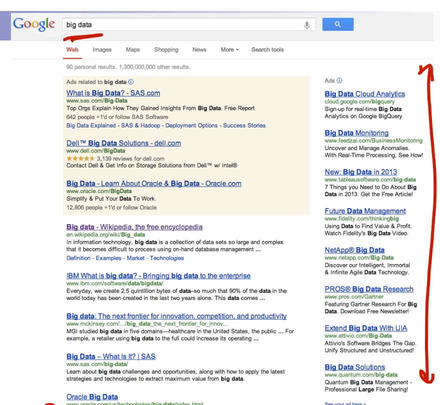
Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Emily Fox
January 14th, 2014

Ad Placement Strategies

RECAP



- Companies bid on ad prices
  - $C_1 \to \$10$
  - $C_2 \to \$20$
  - $C_3 \to \$100$
- Which ad wins? (many simplifications here)
  - Naively: $C_3 \to \$100$
  - But: paid on clicks
  - Instead:

e.g. $\begin{cases} P(\text{click} \mid C_3) = 0.01 \\ P(\text{click} \mid C_2) = 0.5 \end{cases}$

$E[\$_3] = 0.01 \times 100 = \$1$

$E[\$_2] = 0.5 \times 20 = \$10$

Learning Problem for Click Prediction

- Prediction task: $X :\to [0,1]$ $\quad P(\text{Click}=1 \mid X) \leftarrow$ want to model this
- Features:

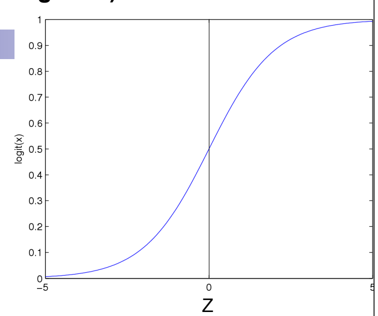
$X = (\text{feats of page},\ \text{feats ad},\ \text{feats user})$ ← features

$(X, Y) \to (\text{webpage1}, \text{ad 7}, \text{user25}, \text{time 12}) \to \text{click}=1$

- Data:
  - Batch: fixed dataset $(X^1, Y^1) \ldots (X^N, Y^N)$
  - Online: data as a stream
    user arrives at a page $\to x^t \to$ predict $\hat{y}$, click? $\to$ observe y^t
- Many approaches (e.g., logistic regression, SVMs, naïve Bayes, decision trees, boosting,…)
  - Focus on logistic regression; captures main concepts, ideas generalize to other approaches

Logistic Regression

Logistic function (or Sigmoid): $\dfrac{1}{1+exp(-z)}$



- Learn P(Y|**X**) directly
  - Assume a particular functional form
  - Sigmoid applied to a linear function of the data:

$$P(Y=0|X,W) = \frac{1}{1+exp(\underbrace{w_0+\sum_i w_i X_i}_{\text{linear function of features}})}$$

log odds ← log linear model

Features can be discrete or continuous!

Standard v. Regularized Updates

- Maximum conditional likelihood estimate

gradient ascent

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[\prod_{j=1}^{N} P(y^j \mid \mathbf{x}^j, \mathbf{w}) \right]$$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

- Regularized maximum conditional likelihood estimate

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[\prod_j P(y^j | \mathbf{x}^j, \mathbf{w})) \right] - \frac{\lambda}{2} \sum_{i>0} w_i^2$$

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

reg derivative ← move towards zero

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:

$$E_{\mathbf{x}}[\ell(\mathbf{w}, \mathbf{x})] = E_{\mathbf{x}}\left[\ln P(y|\mathbf{x}, \mathbf{w}) - \frac{\lambda}{2}||\mathbf{w}||_2^2\right]$$

- Batch gradient ascent updates: $\boxed{O(Nd)}$ problem for large or streaming N

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \underbrace{\frac{1}{N} \sum_{j=1}^{N} x_i^{(j)} [y^{(j)} - P(Y = 1 | \mathbf{x}^{(j)}, \mathbf{w}^{(t)})]}_{\text{avg all datapoint}} \right\}$$

- Stochastic gradient ascent updates: $\boxed{O(d)}$
  - Online setting: $x^{(t)}, y^{(t)}$

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

$x_t = (0\ 0\ 0\ 1\ \ 0\ 0\ 0\ \ 1\ \ 0\ 0\ 0\ 1\ 0)$

AdaGrad in Euclidean Space

- For $\mathcal{W} = \mathbb{R}^d$, $w^{(t+1)} \leftarrow w^{(t)} - \eta\, \mathrm{diag}(A_t)^{-1} g_t$

  no constraints on w

- For each feature dimension,   adaptive step size

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

where

$$\eta_{t,i} = \eta / A_{t,ii}$$

in sparse case, step size larger when seeing a rare feature

- That is,

$$w_i^{(t+1)} \leftarrow w_i^{(t)} - \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$$

- Each feature dimension has it's own learning rate!
  - Adapts with *t*
  - Takes geometry of the past observations into account
  - Primary role of η is determining rate the first time a feature is encountered

Problem 1: Complexity of Update Rules for Logistic Regression

- Logistic regression update: stoch. grad. asc.

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

- 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

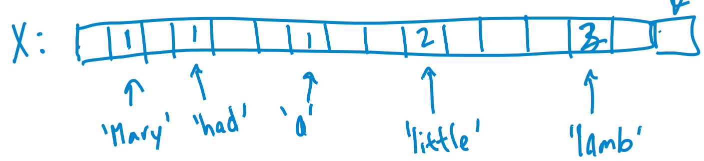
  What if we have 1B features??

- 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…"

'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?

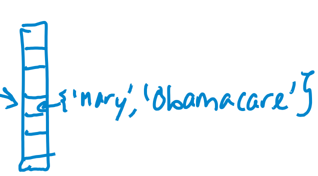
- Hashing & Sketching!
  - Addresses both dimensionality issues and new features in one approach!
- Let's start with a much simpler problem: Is a string in our vocabulary?
  - Membership query
- How do we keep track?
  - Explicit list of strings → scan the list
    - Very slow

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

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

h('Mary') = 7

h('Obamacare') = 7



Hash Functions and Hash Tables

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

$h(i)$ … $\underbrace{\hspace{6cm}}_{m}$

- 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) ⟶ 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
- 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

©Emily Fox 2014 11

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*:

  $h(\text{'hat'}) = 2 \qquad h(\text{'Mary'}) = 7$

  $V = \underbrace{\hspace{6cm}}_{m}$ initially, $V = 0$

- Query Q(i): Element i in set?
  - $V(h(i)) = 0 \Rightarrow Q(i) = \text{no!}$
  - $V(h(i)) = 1 \Rightarrow Q(i) = \text{probably yes}$
- Collisions:

  $h(\text{'Obamacare'}) = 7 \Rightarrow V(h(\text{'Obamacare'})) = V(7) = 1$ ← bc we saw 'Mary'

  but Obamacare not in set
- Guarantee: One-sided errors, but may make many mistakes
  - How can we improve probability of correct answer? probability of collision $= \frac{1}{m}$

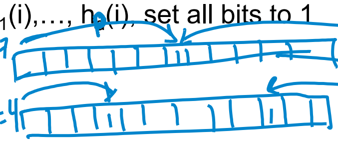
©Emily Fox 2014 12

Bloom Filter: Multiple Hash Tables

- Single hash table -> Many false positives

- Multiple hash tables with independent hash functions
  - Apply $h_1(i),\dots, h_p(i)$, set all bits to 1

$h_1(\text{'Mary'})=7$ $h_1(\text{'Obamacare'})=7$



$h_2(\text{'Mary'})=4$ $h_2(\text{'Obamacare'})=9$

- Query Q(i)?

if $\forall_j$ $h_j(i)=1$

$Q(i)$ = very probably yes

else $Q(i)$ = no

'Mary' + 'Obamacare' collide in h_1 but not h_2

- 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/ # hash fcns

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

Sketching Counts

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: $(1-\eta_t\lambda)\, w_i^{(t)} + x_i^{(t)}\, \eta_t (y^{(t)} - P(\ ))$

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

- 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 $w^{(t)}$ by $(1-\eta_t\lambda)$

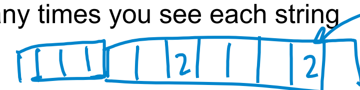
For all $x_i^{(t)} \neq 0$
- $w_i^{(t+1)}$ += $x_i^{(t)} \cdot$ const $\leftarrow \eta_t(y^{(t)} - P(Y=1|\cdots))$

- 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:

h('little')



h('Mary') = 7 h('ObamaCare') = 7

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

See 'Mary' 'Obamacare' $\Rightarrow$ 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:

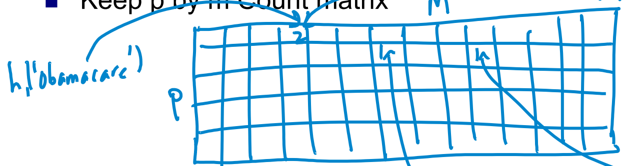
$$Count[j] = \sum_{i: h(i)=j} a_i$$ true counts

over-est.

$Q(i) \rightarrow$ 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*:

row j^{th} hash fcn   input

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

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 2014   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:

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

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

'Mary' counts

$$\overline{\text{'Mary'}} \quad X_{ij} = \sum_{k \neq i} I_{ijk}\, a_k \quad \text{add their counts} \qquad Count[j, h_j(i)] = a_i + X_{ij}$$

sum over words ≠ 'Mary'

colliding counts

- Bounding colliding mass:

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

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

©Emily Fox 2014   20

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

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

$$P(\hat{a}_i > a_i + \epsilon n) = P(\forall j, Count[j, h_j(i)] > a_i + \epsilon n)$$

$$= P(\forall j, \cancel{a_i} + X_{ij} > \cancel{a_i} + \epsilon n)$$

$$\leq P(\forall j, X_{ij} > \underbrace{e}_{\text{"}\alpha\text{"}} E[X_{ij}]) < e^{-p} \leq \underbrace{\delta}_{\text{small prob.}}$$

But updates may be positive or negative

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + \underbrace{x_i^{(t)} [y^{(t)} - P(Y = 1 | \mathbf{x}^{(t)}, \mathbf{w}^{(t)})]}_{\text{pos. or neg., non-integer}} \right\}$$

- Count-Min sketch for positive & negative case
  - a_i no longer necessarily positive
- Update the same: Observe change Δ_i to element *i*:

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

  - Each Count[j,h(i)] no longer an upper bound on a_i
- How do we make a prediction?

$\hat{a}_i := \text{median}_j \; count[j, h_j(i)]$

count[1, $h_1(i)$]
a_i → count[3, $h_3(i)$]
count[2, $h_2(i)$]

- Bound: $|\hat{a}_i - a_i| \leq 3\epsilon ||\mathbf{a}||_1$
  - With probability at least $1-\delta^{1/4}$, where $||\mathbf{a}||_1 = \Sigma_i |a_i|$

Finally, Sketching for LR

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

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

$\forall j,k \quad Count[j,k] = (1-\eta_t \lambda)\, count[j,k]$

$\forall x_i^{(t)} \neq 0$

$\forall j \quad Count[j, h_j(i)] \mathrel{+}= x_i^{(t)} \cdot const$ ← $\eta_t (y^{(t)} - P(Y=1|\ldots))$

- Making a prediction:

Remember our est. of $w_i^{(t)}$: $\text{median}_j \; Count[j, h_j(i)]$

Make pred :

$\text{log odds} = w_0^{(t)} + \sum_{i : x_i \neq 0} \text{median}_j \; count[j, h_j(i)]\, x_i^{(t)}$

- Scales to huge problems, great practical implications…

Hash Kernels

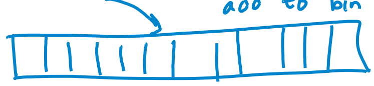
- Count-Min sketch not designed for negative updates
- Biased estimates of dot products

- **Hash Kernels**: Very simple, but powerful idea to remove bias
- Pick 2 hash functions:
  - h : Just like in Count-Min hashing $\quad h : X \to \{1, \ldots, m\}$
  - ξ : Sign hash function $\quad \xi : X \to \{+1, -1\}$
    - Removes the bias found in Count-Min hashing (see homework)

can think of as random projection of X

- Define a "kernel", a projection ϕ for **x**:

For each non-zero element of X, add to bin $h(j)$: $\xi(j) X_j$

$\phi(x) =$ 

$$\phi_i(x) = \sum_{j : h(j) = i} \xi(j) X_j$$

If $X_j = 7$
$h(j) = 4$
$\xi(j) = -1$
⇓
add -7 to bin 4

Hash Kernels Preserve Dot Products

$$\phi_i(\mathbf{x}) = \sum_{j:h(j)=i} \xi(j)\mathbf{x}_j$$

- Hash kernels provide unbiased estimate of dot-products!

$E_{h,\xi}[\phi(x)\cdot\phi(y)] = x\cdot y$ Pf: by homework

- Variance decreases as O(1/*m*) ← gets better w/ more dims

- Choosing *m*? For ε>0, if

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

log in data size

  - Under certain conditions…
  - Then, with probability at least 1-δ:

$$(1-\epsilon)||\mathbf{x}-\mathbf{x}'||_2^2 \leq ||\phi(\mathbf{x})-\phi(\mathbf{x}')||_2^2 \leq (1+\epsilon)||\mathbf{x}-\mathbf{x}'||_2^2$$