ECS289: Scalable Machine Learning

Cho-Jui Hsieh
UC Davis

Oct 25, 2016
Outline

- Nearest Neighbor Search
- Maximum Inner Product Search
K-nearest neighbor classification

Predict the label by voting among K-nearest neighbors.

Prediction time: Find the nearest training sample

1 billion samples, each distance evaluation requires 1 micro second

⇒ 1000 secs per prediction
Nearest Neighbor Search

- Given a database with $n$ vectors $\{x_1, \ldots, x_n \in \mathbb{R}^d\}$
- Given a query point $v$, how to find the closest points in the database?
- Linear search:
  - Compute $\|x_i - v\|$ for all $i$
  - Need $O(nd)$ time
- Can we do better?
Tree-based Nearest Neighbor Search
Tree-based Approaches

- KD Tree (Bentley, 1975)

Preprocessing:
- Divide the points in half by a line perpendicular to one of the axes
- Stop when there is only one or few points in a leaf

(Picture from http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf)
KD-tree: Construction

- Sort the data points in each dimension
- Requires $O(dn)$ time to find the best split in each level
  (Scan over the sorted list for each dimension)
- Balanced construction: $O(\log n)$ levels
- Total time: $O(dn \log n)$
KD-tree: Query

- Recursively traverse down the tree.
- Assume the current nearest distance is $r$ and
  \[ |\text{split}_{-}\text{value} - x.\text{split}_{-}\text{index}| < r \]
  \[ \Rightarrow \text{Only go down to the current branch} \]
- Otherwise, need to check both branches.
- May need to check $O(n)$ nodes in high dimensional cases
PCP Tree: Handling high dimensional case

- Partition by PCA directions (not perpendicular to axes)
- Work much better on real data.

(Picture from http://andrewd.ces.clemson.edu/courses/cpsc805/references/nearest_search.pdf)
Locality Sensitive Hashing
Locality Sensitive Hashing (LSH)

Main Idea:
- Randomly hash data to a small number of bins
- Nearest points will have high probability to lie in the same bin
- For a query point, search for points only within the same bin
**Definition of LSH**

- Let $\mathcal{F}$ be a family of functions that map elements from input space to bucket $\{1, 2, \ldots, s\}$.
- $\mathcal{F}$ is an LSH family: if a hash function $h$ chosen uniformly at random from $\mathcal{F}$:
  - If $d(p, q) \leq R$, then $h(p) = h(q)$ with probability $\geq P_1$,
  - If $d(p, q) \geq cR$, then $h(p) = h(q)$ with probability $\leq P_2$.
- Such family $\mathcal{F}$ is called $O(R, cR, P_1, P_2)$-sensitive ($P_1 > P_2$).
LSH-amplification

- Amplification:
  - AND operation of $k$ functions:
    \[
    \text{AND}_{h_1, \ldots, h_k}(p) = \text{AND}_{h_1, \ldots, h_k}(q) \iff h_i(p) = h_i(q), \ \forall i = 1, \ldots, k
    \]
    \[
    (R, cR, P_1, P_2)\text{-LSH} \Rightarrow (R, cR, P_1^k, P_2^k)\text{-LSH}
    \]
  - OR operation of $m$ functions:
    \[
    \text{OR}_{h_1, \ldots, h_k}(p) = \text{OR}_{h_1, \ldots, h_k}(q) \iff h_i(p) = h_i(q) \text{ for some } i
    \]
    \[
    (R, cR, P_1, P_2)\text{-LSH} \Rightarrow (R, cR, 1 - (1 - P_1)^m, 1 - (1 - P_2)^m)\text{-LSH}
    \]
  - Combined together:
    \[
    (R, cR, P_1, P_2) - \text{LSH} \Rightarrow (R, cR, 1 - (1 - P_1^k)^m, 1 - (1 - P_2^k)^m) - \text{LSH}
    \]
  - Example:
    \[
    (0.8, 0.5) \Rightarrow (1 - (1 - 0.8^8)^{15}, 1 - (1 - 0.5^8)^{15}) = (0.93, 0.057)
    \]
LSH: Hamming distance

- Each point $x = \{0, 1\}^d$
- Hash function: sample a bit and map to 0/1
LSH: Hamming distance

- For each $h_i$, sample $k$ elements and map to $2^k$ bins
- OR-function of $h_1, \ldots, h_m$
- Two points “collide” if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query

Query: 0 0 1 0 1 0 0 0 0

- $h1$ (bit 2, 4)
- $h2$ (bit 3, 5)
LSH: Hamming distance

- For each $h_i$, sample $k$ elements and map to $2^k$ bins
- OR-function of $h_1, \ldots, h_m$
- Two points “collide” if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query
LSH: Hamming distance

- For each $h_i$, sample $k$ elements and map to $2^k$ bins
- OR-function of $h_1, \ldots, h_m$
- Two points “collide” if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query
LSH: Hamming distance

- For each $h_i$, sample $k$ elements and map to $2^k$ bins
- OR-function of $h_1, \ldots, h_m$
- Two points “collide” if $h_i(p) = h_i(q)$ for some $i = 1, \ldots, m$
- Linear search on all the points that collide with the query
LSH: Random Projections

- Hash function that preserves Euclidean distance:

  \[ h_{u,b}(x) = \left\lfloor \frac{u^T x + b}{w} \right\rfloor \]

  where \( \lfloor \cdot \rfloor \) is the floor operation, \( w \) is the width of a bucket, \( u \) sampled from \( N(0, 1) \), \( b \) sampled from \( (0, w) \)

- Amplified by AND/OR operations.
**Summary**

<table>
<thead>
<tr>
<th></th>
<th>Query Time</th>
<th>Space Used</th>
<th>Perprocessing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>KD-Tree</td>
<td>$O(2^d \log n)$</td>
<td>$O(n)$</td>
<td>$O(nd \log n)$</td>
</tr>
<tr>
<td>LSH</td>
<td>$O(kmd + \text{linear search})$</td>
<td>$O(mkn)$</td>
<td>$O(mknd)$</td>
</tr>
</tbody>
</table>
Maximum Inner Product Search (MIPS)
Matrix factorization models for Recommender systems:

- $u_i$: latent features for user $i = 1, \ldots, m$
- $v_j$: latent features for item $j = 1, \ldots, n$
- $u_i^T v_j$: approximate the rating $R_{ij}$

Prediction: Recommend an item to user 1
- Select $j^* = \arg\max_{j=1,\ldots,n} u_1^T v_j$
- Linear search: $O(nk)$ time

Similar application in multiclass classification.
Maximum Inner Product Search (MIPS)

- Given a database with \( n \) candidate vectors
  \( \mathcal{X} := \{ x_i \in \mathbb{R}^d, \ i = 1, \ldots, n \} \).
- Given a query vector \( v \), find
  \[
  \arg \max_{x_i \in \mathcal{X}} v^T x_i
  \]
- Linear search: \( O(nd) \) time
- Can we reduce the query time?
Maximum Inner Product Search (MIPS)

- Cannot directly use LSH for Euclidean distance
  \[ \|v - x_i\|^2 = \|v\|^2 + \|x_i\|^2 - 2v^T x_i \]
  \[ \|v\|^2 \] is fixed for a query point, but \[ \|x_i\|^2 \] can vary
- Nearest neighbor search: return
  \[ \arg \max_{x_i} 2v^T x_i - \|x_i\|^2 \]
Maximum Inner Product Search (MIPS)

- Cannot directly use LSH for Euclidean distance
  - \( \|v - x_i\|^2 = \|v\|^2 + \|x_i\|^2 - 2v^T x_i \)
  - \( \|v\|^2 \) is fixed for a query point, but \( \|x_i\|^2 \) can vary
- Nearest neighbor search: return
  \[
  \arg\max_{x_i} 2v^T x_i - \|x_i\|^2
  \]
- Assymetric reduction to NN: define the mappings \( P(\cdot), Q(\cdot) \) such that
  \[
  \|P(x_i) - Q(v)\|^2 \text{ proportional to } -x_i^T v
  \]
  We can then find MIPS by
  \[
  \arg\min_{x_i} \|P(x_i) - Q(v)\|^2
  \]
Reduction to NN: L2-ALSH

- Proposed in (Shrivastava and Li, “Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS)”). NIPS 2014
- Define

\[ P(x) = [x; \|x\|^2; \|x\|^4; \cdots; \|x\|^{2m}] \quad Q(v) = [v; \frac{1}{2}; \frac{1}{2}; \cdots; \frac{1}{2}] \]

So we have

\[ \|P(x_i) - Q(v)\|^2 = -2v^T x_i + \|v\|^2 + \frac{m}{4} + \|x_i\|^{2m+1} \]

- By normalizing the database, we can make \( \max_i \|x_i\| < 1 \)

\[ \|x_i\|^{2m+1} \rightarrow 0 \text{ when } m \rightarrow \infty \]

- Using this mapping, MIPS \( \approx \) NN-search when \( m \) is large

(applying LSH on \( P(x_1), \cdots, P(x_n) \) and query \( Q(v) \))
SIMPLE-LSH (discussed in (Neyshabur and Srebro, “On Symmetric and Asymmetric LSHs for Inner Product Search”. ICML 2015))

Original proposed in (Bachrach et al., “Speeding Up the Xbox Recommender System Using a Euclidean Transformation for Inner-Product Spaces”. Recsys, 2014.)

A simple way to define \( P(\cdot) \) and \( Q(\cdot) \):

\[
P(x) = [x; \sqrt{M - \|x\|^2}] \quad Q(v) = [v; 0]
\]

where \( M = \max_i \|x_i\|^2 \)

Easy to see \( \|P(x_i) - Q(v)\|^2 = M + \|v\|^2 - 2x_i^T v \)

LSH with \( P(x_1), \ldots, P(x_n) \) and query \( Q(v) \)
Some Comparisons

- Reduction to NN + PCA-tree (Bachrach et al., 2014).

(Figure from Bachrach et al., 2014)
Assume $\mathbf{v}$ and $\{x_1, \cdots, x_n\}$ are all nonnegative.

Sampling approach: sample according to the probability

$$P(i) \sim x_i^T \mathbf{v} = \sum_t X_{it} v_t$$

We can sample by the joint distribution

$$P(i, t) \sim v_t X_{it} = v_t X_{it} / \left( \sum_i \sum_t X_{it} v_t \right)$$

$$= \frac{v_t \left( \sum_i X_{it} \right) X_{it}}{\sum_t v_t \left( \sum_i X_{it} \right) \sum_i X_{it}}$$

$$= P_{\mathbf{v}}(t) P_t(i)$$

And then $P(i) = \sum_t P(i, t)$. 
Directly Solving MIPS: Sampling Approach

- **Sampling MIPS: pre-processing**
  - Compute $R_i = \sum_i X_{it}$
  - Construct distributions $P_t(i) \sim \frac{X_{it}}{\sum_i X_{it}}$ for all $t$.

- **Query: (normalize $\boldsymbol{v}$)**
  - Construct distribution $P_v(t) \sim v_t R_t$
  - Sample many $(i, t)$ pairs by (1) sampling $t$ by $P_v(t)$ (2) sampling $j$ by $P_t(i)$
  - Output the top-$T$ sampled index $i$.

- What’s the time complexity?
Proposed in (Ballard et al., “Diamond Sampling for Approximate Maximum All-pairs Dot-product (MAD) Search”. ICDM 2015.)

Sampling approach for MAD problem:

$$\max_{i,j} |x_i^T x_j|$$

Can handle negative values in $x_i, x_j$. 
Fast Sampling Algorithms
Sampling from a given distribution

- Sampling from a discrete probability distribution with $n$ indices with distribution $p_1, p_2, \ldots, p_n$
  
  (Assume this is normalized, so $\sum_i p_i = 1$)

- What’s the time complexity for generating one sample?
Linear Search

- Randomly generate a number $s \in [0, 1]$ uniformly
- Constant Time
- Construct a cummulated distribution

\[ c_0 = 0, \quad c_1 = c_0 + p_1, \quad c_2 = c_1 + p_2, \ldots, \quad c_n = c_{n-1} + p_n \]

- Find the index $i$ such that $s \in [c_{i-1}, c_i]$.
- $O(n)$ time
Binary Search

- **Preprocessing:**
  
  Construct cummulated distribution $c_0, c_1, \ldots, c_n$

  $O(n)$

- **Sampling:**
  
  Randomly generate a number $s \in [0, 1]$ uniformly

  Binary search to find $i$ such that $s \in [c_{i-1}, c_i]$

  $O(\log n)$ time
Alias Method

- **Preprocessing:**
  - Create an Alias Table
  - $O(n)$ time (Vose’s algorithm)

- **Sampling:**
  - Generate a uniform random number in $\{1, 2, \ldots, n\}$
  - Generate another random number to determine the index
  - *Constant time!*
Alias Method
Alias Method
Alias Method
Alias Method
Alias Method
**F+ Tree**

- Sampling from distribution $p_1, \ldots, p_n$
- But one of $p_i$ will change at each time (or once every few iterations)
- Alias Table does not work (need to reconstruct the whole table)
- F+ Tree (Fenwick Tree)
  - Sampling: $O(\log n)$ time
  - Update: $O(\log n)$ time
### F+ Tree

#### Data Structure Space

<table>
<thead>
<tr>
<th>Method</th>
<th>$c_T = p^T 1$:</th>
<th>Initialization Time</th>
<th>Generation Time</th>
<th>Parameter Update Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSearch</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>BSearch</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>Alias Method</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
</tr>
<tr>
<td>F+tree Sampling</td>
<td>$O(T)$</td>
<td>$O(1)$</td>
<td>$O(T)$</td>
<td>$O(T)$</td>
</tr>
</tbody>
</table>

#### Diagrams

(a) F+tree for $p = [0.3, 1.5, 0.4, 0.3]^T$

(b) Sampling

(c) Updating (with $\delta = 1.0$)
Questions?