Case Study 1: Estimating Click Probabilities

Database Issues

Tackling an Unknown Number of Features with Sketching

Machine Learning for Big Data
CSE547/STAT548, University of Washington
Sham Kakade
April 7th, 2015

Problem 1: Complexity of LR Updates

- Logistic regression update:

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

- Complexity of updates:
  - Constant in number of data points
  - In number of features?
    - Problem both in terms of computational complexity and sample complexity

1B features

- What can we do with very high dimensional feature spaces?
  - Kernels not always appropriate, or scalable
  - What else?

What Next?

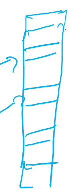
- Hashing & Sketching!
  - Addresses both dimensionality issues and new features in one approach!
- Let's start with a much simpler problem: Is a string in our vocabulary?
  - Membership query
- How do we keep track?
  - Explicit list of strings
    - Very slow

{'mary', 'had', 'a', ... 'lamb'}

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

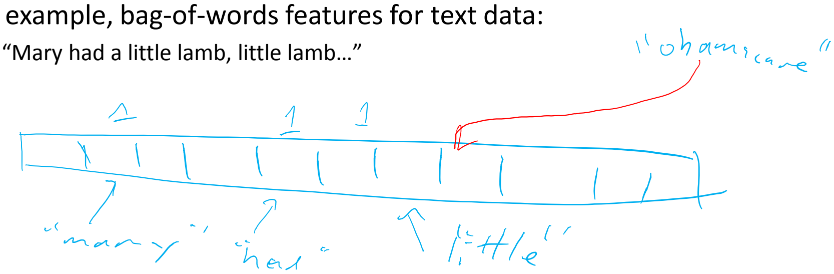
h('mary')

h('obamacare')



Problem 2: Unknown Number of Features

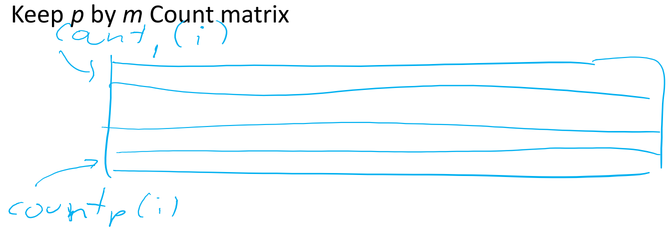
- For example, bag-of-words features for text data:
  - "Mary had a little lamb, little lamb…"



- What's the dimensionality of **x**? ← size of vocab.
- 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

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*:

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

row index.

input

Querying the Count-Min Sketch

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

want the count.

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

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

  - Thus:

    tightest upper bound.

  - Return:

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

Analysis of Count-Min Sketch

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

- Set: $O\left(\frac{1}{\varepsilon}\right)$

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

$O\left(\frac{n}{\varepsilon}\right)$

- Then, after seeing n elements:

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

want.

- With probability at least 1-δ

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*:

$= O(\varepsilon)$

- Bounding expected value:

$$E[I_{ijk}] = Pr(h_j(i) = h_j(k)) = \frac{1}{m}$$

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

$$X_{ij} = \sum_{k \neq i} I_{ijk} \, a_k$$

- Bounding colliding mass:

$$E[X_{ij}] \leq O(n\varepsilon)$$

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

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

$\Pr(Z \geq a) \leq \frac{E[Z]}{a}$ Z positive r.v.

- Applying to the Count-Min sketch:

union bound

But updates may be positive or negative

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

- 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?

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

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

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:
- Making a prediction:
- Scales to huge problems, great practical implications…

Hash Kernels

$x \cdot x'$

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

$\approx E[\phi(x) \cdot \phi(x')]$

- **Hash Kernels**: Very simple, but powerful idea to remove bias
- Pick 2 hash functions:
  - h : Just like in Count-Min hashing $h : \text{string} \rightarrow \{1, \ldots, m\}$
  - ξ : Sign hash function $\xi : \text{string} \rightarrow \{-1, 1\}$
    - Removes the bias found in Count-Min hashing (see homework)
- Define a "kernel", a projection ϕ for **x**:

add $\xi(j)$ (c_j) to index $h(j)$

see j c_j times

$\phi =$

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

Hash Kernels Preserve Dot Products

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

a – counts

- Hash kernels provide unbiased estimate of dot-products!

$\mathop{E}_{h,\xi}[\phi(x)\cdot\phi(y)] = x\cdot y$

pf: HW!

same as

Rand Proj. arguments/

J.L.

- Variance decreases as O(1/*m)*

- Choosing *m*? For ε>0, if

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

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

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

 13

Learning With Hash Kernels

- Given hash kernel of dimension *m*, specified by *h* and *ξ*
  – Learn *m* dimensional weight vector
- Observe data point **x**
  – Dimension does not need to be specified a priori!
- Compute *ϕ*(**x**):
  – Initialize *ϕ*(**x**)
  – For non-zero entries *j* of $\mathbf{x}_j$:

$Pr(Y \mid \phi(x), w^t)$

$= e^{\phi(x)\cdot w^t} / 1 + e^{\phi(x)\cdot w^t}$

- Use normal update as if observation were *ϕ*(**x**), e.g., for LR using SGD:

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

 14

Interesting Application of Hash Kernels: Multi-Task Learning

- Personalized click estimation for many users:
  - One global click prediction vector **w**:

    $\frac{e^{w \cdot x}}{1+e^{w \cdot x}}$

    - But…
  - A click prediction vector $\mathbf{w}_u$ per user *u*:

    instead

    $\frac{e^{w_u \cdot x}}{1+e^{w_u \cdot x}}$

    - 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 $\mathbf{w}_u$ per user:

  $\frac{e^{(w+w_u)\cdot x}}{1+e^{\dots}}$

  - Prediction for user *u*:

    $(w+w_u)\cdot x = w\cdot x + w_u \cdot x$
  - If we know little about user *u*:

    $w \cdot x$
  - After a lot of data from user *u*:

    $w_u \neq 0$, get user specific info.

 15

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!

 16

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*:

$d + (\#\text{users}) \times d$

$$z_{(x,u)} = (\underbrace{x_1, \dots x_d}_{\text{global}}, \underbrace{0, 0 0}_{d}, \dots, \underbrace{x_1 \dots x_d}_{\text{user } u}, 0, 0, \}$$

- Estimating click probability as desired:

$$w = (\bar{w}, \bar{w}, \dots \bar{w}_u, \dots \bar{w}_{\#\text{users}})$$

← "weight vect for z" $z_{(x,u)} \cdot \underline{w} = w \cdot x + w_u \cdot x$

- Address huge dimensionality, new words, and new users using hash kernels:

Simple Trick for Forming Projection $\phi(\mathbf{x},u)$

don't form $z_{(x,u)}$

- Observe data point **x** for user *u*
  - Dimension does not need to be specified a priori and user can be new!

hash both

- Compute $\phi(\mathbf{x},u)$:
  - Initialize $\phi(\mathbf{x},u)$ ← 0
  - For non-zero entries *j* of $\mathbf{x}_j$:
    - E.g., j='Obamacare'
    - Need two contributions to ϕ:
      - Global contribution
      - Personalized Contribution
    - Simply:

'obamacare' and 'obamacare_7' = $g(\text{'obamacare'})\, x_j$

$\phi_h(\text{'obamacare'})$

$\phi_h(\text{'obamacare\_7'})$ += $g(\text{'obamacare\_7'}) \cdot x_j$

user

- Learn as usual using $\phi(\mathbf{x},u)$ instead of $\phi(\mathbf{x})$ in update function

Results from Weinberger et al. on Spam Classification: Effect of *m*

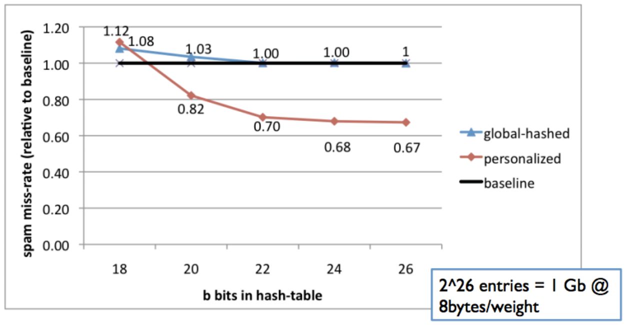


Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (*global-hashed*) converges relatively soon, showing that the distortion error ϵ_d vanishes.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

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

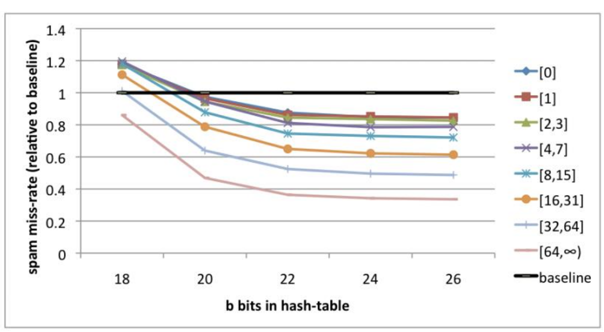


Figure 3. Results for users clustered by training emails. For example, the bucket [8, 15] consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up-to 65% reduction in spam), even users 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)

database

learning

neat