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 7th, 2015

Motivating AdaGrad (Duchi, Hazan, Singer 2011)

• Assuming \( \mathbf{w} \in \mathbb{R}^d \), standard stochastic (sub)gradient descent updates are of the form:

\[
\mathbf{w}_i^{(t+1)} \leftarrow \mathbf{w}_i^{(t)} - \eta_t g_{t,i}
\]

• Should all features share the same learning rate?

  \begin{itemize}
  \item \textcolor{red}{\underline{\textbf{No}}} \text{ maybe instead: } \eta_{t,i} \text{ specific to feature } i
  \end{itemize}

• Often have high-dimensional feature spaces

  \begin{itemize}
  \item Many features are irrelevant
  \item Rare features are often very informative
  \end{itemize}

• Adagrad provides a feature-specific adaptive learning rate by incorporating knowledge of the geometry of past observations
AdaGrad Algorithm

- At time $t$, estimate optimal (sub)gradient modification $A$ by
  \[ A_t = \left( \sum_{\tau=1}^{t} g_{\tau} g_{\tau}^T \right)^{\frac{1}{2}} \]
  in $d$ dims, matrix $\sqrt{\text{diag}(A_t)}$ is $O(d^3)$

- For $d$ large, $A_t$ is computationally intensive to compute. Instead,
  \[ \text{diag}(A_t) = A_t = \begin{pmatrix} A_{11} & 0 & \cdots & 0 \\ 0 & \ddots & \vdots & \vdots \\ 0 & \cdots & A_{dd} \end{pmatrix}, \quad A_{t,ii} = \sqrt{g_{ii}^2} \]

- Then, algorithm is a simple modification of normal updates:
  \[
  w(t+1) = \arg\min_{w \in \mathcal{W}} ||w - (w(t) - \eta \text{diag}(A_t)^{-1} g_t)||^2_{\text{diag}(A_t)}
  \]

AdaGrad Theoretical Guarantees

- AdaGrad regret bound:
  \[
  \frac{1}{T} \sum_{t=1}^{T} \ell_t(w(t)) - \ell_t(w^*) \leq 2R_{\infty} \sum_{i=1}^{d} ||g_1:T,i||_2
  \]
  with $R_{\infty} := \max_{i} ||w(t) - w^*||_{\infty}$

- In stochastic setting:
  \[
  \frac{1}{T} \sum_{t=1}^{T} \ell_t(w(t)) - \ell(w^*) \leq 2R_{\infty} \sum_{i=1}^{d} \mathbb{E}[||g_1:T,i||_2]
  \]

- This really is used in practice!
- Many cool examples. Let’s just examine one…
AdaGrad Theoretical Example

- Expect to out-perform when gradient vectors are sparse
- SVM hinge loss example:
  \[
  \ell_t(w) = [1 - y^t \cdot \langle x^t, w \rangle] + \text{Hinge loss:}
  \]
  \[\text{Example}
  \]
  \[\text{Expect to out-perform when gradient vectors are}
  \]
  \[\text{sparse}
  \]
  \[\text{SVM hinge loss example:}
  \]
  \[\text{If } x_j^t \neq 0 \text{ with probability } \alpha \]
  \[\text{Previously best known method:}
  \]
  \[E \left[ \ell \left( \frac{1}{T} \sum_{t=1}^{T} w^{(t)} \right) \right] - \ell(w^*) = O \left( \frac{||w^*||_\infty}{\sqrt{T}} \cdot \max \{ \log d, d^{1-\alpha/2} \} \right)
  \]

Neural Network Learning

- Very non-convex problem, but use SGD methods anyway
  \[\ell(w, x) = \log(1 + \exp(\langle [p(\langle w_1, x_1 \rangle) \cdots p(\langle w_k, x_k \rangle)], x_0 \rangle))
  \]
  \[p(\alpha) = \frac{1}{1 + \exp(\alpha)}
  \]

Distributed, \(d = 1.7 \cdot 10^9\) parameters. SGD and AdaGrad use 80 machines (1000 cores), L-BFGS uses 800 (10000 cores)

Images from Duchi et al. ISMP 2012 slides
What you should know about Logistic Regression (LR) and Click Prediction

• Click prediction problem:
  – Estimate probability of clicking
  – Can be modeled as logistic regression
• Logistic regression model: Linear model
• Gradient ascent to optimize conditional likelihood
• Overfitting + regularization
• Regularized optimization
  – Convergence rates and stopping criterion
• Stochastic gradient ascent for large/streaming data
  – Convergence rates of SGD
• AdaGrad motivation, derivation, and algorithm

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 if we have 1B features?
• What can we with very high dimensional feature spaces?
  – Kernels not always appropriate, or scalable
  – What else?
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: \( \text{mod} \)
  - \( h(i) = (a \cdot i + b) \% m \)
    - \( a = 7, b = 11, m = 32 \)
    - \( h(i) = (7 \cdot i + 11) \% 32 \)
    - 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 -> 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 \( m \):
  - \( \text{Query } Q(i): \text{ Element } i \text{ in set?} \)
    - \( V(h(i)) = 0 \Rightarrow Q(i) = \text{ no!} \)
    - \( V(h(i)) = 1 \Rightarrow Q(i) = \text{ probably yes (or if } m \text{ small we could have no idea.)} \)

- Collisions:
  - \( h(\text{Obama care}) = 7 \Rightarrow V(h(\text{Obama care})) = V(7) = 1 \)
    - \text{but 'Obama care' not in set}

- 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 → Many false positives
- Multiple hash tables with independent hash functions
  - Apply $h_1(i), \ldots, h_p(i)$, set all bits to 1
  
  - Query $Q(i)$?
    
    $\forall j \ h_j(i) = 1$
    $Q(i) = \text{very probably yes}$
    $\text{else } Q(i) = \text{no}$

- 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:
  
  $m = \frac{n \log_2 \frac{1}{\delta}}{\ln 2} \approx 1.5n \log_2 \frac{1}{\delta}$

  $p = \log_2 \frac{1}{\delta}$

  Prob. of mistakes exp. decreasing w/ # of hash tables

  Single hash table: $\frac{1}{m}$ by making hash table longer
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:
  $$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)})] \right\}$$
- Must keep track of (weighted) counts of each feature:
  - E.g., with sparse data, for each non-zero dimension $i$ in $x^{(t)}$:
  - For all entries of hash $h^{(t)}$ by $(1-\eta t)$
  - For all $x_i^{(t)} \neq 0$
    - $w_i^{(t+1)} = x_i^{(t)} \cdot \text{const} \leftarrow \lambda (y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)})$)
- Can we generalize the Bloom Filter?

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 $i$
    - $\hat{a}_i = \text{true counts}$
    - $Q(i) \rightarrow \text{return } \hat{a}_i = \text{true count of } i$
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

- Query $Q(i)$?
  - What is in $\text{Count}[j,k]$?
    \[
    \text{Count}[j,k] = \sum_{i : h_j(i) = k} a_i
    \]
  - Thus:
    \[
    Q(i) \text{ of each } \text{Count}[j,h_j(i)] \geq a_i
    \]
  - Return:
    \[
    \hat{a}_i = \min_j \text{Count}[j, h_j(i)] \geq a_i
    \]
    
©Emily Fox 2015
Analysis of Count-Min Sketch

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

- Set:
  \[ m = \left\lceil \frac{e}{\epsilon} \right\rceil \quad p = \left\lceil \frac{\ln \frac{1}{\delta}}{\epsilon} \right\rceil \]
  - length of each hash
  - # of hashes
  - false pos. rate

- Then, after seeing n elements:
  \[ a_i \leq \hat{a}_i \leq a_i + \epsilon n \]
  - high prob. statement

- With probability at least 1-\delta

Proof of Count-Min for Point Query with Positive Counts: Part 1 – Expected Bound

- \( I_{i,k} \) = indicator that \( i \) & \( k \) collide on hash \( j \):
  \[ (i \neq k) \land (h_j(i) = h_j(k)) \]
  \[ m = \left\lceil \frac{e}{\epsilon} \right\rceil \]

- Bounding expected value:
  \[ E[I_{i,j,k}] = P(h_j(i) = h_j(k)) = \frac{1}{m} \leq \frac{\epsilon}{e} \]

- \( X_{i,j} \) = total colliding mass on estimate of count of \( i \) in hash \( j \):
  \[ X_{i,j} = \sum_{k \neq i} I_{i,j,k} \alpha_k \]
  \[ \text{add their counts if collide} \]

- Bounding colliding mass:
  \[ E[X_{i,j}] = \sum_{k \neq i} \alpha_k \frac{E[I_{i,j,k}]}{m} \leq \frac{ne}{e} \]
  \[ \text{sum over words \neq 'Mary'} \]

- 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: \( \text{Count}[j, h_j(i)] = a_i + X_{i,j} \quad E[X_{i,j}] \leq \frac{e}{n} \)

- Markov inequality: For \( z_1, \ldots, z_k \) positive iid random variables

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

- Applying to the Count-Min sketch:

\[
P(\hat{a}_i > a_i + \epsilon n) = P(\forall j, \text{Count}[j, h_j(i)] > a_i + \epsilon n) = P(\forall j, a_i + X_{i,j} > a_i + \epsilon n) \leq P(\forall j, X_{i,j} > \epsilon E[X_{i,j}]) < e^{-p} \leq \delta
\]