COMP 480/580 — Probabilistic Algorithms and Data Structure Feb 25,2020

Lecture 11:Minwise Hashing and Efficient Search

Lecturer: Anshumali Shrivastava *Scribe By: Yawen Guo*

Situation:

Suppose we need to find out similar images of a query image from a large database. Which algorithms should we use? This is one of the large scale search problems. There are many solutions to this problem. We might use hashing algorithms to solve it. Just remove the duplicates from an array of n integers. However, if we also want to find out near duplicates, which algorithm should we choose?

Goal:

We want to find the most similar image x^* to query image q. And require speed and accuracy.

1 Similarity or Near-Neighbor Search

1.1 Methodology

This is a common solution to calculate similarity. Assume there are a collection C and a distance metric. For any query q, compute

$$x^* = \arg\min_{x \in C} sim(q, x)$$

x^* represents the closest item.

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

$$sim(x, y) = \frac{1}{1 + d(x, y)}$$

There are many distance function, for example, let us use the simple function to compute the distance between two points. So when x and y are close, $d(x, y)$ is small.
As for the similarity measure (sim(x,y)), when x and y are close, the similarity will be large.

1.2 Analysis

The query time is O(nD) per query, when n is the size of C while D is the dimension of an image. Querying is a very common operation with very larger number of n and D. Therefore, this exhaustive search solution performs badly. To obtain better performance, we should allow algorithms return an approximate solution $\hat{x}$.

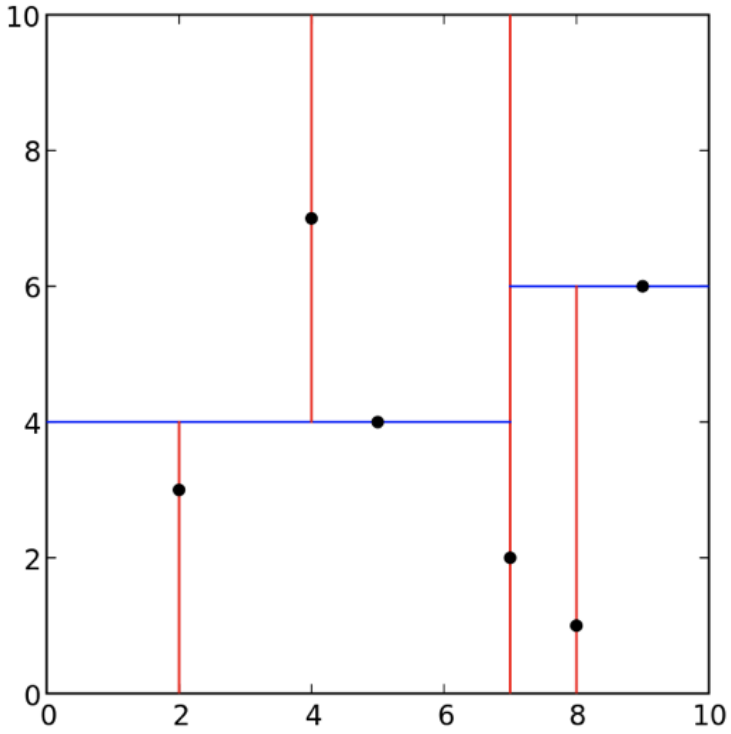

Figure 1: Space Partitioning Methods

2 Space Partitioning Methods

2.1 Methodology

Using trees. We can organize the database into a tree structure, where the branches of the tree partition the data space. To search for a query, we can perform an efficient search through the space partitions until we find the partition that contains similar points.

2.2 Analysis

However, this solution only perform well in low dimensions and can't efficiently be applied to high dimensions. If D is bigger than 10, the query time will be exhaustive.

Motivating problem: Search Engines

Assume we want to correct/suggest a user-typed query in real time. Suppose a user want to search for "Nike shoes" but mistakenly typed "Niki shoes". We observe other users behaviors and know that the brand should be 'Nike'. Since these two words has high similarity, we should suggest the correct word.
Moreover, if we store statistically significant query strings in a database D and we are given a new user typed query q, we need to find the closest string $s \in D$ to q. We want to do this in real time, which means we must limit the latency to 20ms. For an exact solution, a cheap distance function takes 400 ms to compute, so we can't afford even one similarity computation! In this case, we can use locality sensitive hashing to get an approximate solution and obtain

performance 210000 times better than the exact solution.

3 Locality Sensitive Hashing

Classical Hashing:
1. If $x = y \rightarrow h(x) = h(y)$
2. If $x \neq y \rightarrow h(x) \neq h(y)$

Locality Sensitive Hashing(LSH):
1. If $sim(x, y)$ is high $\rightarrow$ Probability of h(x) = h(y) is high
2. If $sim(x, y)$ is low $\rightarrow$ Probability of h(x) = h(y) is low

3.1 Methodology

A locality sensitive hash function has a collision probability that is related to the distance between elements. There are many families of similarities that we can use for sim(x,y), if a similarity measure has a corresponding LSH function h($\cdot$), then we can use it.

3.2 Jaccard Similarity Measure

Given two sets: S_1 and S_2, the Jaccard Similarity is

$$sim(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$$

Simple example:

$$S_1 = \{3, 10, 15, 19\}, S_2 = \{4, 10, 15\}$$
$$J = \frac{2}{5}$$

What about strings?
Suppose we have sets of characters or strings. Then we can use N-grams. A set of N-grams is the set of all contiguous n-character token in a string. We will use 3-grams for our application.
String iphone 6$\rightarrow$ {iph,pho,hon,one,ne,e 6}
What is Jaccard distance(3-gram)

- amazon vs anazon
  {ama, maz, aza, zon} vs {ana, naz, aza, zon} = $\frac{1}{3}$
  {$am, ama, maz, aza, zon, on.} vs {$an, ana, naz, aza, zon, on.} = $\frac{1}{3}$
- amazon vs amazom
  {ama, maz, aza, zon}vs {ama, maz, aza, zom}= $\frac{3}{5}$
  {$am, ama, maz, aza, zon, on.} vs {$am, ama, maz, aza, zom, om.}=$\frac{1}{2}$
- amazon vs random
  0

4 Minwise Hashing

4.1 Random Sampling with Universal Hashing

Question: Given the string "Amazon", the set of 3-grams "Ama","maz","azo","zon", and a random hash function U: string → [0-R], can we get a random element of the set?
Observe that:

$$Pr(h(s) = c) = \frac{1}{R}$$

To get a random alement of the set, we can perform random sampling using universal hashing. We hash every token using U and pick the token that has minimum or maximum hash value. For instance, {U(Ama),U(maz),U(azo),U(zon)}= {10,2005,199,2}. So our choice will be "zon".

4.2 Minwise Hashing

Suppose we have a set S with L elements: $S = \{s_1, s_2, ...s_L\}$. To compute the minwise hash value of S, we compute the set of random hash values U(*) of each element and we take the minimum value.

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

Properties:
1.Minwise Hashing can be applied to any set.
2.Pr(MinHash(S_1)=MinHash(S_2))=$\frac{|S_1 \cap S_2|}{|S_1 \cup S_2|}$. It means the collision property is equal to the Jaccard similarity.
Proof: Under randomness of hash function U.
Fact1: For any set, the element with minimum hash is a random sample. Consider set $|S_1 \cup S_2|$, and sample a random element e using U.
Claim1:$e \in |S_1 \cap S_2|$ if and only if $Minhash(S_1) = Minhash(S_2)$

4.3 Estimate Similarity Efficiently

Pr(MinHash(S_1)=MinHash(S_2))=$\frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = J$
Given 50 minhashes of S_1 and S_2. How can we estimate J?
Let us first assume the memory is 50 numbers.

$$x_i = \begin{cases} 1, & \text{if } Minhash_i(S_1) = Minhash_i(S_2) \\ 0, & \text{if } Minhash_i(S_1) \neq Minhash_i(S_2) \end{cases}$$

$$\hat{J} = \frac{1}{50} \sum_{i=1}^{5} 0x_i$$

$$E(\hat{J}) = J$$

$$Var(\hat{J}) = \frac{J(1-J)}{50}$$

When J =0.8, the standard deviation is roughly 0.05.