Case Study 1: Estimating Click Probabilities

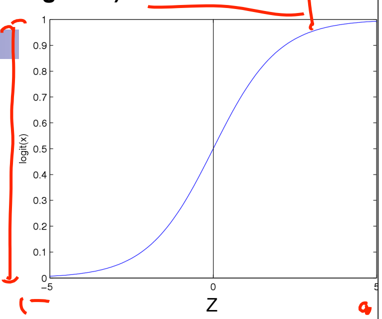
L2 Regularization for Logistic Regression

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:

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

interval [0,1]

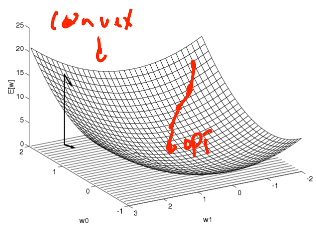
linear function of the features

$w_0 + \sum_i w_i X_i$ ← 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: $\nabla_{\mathbf{w}} l(\mathbf{w}) = [\frac{\partial l(\mathbf{w})}{\partial w_0}, \ldots, \frac{\partial l(\mathbf{w})}{\partial w_n}]'$

Step size, η>0

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

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

- 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

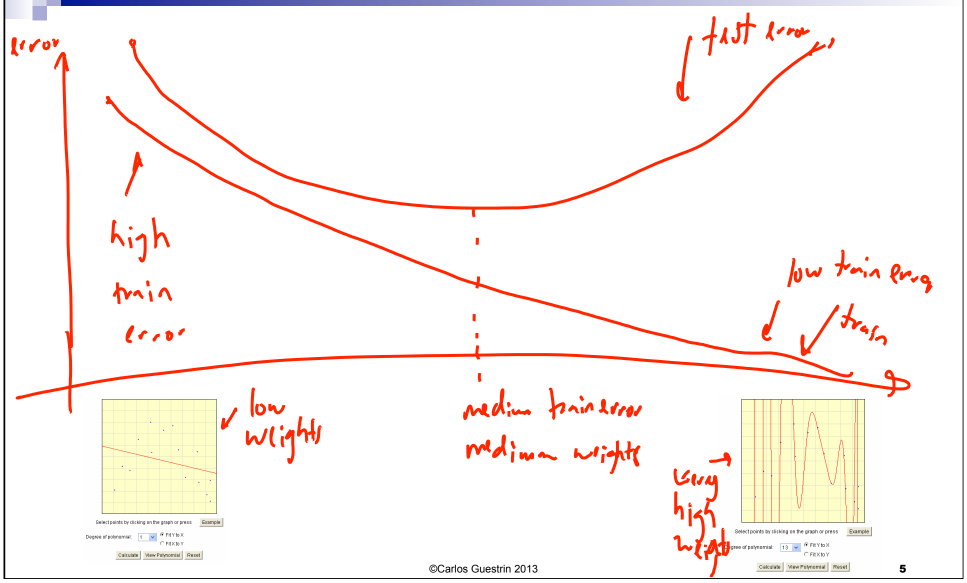
$$w_0^{(t+1)} \leftarrow w_0^{(t)} + \eta \sum_j [y^j - \hat{P}(Y^j = 1 \mid \mathbf{x}^j, \mathbf{w}^{(t)})]$$

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

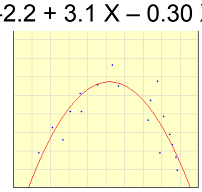
Test set error as a function of model complexity

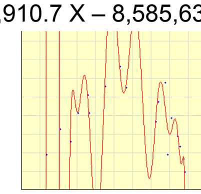


Regularization in linear regression

- 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 + …

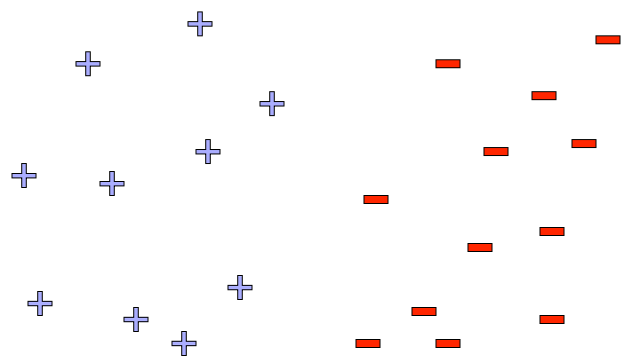




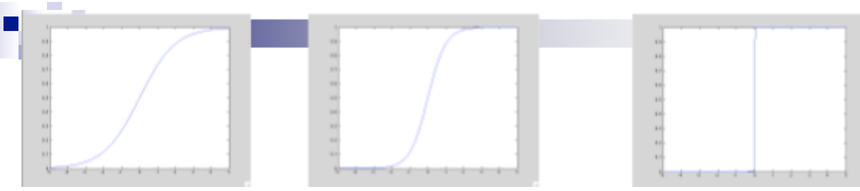
- Regularized least-squares (a.k.a. ridge regression), for $\lambda > 0$:

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

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:

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

- Practical note about w_0:

- Gradient of regularized likelihood:

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

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

- Regularized logistic regression is strongly concave
  - Negative second derivative bounded away from zero:
- Strong concavity (convexity) is super helpful!!
- For example, for strongly concave *l*(**w**):

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

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/ϵ^2)
- If gradient of func Lipschitz: O(1/ϵ)
- If func is strongly convex: O(ln(1/ϵ))

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

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

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:

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:

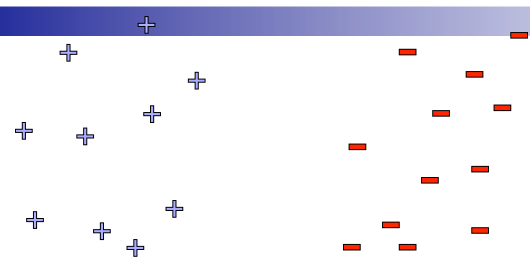
The Perceptron Algorithm [Rosenblatt '58, '62]

- 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:
- How many mistakes is it going to make?

Linear Separability: More formally, Using Margin



- Data linearly separable, if there exists
  - a vector
  - a margin
- Such that

Perceptron Analysis: Linearly Separable Case

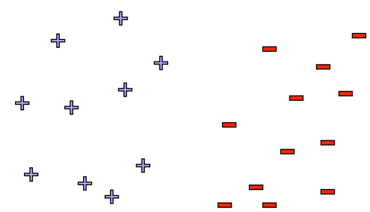
- Theorem [Block, Novikoff]:
  - Given a sequence of labeled examples:
  - Each feature vector has bounded norm:
  - If dataset is linearly separable:
- Then the number of mistakes made by the online perceptron on this sequence is bounded by

Perceptron Proof for Linearly Separable case

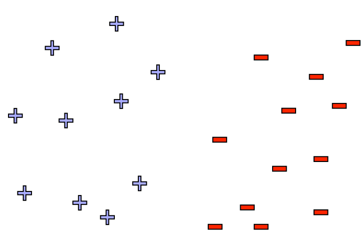
- Every time we make a mistake, we get gamma closer to w^*:
  - Mistake at time t: $w^{(t+1)} = w^{(t)} + y^{(t)} x^{(t)}$
  - Taking dot product with w^*:
  - Thus after k mistakes:
- Similarly, norm of $w^{(t+1)}$ doesn't grow too fast:
  - $||\mathbf{w}^{(t+1)}||^2 = ||\mathbf{w}^{(t)}||^2 + 2y^{(t)}(\mathbf{w}^{(t)}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

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?



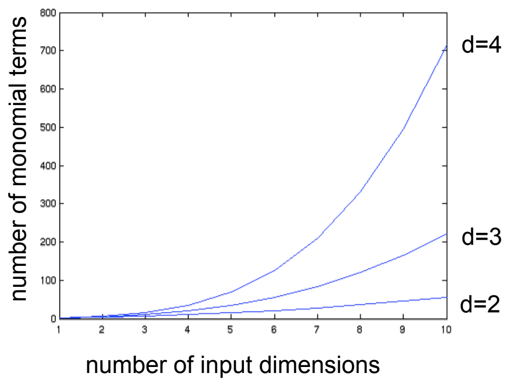
Use features of features of features of features….

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

Feature space can get really large really quickly!

Higher order polynomials

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



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:
- 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:
- Prediction rule now:
- When using high dimensional features:

Dot-product of polynomials

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

Finally the Kernel Trick!!! (Kernelized Perceptron

- Every time you make a mistake, remember $(x^{(t)},y^{(t)})$
- Kernelized Perceptron prediction for **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

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

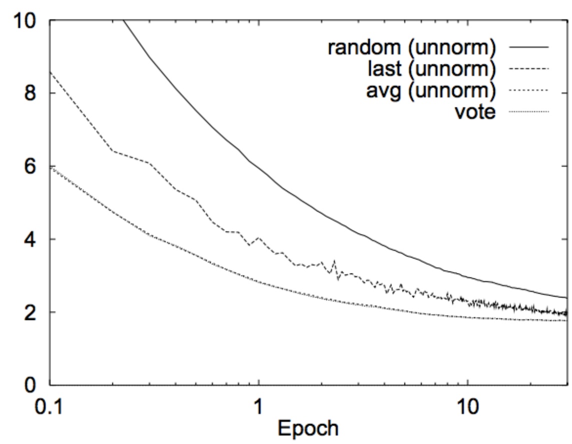
- Sigmoid

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

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