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 $\boxed{\text{AdaGrad}}$ (Duchi, Hazan, Singer 2011)

adaptive gradient

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

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta_t g_{t,i}$$

"step size"
"learning rate"

- Should all features share the same learning rate?

maybe instead: $\eta_{t,i}$ specific to feature i

- Often have high-dimensional feature spaces
  - Many features are irrelevant → small learning rate
  - Rare features are often very informative

- 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

estimate of A at time t

up to time t

past gradients

$$A_t = \left(\sum_{\tau=1}^{t} g_\tau g_\tau^T\right)^{\frac{1}{2}}$$

in d dims, matrix $\sqrt{\ }$ is $O(d^3)$

- For d large, A_t is computationally intensive to compute. Instead,

$$\mathrm{diag}(A_t) \qquad A_t = \begin{pmatrix} A_{11} & & 0 \\ & \ddots & \\ 0 & & A_{dd} \end{pmatrix} \qquad A_{t,ii} = \sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}$$

- Then, algorithm is a simple modification of normal updates:

$$\mathbf{w}^{(t+1)} = \arg\min_{\mathbf{w}\in\mathcal{W}} ||\mathbf{w} - (\mathbf{w}^{(t)} - \eta \mathrm{diag}(A_t)^{-1} g_t)||^2_{\mathrm{diag}(A_t)}$$

$$\eta_t \to \eta A^{-1} \to \eta\, \mathrm{diag}(A_t)^{-1}$$

weigh dimensions by sqrt of sum of past grad. in that dim.

AdaGrad Theoretical Guarantees

- AdaGrad regret bound:

radius of space

$$R_\infty := \max_t ||\mathbf{w}^{(t)} - \mathbf{w}^*||_\infty$$

selected $w^{(t)}$'s vs. best in retrospect

$$R(T) = \sum_{t=1}^{T} \ell_t(\mathbf{w}^{(t)}) - \ell_t(\mathbf{w}^*) \leq 2R_\infty \sum_{i=1}^{d} ||g_{1:T,i}||_2$$

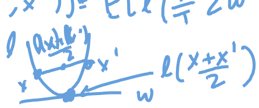
upper bound

Jensen's inequal.

  - In stochastic setting: $\ell(w) = E_x[\ell(w,x)]$ and $\ell_t(w) = \ell(w, x^t)$. Then,

$$\frac{E[R(T)]}{T} = \mathbb{E}\left[\ell\left(\frac{1}{T}\sum_{t=1}^{T} w^{(t)}\right)\right] - \ell(\mathbf{w}^*) \leq \frac{2R_\infty}{T}\sum_{i=1}^{d}\mathbb{E}[||g_{1:T,j}||_2]$$

$$E\left[\frac{1}{T}\sum_t \ell(w^{(t)}, x^t)\right] \geq E\left[\ell\left(\frac{1}{T}\sum w^{(t)}, x^t\right)\right]$$



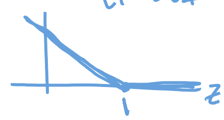
$$\frac{1}{T} E_{x^{1:T}}\left[\sum_t \ell(w^*, x^t)\right] = \frac{1}{T}\sum_t \underbrace{E_{x^t}[\ell(w^*, x^t)]}_{\ell(w^*)} = \ell(w^*)$$

- 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:

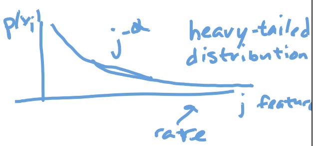
Hinge loss: $[1-z]_+$



$$\ell_t(\mathbf{w}) = [1 - \underbrace{y^t \langle \mathbf{x}^t, \mathbf{w} \rangle}_{\text{"z"}}]_+$$

$$\mathbf{x}^t \in \{-1, 0, 1\}^d$$

- If $x_j^t \neq 0$ with probability $\propto j^{-\alpha}, \quad \alpha > 1$



$$\mathbb{E}\left[\ell\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^{(t)}\right)\right] - \ell(\mathbf{w}^*) = \mathcal{O}\left(\frac{||\mathbf{w}^*||_\infty}{\sqrt{T}} \cdot \max\{\log d, d^{1-\alpha/2}\}\right)$$

same

For α large, adagrad can be exp. better in d

- Previously best known method: $\mathbb{E}\left[\ell\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^{(t)}\right)\right] - \ell(\mathbf{w}^*) = \mathcal{O}\left(\frac{||\mathbf{w}^*||_\infty}{\sqrt{T}} \cdot \sqrt{d}\right)$

Neural Network Learning

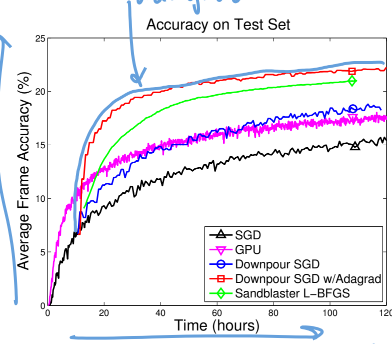
- Very non-convex problem, but use SGD methods anyway

$$\ell(w, x) = \log(1 + \exp(\langle [p(\langle w_1, x_1 \rangle) \cdots p(\langle w_k, x_k \rangle)], x_0 \rangle))$$

$$p(\alpha) = \frac{1}{1 + \exp(\alpha)}$$

adagrad

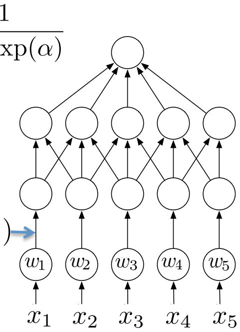
accuracy



running time

(Dean et al. 2012)

$p(\langle w_1, x_1 \rangle)$



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

focus

Problem 1: Complexity of LR Updates

- Logistic regression update:

stochastic gradient ascent

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

- Complexity of updates:
  - Constant in number of data points ✓
  - In number of features? $O(d)$
    - 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 "kernel trick"
  - What else?

Problem 2: Unknown Number of Features

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

X: 'Mary' 'had' 'a' 'little' 'lamb' 'Obamacare'

- What's the dimensionality of **x**? size of vocabulary… millions
- What if we see new word that was not in our vocabulary?
  - Obamacare
  - Theoretically, just keep going in your learning, and initialize $\mathbf{w}_{\text{Obamacare}} = 0$
  - 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 → scan the list
    - Very slow

{'Mary', 'had', 'a', 'little', 'lamb', 'Obamacare'}

  - Fancy Trees and Tries
    - Hard to implement and maintain
  - Hash tables?

h('Mary') = 7 → {'Mary', 'Obamacare'}
h('Obamacare') = 7

Hash Functions and Hash Tables

- $\boxed{\text{Hash functions}}$ map **keys** to integers (bins): $h: X \rightarrow \{1, \ldots, m\}$
  - Keys can be integers, strings, objects,…

$h(i)$

m

- Simple example: **mod**
  - $h(i) = (a.i + b) \% m$

    $a = 7 \quad b = 11 \quad m = 32$

    $i = 4 \Rightarrow h(i) = 39 \% 32 = 7$
  - Random choice of (a,b) (usually primes) $\longrightarrow$ random hash fcn
  - If inputs are uniform, bins are uniformly used
  - From two results can recover (a,b), so not pairwise independent -> Typically use fancier hash functions

    $\{h(i), h(j)\}$ ← need this for our theory
- $\boxed{\text{Hash table:}}$
  - 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*:

$h(\text{'had'})$ $\quad$ $h(\text{'Mary'}) = 7$

$v =$ $\quad$ initially, $v = 0$

m

- Query *Q(i)*: Element *i* in set?
  - $V(h(i)) = 0 \Rightarrow Q(i) = \text{no!}$
  - $V(h(i)) = 1 \Rightarrow Q(i) = \text{probably yes}$ (or if m small compared to vocab. then "I have no idea")
- Collisions:

  $h(\text{'Obamacare'}) = 7 \Rightarrow V(h(\text{'obamacare'})) = V(7) = 1$

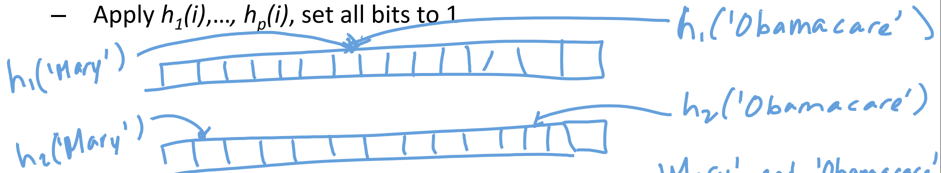
  but 'Obamacare' not in set

  bc we saw 'Mary'

  $\boxed{\text{prob. of collision} = 1/m}$
- 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



'Mary' and 'Obamacare' collide on h_1, but not h_2

- Query $Q(i)$?

  $\forall j \quad h_j(i) = 1$

  $Q(i)$ = very probably yes

  else $Q(i)$ = no

- Significantly decrease probability of false positives

Analysis of Bloom Filter

- Want to keep track of n elements with false positive probability of δ>0… how large m & p?

  dim of each # of hashes

- Simple analysis yields:

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

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

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

single hash table: $\frac{1}{m}$ by making hash table longer

Sketching Counts

For LR: Want sketch to keep track of $w^{(t)}$

- 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:

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

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

For all entries of hash
- multiply current $w_i^{(t)}$ by $(1-\eta_t\lambda)$

For all $x_i^{(t)} \neq 0$
- $w_i^{(t+1)}$ += $x_i^{(t)} \cdot \text{const}$

const $= \eta_t(y^{(t)} - P(Y=1|\ldots))$

- Can we generalize the Bloom Filter?

Count-Min Sketch: single vector

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

| 1 | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 2 | 1 |

h('little')

h('Mary') = 7   h('Obamacare') = 7

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

see 'Mary' 'Obamacare' → count[7] = 2

Q('Mary') = Count[7] = 2 > 1

true count of 'Mary'

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

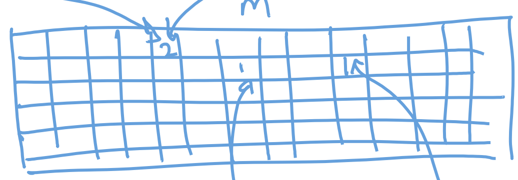
true counts

$$Count[j] = \sum_{i:h(i)=j} a_i$$

$Q(i) \longrightarrow$ return $\hat{a}_i = Count[h(i)] \geq a_i$

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:

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

jth hash fcn

Querying the Count-Min Sketch

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

- Query Q(i)?
  - What is in *Count*[*j*,*k*]?

$$Count[j,k] = \sum_{i: h_j(i) = k} a_i$$

  - Thus:

$Q(i)$

each $Count[j, h(i)] \geq a_i$

  - Return:

$$\hat{a}_i \triangleq \min_j Count[j, h_j(i)] \geq a_i$$

← tightest upper bound

Analysis of Count-Min Sketch

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

- Set:

$$m = \left\lceil \frac{e}{\epsilon} \right\rceil \qquad p = \left\lceil \ln \frac{1}{\delta} \right\rceil$$

length of each hash   # of hashes   false pos. rate

- Then, after seeing n elements:

$$a_i \leq \hat{a}_i \leq a_i + \epsilon n$$

- With probability at least 1-δ

high. prob. statement

©Emily Fox 2015 19

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

- $I_{i,j,k}$ = indicator that *i* & *k* collide on hash *j*:

$$(i \neq k) \wedge (h_j(i) = h_j(k))$$

- Bounding expected value:

$$E[I_{ijk}] = P(h_j(i) = h_j(k)) = \frac{1}{m} \leq \frac{\epsilon}{e} \qquad m = \left\lceil \frac{e}{\epsilon} \right\rceil$$

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

$$\text{'Mary'}\ X_{ij} = \sum_{k \neq i} I_{ijk} a_k$$

add their counts if collide

sum over words ≠ 'Mary'

$Count[j, h_j(i)] = a_i + X_{ij}$

'Mary' counts

- Bounding colliding mass:

$$E[X_{ij}] = \underbrace{\sum_{k \neq i} a_k}_{n} E[I_{ijk}] \leq \frac{n\epsilon}{e}$$

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

©Emily Fox 2015 20

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

- What we know: $Count[j, h_j(i)] = a_i + X_{i,j} \qquad E[X_{i,j}] \leq \frac{\epsilon}{e} n$
- Markov inequality: For $z_1,\ldots,z_k$ positive iid random variables

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

$P(z_i > a) < \frac{E[z_i]}{a}$

"a" $= \alpha E[z_i]$

and use ind. of z_i

- Applying to the Count-Min sketch:

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

$p = \lceil \ln \frac{1}{\delta} \rceil$

small prob.