Case Study 1: Estimating Click Probabilities

Stochastic Gradient Descent (continued)

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington

Carlos Guestrin

January 17th, 2013

What is the Perceptron Doing???

- When we discussed logistic regression:
  - Started from maximizing conditional log-likelihood
- When we discussed the Perceptron:
  - Started from description of an algorithm
- What is the Perceptron optimizing????

Perceptron & Stochastic Gradient descent

- Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

complexity in terms of N : O(1)

1 sample estimate of $E[\nabla \ell(w)]$

Stochastic Gradient descent!! :)

with $\eta = 1$

O(N)

- Batch hinge minimization update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \eta \frac{1}{N} \sum_{i=1}^{N} \left\{\mathbb{1}\left[y^{(i)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(i)}) \leq 0\right] y^{(i)}\mathbf{x}^{(i)}\right\}$$

N sample estimate of $E[\nabla \ell(w)]$

Stochastic Gradient Descent: general case

after sufficiently large t, running avg of loss approx. true loss, use for stopping

- Given a stochastic function of parameters: $f(w) = E_x[f(w,x)]$
  - Want to find minimum

$w^* = \arg\min_w E_x[f(w,x)]$

- Start from $\mathbf{w}^{(0)}$; e.g., $w^{(0)} = 0$
- Repeat until convergence:
  - Get a sample data point $\mathbf{x}^{(t)}$
  - Update parameters:

$w^{(t+1)} \leftarrow w^{(t)} - \eta_t \nabla f(w^{(t)}, x^{(t)})$

stopping criteria very hard...

→ theory bounds # iterations in terms of uncomputable constants

→ in practice, use validation set

- Works on the online learning setting!
- Complexity of gradient computation is constant in number of examples!
- In general, step size changes with iterations, e.g., $\eta_t = \frac{k}{t}$ for $k > 0$

Stochastic Gradient Ascent for Logistic Regression

- Logistic loss as a stochastic function:

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

- Batch gradient ascent updates:

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

average over N datapoints

- Stochastic gradient ascent updates:
  - Online setting:

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

↑ 1 point at a time

Convergence rate of SGD

- **Theorem**:
  - (see Nemirovski et al '09 from readings)
  - Let *f* be a strongly convex stochastic function
  - Assume gradient of *f* is Lipschitz continuous and bounded
  - Then, for step sizes:
  - The expected loss decreases as O(1/t):

Convergence rates for gradient descent/ascent versus SGD

- Number of Iterations to get to accuracy

$$\ell(\mathbf{w}^*) - \ell(\mathbf{w}) \leq \epsilon$$

- Gradient descent:
  - If func is strongly convex: O(ln(1/ϵ)) iterations
- Stochastic gradient descent:
  - If func is strongly convex: O(1/ϵ) iterations
- Seems exponentially worse, but much more subtle:
  - Total running time, e.g., for logistic regression:
    - Gradient descent:
    - SGD:
    - SGD can win when we have a lot of data
  - And, when analyzing true error, situation even more subtle… expected running time about the same, see readings

What you need to know

- Perceptron is optimizing hinge loss
- Subgradients and hinge loss
- (Sub)gradient decent for hinge objective
- Objective functions in ML as expectations
- Gradient estimation, rather than objective estimation
- Stochastic gradient descent -> estimate gradient from single training example
  - Mini-batches possible and useful
- Stochastic gradient ascent for logistic regression
- Analysis of stochastic gradient descent
  - Decreasing step size fundamental here
- Comparing analysis of stochastic gradient descent with gradient descent

Case Study 1: Estimating Click Probabilities

Tackling an Unknown Number of Features with Sketching

Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington

Carlos Guestrin

January 17th, 2013

Problem1: Complexity of Update Rules for LR and Perceptron

- Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- 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!
- Recall Perceptron update:

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Must keep track of counts of each feature (weighed by $y^{(t)}$):
  - 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 d by m Count matrix

- d hash functions:
  - Just like in Bloom Filter, decrease errors with multiple hashes
  - Every time see string *i*:

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

Querying the Count-Min Sketch

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

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

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

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

- Applying to the Count-Min sketch:

But Our Updates may be positive or Negative

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

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

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

  - 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 Perceptron

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbb{1}\left[y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) \leq 0\right] y^{(t)}\mathbf{x}^{(t)}$$

- Never need to know size of vocabulary!
- Make a mistake, update Count-Min matrix:
- Making a prediction:
- Scales to huge problems, great practical implications… More next time

What you need to know

- Update rules for perceptron and SGD problematic in high dimensions:
  - Complexity linear in number of features
  - What if we have a new feature?
- 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 Perceptron Learning and Prediction