \cdot \frac{1}{2^{2^{(h-2)}}}$. This simplifies to 1 exactly when $h = lglg(n) + 2$. From this point on, we can no longer select two bins with at least that height, as only one such bin exists. Thus, we bound the largest bin to have height = $O(lglg(N))$.

For additional reading and the formal proof, see [2] and [3].

References

[1] Wikipedia Contributors. *Precision and recall*. Wikipedia

[2] Gregory Valiant, Mary Wootters. *Balls in Bins and Power-of-Two-Choices*. CS265/CME309: Randomized Algorithms and Probabilistic Analysis. Stanford University, January 12, 2025

[3] Michael Mitzenmacher. *The power of two choices in randomized load balancing*. IEEE Transactions on Parallel and Distributed Systems, 12(10):1094–1104, 2001.