COMP 480/580 — Probabilistic Algorithms and Data Structure Oct 17, 2024

Lecture 14

Lecturer: Anshumali Shrivastava

Scribe By: Hongyi Li(hl93), Yiting Chen(yc203), Jinbiao Wei(jw142), Jiawen Wei(jw143)

1 Introduction

Here we have the problem of implementing large-scale image search in a database involves efficiently storing, indexing, and retrieving images based on their visual content. This is crucial for applications like reverse image search, content-based image retrieval (CBIR), and recommendation systems.

Some of the potential challenges are:

- **High Dimensionality**: Feature vectors can be large, making indexing and retrieval computationally intensive.
- **Storage Requirements**: Large datasets require significant storage, necessitating efficient data compression and management strategies.
- **Latency Constraints**: Real-time applications demand quick response times, which can be challenging with large-scale data.

In classical hashing, we define a hash function such that if $x = y \rightarrow h(x) = h(y)$ and conversely if $x \neq y \rightarrow h(x) \neq h(y)$. However, this technique provides no assurance that similar outputs will be mapped to similar values in the output space.

To satisfy this need to efficiently find approximate nearest neighbors in high-dimensional data spaces, we need to introduce the idea of Locality Sensitivity Hashing (LSH) [1]. LSH defines a whole new class of hash functions (denoted *sim*) with the property that the probability of a collision between two values directly correlates to the similarity of the respective items. In other words, $sim(x, y)$ is high $\rightarrow Pr(h(x) = h(y))$ is high and $sim(x, y)$ is low $\rightarrow Pr(h(x) = h(y))$ is low. Conversely, $h(x) = h(y) \rightarrow sim(x, y)$ is high and $h(x) \neq h(y) \rightarrow sim(x, y)$ is low.

2 Jaccard Similarity

First, let's introduce some basic terms that would help us better understand LSH.

2.1 Mathematical Definition:

For two sets A and B, the Jaccard similarity $J(A, B)$ is calculated as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- $|A \cap B|$: The number of elements common to both sets (the intersection).
- $|A \cup B|$: The total number of unique elements in both sets (the union).

The result ranges between 0 and 1:

- 0: No shared elements between the sets.
- 1: The sets are identical.

3 N-grams

3.1 N-grams Overview

- **N-grams**: These are contiguous sequences of n items (characters, words, etc.) from a given text or speech sequence.
- In this case, **character 3-grams** are used. A 3-gram splits a string into all contiguous sequences of 3 characters. For example:
  - The string `"iphone 6"` is split into:

$$\{\texttt{iph}, \texttt{pho}, \texttt{hon}, \texttt{one}, \texttt{ne}, \texttt{e 6}\}$$

3.2 Example: Jaccard Similarity Calculation

1. **"amazon" vs "anazon"**:
   - The 3-grams for "amazon" and "anazon" are:
     - **"amazon"**: {ama, maz, aza, zon}
     - **"anazon"**: {ana, naz, aza, zon}
   - **Intersection** (common elements): {aza, zon} (2 elements).
   - **Union** (all unique elements): {ama, maz, aza, zon, ana, naz} (6 elements).
   - **Jaccard Similarity**:

$$J(\text{"amazon"}, \text{"anazon"}) = \frac{2}{6} = \frac{1}{3}$$

4 Minhash

4.1 Minwise Hashing (Minhash)

- **Document** : $S = \{\texttt{ama}, \texttt{maz}, \texttt{azo}, \texttt{zon}, \texttt{on}\}$.
- Generate Random U_i: $\mathit{Strings} \rightarrow N$.
- $U_i(S) = \{U_i(\texttt{ama}), U_i(\texttt{maz}), U_i(\texttt{azo}), U_i(\texttt{zon}), U_i(\texttt{on})\}$
- Let's say $U_i(S) = \{153, 283, 505, 128, 292\}$
- Then Minhash: $h_i = \min h_i(S) = 128$

4.2 Properties of Minwise Hashing

Claim:

$$Pr(\text{Minhash}(S_1) = \text{Minhash}(S_2)) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = J$$

Proof: Let us consider a hash function U which is used to calculate the minwise hash of the sets S_1 and S_2. Let e be the element that has the smallest hash value in $S_1 \cup S_2$. It's easy to see that Minwise(S_1) $= U(e)$ if and only if $e \in S_1$ and Minwise(S_1) $= U(e)$ if and only if $e \in S_2$. However, it is also possible for Minwise(S_1) $=$ Minwise(S_2) $= U(e)$ if and only if $e \in S_1 \cap S_2$. If we say $|S_1 \cap S_2| = N$ and $|S_1 \cup S_2| = M$, then there are M possible choices for e with N of which resulting in a collision, so the probability of a collision is:

$$\frac{N}{M} = J$$

5 Locality Sensitive Hashing (LSH)

5.1 What is LSH and Why do we need it?

Hashing is a method that maps input data (like strings or numbers) to a fixed-size value, called a hash, using a hash function. The goal is to store or retrieve data efficiently by distributing it across buckets in a hash table. Traditional hashing ensures that different inputs produce distinct hashes to minimize collisions.

$$\begin{aligned} &\text{if} \quad x = y \to h(x) = h(y) \\ &\text{if} \quad x \neq y \to h(x) \neq h(y) \end{aligned}$$

However, in real-world applications, if you look at even the same products, they have different descriptions, which means the exactness is often broken during data retrieval, and we want to extend it to something that is not exactly the same as some sort of a near implicate.

Thus, the main idea behind Locality Sensitive Hashing (LSH) is to use hash functions that respect the similarity between data points. By hashing data such that similar points map to the same bucket, we reduce the search space, making it possible to perform approximate nearest-neighbor searches efficiently in large, high-dimensional datasets. The algorithm balances speed and accuracy, ensuring that you retrieve a subset of candidates quickly, which you can refine further using exact similarity measures if needed.

$$\text{if} \quad sim(x, y) \quad \text{is high/low} \to \text{Probability of} \quad h(x) = h(y) \quad \text{is high/low}$$

5.2 LSH Algorithm Introduction

The LSH algorithm aims to efficiently find approximate nearest neighbors for high-dimensional data by mapping similar data points to the same hash buckets.

Create Hashing Tables

The first step of LSH is creating hashing tables, which consist of **L** independent hash tables and each hash table contains **K** hashing functions. For each hash table $\mathbf{T}^i, i = 1, 2, ..., L$, $\mathbf{T}^i = \{h_1^i, h_2^i, ..., h_k^i\}$. The goal is to ensure that similar data points hash to the same bucket across multiple tables. Since a single hash function or table may not capture all relevant neighbors, multiple independent hash tables are used to increase the chance of finding similar items. A larger L leads to a better performance of catching all similar points.

5.2.1 Data Pre-processing

After the hash tables have been created, we need to process each data point x in the dataset and insert it into the corresponding bucket in each hash table: $\{h_{(i-1)K+1}(x), ..., h_{iK}(x)\}$ for each $\mathbf{T}^i$.

1. Compute the hash values using the set of hash functions for each table.
2. Insert the data point x into the corresponding bucket in each hash table.

5.2.2 Data Query

During the query phase, given a query point q, we first need to take the union of L buckets from each hash table and then get the best elements from the union based on similarity with q. The expectation is that this unit is not too big, definitely not as big as the database, because otherwise the whole point is lost. There are two things that are very important regarding the query phase of LSH: (1)the union is not very big. (2) the chance of missing a very good candidate is very small. Thus, we need to select carefully on the hash family to fulfill the two requirements.

6 Analysis of LSH

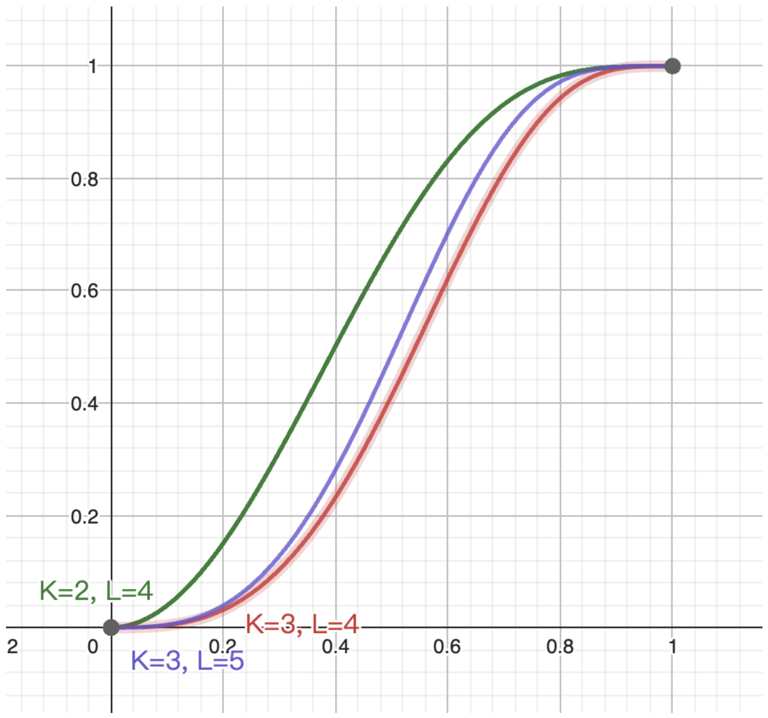
We now consider the probability of collision with locality sensitive hashing. That is, how likely are two elements hashed to the same bucket of a table. Fix x and let q be arbitrary. Recall

$$P(\text{minhash}(x)\text{=}\text{minhash}(q)) = Jaccard$$

Denote this probability by p_x. Then we have:

- p_x^K is the probability that x and q are mapped to the same bucket in one table.
- $1 - p_x^K$ is the probability that x and q are not mapped to the same bucket in one table.
- $(1 - p_x^K)^L$ is the probability that x and q are not mapped to the same bucket in any of the L tables.
- $1 - (1 - p_x^K)^L$ is the probability that x and q are mapped to the same bucket in at least one of the L tables i.e. probability of collision.

Note that $1 - (1 - p_x^K)^L$ is monotonically increasing on the interval $0 \leq p_x \leq 1$



That is, the more similar x and q are, the more likely we will retrieve q in a query of x. LSH based search achieves O(n^ρ), where ρ is function of similarity threshold (how similar two objects need to be to be considered ”neighbors” for retrieval) and gap (the difference in similarity between items you want to retrieve and items that shouldn’t be retrieved). When the similarity threshold is high, only highly similar items are retrieved, which leads to fewer candidates being compared and thus near constant time search.

6.1 Choice of K, L

As mentioned before, K, which controls the quality of each bucket, and L, which controls the failure probability. The selection of K and L is crucial for balancing efficiency and accuracy. Increasing K reduces the number of candidates retrieved exponentially, while increasing L increases the chances of retrieving relevant candidates linearly. In practice, it is suggested that $K = \approx log(n)$ and $L = \sqrt{n}$, where n is the size of the database.

7 Cosine Similarity Introduction

Cosine similarity is a measure of similarity between two vectors, often used in information retrieval and text mining to compare documents or items represented as vectors. Given two vectors $\mathbf{x}$ and $\mathbf{y}$, the cosine similarity is defined as:

$$\text{cos\_sim}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{||\mathbf{x}||\,||\mathbf{y}||}$$

Where:

- $\mathbf{x} \cdot \mathbf{y}$ is the dot product of the two vectors.
- $||\mathbf{x}||$ and $||\mathbf{y}||$ are the magnitudes (norms) of vectors $\mathbf{x}$ and $\mathbf{y}$, respectively.

The value of cosine similarity ranges from -1 to 1. A value of 1 implies that the two vectors are identical (pointing in the same direction), while -1 implies they are completely opposite. A value of 0 implies that the vectors are orthogonal (no similarity).

8 Cosine Similarity and LSH

One variant of LSH, called *cosine LSH*, is based on cosine similarity. In this method, instead of using Jaccard similarity, cosine similarity is used to determine the likelihood of two vectors being similar.

The key idea in cosine LSH is to use random hyperplanes to project high-dimensional data points onto a lower-dimensional space. The vectors' relative angles, represented by cosine similarity, dictate whether they hash to the same bucket or not.

For a given vector $\mathbf{x}$, the hash function $h(x)$ can be defined as follows:

$$h(\mathbf{x}) = \begin{cases} 1, & \text{if the vector lies on the same side of a randomly chosen hyperplane crossing the origin,} \\ 0, & \text{otherwise.} \end{cases}$$

This means that the random hyperplane partitions the space into two regions. If both vectors $\mathbf{x}$ and $\mathbf{y}$ are on the same side of the hyperplane, they hash to the same value $h(\mathbf{x}) = h(\mathbf{y}) = 1$; otherwise, they hash to different values.

For two vectors $\mathbf{x}$ and $\mathbf{y}$, let θ be the angle between them. The probability that their random hash functions (using hyperplane projections) are equal is given by:

$$\mathbb{P}(h(\mathbf{x}) = h(\mathbf{y})) = 1 - \frac{\theta}{\pi}$$

The proof of the formula is straightforward. Consider two vectors $\mathbf{x}$ and $\mathbf{y}$ with an angle θ between them. The random hyperplane partitions the space into two regions. For the hash values of $\mathbf{x}$ and $\mathbf{y}$ to differ, the hyperplane must lie between them. The probability of this happening corresponds to the proportion of the unit circle (or spherical space in higher dimensions) that is "swept out" by the angle θ. This proportion is simply $\frac{\theta}{\pi}$, since θ is measured in radians and a full circle is π radians.

Thus, the probability that the two vectors are separated by the hyperplane is $\frac{\theta}{\pi}$. Conversely, the probability that both vectors $\mathbf{x}$ and $\mathbf{y}$ lie on the same side of the hyperplane is $1 - \frac{\theta}{\pi}$.

References

[1] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *IEEE Signal processing magazine*, 25(2):128–131, 2008.