Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Hash Kernels

- Count-Min sketch not designed for negative updates
- Biased estimates of dot products
- **Hash Kernels**: Very simple, but powerful idea to remove bias
  - Pick 2 hash functions:
    - $h$: Just like in Count-Min hashing
    - $\xi$: Sign hash function
      - Removes the bias found in Count-Min hashing (see homework)
  - Define a “kernel”, a projection $\phi$ for $x$:
    \[
    \phi_i = \sum_{j: h(j) = i} \xi(j) a_j
    \]
Hash Kernels Preserve Dot Products

\[ \phi_i(x) = \sum_{j: h(j) = i} \xi(j)x_j \]

• Hash kernels provide unbiased estimate of dot-products!

\[ \mathbb{E}_{h, \xi} \left[ \phi(x) \cdot \phi(y) \right] = x \cdot y \]

• Variance decreases as \( O(1/m) \)

• Choosing \( m \)? For \( \varepsilon > 0 \), if

\[
m = \mathcal{O} \left( \frac{\log N}{\varepsilon} \right)
\]

– Under certain conditions...
– Then, with probability at least 1-\( \delta \):

\[
(1 - \varepsilon)\|x - x'\|^2_2 \leq \|\phi(x) - \phi(x')\|^2_2 \leq (1 + \varepsilon)\|x - x'\|^2_2
\]

Interesting Application of Hash Kernels: Multi-Task Learning

• Personalized click estimation for many users:
  – One global click prediction vector \( w \):
    • But...
    – A click prediction vector \( w_u \) per user \( u \):
      • But...

• Multi-task learning: Simultaneously solve multiple learning related problems:
  – Use information from one learning problem to inform the others

• In our simple example, learn both a global \( w \) and one \( w_u \) per user:
  – Prediction for user \( u \):
    \( (w + w_u) \cdot x \approx w \cdot x + w_u \cdot x \)
    • If we know little about user \( u \):
    • After a lot of data from user \( u \):
Hash Kernels for Multi-Task Learning

- Simple, pretty solution with hash kernels:
  - Very multi-task learning as (sparse) learning problem with (huge) joint data point $z$ for point $x$ and user $u$:

$$Z_{x,u} = (x_1, \ldots, x_d, 0, 0, \ldots, x_i, 0, \ldots, 0, \delta)$$

- Estimating click probability as desired:

$$w = (w_1, w_2, \ldots, w_n, \ldots)$$

- Address huge dimensionality, new words, and new users using hash kernels:

Simple Trick for Forming Projection $\phi(x,u)$

- Observe data point $x$ for user $u$:
  - Dimension does not need to be specified a priori and user can be new!

- Compute $\phi(x,u)$:
  - Initialize $\phi(x,u) = 0$
  - For non-zero entries $j$ of $x$:
    - E.g., $j$='Obamacare'
    - Need two contributions to $\phi$:
      - Global contribution
      - Personalized Contribution
    - Simply:

- Learn as usual using $\phi(x,u)$ instead of $\phi(x)$ in update function
What you need to know

- Hash functions
- Bloom filter
  - Test membership with some false positives, but very small number of bits per element
- Count-Min sketch
  - Positive counts: upper bound with nice rates of convergence
  - General case
- Application to logistic regression
- Hash kernels:
  - Sparse representation for feature vectors
  - Very simple, use two hash function (Can use one hash function...take least significant bit to define $\xi$)
  - Quickly generate projection $\varphi(x)$
  - Learn in projected space
- Multi-task learning:
  - Solve many related learning problems simultaneously
  - Very easy to implement with hash kernels
  - Significantly improve accuracy in some problems (if there is enough data from individual users)

Case Study 2: Document Retrieval

Task Description:
Finding Similar Documents
Document Retrieval

- **Goal:**Retrieve documents of interest

- **Challenges:**
  - Tons of articles out there
  - How should we measure similarity?

Task 1: Find Similar Documents

- **To begin...**
  - **Input:** Query article
  - **Output:** Set of $k$ similar articles
Document Representation

- Bag of words model

\[ X = \left[ \frac{w_{c_1}}{w_{c_2}} \ldots \frac{w_{c_i}}{w_{c_j}} \right] \in \mathbb{R}^d \]

ignore order of words

1-Nearest Neighbor

- Articles

\[ X = \{ x^1, \ldots, x^n \} \in \mathbb{R}^d \]

- Query:

\[ \times \]

- 1-NN

□ Goal: find article in X "closest" to \( x \)

□ Formulation:

\[ x_{NN} = \text{arg min}_x d(x; x_{\text{query}}) \]
**k-Nearest Neighbor**

- **Articles** \( X = \{x^1, \ldots, x^N\}, \quad x^i \in \mathbb{R}^d \)
- **Query:** \( x \in \mathbb{R}^d \)

**k-NN**

- **Goal:** Find \( k \) articles in \( X \) closest to \( x \)

- **Formulation:**
  \[
  X_{\text{NN}} = \{x_1^* \ldots x_k^*\} \quad \text{s.t.} \quad \forall x \notin X_{\text{NN}}
  \]

---

**Distance Metrics – Euclidean**

\[
d(u, v) = \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2}
\]

Or, more generally,

\[
d(u, v) = \sqrt{\sum_{i=1}^{d} \sigma_i^2(u_i - v_i)^2}
\]

Equivalently,

\[
d(u, v) = \sqrt{(u - v)^\top \Sigma (u - v)}
\]

where

\[
\Sigma = \begin{bmatrix}
\sigma_1^2 & 0 & \cdots & 0 \\
0 & \sigma_2^2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_d^2
\end{bmatrix}
\]

**Other Metrics...**

- Mahalanobis, Rank-based, Correlation-based, cosine similarity...
Notable Distance Metrics
(and their level sets)

- **L₁ norm (absolute)**
- **L₂ norm (scaled Euclidean)**
- **Mahalanobis**

(Σ is general symmetric positive definite matrix, on previous slide = diagonal)

Euclidean Distance + Document Retrieval

- Recall distance metric
  \[ d(u, v) = \sqrt{\sum_{i=1}^{d}(u_i - v_i)^2} \]

- What if each document were \( \alpha \) times longer?
  - Scale word count vectors
    \[ u, v \in \mathbb{R}^d \]
    \[ \|u\|_2 = 1, \|v\|_2 = 1 \]
  - What happens to measure of similarity?

- Good to normalize vectors
  \[ \|u\|_2 = 1, \|v\|_2 = 1 \]
Issues with Document Representation

- Words counts are **bad** for standard similarity metrics

  ![Image of The Panama Papers]

- Term Frequency – Inverse Document Frequency (tf-idf)
  - Increase importance of rare words

TF-IDF

- Term frequency:
  \[
  \text{tf}(t, d) = \frac{\text{# of occur. of } t \text{ in a doc. } d}{\text{document}}
  \]
  - Could also use \(0, 1, 1 + \log f(t, d), \ldots\)
- Inverse document frequency:
  \[
  \text{idf}(t, D) = \log \frac{|X|}{1 + |\{d \in X : t \text{ in } d\}|}
  \]
- tf-idf:
  \[
  \text{tfidf}(t, d, D) = \text{tf}(t, d) \times \text{idf}(t, D)
  \]
  - High for document \(d\) with high frequency of term \(t\) (high “term frequency”) and few documents containing term \(t\) in the corpus (high “inverse doc frequency”)

© Sham Kakade 2016
Issues with Search Techniques

- Naïve approach:
  - **Brute force search**
    - Given a query point $x^*$
    - Scan through each point $x_i$
    - $O(N)$ distance computations per 1-NN query!
    - $O(N \log k)$ per $k$-NN query!

- What if $N$ is huge???
  (and many queries)

Think about Web Search/Image Search

- How big is $N$?
  \[
  N = \text{# web pages} = \text{# images on web.}
  \]

- How fast do we desire to do recall?
Intuition (?): NN in 1D and Sorting

- How do we do 1-NN searches in 1 dim?

- Pre-processing time:
  \[ O(N \log N) \]

- Query time:
  \[ O(\log N) \text{ (for 1NN)} \]

---

KD-Trees

- Smarter approach: **kd-trees**
  - Structured organization of documents
    - Recursively partitions points into axis aligned boxes.
  - Enables more efficient pruning of search space
    - Examine nearby points first.
    - Ignore any points that are further than the nearest point found so far.

- *kd-trees* work “well” in “low-medium” dimensions
  - We’ll get back to this...
KD-Tree Construction

- Start with a list of $d$-dimensional points.

<table>
<thead>
<tr>
<th>Pt</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>4.31</td>
</tr>
<tr>
<td>3</td>
<td>0.13</td>
<td>2.85</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Split the points into 2 groups by:
  - Choosing dimension $d_j$ and value $V$ (methods to be discussed...)
  - Separating the points into $x_{d_j} > V$ and $x_{d_j} \leq V$. 

©Sham Kakade 2016
Consider each group separately and possibly split again (along same/different dimension).

- Stopping criterion to be discussed...
Continue splitting points in each set
- creates a binary tree structure
- Each leaf node contains a list of points

Keep one additional piece of information at each node:
- The (tight) bounds of the points at or below this node.
KD-Tree Construction

- Use heuristics to make splitting decisions:
- Which dimension do we split along?
- Which value do we split at?
- When do we stop?

Many heuristics...

- median heuristic
- center-of-range heuristic
**Nearest Neighbor with KD Trees**

- Traverse the tree looking for the nearest neighbor of the query point.

---

**Examine nearby points first:**
- Explore branch of tree closest to the query point first.
Examine nearby points first:
- Explore branch of tree closest to the query point first.

When we reach a leaf node:
- Compute the distance to each point in the node.
When we reach a leaf node:

- Compute the distance to each point in the node.

Then backtrack and try the other branch at each node visited.
Each time a new closest node is found, update the distance bound.

Using the distance bound and bounding box of each node:
- Prune parts of the tree that could NOT include the nearest neighbor.
Nearest Neighbor with KD Trees

- Using the distance bound and bounding box of each node:
  - Prune parts of the tree that could NOT include the nearest neighbor
Complexity

- For (nearly) balanced, binary trees...
- Construction
  - Size: \( O(N) \)
  - Depth: \( O(\log N) \) (under some assumptions)
  - Median + send points left-right: \( O(N \log N) \)
  - Construction time: \( O(N \log N) \)
- 1-NN query
  - Traverse down tree to starting point: \( O(\log N) \)
  - Maximum backtrack and traverse: \( O(N) \)
  - Complexity range: \( O(\log N) \) to \( O(N) \)

Under some assumptions on distribution of points, we get \( O(\log N) \) but exponential in \( d \) (see citations in reading)
Complexity for $N$ Queries

- Ask for nearest neighbor to each document
- Brute force 1-NN: $O(N^2)$
- kd-trees:

$$O(N^2) \rightarrow O(N \log N)$$

(if each query was $O(dN)$ instead)

---

Inspections vs. $N$ and $d$

- $O(N)$

- $O(N \log N)$ for $d$-trees

- Great for low dimensions.
Exactly the same algorithm, but maintain distance as distance to furthest of current \( k \) nearest neighbors

- Complexity is:
  \[ O(k \log n) \]

Before: Prune when distance to bounding box >  
Now: Prune when distance to bounding box >  
Will prune more than allowed, but can guarantee that if we return a neighbor at distance \( \gamma \), then there is no neighbor closer than \( r/\alpha \).

In practice this bound is loose...Can be closer to optimal.

Saves lots of search time at little cost in quality of nearest neighbor.
Cover trees (+ ball trees)

- What about exact NNs searches in high dimensions?
- Idea: utilize triangle inequality of metric (so allow for arbitrary metric)
- cover-tree guarantees:

Cover trees: what does the triangle inequality imply?
Cover trees: data structure

Wrapping Up – Important Points

**kd-trees**
- Tons of variants
  - On construction of trees (heuristics for splitting, stopping, representing branches...)
  - Other representational data structures for fast NN search (e.g., cover trees, ball trees,...)

**Nearest Neighbor Search**
- Distance metric and data representation are crucial to answer returned

**For both...**
- High dimensional spaces are hard!
  - Number of kd-tree searches can be exponential in dimension
    - Rule of thumb... $N \gg 2^d$... Typically useless.
  - Distances are sensitive to irrelevant features
    - Most dimensions are just noise $\Rightarrow$ Everything equidistant (i.e., everything is far away)
    - Need technique to learn what features are important for your task
What you need to know

- Document retrieval task
  - Document representation (bag of words)
  - tf-idf
- Nearest neighbor search
  - Formulation
  - Different distance metrics and sensitivity to choice
  - Challenges with large $N$
- $kd$-trees for nearest neighbor search
  - Construction of tree
  - NN search algorithm using tree
  - Complexity of construction and query
  - Challenges with large $d$

Acknowledgment

- This lecture contains some material from Andrew Moore’s excellent collection of ML tutorials:
  - [http://www.cs.cmu.edu/~awm/tutorials](http://www.cs.cmu.edu/~awm/tutorials)
- In particular, see:
  - [http://grist.caltech.edu/sc4devo/.../files/sc4devo_scalable_datamining.ppt](http://grist.caltech.edu/sc4devo/.../files/sc4devo_scalable_datamining.ppt)