Case Study 1: Estimating Click Probabilities

Problem 1: Complexity of LR Updates

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

- Complexity of updates:
  - Constant in number of data points
  - In number of features?
    - Problem both in terms of computational complexity and sample complexity
  - What 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
  - Fancy Trees and Tries
    - Hard to implement and maintain
  - Hash tables?

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
Count-Min Sketch: general case

- Keep \( p \) by \( m \) Count matrix
  \[
  \begin{array}{ccc}
  \text{Count}_0(i) \\
  \vdots \\
  \text{Count}_m(i) \\
  \end{array}
  \]
- \( p \) 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

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

- Query \( Q(i) \)?
  - What is in \( \text{Count}[j,k] \)?
    \[
    \text{Count}[j,k] = \sum_{i : h_j(i) = k} a_i \geq a_i
    \]
  - Thus:
    \[
    \text{tightest upper bound.}
    \]
  - Return:
    \[
    a_i = \min_{j} \text{Count}(j, h_j(i)) \geq a_i
    \]
Analysis of Count-Min Sketch

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

- Set:
  \[ 0 \left( \frac{1}{\epsilon} \right) \]
  \[ m = \left\lceil \frac{e}{\epsilon} \right\rceil \quad p = \left\lceil \ln \frac{1}{\delta} \right\rceil \quad O \left( \frac{n}{\delta} \right) \]

- Then, after seeing \( n \) elements:
  \[ \hat{a}_i \leq a_i + \epsilon n \]

- With probability at least \( 1 - \delta \)

---

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

- \( l_{i,j,k} \) = indicator that \( i \) & \( k \) collide on hash \( j \):
  \[ = O(\epsilon) \]

- Bounding expected value:
  \[ \mathbb{E} \left[ \sum_{j} l_{i,j,k} \right] = \mathbb{P}(h_i(i) = h_j(k)) = \frac{1}{m} \]

- \( X_{i,j} \) = total colliding mass on estimate of count of \( i \) in hash \( j \):
  \[ X_{i,j} = \sum_{k \in \mathbb{X}_i} l_{i,j,k} \]

- Bounding colliding mass:
  \[ \mathbb{E}(X_{i,j}) \leq O(n \epsilon) \]

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

• Markov inequality: For \( z_1, \ldots, z_k \) positive iid random variables
  \[ P(\forall z_i : z_i > \alpha \mathbb{E}[z_i]) < \alpha^{-k} \quad \mathbb{E} \left[ \sum_{i=1}^{k} z_i \right] \leq \mathbb{E} \left[ \sum_{i=1}^{k} \alpha z_i \right] \]

• Applying to the Count-Min sketch:

But updates may be positive or negative

\[ w^{(t+1)}_i \leftarrow w^{(t)}_i + \eta_k \left\{ -\lambda w^{(t)}_i + x^{(t)}_i y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)}) \right\} \]

• Count-Min sketch for positive & negative case  
  \( a_i \) no longer necessarily positive

• Update the same: Observe change \( \Delta_i \) to element \( i \):
  \[ \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + \Delta_i \]

  – Each \( \text{Count}[j, h(j)] \) no longer an upper bound on \( a_j \)

• How do we make a prediction?

\[ \hat{a}_i = \text{median}(\text{count} + \sum_{j=1}^{p} h_j(a_j)) \]

• Bound:  
  \[ |\hat{a}_i - a_i| \leq 3\varepsilon \|a\|_1 \]

  – With probability at least 1-\( \delta^{1/4} \), where \( \|a\|_1 = \sum_i |a_i| \)
Finally, Sketching for LR

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

- 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
  - \( \xi \): Sign hash function
    - Removes the bias found in Count-Min hashing (see homework)

- Define a “kernel”, a projection \( \phi \) for \( x \):
  \[
  \phi_i = \sum_j E_{\xi}(x_s h(j)) \xi(j) a_{ij}
  \]
Hash Kernels Preserve Dot Products

\[ \phi_i(x) = \sum_{j: h(j) = i} \xi(j) x_j \]

- Hash kernels provide unbiased estimate of dot-products!
- Variance decreases as \(O(1/m)\)
- Choosing \(m\)? For \(\varepsilon > 0\), if:
  \[ m = \mathcal{O} \left( \frac{\log N}{\varepsilon^2} \right) \]
  - Under certain conditions...
  - Then, with probability at least 1-\(\delta\):
  \[ (1 - \varepsilon) \|x - x'\|^2 \leq \|\phi(x) - \phi(x')\|^2 \leq (1 + \varepsilon) \|x - x'\|^2 \]

Learning With Hash Kernels

- Given hash kernel of dimension \(m\), specified by \(h\) and \(\xi\)
  - Learn \(m\) dimensional weight vector
- Observe data point \(x\)
  - Dimension does not need to be specified a priori!
- Compute \(\phi(x)\):
  - Initialize \(\phi(x)\)
  - For non-zero entries \(j\) of \(x_j\):
    \[ \phi(x) \cdot \omega \leftarrow \frac{\phi(x) \cdot \omega}{1 + \varepsilon \phi(x) \cdot \omega} \]
- Use normal update as if observation were \(\phi(x)\), e.g., for LR using SGD:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \phi(x^{(t)}) y^{(t)} - P(Y = 1|\phi(x^{(t)}), w^{(t)}) \right\} \]
Interesting Application of Hash Kernels: Multi-Task Learning

• Personalized click estimation for many users:
  – One global click prediction vector \( \mathbf{w} \):
    
    \[
    \frac{e^{\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}}
    \]

    • But...
    – A click prediction vector \( \mathbf{w}_u \) per user \( u \):
      
      \[
      \frac{e^{\mathbf{w}_u^\top \mathbf{x}}}{1 + e^{\mathbf{w}_u^\top \mathbf{x}}}
      \]

    • 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 \( \mathbf{w} \) and one \( \mathbf{w}_u \) per user:
  – Prediction for user \( u \):
    
    \[
    \frac{e^{(\mathbf{w} + \mathbf{w}_u)^\top \mathbf{x}}}{1 + e^{(\mathbf{w} + \mathbf{w}_u)^\top \mathbf{x}}}
    \]

  – If we know little about user \( u \):
  – After a lot of data from user \( u \):

©Sham Kakade 2016

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 $u$:

$$z_{x,u} = (x_1 \cdots x_d, 0, 0, \cdots, x_1 \cdots x_d, 0, 0, \cdots)$$

- Estimating click probability as desired:

$$w = \left( \bar{w}, \bar{w}, \cdots, \bar{w}, \cdots \bar{w}_{\#user} \right)$$

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

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

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

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

- Learn as usual using $\phi(x,u)$ instead of $\phi(x)$ 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 $\epsilon_d$ 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 \( \xi \))
  - Quickly generate projection \( \varphi(x) \)
  - 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)