COMP 542 — Large Scale Machine Learning Jan 18, 2016

Lecture 5: Large-Scale Search: Locality Sensitive Hashing (LSH)

Lecturer: Anshumali Shrivastava *Scribe By: Jing Guo*

1 Large-scale image search problem

Nowadays, there exist hundreds of millions of images online. These images are either stored in web pages, or databases of companies, such as Facebook, Flickr, etc. It is challenging to quickly find similar images from these huge repositories. This is because:

- The repositories are huge. Facebook has around 10 billion images [2]. These images have different resolution, dimension.
- Images are very high-dimensional objects. Using pixel vectors to represent images is not enough, since the key objects in an image may rotate, shift, zoom in and zoom out, etc. All these alternations will make a huge impact on the pixel vector representation, however, these spurious transformations do not change the semantic similarity of the images.
- Comparing images is a very common and frequently required application, such as the image search in Google. The search query on the database is expected to be fast and accurate.

The image search problem is an example of classical near-neighbor search (or similarity search) problem: given a collection C and a similarity matrix. For a query q, find:

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

2 Limitations

Tree-based machine learning algorithm are popular in partitioning and clustering objects, see Figure 1. However, for high-dimensional objects, partitioning is not efficient. This is because of the infamous *curse of dimensionality*, space partitions grow exponentially with the dimensionality. For example, if an image has 10 features, and each feature is partitioned into say 2 groups. This representation will lead to 2^{10} partitions. Therefore, space partitioning methods are not efficient for image query.

3 Hashing algorithm

3.1 Detour: Motivation from a simple problem

Suppose we have an array of integers. We are asked to find whether the integer array contains a given integer. We can have three different algorithms to solve this simple problem.

- *O(N) Solution.* We traverse the integer array to compare each element with the given integer. The traversal costs O(N) time.

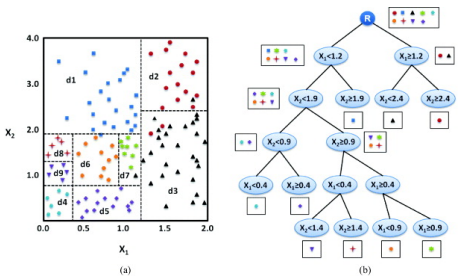

Figure 1: Example of using decision tree to partition. The figure are taken from [1].

- *O(logN) Solution.* To reduce time complexity, we can do some pre-processing on the integer array before the query. We sort the array. Therefore, we can do a binary search later on for the given integer. Although it costs time to sort the array during the pre-processing, it significantly increases the efficiency for the query.
- *O(1) solution* To further improve the efficiency, we can use hashmap during the pre-processing. Thus we will have a hash table to look up. Given a certain integer, it will cost O(1) time on average.

Back to the image search problem, we can also use hashing algorithms to improve the searching efficiency. However, different from traditional hashing, the hashing algorithm to be used in this situation is to compare the similarity between two images, rather than exact duplicates. This is because that two very similar images will have different hash values when using a traditional hashing algorithm, and it won't meet our similarity search requirements. We need some generalization.

3.2 Standard hashing

A hash function is a mapping that takes a vector x and maps it to a discrete key.

$$h : R^D \mapsto 1, 2, 3, 4, ..., k$$

For a standard hashing algorithm, such as the hash function used in Java's HashMap, it follows:

$$if \quad x = y, \quad then \quad h(x) = h(y)$$
$$if \quad x \neq y, \quad then \quad h(x) \neq h(y)$$

However, the standard hashing algorithm will not meet our requirement. We want to evaluate the similarity between two images using hashing. The standard hash function will return different values for two very similar but not identical images.

3.3 Locality sensitive hashing: Generalized Hashing

Locality sensitive hashing (LSH) is a hashing algorithm with probabilistic relaxation. LSH follows:

$$if \ \boldsymbol{Similarity(x, y)} \ is \ high, \ then \ the \ \boldsymbol{Probability(h(x) = h(y))} \ is \ high$$

if $\boldsymbol{Similarity(x, y)}$ *is low, then the* $\boldsymbol{Probability(h(x) = h(y))}$ *is low*

Different from traditional hashing algorithm, LSH maximizes the probability of two similar objects falling into the same bucket. By using LSH, we can pre-process the images database. In such way, we can reduce the dimensionality of image data. During an image query, instead of computing the similarity of two images, we can use LSH to compute the hash value to evaluate the similarity between images.

4 Formal LSH

A popular technique for efficient approximate near-neighbor search, uses the underlying theory of *Locality Sensitive Hashing* (LSH), which is a formalization of the above notion. A good reading is Chapter 3 in the book [3], online free copy at `http://www.langtoninfo.com/web_content/9781107015357_frontmatter.pdf`.

LSH is a family of functions, with the property that similar input objects in the domain of these functions have a higher probability of colliding in the range space than non-similar ones. Consider $\mathcal{H}$ a family of hash functions mapping $\mathbb{R}^D$ to a discrete set $[0, R-1]$.

Definition 1 Locality Sensitive Hashing (LSH) Family *A family* $\mathcal{H}$ *is called* (S_0, cS_0, p_1, p_2)*-sensitive if for any two point* $x, y \in \mathbb{R}^d$ *and* h *chosen uniformly from* $\mathcal{H}$ *satisfies the following:*

- *if* $Sim(x, y) \geq S_0$ *then* $Pr_{\mathcal{H}}(h(x) = h(y)) \geq p_1$
- *if* $Sim(x, y) \leq cS_0$ *then* $Pr_{\mathcal{H}}(h(x) = h(y)) \leq p_2$

5 Basic Idea for Similarity Search

We use the LSH in pre-processing the images. In the images repositories, we use LSH to compute the hash value for each image and organize them into hash tables. To increase the accuracy, we can repeat this process several times, and get multiple hash tables. When an image query comes, we use the same LSH algorithm to compute its hash value, and refer to the pre-processed hash tables to fetch the similar images.

Since LSH maximizes the probability of two similar images falling into the same bucket, the probability of the requested images falls into the bucket containing its similar images would be very high. Next lecture for more details.

References

[1] http://cdn.iopscience.com/images/1749-4699/5/1/015004/full/csd422281fig1.jpg.

[2] https://www.facebook.com/notes/facebook-engineering/10-billion-photos/30695603919.

[3] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets.* Cambridge University Press, 2014.