Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

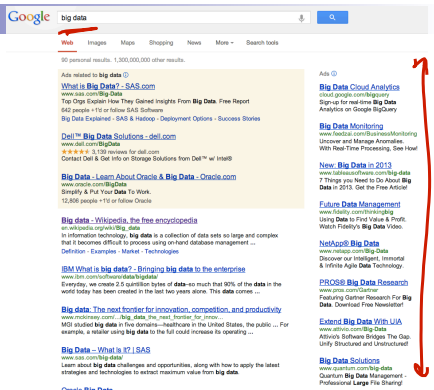
Machine Learning for Big Data

CSE547/STAT548, University of Washington

Emily Fox

January 14th, 2014

Ad Placement Strategies



- Companies bid on ad prices
  - $c_1 \rightarrow \$10$
  - $c_2 \rightarrow \$20$
  - $c_3 \rightarrow \$100$
- Which ad wins? (many simplifications here)
  - Naively: $c_3 \rightarrow \$100$
  - But: paid on clicks
  - Instead:

e.g. $P(\text{click} \mid c_3) = 0.01$ $\quad E[\$_3] = 0.01 \times 100 = \$1$

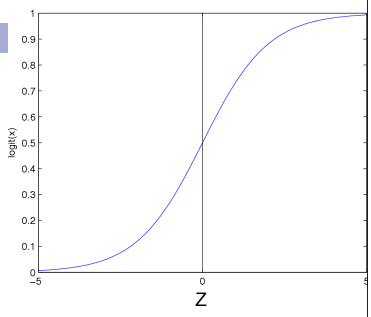
$P(\text{click} \mid c_2) = 0.5$ $\quad E[\$_2] = 0.5 \times 20 = \$10$

Learning Problem for Click Prediction

- Prediction task: $X \rightarrow [0,1]$ $p(click=1|X)$
- Features: $X = (feats\ of\ page, feats\ ad, feats\ user)$
- Data: $(X, Y) \rightarrow (webpage1, ad7, user25, time\ 12) \rightarrow click=1$ (Y)
  - Batch: fixed dataset $(X^1, Y^1) \ldots (X^N, Y^N)$
  - Online: data as a stream: user arrives at a page $\rightarrow X^t \rightarrow$ predict $\hat{Y}$, click? $\rightarrow$ 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): $\frac{1}{1+exp(-z)}$



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

$$P(Y=0|X,W) = \frac{1}{1+exp(w_0+\sum_i w_i X_i)}$$

linear function of features

Features can be discrete or continuous!

Standard v. Regularized Updates

- Maximum conditional likelihood estimate

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

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

- 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. pushes ← more towards zero

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:

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

- Batch gradient ascent updates: O(Nd)

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

avg all data points

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

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

1 data point at a time

AdaGrad in Euclidean Space

- For $\mathcal{W} = \mathbb{R}^d$,
- For each feature dimension,

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

where

$$\eta_{t,i} = \frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}}$$

- 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:

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

- Complexity of updates:
  - Constant in number of data points
  - In number of features?
    - Problem both in terms of computational complexity and sample complexity
- 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 $\mathbf{w}_{\text{Obamacare}} = 0$
  - 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
    - 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) % m
  - 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*:
- Query Q(i): Element i in set?
  -
  -
- 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 -> Many false positives
- Multiple hash tables with independent hash functions
  - Apply $h_1(i),\ldots, h_d(i)$, set all bits to 1
- Query Q(i)?
- 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 & d?
- Simple analysis yields:

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

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

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:

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

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

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

  - 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*:

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

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]?
  - Thus:
  - Return:

Analysis of Count-Min Sketch

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

- Set:

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

- Then, after seeing n elements:

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

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

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

- Applying to the Count-Min sketch:

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?
- 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

- 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
  - ξ : Sign hash function
    - Removes the bias found in Count-Min hashing (see homework)
- Define a "kernel", a projection ϕ for $\mathbf{x}$:

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!
- 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:

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

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 $\mathbf{x}_j$:
- Use normal update as if observation were ϕ(**x**), e.g., for LR using SGD:

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

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 $\mathbf{w}_u$ 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 $\mathbf{w}_u$ 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:
  - Desired effect achieved if *j* includes both
    - just word (for global **w**)
    - word,user (for personalized $\mathbf{w}_u$)

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

- Observe data point **x** for user *u*
  - Dimension does not need to be specified a priori and user can be unknown!
- Compute $\phi(\mathbf{x},u)$:
  - Initialize $\phi(\mathbf{x},u)$
  - For non-zero entries *j* of $\mathbf{x}_j$:
    - E.g., j='Obamacare'
    - Need two contributions to ϕ:
      - Global contribution
      - Personalized Contribution
    - Simply:
- 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*

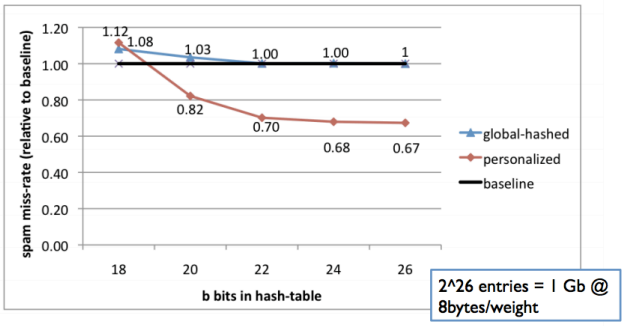


Figure 2. The decrease of uncaught spam over the baseline classifier averaged over all users. The classification threshold was chosen to keep the not-spam misclassification fixed at 1%. The hashed global classifier (*global-hashed*) converges relatively soon, showing that the distortion error ϵ_d vanishes.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

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

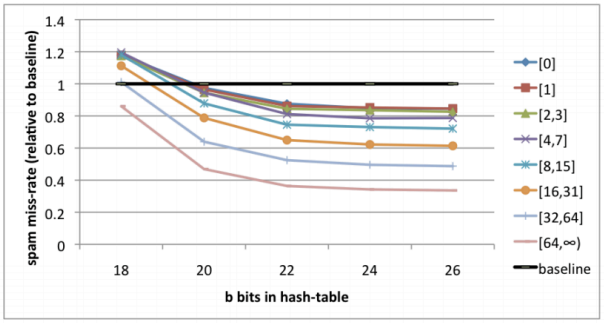


Figure 3. Results for users clustered by training emails. For example, the bucket $[8, 15]$ consists of all users with eight to fifteen training emails. Although users in buckets with large amounts of training data do benefit more from the personalized classifier (up-to 65% reduction in spam), even users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