Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
April 9th, 2015

Problem 1: Complexity of LR Updates

- Logistic regression update:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)})] \right\} \]

- Complexity of updates:
  - Constant in number of data points
  - In number of features?
    - Problem both in terms of computational complexity and sample complexity
  - What can we with very high dimensional feature spaces?
    - Kernels not always appropriate, or scalable
    - What else?

©Emily Fox 2015
Problem 2: Unknown Number of Features

- For example, bag-of-words features for text data:
  - “Mary had a little lamb, little lamb…”

- What’s the dimensionality of \( x \)?
- What if we see new word that was not in our vocabulary?
  - Obamacare
    - Theoretically, just keep going in your learning, and initialize \( w_{\text{Obama}} = 0 \)
    - In practice, need to re-allocate memory, fix indices,... A big problem for Big Data

Count-Min Sketch: general case

- Keep \( p \) by \( m \) Count matrix

- \( p \) hash functions:
  - Just like in Bloom Filter, decrease errors with multiple hashes
  - Every time see string \( i \):
    \[
    \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + 1
    \]
Querying the Count-Min Sketch

\[ \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + 1 \]

• Query Q(i)?
  – What is in Count[j,k]?
    \[ \text{Count}[j, k] = \sum_{i: h_j(i) = k} a_i \]
  – Thus:
    \[ Q(i) \]
    each \( \text{Count}[j, h_j(i)] \geq a_i \]
  – Return:
    \[ \hat{a}_i \triangleq \min_j \left\{ \text{Count}[j, h_j(i)] \geq a_i \right\} \]
    tightest upper bound

But updates may be positive or negative

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ \begin{array}{l}
-\lambda w_i^{(t)} + x_i^{(t)} y^{(t)} - P(Y = 1|\mathbf{x}^{(t)}, \mathbf{w}^{(t)})
\end{array} \right\} \]

• Count-Min sketch for positive & negative case
  – \( a_i \) no longer necessarily positive
• Update the same: Observe change \( \Delta_i \) to element \( i \):
  \[ \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + \Delta_i \]
  – Each \( \text{Count}[j,h_j(i)] \) no longer an upper bound on \( a_i \)
• How do we make a prediction?

• Bound: \[ |\hat{a}_i - a_i| \leq 3\epsilon ||a||_1 \]
  – With probability at least \( 1-5^{1/4} \), where \( ||a|| = \sum_i |a_i| \)
Finally, Sketching for LR

\[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)})] \right\} \]

- Never need to know size of vocabulary!
- At every iteration, update Count-Min matrix:

- Making a prediction:

- Scales to huge problems, great practical implications...

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 \( x \):
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!

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

Learning With Hash Kernels

- Given hash kernel of dimension \( m \), specified by \( h \) and \( \xi \)
  - Learn \( m \) dimensional weight vector
- Observe data point \( x \)
  - Dimension does not need to be specified a priori!
- Compute \( \phi(x) \):
  - Initialize \( \phi(x) \)
  - For non-zero entries \( j \) of \( x \):
- Use normal update as if observation were \( \phi(x) \), e.g., for LR using SGD:
  \[
  w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + \phi_i(x^{(t)})[y^{(t)} - P(Y = 1|\phi(x^{(t)}), w^{(t)})] \right\}
  \]
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 $u$:

  • If we know little about user $u$:
  • After a lot of data from user $u$:

Problems with Simple Multi-Task Learning

• Dealing with new user is annoying, just like dealing with new words in vocabulary

• Dimensionality of joint parameter space is HUGE, e.g.
  personalized email spam classification from Weinberger et al.:
  – 3.2M emails
  – 40M unique tokens in vocabulary
  – 430K users
  – 16T parameters needed for personalized classification!
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 $u$:

• Estimating click probability as desired:

• 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 $\phi(x,u)$
  – 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
Results from Weinberger et al. on Spam Classification: Effect of $m$

Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (global-hashed) converges relatively soon, showing that the distortion error $\epsilon_d$ vanishes. The personalized classifier results in an average improvement of up to 30%.

Results from Weinberger et al. on Spam Classification: Multi-Task Effect

Figure 3. Results for users clustered by training emails. For example, the bucket [8, 15] consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up to 65% reduction in spam), even users that did not contribute to the training corpus at all obtain almost 20% spam-reduction.
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

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
April 9th, 2015

©Emily Fox 2015
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** \( X = \{x^1, \ldots, x^N\}, \quad x^i \in \mathbb{R}^d \)
- **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,

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

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)

- Scaled Euclidean ($L_2$)
- Mahalanobis
  ($\Sigma$ is general symmetric positive definite matrix, on previous slide = diagonal)

- $L_1$ norm (absolute)
- $L_\infty$ (max) norm

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 \( \{0, 1\}, 1 + \log f(t, d) \ldots \)

- Inverse document frequency:
  \[ idf(t, D) = \]

- \( \text{tf-idf} \):
  \[ \text{tfidf}(t, d, D) = \]

  - 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)

---

**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>

KD-Tree Construction

- 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 $x^i_{d_j} \leq V$. 

<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>3</td>
<td>0.13</td>
<td>2.85</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>4.31</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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
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
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)

©Emily Fox 2015
Complexity for $N$ Queries

- Ask for nearest neighbor to each document
- Brute force 1-NN:
- $kd$-trees:

Inspections vs. $N$ and $d$
K-NN with KD Trees

- 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 $> r$
- **Now:** Prune when distance to bounding box $> r$
- 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.
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., 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)