COMP580 Scribe Assignment

Nikhil Chigali(nc71), Jeffrey Joan Sam(jj116)

3rd October 2023

1 Large Scale Image Search

Large-scale image search involves efficiently searching a large database of images to find the most similar images to a given query image. The goal is to retrieve relevant images quickly and accurately.

2 Space Partitioning Methods

2.1 Trees and Efficiency in Near-Neighbor Search

Many near-neighbor search methods utilize a tree structure to organize the database. In this structure, the branches of the tree divide the data space into partitions, enabling efficient search for a given query. The approach involves navigating through these space partitions until a partition containing similar points is found. This method is precise and operates effectively in low-dimensional spaces.

2.2 Efficiency Challenges in High Dimensions

However, as we transition to higher dimensions, the efficiency of space partitioning methods diminishes. In such high-dimensional spaces (D ¿ 10), the query time becomes comparable to that of an exhaustive search. The fundamental reason behind this challenge lies in high-dimensional spaces, where the volume of the space increases exponentially with the dimensionality. As a result, the partitions become less effective in segregating the data, making it harder to quickly identify relevant partitions for a given query. This issue severely hampers the efficiency and practicality of space partitioning methods in high-dimensional scenarios. Thus, alternative approaches and techniques are needed to address this efficiency problem in high-dimensional near-neighbor search.

3 Locality Sensitive Hashing

Locality Sensitive Hashing is the solution to the high dimension problem in Space Partitioned methods. It provides us with an approximation that is much

faster and more efficient.

3.1 Motivating Problem: Search Engine Autocorrection

In the context of real-time query correction, where a user might mistype a query like "amaozn" instead of "amazon", we aim to swiftly suggest the correct query by leveraging a database of common query strings. The challenge is achieving this within a 20ms latency constraint, where traditional exact similarity computation is too slow, taking 400ms per computation. To address this, we employ Locality Sensitive Hashing (LSH), a technique that hashes similar items to the same bucket with high probability.

Using LSH, we hash the user's query and compare it against pre-hashed common query strings. This provides us with a set of potential corrections or suggestions. Rather than performing exact similarity computations, which are time-prohibitive, we estimate similarity using a cost-effective distance function based on hash values. This approximation allows us to select the best-matching query efficiently, meeting the real-time requirement and achieving a performance improvement of 210,000 times compared to exact similarity computation. By applying LSH and approximate similarity estimation, we provide users with rapid and accurate query corrections, enhancing their search experience.

3.2 Locality Sensitive Hash Functions

Classical Hash Functions

1. if $x = y \rightarrow h(x) = h(y)$
2. if $x \neq y \rightarrow h(x) \neq h(y)$

Locality-sensitive hashing on the other hand gives Collision probability which represents the similarity distance between elements.

Locality Sensitive Hash Functions

1. if $sim(x, y)$ is high $\rightarrow Pr(h(x) = h(y))$ is high
2. if $sim(x, y)$ is low $\rightarrow Pr(h(x) = h(y))$ is low

3.3 Jaccard Similarity Measure

Jaccard similarity is a widely used measure to quantify the similarity between two sets. It is particularly applicable in various fields, including information retrieval, data mining, and natural language processing, to assess the overlap or similarity between two collections of items. The Jaccard similarity coefficient is defined as the size of the intersection of sets divided by the size of their union.

Formula for Jaccard Similarity

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

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

Where:

- $|A \cap B|$ represents the size of the intersection of sets A and B, i.e., the number of elements common to both sets.
- $|A \cup B|$ represents the size of the union of sets A and B, i.e., the total number of distinct elements in both sets.

Interpreting Jaccard Similarity

The Jaccard similarity ranges from 0 to 1.

- $J(A, B) = 0$ implies that the sets A and B have no common elements, indicating complete dissimilarity.
- $J(A, B) = 1$ indicates that the sets A and B are identical, implying maximum similarity.

Let's consider two sets of numbers, A and B: $A = \{2, 4, 6, 8, 10\}$
$B = \{4, 6, 8, 12, 14\}$
We'll calculate the Jaccard similarity $J(A, B)$ using the formula:

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

where $|A \cap B|$ is the size of the intersection of sets A and B, and $|A \cup B|$ is the size of the union of sets A and B.

1. **Intersection ($|A \cap B|$):** - The numbers common to both sets A and B are 4, 6, and 8, so $|A \cap B| = 3$.

2. **Union ($|A \cup B|$):** - The numbers in the union of sets A and B are 2, 4, 6, 8, 10, 12, and 14, so $|A \cup B| = 7$.

Now, let's plug these values into the Jaccard similarity formula:

$$J(A, B) = \frac{3}{7} \approx 0.4286$$

So, the Jaccard similarity between sets A and B is approximately 0.4286.

3.4 N-gram representation to form sets

Let's create contiguous character 3-gram representations (3-grams or trigrams) for the string "samsungs23," we'll slide a window of size 3 characters through the string, considering each group of three consecutive characters as a 3-gram.
String: "samsungs23"
Set for "samsungs23":

$$\{"sam", "ams", "msu", "sun", "ung", "ngs", "gs2", "s23"\}$$

3.5 Jaccard similarity on N-gram set

3.5.1 "amazon" vs "anazon"

- **3-Grams for "amazon":**
  - Set of 3-grams: {"ama","maz","azo","zon"}
- **3-Grams for "anazon":**
  - Set of 3-grams: {"ana","naz","aza","zon"}
- **Intersection and Union:**
  - Intersection ($|A \cap B|$): 2 3-grams common to both sets: "azo" and "zon"
  - Union ($|A \cup B|$): Total 6 unique 3-grams in both sets
- **Jaccard Similarity:**
  - $J(\text{"}amazon\text{"},\text{"}anazon\text{"}) = \frac{2}{6} \approx 0.3333$

3.5.2 "amazon" vs "amazom"

- **3-Grams for "amazon":**
  - Set of 3-grams: {"ama","maz","azo","zon"}
- **3-Grams for "amazom":**
  - Set of 3-grams: {"ama","maz","azo","zom"}
- **Intersection and Union:**
  - Intersection ($|A \cap B|$): 3 3-grams common to both sets: "ama", "maz", "azo"
  - Union ($|A \cup B|$): Total 5 unique 3-grams in both sets
- **Jaccard Similarity:**
  - $J(\text{"}amazon\text{"},\text{"}amazom\text{"}) = \frac{3}{5} = 0.6$

3.5.3 "amazon" vs "random"

- **3-Grams for "amazon":**
  - Set of 3-grams: {"ama","maz","azo","zon"}
- **3-Grams for "random":**
  - Set of 3-grams: {"ran","and","ndo","dom"}
- **Intersection and Union:**

- Intersection ($|A \cap B|$): 0 common 3-grams between both sets
- Union ($|A \cup B|$): Total 8 unique 3-grams in both sets

- **Jaccard Similarity:**
  - $J(\text{"}amazon\text{"}, \text{"}random\text{"}) = \frac{0}{8} = 0.0$

3.6 Random Sampling with Universal Hashing

Given the string "Amazon" and the set of 3-grams $\{\text{"}Ama\text{"}, \text{"}maz\text{"}, \text{"}azo\text{"}, \text{"}zon\text{"}\}$, we can use a random hash function $U : string \to [0, R]$ to obtain a random element from the set. The probability of U mapping a string s to a specific value c is $\frac{1}{R}$.

Random Element Selection via Universal Hashing

To select a random element from the set, we hash every 3-gram token using the universal hash function U and then choose the token with either the minimum or maximum hash value.

Example:

Consider the hash values obtained for each 3-gram:

$$\{U(\text{"}Ama\text{"}), U(\text{"}maz\text{"}), U(\text{"}azo\text{"}), U(\text{"}zon\text{"})\} = \{10, 2005, 199, 2\}$$

In this example, we have chosen "zon" as the random element based on the minimum hash value.

This method demonstrates how universal hashing and random sampling can be used to obtain a random element from a set of 3-grams associated with a given string.

4 Minwise Hashing for Set S

In Minwise Hashing, we compute the minwise hash value of a set S by generating random hash values $U(\cdot)$ for each element in the set and selecting the minimum hash value. The MinHash value for set S is calculated as follows:

$$MinHash(S) = \min\{U(s_1), U(s_2), \ldots, U(s_L)\}$$

Example

Consider a document $S = \{ama, maz, azo, zon, on.\}$. We generate random hash functions $U_i : String \to N$ and obtain hash values for each element in S:

$$U_i(S) = \{U_i(ama), U_i(maz), U_i(azo), U_i(zon), U_i(on.)\} = \{153, 283, 505, 128, 292\}$$

In this example, the MinHash value is 128.

4.1 Properties of Minwise Hashing

1. **Applicability:**
   - Minwise Hashing can be applied to any set, making it a versatile technique for similarity estimation.
2. **Collision Probability and Jaccard Similarity:**
   - The probability that the MinHash values of two sets, $S1$ and $S2$, are equal ($Pr(MinHash(S1) = MinHash(S2))$) is equal to the Jaccard similarity of the sets ($\frac{|S1 \cap S2|}{|S1 \cup S2|}$).
   - Mathematically, $Pr(MinHash(S1) = MinHash(S2)) = \frac{|S1 \cap S2|}{|S1 \cup S2|}$.

4.2 Proof: MinHash is Locality Sensitive

- **Objective**: Prove that MinHash collision probability equals Jaccard similarity.
- **Key Observations**:
  1. **Minimum Hash Element**: Let e be the element with the smallest hash value in $S1 \cup S2$ (belonging to both sets if applicable).
  2. **MinHash Comparisons**: MinHash(S1) or MinHash(S2) will be $U(e)$ (hash of the minimum element) based on the set e belongs to.
  3. **Randomness of e**: e is a randomly selected element from $S1 \cup S2$ due to universal hashing.
- **Collision Probability**:
  - Each of the N possible e choices has $Pr(U(e) collides) = \frac{M}{N}$, where M is the common elements in $S1$ and $S2$.
  - **Hence**: $Pr(MinHash collision) = \frac{M}{N}$, which simplifies to $Pr(MinHash collision) = \frac{|S1 \cap S2|}{|S1 \cup S2|}$.
- **Conclusion**:
  - The collision probability in MinHash equates to the Jaccard similarity, implying : $Pr(MinHash collision) = \frac{|S1 \cap S2|}{|S1 \cup S2|}$.