Case Study 1: Estimating Click Probabilities

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: $\text{paid on clicks}$
  - Instead:
    - $\Pr(\text{click} | c_3) = 0.01$
    - $\Pr(\text{click} | c_2) = 0.5$
    - $E[\$3] = 0.01 \times 100 = \$1$
    - $E[\$2] = 0.5 \times 20 = \$10$
Learning Problem for Click Prediction

- Prediction task: $y \rightarrow \{0,1\}$
  - $p(\text{click}=1 | x)$

- Features:
  - $(\text{features of page, features ad, features user})$

- Data:
  - Batch: $(x_i, y_i) \rightarrow (\text{webpage}_1, \text{ad} \_2, \text{user}_3, \text{time}_4) \rightarrow \text{click}$
  - Online: data as a stream
    - as user arrives at a page, predict $y$, click?

- 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

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

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

Logistic function (or Sigmoid):

$$\frac{1}{1 + \exp(-z)}$$

Features can be discrete or continuous!
Standard v. Regularized Updates

- Maximum conditional likelihood estimate
  \[ w^* = \arg\max_w \ln \prod_{j=1}^{N} P(y^j | x^j, w) \]
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 | x^j, w)] \]

- Regularized maximum conditional likelihood estimate
  \[ w^* = \arg\max_w \ln \prod_{j} P(y^j | x^j, w) - \frac{\lambda}{2} \sum_{i>0} w_i^2 \]
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \sum_j x_i^j [y^j - \hat{P}(Y^j = 1 | x^j, w)] \right\} \]

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:
  \[ E_{\mathbf{x}} [\ell(w, x)] = E_{\mathbf{x}} [\ln P(y|x, w) - \frac{\lambda}{2} ||w||^2_2] \]

- Batch gradient ascent updates:
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \left\{ -\lambda w_i^{(t)} + \frac{1}{N} \sum_{j=1}^{N} x_i^{(j)} [y^{(j)} - \hat{P}(Y = 1|x^{(j)}, w^{(t)})] \right\} \]

- Stochastic gradient ascent updates:
  - Online setting: \( x^{(t)}, y^{(t)} \)
    \[ w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta_t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - \hat{P}(Y = 1|x^{(t)}, w^{(t)})] \right\} \]
AdaGrad in Euclidean Space

- For $W = \mathbb{R}^d$, no constraints on $w$
- For each feature dimension,
  
  \[ w_i^{(t+1)} \leftarrow w_i^{(t)} - \eta_t g_{t,i} \]

  where
  \[ \eta_{t,i} = \eta A_{t,i} \]

  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 its own learning rate!

- Adapts with $t$
- Takes geometry of the past observations into account
- Primary role of $\eta$ is determining rate the first time a feature is encountered

Problem 1: Complexity of Update Rules for Logistic Regression

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

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

  What if we have 18 features?

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

©Emily Fox 2014
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

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?

©Emily Fox 2014
Hash Functions and Hash Tables

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

- Simple example: mod
  - \( h(i) = (a \cdot i + b) \mod m \)
  - \( a = 7, \ b = 11, \ m = 32 \)
  - \( i = 4 \Rightarrow h(i) = 39 \mod 32 = 7 \)
  - 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 \( m \):
  - \( v \) 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:
  - \( h(\text{ObamaCare}) = 7 \Rightarrow v(h(\text{ObamaCare})) = 1 \)
  - \( h(\text{Mary}) = 7 \Rightarrow v(h(\text{Mary})) = 0 \)

- 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 \( Q(i) \)?
  - if \( \forall j \ h_j(i) = 1 \)
    - \( Q(i) = \text{very probably yes} \)
  - else \( Q(i) = \text{no} \)

- Significantly decrease probability of false positives

Analysis of Bloom Filter

- Want to keep track of \( n \) elements with false positive probability of \( \delta > 0 \)... how large \( m \) & \( d \)?

- Simple analysis yields:

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

\[
d = \log_2 \frac{1}{\delta}
\]
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:
  \[
  w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta t \left\{ -\lambda w_i^{(t)} + x_i^{(t)} [y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)})] \right\}
  \]
- Must keep track of (weighted) counts of each feature:
  - E.g., with sparse data, for each non-zero dimension \( i \) in \( x^{(t)} \):
    \[
    \text{For all entries of hash - multiply current } w_i^{(t)} \text{ by } (1 - \eta t \lambda)
    \]
    \[
    \text{For all } x_i^{(t)} \neq 0
    \]
    \[
    w_i^{(t+1)} + = x_i^{(t)} \cdot \text{const } \equiv \sum \left( y^{(t)} - P(Y = 1|x^{(t)}, w^{(t)}) \right)
    \]
- 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 \( i \):
    \[
    \text{Count}[h(i)] \leftarrow \text{Count}[h(i)] + 1
    \]
- See 'Mary', 'ObamaCare'
  \[
  \text{Count}['Mary'] = 2 \quad \text{Count}['ObamaCare'] = 7
  \]
  \[
  Q('Mary') = \text{count}['Mary'] = 2 > 1
  \]
- Again, collisions could be a problem:
  - \( a_i \) is the count of element \( i \)

\[
\text{Count}[j] = \sum_{i : h(i) = j} a_i \quad \text{true counts}
\]
\[
\text{Q}(i) \rightarrow \text{return } a_i = \text{count}[h(i)] \geq a_i
\]
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 \( i \):
    \[
    \forall j \in \{1, \ldots, p\} : \text{Count}[j, h_j(i)] \leftarrow \text{Count}[j, h_j(i)] + 1
    \]

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 \( \text{Count}[j,k] \)?
    \[
    \text{Count}[j, k] = \sum_{i : h_j(i) = k} a_i
    \]
  - Thus:
    \[
    Q(i) = \text{each } \text{count}[j, h(i)] \geq a_i
    \]
  - Return:
    \[
    \hat{a_i} = \min_j \text{Count}[j, h_j(i)] \geq a_i
    \]
    \( \hat{a_i} \) is the tightest upper bound for \( a_i \).
Analysis of Count-Min Sketch

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

- Set:
  \[ m = \left\lceil \frac{\epsilon}{\delta} \right\rceil \]
  \[ p = \left\lceil \ln \frac{1}{\delta} \right\rceil \]

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

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

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 \( j \):
  \[ (i \neq k) \land (h_j(i) = h_j(k)) \]

- Bounding expected value:
  \[ E[I_{i,j,k}] = P(h_j(i) = h_j(k)) = \frac{1}{m} \leq \frac{\epsilon}{\delta} \]

- \( X_{i,j} \) = total colliding mass on estimate of count of \( i \) in hash \( j \):
  \[ \overbrace{\sum_{k \neq i} I_{i,j,k}}^{\text{many counts}} a_k \text{add their counts} \]
  \[ \text{Count}_{i,j} = a_i + X_{i,j} \]

- Bounding colliding mass:
  \[ E[X_{i,j}] = \sum_{k \neq i} a_k E[I_{i,j,k}] \leq \frac{n \epsilon}{\delta} \]

- 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: \( \text{Count}[j, h_j(i)] = a_i + X_{i,j} \), \( E[X_{i,j}] \leq \frac{\epsilon}{n} \)

- 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:
  \[
P(a_i > a_i + \epsilon n) = P(\forall j, \text{Count}[j, h_j(i)] > a_i + \epsilon n)
  = P(\forall j, a_i + X_{i,j} > a_i + \epsilon n)
  \leq P(\forall j, X_{i,j} > \epsilon E[X_{i,j}]) < e^{-p} \leq \delta
  \]

But updates may be positive or negative

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

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

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

- How do we make a prediction?
  \( \hat{a}_i = \text{median count} \left[ j, h_j(i) \right] \)

- Bound: \( |\hat{a}_i - a_i| \leq 3\epsilon ||a||_1 \)
  - With probability at least \( 1-5^{1/4} \), where \( ||a|| = \sum_i |a_i| \)
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:

\[
\begin{align*}
&V_{j,k} \text{ Count}_j[k] = (1-h) \text{ count}_j[k] \\
&V_{j,k} \neq 0 \\
&V_j \text{ Count}_j[h_j(i)] + = X_i^{(t)} \cdot \text{ const} \\
&\text{Remember our est. of } w_0^{(t)} : \text{ median Count}_j[h_j(i)]
\end{align*}
\]

- Making a prediction:

\[
\text{Make pred: } -\log \text{ odds } = w_0^{(t)} + \sum_{i: x_i \neq 0} \text{ median Count}_j[h_j(i)] X_i^{(t)}
\]

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

- Define a “kernel”, a projection \( \phi \) for \( x \):

\[
\phi(x) = \begin{cases} 1 & \text{for each non-zero element of } x_i \\
0 & \text{add to bin } h(j) \end{cases} \\
\phi_i(x) = \sum_j \delta_{i,j} x_j
\]
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!

\[ E_{h \sim f} [\phi(x) \cdot \phi(y)] = x \cdot y \]  \hspace{1cm} \text{pf: by homework}

- Variance decreases as \( O(1/m) \) \hspace{1cm} \text{gets better w/ more dims}

- Choosing \( m \)? For \( \epsilon > 0 \), if

\[ m = \mathcal{O} \left( \frac{\log N}{\epsilon^2} \right) \] \hspace{1cm} \text{log in data size}

\[ (1 - \epsilon) ||x - x'||_2^2 \leq ||\phi(x) - \phi(x')||_2^2 \leq (1 + \epsilon) ||x - x'||_2^2 \]