Case Study 1: Estimating Click Probabilities

L2 Regularization for Logistic Regression

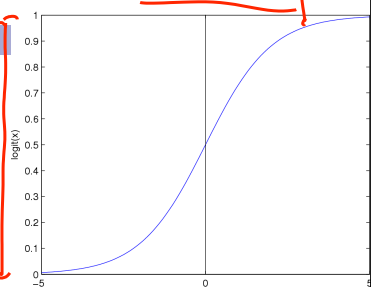
Machine Learning/Statistics for Big Data
CSE599C1/STAT592, University of Washington

Carlos Guestrin

January 10th, 2013

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:

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

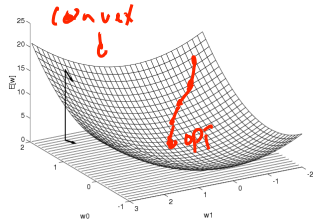
linear function of the features

$w_0 + \sum_i w_i X_i$ — not bounded, could be negative

Features can be discrete or continuous!

Optimizing concave function – Gradient ascent

- Conditional likelihood for Logistic Regression is concave. Find optimum with gradient ascent



Gradient: $\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_n}]'$

Update rule: $\Delta \mathbf{w} = \eta \nabla_{\mathbf{w}} l(\mathbf{w})$

Step size, $\eta > 0$

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \eta \frac{\partial l(\mathbf{w})}{\partial w_i}$$

- Gradient ascent is simplest of optimization approaches
  - e.g., Conjugate gradient ascent much better (see reading)

Gradient Ascent for LR

Gradient ascent algorithm: iterate until change < ε

$$w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \sum_j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

For i=1,…,n,

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

repeat

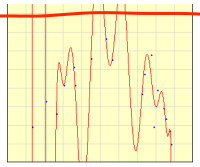
Test set error as a function of model complexity

Regularization in linear regression

- Overfitting usually leads to very large parameter choices, e.g.:

-2.2 + 3.1 X – 0.30 X^2

-1.1 + 4,700,910.7 X – 8,585,638.4 X^2 + …



- Regularized least-squares (a.k.a. ridge regression), for λ>0:

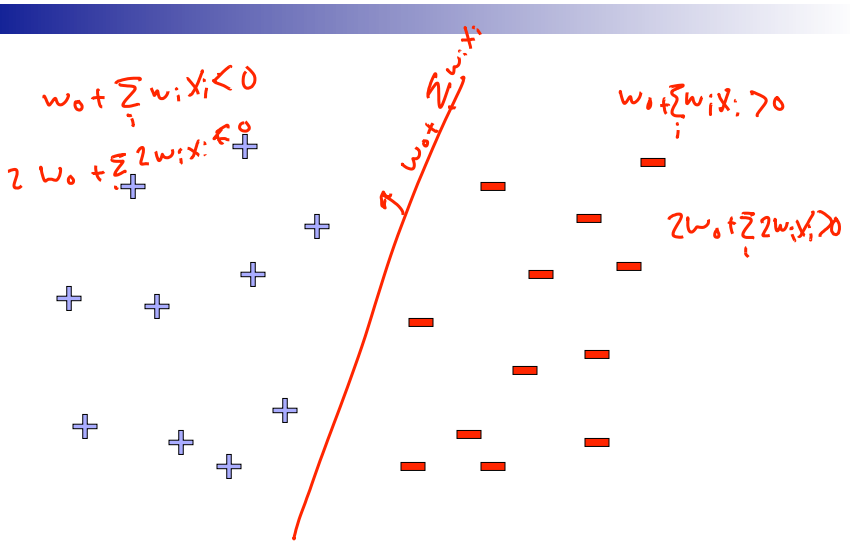
$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

squared error on train data

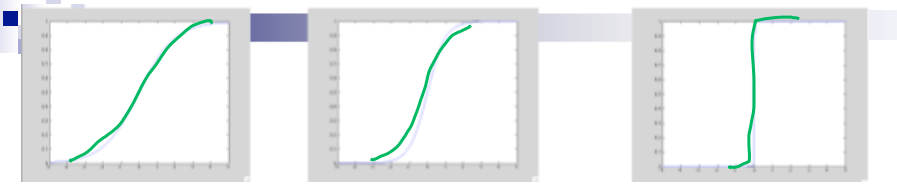
regularization term

$\|w\|_2^2$

Linear Separability



Large parameters → Overfitting



$$\frac{1}{1+e^{-x}} \qquad \frac{1}{1+e^{-2x}} \qquad \frac{1}{1+e^{-100x}}$$

- If data is linearly separable, weights go to infinity
- Leads to overfitting:

- Penalizing high weights can prevent overfitting…

Regularized Conditional Log Likelihood

- Add regularization penalty, e.g., L_2:

$$\ell(\mathbf{w}) = \ln \prod_j P(y^j|\mathbf{x}^j, \mathbf{w}) - \frac{\lambda}{2}||\mathbf{w}||_2^2$$

- Practical note about w_0:

- Gradient of regularized likelihood:

$$\frac{\partial \ell(\mathbf{w})}{\partial w_i} = \frac{\partial}{\partial w_i} \ln \prod_j P(y^j|\mathbf{x}^j, \mathbf{w}) - \lambda w_i$$

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

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \ln \left[\prod_j P(y^j|\mathbf{x}^j, \mathbf{w})) \right] - \lambda \sum_{i>0} w_i^2$$

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

Stopping criterion

for LR, strong concavity constant $\gamma \geq \lambda$

$$\ell(\mathbf{w}) = \ln \prod_j P(y^j|\mathbf{x}^j, \mathbf{w})) - \lambda||\mathbf{w}||_2^2$$

- Regularized logistic regression is strongly concave
  - Negative second derivative bounded away from zero:

$f(x)$: concave $\Leftrightarrow -f''(x) \geq 0$, twice diff

strongly concave: $-f''(x) \geq \gamma \; ; \; \gamma > 0$

- Strong concavity (convexity) is super helpful!!
- For example, for strongly concave $l(\mathbf{w})$:

$$\ell(\mathbf{w}^*) - \ell(\mathbf{w}) \leq \frac{1}{2\lambda}||\nabla\ell(\mathbf{w})||_2^2$$

$\ell(\mathbf{w}^*)$: true; $\ell(\mathbf{w})$: estimate; $\leq \epsilon$; norm of gradient

$||\nabla\ell(w)||_2^2 \leq 2\lambda\epsilon$

Convergence rates for gradient descent/ascent

- Number of Iterations to get to accuracy

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

- If func Lipschitz: $O(1/\epsilon^2)$

$||f(x) - f(y)|| \leq K||x-y||$

- If gradient of func Lipschitz: $O(1/\epsilon)$
- If func is strongly convex: $O(\ln(1/\epsilon))$

constant step size

exponentially fewer iterations

What you should know about Logistic Regression (LR) and Click Prediction

- Click prediction problem:
  - Estimate probability of clicking
  - Can be modeled as logistic regression
- Logistic regression model: Linear model
- Optimize conditional likelihood
- Gradient computation
- Overfitting
- Regularization
- Regularized optimization
- Convergence rates and stopping criterion

Case Study 1: Estimating Click Probabilities

Online Learning Perceptron Algorithm Kernels

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

Carlos Guestrin

January 10th, 2013

Challenge 1: Complexity of Computing Gradients

N - data points

d ⇒ dims

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

estimate probability

O(d)

complexity is O(Nd)

in big data d is large

N can be huge

just to take a small η step

Challenge 2: Data is streaming

- Assumption thus far: **Batch data**
- But, click prediction is a streaming data task:
  - User enters query, and ad must be selected:
    - Observe $\mathbf{x}^j$, and must predict y^j
  - User either clicks or doesn't click on ad:
    - Label y^j is revealed afterwards
      - Google gets a reward if user clicks on ad
  - Weights must be updated for next time:

must predict y^j ✓ online

click?

reward

$w^{(t+1)} \leftarrow w^{(t)} + \Delta$

Online Learning Problem

- At each time step t:
  - Observe features of data point:
    - Note: many assumptions are possible, e.g., data is iid, data is adversarially chosen… details beyond scope of course
  - Make a prediction:
    - Note: many models are possible, we focus on linear models
    - *For simplicity, use vector notation*
  - Observe true label:
    - Note: other observation models are possible, e.g., we don't observe the label directly, but only a noisy version... Details beyond scope of course
  - Update model:

The Perceptron Algorithm [Rosenblatt '58, '62]

- Classification setting: y in {-1,+1}
- Linear model
  - Prediction:
- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe features:
    - Make prediction:
    - Observe true class:
    - Update model:
      - If prediction is not equal to truth

Mistake Bounds

- Algorithm "pays" every time it makes a mistake:

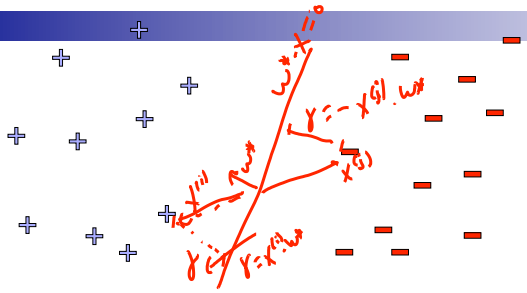
  Loss function. number of mistakes made

  ⇒ Google pays for its mistakes

- How many mistakes is it going to make?

  → Mistake Bound

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector : $\exists\, w^*$, $\|w^*\| = 1$
  - a margin $\gamma > 0$
- Such that

  $\forall t$ if $y^{(t)} = +1$, $w^* \cdot x^{(t)} \geq \gamma$

  if $y^{(t)} = -1$, $-w^* \cdot x^{(t)} \geq \gamma$

  $\forall t$ $y^{(t)}(w^* \cdot x^{(t)}) \geq \gamma$

Perceptron Analysis: Linearly Separable Case

- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples: $(x^{(1)}, y^{(1)}), \ldots, (x^{(T)}, y^{(T)})$
  - Each feature vector has bounded norm: $\forall t \ \|x^{(t)}\| \le R$
  - If dataset is linearly separable: $\exists w^*, \|w^*\| = 1, \forall t \ y^{(t)}(w^* \cdot x^{(t)}) \ge \gamma, \gamma > 0$
- Then the number of mistakes made by the online perceptron on this sequence is bounded by $\left(\frac{R}{\gamma}\right)^2$

wow!! ← constant, independent of T and dim of data d

Perceptron Proof for Linearly Separable case

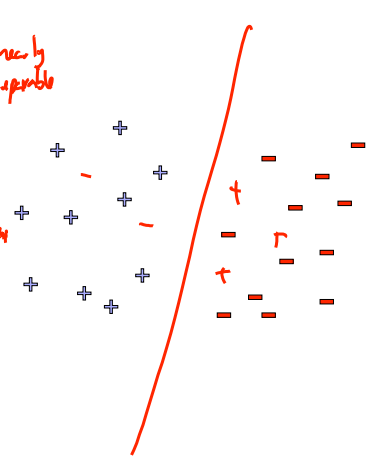
$a \cdot b \le \|a\| \|b\|$

- Every time we make a mistake, we get gamma closer to w^*:
  - Mistake at time t: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$
  - Taking dot product with w^*: $w^* \cdot w^{(t+1)} = w^* \cdot w^{(t)} + y^{(t)}(w^* \cdot x^{(t)}) \ge w^* \cdot w^{(t)} + \gamma$
  - Thus after k mistakes: $w^* \cdot w^{(t+1)} \ge k\gamma$
- Similarly, norm of $w^{(t+1)}$ doesn't grow too fast: I made a mistake: $y^{(t)}(w^{(t)} \cdot x^{(t)}) \le 0$
  - $||\mathbf{w}^{(t+1)}||^2 = ||\mathbf{w}^{(t)}||^2 + 2y^{(t)}(\mathbf{w}^{(t)} \cdot \mathbf{x}^{(t)}) + ||\mathbf{x}^{(t)}||^2 \le ||w^{(t)}||^2 + R^2$
  - Thus, after k mistakes: $\|w^{(t+1)}\|^2 \le kR^2$
- Putting all together: $k\gamma \le w^* \cdot w^{(t+1)} \le \|w^*\| \, \|w^{(t+1)}\| \le \sqrt{k} R$

$\Rightarrow k \le \left(\frac{R}{\gamma}\right)^2$ ← wow!!

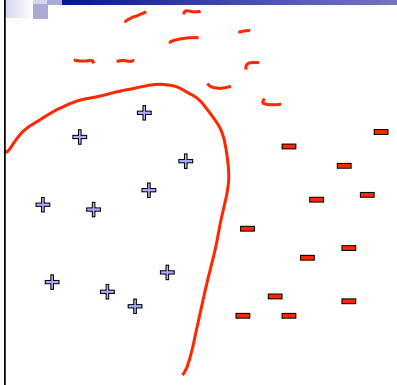
Beyond Linearly Separable Case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an oblivious adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data
  - Constant cost per iteration
    - Converges in O(1/ε)



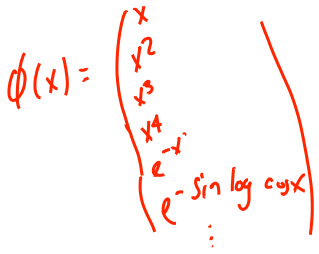
- However, real world not linearly separable
  - Can't expect never to make mistakes again
  - Analysis extends to non-linearly separable case
  - Very similar bound, see Freund & Schapire from Readings
  - Converges, but ultimately may not give good accuracy (make many many many mistakes)

What if the data is not linearly separable?



Use features of features of features….

$$\Phi(\mathbf{x}) : R^m \mapsto F$$



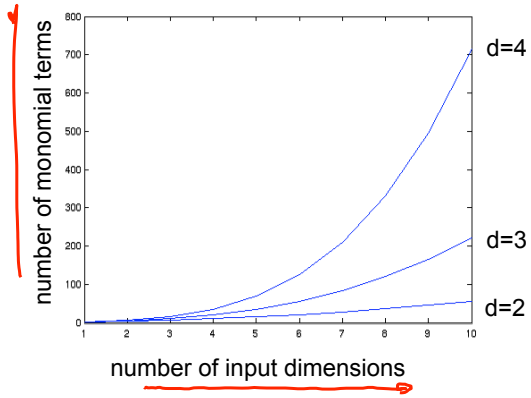
Feature space can get really large really quickly!

Higher order polynomials

$$\text{num. terms} = \binom{d+m-1}{d} = \frac{(d+m-1)!}{d!(m-1)!}$$

dim of ϕ

m – input features
d – degree of polynomial



grows fast!
d = 6, m = 100
about 1.6 billion terms

Perceptron Revisited

- Given weight vector $w^{(t)}$, predict point **x** by:

$$\hat{y} = \text{sign}(w^{(t)} \cdot x)$$

- Mistake at time *t*: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$
- Thus, write weight vector in terms of mistaken data points only:
  - Let $M^{(t)}$ be time steps up to *t* when mistakes were made:

$$w^{(t)} = \sum_{i \in M(t)} y^{(i)} x^{(i)}$$

- Prediction rule now:

$$\text{sign}(w^{(t)} \cdot x) = \text{sign}\left(\left(\sum_{i \in M(t)} y^{(i)} x^{(i)}\right) \cdot x\right) = \text{sign}\left(\sum_{i \in M(t)} y^{(i)} x^{(i)} \cdot x\right)$$

- When using high dimensional features:

$$\text{sign}\left(\sum_{i \in M(t)} y^{(i)} \phi(x^{(i)}) \cdot \phi(x)\right)$$

list of mistakes ever made

dot product between x and mistake i

Dot-product of polynomials

$u = (u_1, u_2)$
$v = (v_1, v_2)$

$\Phi(\mathbf{u}) \cdot \Phi(\mathbf{v}) =$ polynomials of degree exactly d

$d=1$ $\phi(u) \cdot \phi(v) = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2 = u \cdot v$

$d=2$ $\phi(u) \cdot \phi(v) = \begin{pmatrix} u_1^2 \\ u_1 u_2 \\ u_2 u_1 \\ u_2^2 \end{pmatrix} \cdot \begin{pmatrix} v_1^2 \\ v_1 v_2 \\ v_2 v_1 \\ v_2^2 \end{pmatrix} = u_1^2 v_1^2 + 2 u_1 u_2 v_1 v_2 + u_2^2 v_2^2$
$= (u_1 v_1 + u_2 v_2)^2 = (u \cdot v)^2$

proof by one step of induction

if $\phi(\cdot)$ is poly of degree exactly d,

$\phi(u) \cdot \phi(v) = (u \cdot v)^d$

compute in time of basically $u \cdot v$

Finally the Kernel Trick!!! (Kernelized Perceptron

- Every time you make a mistake, remember $(x^{(t)}, y^{(t)})$

- Kernelized Perceptron prediction for $\mathbf{x}$:

$$\text{sign}(\mathbf{w}^{(t)} \cdot \phi(\mathbf{x})) = \sum_{i \in M^{(t)}} \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x})$$
$$= \sum_{i \in M^{(t)}} k(\mathbf{x}^{(i)}, \mathbf{x})$$

Polynomial kernels

- All monomials of degree d in O(d) operations:

$$\Phi(\mathbf{u})\cdot\Phi(\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d = \text{polynomials of degree exactly d}$$

- How about all monomials of degree up to d?
  - Solution 0:
  - Better solution:

Common kernels

- Polynomials of degree exactly d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v})^d$$

- Polynomials of degree up to d

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^d$$

- Gaussian (squared exponential) kernel

$$K(\mathbf{u},\mathbf{v}) = \exp\left(-\frac{||\mathbf{u}-\mathbf{v}||}{2\sigma^2}\right)$$

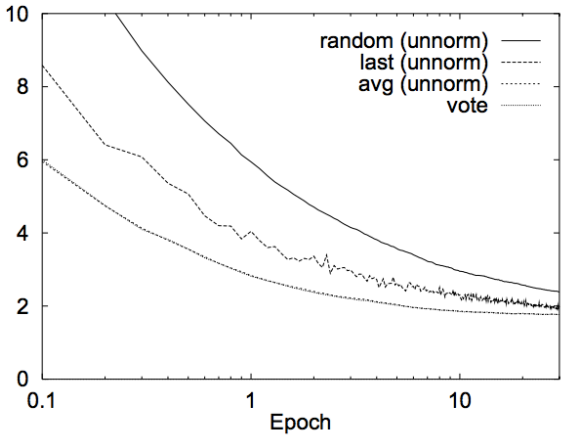
- Sigmoid

$$K(\mathbf{u},\mathbf{v}) = \tanh(\eta\mathbf{u}\cdot\mathbf{v}+\nu)$$

Fundamental Practical Problem for All Online Learning Methods: **Which weight vector to report?**

- Suppose you run online learning method and want to sell your learned weight vector… Which one do you sell???
- Last one?
-
-
-

Choice can make a huge difference!!



[Freund & Schapire '99]

What you need to know

- Notion of online learning
- Perceptron algorithm
- Mistake bounds and proofs
- The kernel trick
- Kernelized Perceptron
- Derive polynomial kernel
- Common kernels
- In online learning, report averaged weights at the end