Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Sham Kakade
April 12th, 2015
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 \( x \):

\[
\phi \cdot = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}
\]

\[
\phi_i = \sum_j \xi(j) \alpha_j
\]
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!

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

- Variance decreases as \( O(1/m) \)

- Choosing \( m \)? For \( \epsilon > 0 \), if

\[ m = \mathcal{O}\left(\frac{\log \frac{N}{\delta}}{\epsilon^2}\right) \]

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

\[ (1 - \epsilon)\|x - x'\|_2^2 \leq \|\phi(x) - \phi(x')\|_2^2 \leq (1 + \epsilon)\|x - x'\|_2^2 \]
Interesting Application of Hash Kernels: Multi-Task Learning

• Personalized click estimation for many users:
  – One global click prediction vector \( \mathbf{w} \):
    
    \[
    \frac{e^{\mathbf{w} \cdot \mathbf{x}}}{1 + e^{\mathbf{w} \cdot \mathbf{x}}}
    \]

    • But...
  – A click prediction vector \( \mathbf{w}_u \) per user \( u \):
    
    \[
    \frac{e^{\mathbf{w}_u \cdot \mathbf{x}}}{1 + e^{\mathbf{w}_u \cdot \mathbf{x}}}
    \]

    • 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 \( \mathbf{w} \) and one \( \mathbf{w}_u \) per user:
  – Prediction for user \( u \):
    
    \[
    (\mathbf{w} + \mathbf{w}_u) \cdot \mathbf{x} = \mathbf{w} \cdot \mathbf{x} + \mathbf{w}_u \cdot \mathbf{x}
    \]

    – If we know little about user \( u \):
      
      \[
      \mathbf{w} \cdot \mathbf{x}
      \]

    – After a lot of data from user \( u \):
      
      \[
      \mathbf{w}_u \neq 0 \quad \text{get user-specific}
      \]

©Sham Kakade 2016
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 $u$:

$$Z(x, u) = (x, \ldots, x_d, 0, 0, \ldots, x, \ldots, x_d, 0, 0, \ldots, 3)
\quad \text{user } u
\quad \text{dim } d$$

• Estimating click probability as desired:

$$w = (w, \overline{w}, \ldots, \overline{w}_u, \ldots, \overline{w})$$

$\overline{w}$ is the weight vector for $z$.

• 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 \( x_j \):
    - E.g., \( j = \text{‘Obamacare’} \)
    - 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
1-Nearest Neighbor

- Articles

- Query:

- 1-NN
  - Goal:

- Formulation:
**k-Nearest Neighbor**

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

- Query: \( x \in \mathbb{R}^d \)

- **k-NN**
  - Goal:

- Formulation:
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,

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

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)

- **L_1 norm (absolute)**
- **L_∞ (max) norm**
- **Scaled Euclidian (L_2)**
- **Mahalanobis**
  
  (Σ is general sym pos def 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
  - What happens to measure of similarity?

- Good to normalize vectors
Issues with Document Representation

- Words counts are bad for standard similarity metrics

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

- Term frequency:

\[ tf(t, d) = \]

- Could also use \{0, 1\}, \[1 + \log f(t, d)\], …

- Inverse document frequency:

\[ idf(t, D) = \]

- \(tf-idf\):

\[ tfidf(t, d, D) = \]

- 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”)
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)

©Sham Kakade 2016
Think about Web Search/Image Search

- How big is $N$?
- 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:

- Query time:
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...
Start with a list of $d$-dimensional points.
Split the points into 2 groups by:

- Choosing dimension \( d_j \) and value \( V \) (methods to be discussed...)
- Separating the points into \( x^{i}_{d_j} > V \) and \( x^{i}_{d_j} \leq V \).
Consider each group separately and possibly split again (along same/different dimension).

- Stopping criterion to be discussed...
Consider each group separately and possibly split again (along same/different dimension).

- Stopping criterion to be discussed...
KD-Tree Construction

- 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
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
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:
- Depth:
- Median + send points left right:
- Construction time:

1-NN query
- Traverse down tree to starting point:
- Maximum backtrack and traverse:
- Complexity range:

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:
- kd-trees:
Inspections vs. $N$ and $d$
Exactly the same algorithm, but maintain distance as distance to furthest of current \( k \) nearest neighbors

Complexity is:
Approximate K-NN with KD Trees

- **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 $r$, 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 → 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)