Lecture 1 Supplement: Locality Sensitive Hashing

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

October, 2018
Hash functions and hash tables

Locality Sensitive Hashing

Approximate $r$-neighbor retrieval by LSH

LSH by random projections

Reading: Lecture 16 notes by Moses Charikar, section 3.2
Hash functions and hash codes

Let the data space be $\mathbb{R}^n$, and assume some fixed probability measure on this space.

- A **family of hash functions** is a set $\mathcal{H} = \{ h : \mathbb{R}^n \to \{0, 1\} \}$ with the following properties
  1. For each $h$, $\Pr[h(x) = 1] \approx \frac{1}{2}$
  2. The binary random variables defined by the functions in $\mathcal{H}$ are mutually independent.
     (Or, if $\mathcal{H}$ is not finite, any finite random sample of such random variables is mutually
     independent.)

- Let $h_{1:k}$ be a mutually independent subset of $\mathcal{H}$. We call

$$g(x) = [h_1(x) \ h_2(x) \ldots \ h_k(x)] \in \{0, 1\}^k$$

the **hash code** of $x$.

- Note that the codes $g(x)$ are (approximately) uniformly distributed; the
  probability of any $g \in \{0, 1\}^k$ is about $\frac{1}{2^k}$.

- Useful hash functions must be fast to compute.
A hash table $\mathcal{T}$ is a data structure in which points in $\mathbb{R}^n$ can be stored in such a way that

1. All points with the same code $g$ are in the same bin denoted by $\mathcal{T}_g$. The table need not use space for empty bins.
2. Given any value $g \in \{0, 1\}^k$, we can obtain a point in $\mathcal{T}_g$ or find if $\mathcal{T}_g = \emptyset$ in constant time (independent of the number of points $N$ stored in $\mathcal{T}$).
   Some versions of hash tables return all points in $\mathcal{T}_g$, e.g., as a list, in constant time.
3. It is usually assumed that storing a point $x$ with given code $g(x)$ in a hash table is also constant time.

Hence, using a hash table to store an $x$ or to retrieve something, involves computing $k$ hash functions, then a constant-time access to $\mathcal{T}$.

When $x' \neq x$ and $g(x') = g(x)$ we call this a collision. In some applications (not of interest to us), collisions are to be avoided.
Locality Sensitive Hash Functions and Codes

- A hash function $h$ is **locality sensitive** iff for any $x, x' \in \mathbb{R}^n$
  
  \[ Pr[h(x) = h(x')] \geq p_1 \quad \text{when} \quad ||x - x'|| \leq r \]  
  \[ Pr[h(x) = h(x')] \leq p_2 \quad \text{when} \quad ||x - x'|| \geq cr \]  
  
  with $p_1, p_2, r$ and $c > 1$ fixed parameters (of the family $\mathcal{H}$) and $p_1 > p_2$.

- W.l.o.g., we set $p_1 = p_2^\rho$ for some $\rho < 1$.

- A locality sensitive $h$ makes a weak distinction between points that are close in space vs. points that are far away. A hash code $g$ from locality sensitive hash functions sharpens this distinction, in the sense that the probability of far away points colliding can be made arbitrarily small.
  
  \[ p_{bad} = Pr[g(x) = g(x') \mid ||x - x'|| > cr] \leq p_2^k \]  

- Assume $x$ is not in $\mathcal{T}$; for any $x' \in \mathcal{D}$ which is far from $x$, the probability that $x'$ collides with $x$ is $\leq p_{bad}$.

- We construct $\mathcal{T}$ so that $p_{bad} \leq \frac{1}{N}$ for $N$ the sample size. For this we need Exercise (in Homework 1)
  
  \[ k = \frac{\ln N}{-\ln p_2} \Rightarrow p_{bad} \leq \frac{1}{N} \]  

- Suppose $x' \in \mathcal{T}$ is “close” to $x$. What is the probability that $g(x') = g(x)$?
  
  \[ p_{good} = p_1^k = p_2^{\rho k} = \frac{1}{N^\rho} \]  

  This is the probability that the bin $\mathcal{T}_g(x)$ contains $x'$. 
Approximate $r$-neighbor retrieval by LSH

**Input** $\mathcal{D}$ set of $N$ points, $L$ mutually independent hash codes $g_{1:L}$ of dimension $k$.

**Indexing** Construct $L$ hash tables $\mathcal{T}^{1:L}$, each storing $\mathcal{D}$.

**Retrieval** Given $x$

1. compute $g(x)$
2. for $j = 1, 2, \ldots, L$
   if the bin $\mathcal{T}_{g(x)}^j \neq \emptyset$
     2.1 return some (all) $x'$ from it.
     2.2 stop if a single neighbor is wanted.

Some analysis. We set $L = N^\rho$

- **Indexing time** $\propto kN^{\rho+1}$
- **Retrieval time** $\propto kN^\rho$
- **Space used** $\propto kN^{\rho+1}$

- For each $x' \in \mathcal{D}$ close to $x$, the probability that $x'$ is **NOT** returned for any $j \in 1:L$ is
  \[
  (1 - \frac{1}{N^\rho})^{N^\rho} \approx \frac{1}{e}
  \]  
  This can be made arbitrarily small by multiplying $L$ with a constant.

- For each $x' \in \mathcal{D}$ far from $x$, the probability that $x'$ is **NOT** returned for any $j \in 1:L$ is
  \[
  (1 - \frac{1}{N})^{N^\rho} \approx \left(\frac{1}{e}\right)^{1/N^{1-\rho}} \approx \frac{1}{e^0} = 1
  \]  

- Hence, we are almost sure not to return a far point, and have a significant probability to return a close point when one exists, if no points neither far nor close are in the data. This is why this algorithm is **approximate**: it may also return points with $r < ||x' - x|| \leq cr$. 

How do we find **good** hash functions?

- We need large families of $h$ functions
- that are easy to generate randomly
- and fast to compute for a given $x$

- Generic method to obtain them: *random projections*
Projecting on a random vector

- Data are $x \in \mathbb{R}^n$ as usual.
- Define $h_{a,b} : \mathbb{R}^n \to \mathbb{Z}$ by
  \begin{equation}
  h_{a,b}(x) = \left\lfloor \frac{a^T x + b}{w} \right\rfloor
  \end{equation}
  with $w > 0$ a width parameter, $a \in \mathbb{R}^n$, $b \in [0, w)$.
- Intuitively, $x$ is "projected" on $a^1$, then the result is quantized into bins of width $w$, with a grid origin given by $b$.

- The family of hash functions is $\mathcal{H}_w = \{ h_{a,b} : a \in \mathbb{R}^n, b \in [0, w) \}$.
- Sampling $\mathcal{H}_w$: $a \sim \text{Normal}(0, I_n)$, $b \sim \text{uniform}[0, w)$.
  - Because the Normal distribution is a stable distribution, this ensures that $a^T x$ is distributed as $\text{Normal}(0, ||x||^2)$. Exercise Verify this
  - Hence $a^T x - a^T x'$ is distributed as $\text{Normal}(0, ||x - x'||^2)$. Exercise Verify this
  - Moreover, if hash functions are sampled independently from $\mathcal{H}_w$, (and nothing is known about $x$) then $h_{a,b}(x), h_{a',b'}(x)$ are independent random variables. Exercise Prove this

- This type of hash functions are being widely used by approximate neighbor search algorithms.

\footnote{1a is not necessarily unit length}