Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Announcements:

- HW1 due next week
- updated TA office hours
- Project Proposals due tomo:
  - ‘big data’ questions v.s. ‘real data’ questions

- Today:
  - Review: bloom filter
  - Sketching counts; Hash kernels
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 $w_{\text{Obamacare}} = 0$
  - In practice, need to re-allocate memory, fix indices,… A big problem for Big Data

What Next?

- Hashing & Sketching!
  - Addresses both dimensionality issues and new features in one approach!

- Let’s start with a much simpler problem: Is a string in our vocabulary?
  - Membership query

- How do we keep track?
  - Explicit list of strings
    - Very slow
  - Fancy Trees and Tries
    - Hard to implement and maintain
  - Hash tables?
Hash Functions and Hash Tables

- Hash functions map keys to integers (bins):
  - Keys can be integers, strings, objects, ...

- Simple example: mod
  - \( h(i) = (a_i + b) \mod m \)
    - \( a = 7, b = 11, m = 32 \)
    - \( h(i) = 39 \mod 32 = 7 \)
  - Random choice of \((a,b)\) (usually primes)
  - If inputs are uniform, bins are uniformly used
  - From two results can recover \((a,b)\), so not pairwise independent \(
    \rightarrow \text{Typically use fancier hash functions} \)

- Hash table:
  - Store list of objects in each bin
  - Exact, but storage still linear in size of object ids, which can be very long
    - E.g., hashing very long strings, entire documents

Hash Bit-Vector Table-Based Membership Query

- Approximate queries with one-sided error: Accept false positives only
  - If we say no, element is not in set
  - If we say yes, element is very to be likely in set

- Given hash function, keep binary bit vector \( v \) of length \( m \):
  - Query \( Q(i) \): Element \( i \) in set?
    - \( v(h(i)) = 0 \) \( \Rightarrow \) \( Q(i) = 0 \)
    - \( v(h(i)) = 1 \) \( \Rightarrow \) \( Q(i) = 1 \)

- Collisions:

- Guarantee: One-sided errors, but may make many mistakes
  - How can we improve probability of correct answer?
Bloom Filter: Multiple Hash Tables

- Single hash table $\rightarrow$ Many false positives
- Multiple hash tables with independent hash functions
  - Apply $h_1(i), \ldots, h_d(i)$, set all bits to 1
    \[
    \forall i \in [1..n], \quad h_j(i) = \text{set all bits to } 1
    \]
- Query $Q(i)$?
  \[
  \text{if } \forall_j \quad h_j(i) = 1 \quad \Rightarrow \quad \text{"Yes"}
  \]
  \[
  \text{else} \quad \text{No!}
  \]
- Significantly decrease probability of false positives

Analysis of Bloom Filter

- Want to keep track of $n$ elements with false positive probability of $\delta > 0$... how large $m$ & $p$?

- Simple analysis yields:
  \[
  m = n \log_2 \frac{1}{\delta} \approx 1.5n \log_2 \frac{1}{\delta}
  \]
  \[
  p = \log_2 \frac{1}{\delta}
  \]
  \[
  \Rightarrow \quad \text{prob (false positive)} \leq \delta
  \]
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 6, 2017

Sketching Counts

- Bloom Filter is super cool, but not what we need...
  - We don’t just care about whether a feature existed before, but to keep track of counts of occurrences of features! (assuming \(x_i\) integer)
- Recall the LR update:

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

- Must keep track of (weighted) counts of each feature:
  - E.g., with sparse data, for each non-zero dimension \(i\) in \(x^{(t)}\):

- Can we generalize the Bloom Filter?

©Sham Kakade 2017
Count-Min Sketch: single vector

- Simpler problem: Count how many times you see each string
- Single hash function:
  - Keep Count vector of length $m$
  - every time see string $i$:

$$\text{Count}[h(i)] \leftarrow \text{Count}[h(i)] + 1$$

- Again, collisions could be a problem:
  - $a_i$ is the count of element $i$

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 \( \text{Count}[j,k] \)?

  - Thus:

  - Return:

Analysis of Count-Min Sketch

\[ \hat{a}_i = \min_j \text{Count}[j, h(i)] \geq a_i \]

- Set:
  
  \[ m = \left\lceil \frac{\epsilon}{\epsilon} \right\rceil \quad p = \left\lceil \ln \frac{1}{\delta} \right\rceil \]

- Then, after seeing \( n \) elements:

  \[ \hat{a}_i \leq a_i + \epsilon n \]

- With probability at least \( 1-\delta \)
Proof of Count-Min for Point Query with Positive Counts: Part 1 – Expected Bound

• $l_{i,j,k}$ = indicator that $i$ & $k$ collide on hash $j$:

• Bounding expected value:

• $X_{i,j}$ = total colliding mass on estimate of count of $i$ in hash $j$:

• Bounding colliding mass:

• Thus, estimate from each hash function is close in expectation

Proof of Count-Min for Point Query with Positive Counts: Part 2 – High Probability Bounds

• What we know: $Count[j, h_j(i)] = a_i + X_{i,j}$  
$E[X_{i,j}] \leq \frac{\epsilon}{e} n$

• Markov inequality: For $z_1, ..., z_k$ positive iid random variables

  $P(\forall z_i : z_i > \alpha E[z_i]) < \alpha^{-k}$

• Applying to the Count-Min sketch:
But updates may be positive or negative

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

- Count-Min sketch for positive & negative case
  - \( a \), 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 \( \text{Count}[j,h(i)] \) no longer an upper bound on \( a_i \)
- How do we make a prediction?

- Bound: \( |\hat{a}_i - a_i| \leq 3\varepsilon ||a||_1 \)
  - With probability at least \( 1 - 6\varepsilon^4 \), where \( ||a|| = \Sigma_i |a_i| \)

Finally, Sketching for LR

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

- 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
  $$m = O\left(\frac{\log N}{\epsilon^2}\right)$$
  - 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 $x_j$:

- 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 ξ)
  – Quickly generate projection ϕ(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)