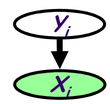
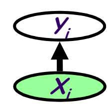
# Generative vs. Discriminative Classifiers

- Goal: Wish to learn f: X → Y, e.g., P(Y|X)

- Generative classifiers (e.g., Naïve Bayes):
  - Assume some functional form for P(X|Y), P(Y)
    This is a ***'generative'*** model of the data!
  - Estimate parameters of P(X|Y), P(Y) directly from training data
  - Use Bayes rule to calculate P(Y|X= x)




- Discriminative classifiers:
  - Directly assume some functional form for P(Y|X)
    This is a ***'discriminative'*** model of the data!
  - Estimate parameters of P(Y|X) directly from training data







# Naïve Bayes vs Logistic Regression

- Consider Y boolean, X continuous, X=<X[1] ... X[m]>
- Number of parameters to estimate:

NB:

$$p(y \mid \mathbf{x}) = \frac{\pi_k \exp\left\{-\sum_j \left(\frac{1}{2\sigma_{k,j}^2}(x_j-\mu_{k,j})^2 - \log\sigma_{k,j} - C\right)\right\}}{\sum_{k'} \pi_{k'} \exp\left\{-\sum_j \left(\frac{1}{2\sigma_{k',j}^2}(x_j-\mu_{k',j})^2 - \log\sigma_{k',j} - C\right)\right\}} \quad **$$

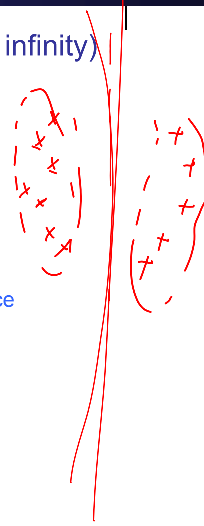
LR:

$$\mu(x) = \frac{1}{1+e^{-\theta^T x}}$$

- Estimation method:
  - NB parameter estimates are uncoupled
  - LR parameter estimates are coupled

# Naïve Bayes vs Logistic Regression

- Asymptotic comparison (# training examples → infinity)
- when model assumptions correct
  - NB, LR produce identical classifiers
- when model assumptions incorrect
  - LR is less biased – does not assume conditional independence
  - therefore expected to outperform NB






# Naïve Bayes vs Logistic Regression

- Non-asymptotic analysis (see [Ng & Jordan, 2002] )
- convergence rate of parameter estimates – how many training examples needed to assure good estimates?

  NB order log m (where m = # of attributes in X)
  LR order m
- NB converges more quickly to its (perhaps less helpful) asymptotic estimates

# Rate of convergence: logistic regression

- Let $h_{Dis,m}$ be logistic regression trained on $n$ examples in $m$ dimensions. Then with high probability:

$$\epsilon(h_{Dis,n}) \leq \epsilon(h_{Dis,\infty}) + O\left(\sqrt{\frac{m}{n}\log\frac{n}{m}}\right)$$

- Implication: if we want $\epsilon(h_{Dis,m}) \leq \epsilon(h_{Dis,\infty}) + \epsilon_0$ for some small constant $\varepsilon_0$, it suffices to pick order $m$ examples

  → Convergences to its asymptotic classifier, in order $m$ examples

  - result follows from Vapnik's structural risk bound, plus fact that the "VC Dimension" of an $m$-dimensional linear separators is $m$



# Rate of convergence: naïve Bayes parameters

- Let any $\varepsilon_1$, $\delta$>0, and any $n \geq 0$ be fixed.
  Assume that for some fixed $\rho_0$ > 0,
  we have that $\rho_0 \leq p(y = T) \leq 1 - \rho_0$

- Let $n = O((1/\epsilon_1^2)\log(m/\delta))$

- Then with probability at least 1-$\delta$, after $n$ examples:

  1. For discrete input, $|\hat{p}(x_i|y=b) - p(x_i|y=b)| \leq \epsilon_1$, $|\hat{p}(y=b) - p(y=b)| \leq \epsilon_1$ for all $i$ and $b$
  2. For continuous inputs, $|\hat{\mu}_{i|y=b} - \mu_{i|y=b}| \leq \epsilon_1$, $|\hat{\sigma}^2_{i|y=b} - \sigma^2_{i|y=b}| \leq \epsilon_1$ for all $i$ and $b$

# Some experiments from UCI data sets

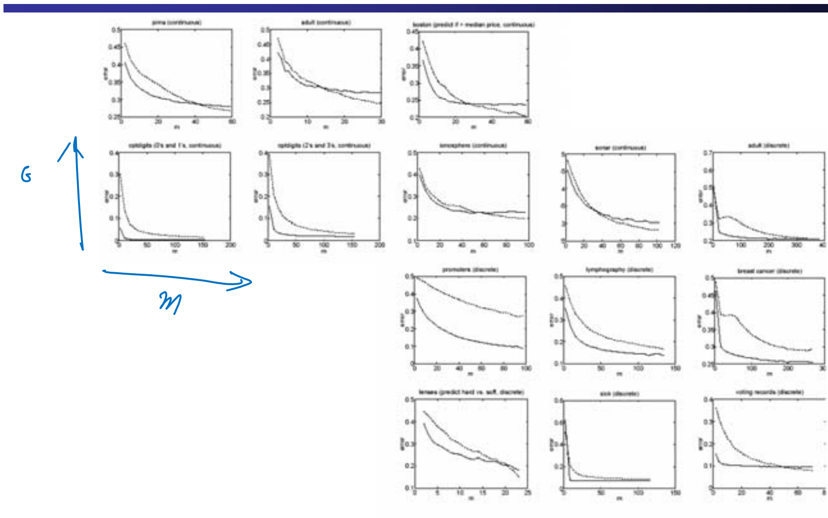


Figure 1: Results of 15 experiments on datasets from the UCI Machine Learnin repository. Plots are of generalization error vs. $m$ (averaged over 1000 randor train/test splits). Dashed line is logistic regression; solid line is naive Bayes.



# Summary

- Naïve Bayes classifier
  - What's the assumption
  - Why we use it
  - How do we learn it
- Logistic regression
  - Functional form follows from Naïve Bayes assumptions
  - For Gaussian Naïve Bayes assuming variance
  - For discrete-valued Naïve Bayes too
  - But training procedure picks parameters without the conditional independence assumption
- Gradient ascent/descent
  - – General approach when closed-form solutions unavailable
- Generative vs. Discriminative classifiers
  - – Bias vs. variance tradeoff

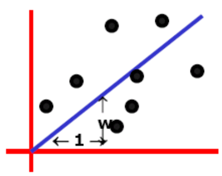
# Machine Learning

**10-701/15-781, Fall 2011**

## Linear Regression and Sparsity

**Eric Xing**

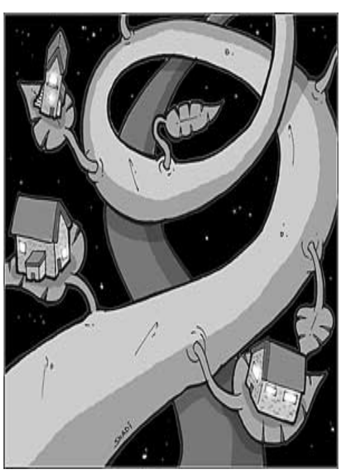
**Lecture 4, September 21, 2011**




**Reading:**

# Machine learning for apartment hunting



- Now you've moved to Pittsburgh!!

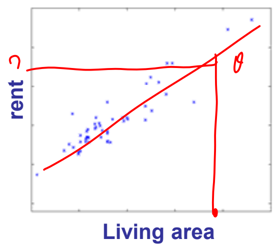
  And you want to find the **most reasonably priced** apartment satisfying your **needs:**

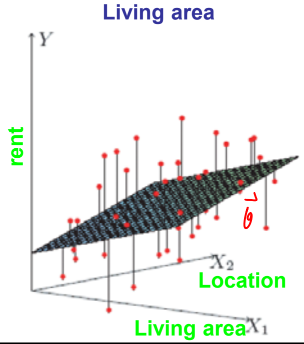
  - square-ft., # of bedroom, distance to campus …

| Living area ($ft^2$) | # bedroom | Rent ($) |
|---|---|---|
| 230 | 1 | 600 |
| 506 | 2 | 1000 |
| 433 | 2 | 1100 |
| 109 | 1 | 500 |
| … | | |
| 150 | 1 | ? |
| 270 | 1.5 | ? |

# The learning problem



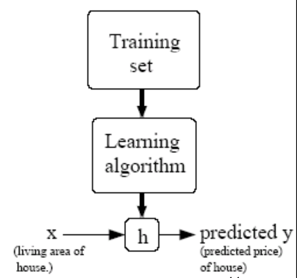




- Features:
  - Living area, distance to campus, # bedroom …
  - Denote as $\mathbf{x}=[x^1, x^2, \ldots x^k]$
- Target:
  - Rent
  - Denoted as $y$
- Training set:

Our goal:

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1 & -- \\ -- & \mathbf{x}_2 & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n & -- \end{bmatrix} = \begin{bmatrix} x_1^1 & x_1^2 & \ldots & x_1^k \\ x_2^1 & x_2^2 & \ldots & x_2^k \\ \vdots & \vdots & \vdots & \vdots \\ x_n^1 & x_n^2 & \ldots & x_n^k \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} - & \mathbf{y}_1 & - \\ - & \mathbf{y}_2 & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{y}_n & - \end{bmatrix}$$






# Linear Regression

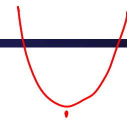
- Assume that Y (target) is a linear function of X (features):
  - e.g.:

$$\hat{y} = \theta_0 + \theta_1 x^1 + \theta_2 x^2$$

  - let's assume a vacuous "feature" $X^0$=1 (this is the intercept term, why?), and define the feature vector to be:
  - then we have the following general representation of the linear function:
- Our goal is to pick the optimal $\theta$ . How!
  - We seek $\theta$ that minimize the following **cost function**:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\hat{y}_i(\vec{x}_i) - y_i)^2$$

# The Least-Mean-Square (LMS) method



- The Cost Function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$

- Consider a **gradient descent** algorithm:

$$\theta_j^{t+1} = \theta_j^{t} - \alpha \frac{\partial}{\partial \theta_j} J(\theta)\Big|_t$$



# The Least-Mean-Square (LMS) method

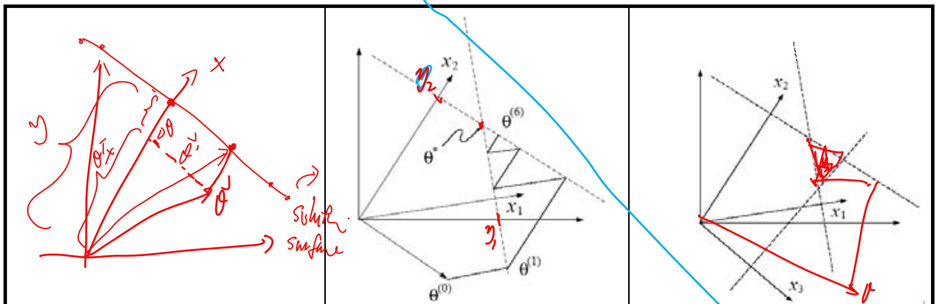
- Now we have the following descent rule:

$$\theta_j^{t+1} = \theta_j^{t} + \alpha \sum_{i=1}^{n}(y_i - \bar{\mathbf{x}}_i^T\theta^t)x_i^j$$

- For a single training point, we have:

  - This is known as the LMS update rule, or the Widrow-Hoff learning rule
  - This is actually a "**stochastic**", "**coordinate**" descent algorithm
  - This can be used as a **on-line** algorithm

## Geometric and Convergence of LMS



| N=1 | N=2 | N=3 |
|---|---|---|

$$\theta^{t+1} = \theta^t + \alpha(y_i - \vec{\mathbf{x}}_i^{\,T}\theta^t)\vec{\mathbf{x}}_i$$

**Claim: when the step size $\alpha$ satisfies certain condition, and when certain other technical conditions are satisfied, LMS will converge to an "optimal region".**

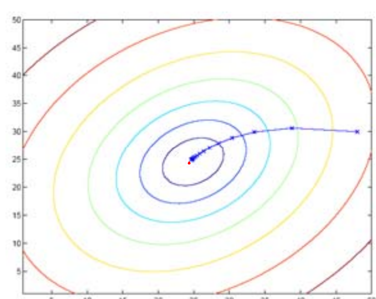


## Steepest Descent and LMS

- Steepest descent
  - Note that:

$$\nabla_\theta J = \left[\frac{\partial}{\partial\theta_1}J, \ldots, \frac{\partial}{\partial\theta_k}J\right]^T = -\sum_{i=1}^{n}(y_n - \mathbf{x}_n^{\,T}\theta)\mathbf{x}_n$$

$$\theta^{t+1} = \theta^t + \alpha\sum_{i=1}^{n}(y_n - \mathbf{x}_n^{\,T}\theta^t)\mathbf{x}_n$$



  - This is as a **batch** gradient descent algorithm

# The normal equations

- Write the cost function in matrix form:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$
$$= \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y})$$
$$= \frac{1}{2}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)$$

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1 & -- \\ -- & \mathbf{x}_2 & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n & -- \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- To minimize $J(\theta)$, take derivative and set to zero:

$$\nabla_\theta J = \frac{1}{2}\nabla_\theta \mathrm{tr}\left(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right)$$
$$= \frac{1}{2}\left(\nabla_\theta \mathrm{tr}\theta^T X^T X\theta - 2\nabla_\theta \mathrm{tr}\vec{y}^T X\theta + \nabla_\theta \mathrm{tr}\vec{y}^T\vec{y}\right)$$
$$= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right)$$
$$= X^T X\theta - X^T\vec{y} = 0$$

$$\Rightarrow X^T X\theta = X^T\vec{y}$$

**The normal equations**

$$\Downarrow$$

$$\theta^* = \left(X^T X\right)^{-1} X^T\vec{y}$$



# Some matrix derivatives

- For $f : \mathbb{R}^{m\times n} \mapsto \mathbb{R}$, define:

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial}{\partial A_{11}} f & \cdots & \frac{\partial}{\partial A_{1n}} f \\ \vdots & \ddots & \vdots \\ \frac{\partial}{\partial A_{1m}} f & \cdots & \frac{\partial}{\partial A_{mn}} f \end{bmatrix}$$

- Trace:

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{ii}, \qquad \mathrm{tr}a = a, \qquad \mathrm{tr}ABC = \mathrm{tr}CAB = \mathrm{tr}BCA$$

- Some fact of matrix derivatives (without proof)

$$\nabla_A \mathrm{tr}AB = B^T, \qquad \nabla_A \mathrm{tr}ABA^T C = CAB + C^T AB^T, \qquad \nabla_A |A| = |A|\left(A^{-1}\right)^T$$

# Comments on the normal equation

- In most situations of practical interest, the number of data points $N$ is larger than the dimensionality $k$ of the input space and the matrix $\mathbf{X}$ is of full column rank. If this condition holds, then it is easy to verify that $X^TX$ is necessarily invertible.
- The assumption that $X^TX$ is invertible implies that it is positive definite, thus the critical point we have found is a minimum.
- What if $\mathbf{X}$ has less than full column rank? → regularization (later).




# Direct and Iterative methods

- Direct methods: we can achieve the solution in a single step by solving the normal equation
  - Using Gaussian elimination or QR decomposition, we converge in a finite number of steps
  - It can be infeasible when data are streaming in in real time, or of very large amount
- Iterative methods: stochastic or steepest gradient
  - Converging in a limiting sense
  - But more attractive in large practical problems
  - Caution is needed for deciding the learning rate $\alpha$

## Convergence rate

- **Theorem**: the steepest descent equation algorithm converge to the minimum of the cost characterized by normal equation:

$$\theta^{(\infty)} = (X^T X)^{-1} X^T y$$

If

$$0 < \alpha < 2/\lambda_{\max}[X^T X]$$

- A formal analysis of LMS need more math-mussels; in practice, one can use a small $\alpha$, or gradually decrease $\alpha$.




## A Summary:

- LMS update rule

$$\theta_j^{t+1} = \theta_j^{t} + \alpha (y_n - \mathbf{x}_n^T \theta^t) x_{n,i}$$

  - Pros: on-line, low per-step cost, fast convergence and perhaps less prone to local optimum
  - Cons: convergence to optimum not always guaranteed
- Steepest descent

$$\theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} (y_n - \mathbf{x}_n^T \theta^t) \mathbf{x}_n$$

  - Pros: easy to implement, conceptually clean, guaranteed convergence
  - Cons: batch, often slow converging
- Normal equations

$$\theta^* = \left(X^T X\right)^{-1} X^T \vec{y}$$

  - Pros: a single-shot algorithm! Easiest to implement.
  - Cons: need to compute pseudo-inverse $(X^T X)^{-1}$, expensive, numerical issues (e.g., matrix is singular ..), although there are ways to get around this …

# Geometric Interpretation of LMS

- The predictions on the training data are:

$$\hat{\vec{y}} = X\theta^* = X\left(X^T X\right)^{-1} X^T \vec{y}$$

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1 & -- \\ -- & \mathbf{x}_2 & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n & -- \end{bmatrix}$$

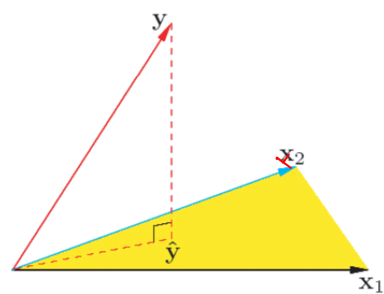
- Note that

$$\hat{\vec{y}} - \vec{y} = \left(X\left(X^T X\right)^{-1} X^T - I\right)\vec{y}$$

and

$$X^T\left(\hat{\vec{y}} - \vec{y}\right) = X^T\left(X\left(X^T X\right)^{-1} X^T - I\right)\vec{y}$$
$$= \left(X^T X\left(X^T X\right)^{-1} X^T - X^T\right)\vec{y}$$
$$= 0 \quad !!$$

$\hat{\vec{y}}$ is the orthogonal projection of $\vec{y}$ into the space spanned by the column of X



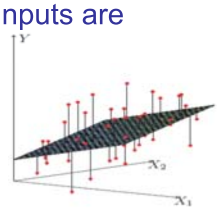

© Eric Xing @ CMU, 2006-2011 23

# Probabilistic Interpretation of LMS

p(y|x)

- Let us assume that the target variable and the inputs are related by the equation:

$$y_i = \theta^T \mathbf{x}_i + \varepsilon_i$$



where $\varepsilon$ is an error term of unmodeled effects or random noise

- Now assume that $\varepsilon$ follows a Gaussian $N(0,\sigma)$, then we have:

$$p(y_i \mid x_i; \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

- By independence assumption:

$$L(\theta) = \prod_{i=1}^{n} p(y_i \mid x_i; \theta) = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n \exp\left(-\frac{\sum_{i=1}^{n}(y_i - \theta^T \mathbf{x}_i)^2}{2\sigma^2}\right)$$

© Eric Xing @ CMU, 2006-2011 24

## Probabilistic Interpretation of LMS, cont.

- Hence the log-likelihood is: $J(\theta)$

$$l(\theta) = n \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2}\frac{1}{2}\sum_{i=1}^{n}(y_i - \theta^T \mathbf{x}_i)^2$$

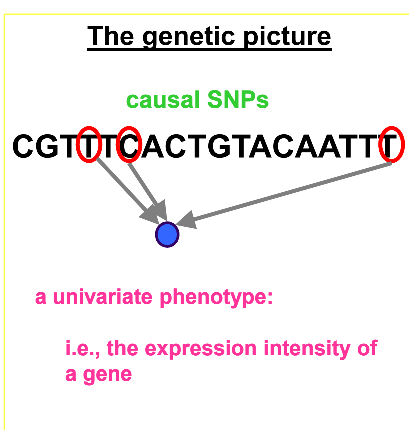
- Do you recognize the last term?

Yes it is: $$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$$

- Thus under independence assumption, LMS is equivalent to MLE of $\theta$ !



## Case study: predicting gene expression



# Association Mapping as Regression

| | Phenotype (BMI) | Genotype |
|---|---|---|
| Individual 1 | 2.5 | . . C . . . . . T . . C . . . . . . . . T . . .<br>. . C . . . . . A . . C . . . . . . . . T . . . |
| Individual 2 | 4.8 | . . G . . . . . A . . G . . . . . . . . A . . .<br>. . C . . . . . T . . C . . . . . . . . T . . . |
| ⋮ | | |
| Individual N | 4.7 | . . G . . . . . T . . C . . . . . . . . T . . .<br>. . G . . . . . T . . G . . . . . . . . T . . . |

Benign SNPs          Causal SNP

# Association Mapping as Regression

| | Phenotype (BMI) | Genotype |
|---|---|---|
| Individual 1 | 2.5 | . . 0 . . . . . 1 . . 0 . . . . . . . 0 . . . |
| Individual 2 | 4.8 | . . 1 . . . . . 1 . . 1 . . . . . . . 1 . . . |
| ⋮ | | |
| Individual N | 4.7 | . . 2 . . . . . 2 . . 1 . . . . . . . 0 . . . |

$$\mathbf{y}_i = \sum_{j=1}^{J} x_{ij}\beta_j$$

SNPs with large $|\beta_j|$ are relevant

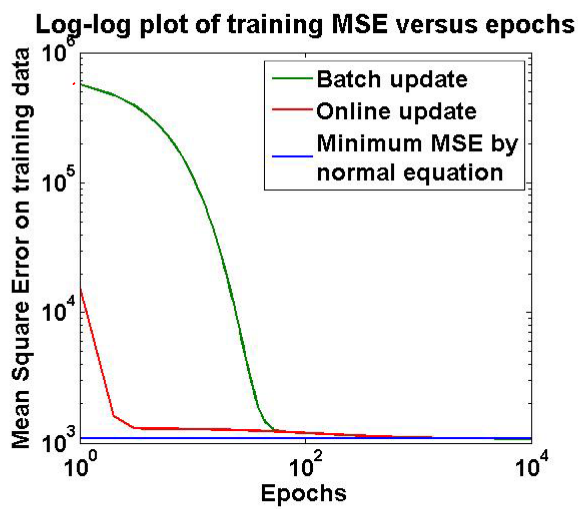
 

## Experimental setup

- Asthama dataset
  - 543 individuals, genotyped at 34 SNPs
  - Diploid data was transformed into 0/1 (for homozygotes) or 2 (for heterozygotes)
  - X=543x34 matrix
  - Y=Phenotype variable (continuous)
- A single phenotype was used for regression
- Implementation details
  - Iterative methods: Batch update and online update implemented.
  - For both methods, step size α is chosen to be a small fixed value ($10^{-6}$). This choice is based on the data used for experiments.
  - Both methods are only run to a maximum of 2000 epochs or until the change in training MSE is less than $10^{-4}$

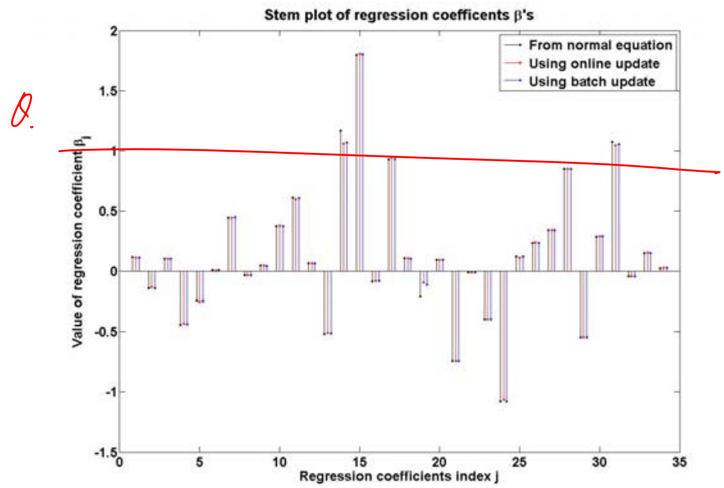



## Convergence Curves




- For the batch method, the training MSE is initially large due to uninformed initialization
- In the online update, N updates for every epoch reduces MSE to a much smaller value.

# The Learned Coefficients



# Multivariate Regression for Trait Association Analysis

| Trait | | Genotype | | Association Strength |
|---|---|---|---|---|
| 2.1 | = | TGAACCATGAAGTA | x | ? |
| $y$ | = | $X$ | x | $\beta$ |

## Multivariate Regression for Trait Association Analysis


$$\beta^* = \arg\min_{\beta} (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)$$

Many non-zero associations:
Which SNPs are truly significant?



## Sparsity

- One common assumption to make **sparsity**.
- **Makes biological sense:** each phenotype is likely to be associated with a small number of SNPs, rather than all the SNPs.
- **Makes statistical sense:** Learning is now feasible in high dimensions with small sample size

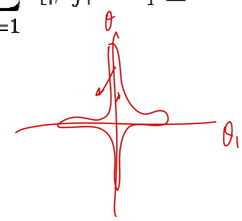
## Sparsity: In a mathematical sense

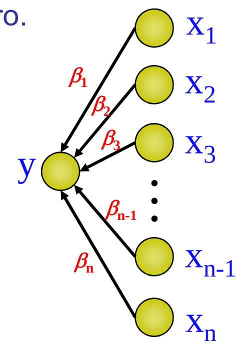
- Consider least squares linear regression problem:
- Sparsity means most of the beta's are zero.

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2$$

subject to:

$$\sum_{j=1}^{p} \mathbb{I}[|\beta_j| > 0] \leq C$$






- But this is not convex!!! Many local optima, computationally intractable.



## L1 Regularization (LASSO)

**(Tibshirani, 1996)**

- A convex relaxation.

Constrained Form

$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2$$

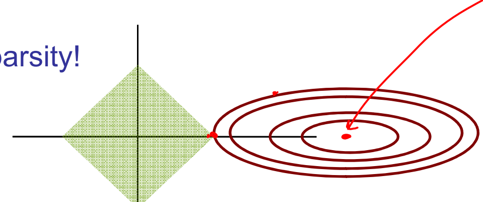
subject to:

$$\sum_{j=1}^{p} |\beta_j| \leq C$$

Lagrangian Form

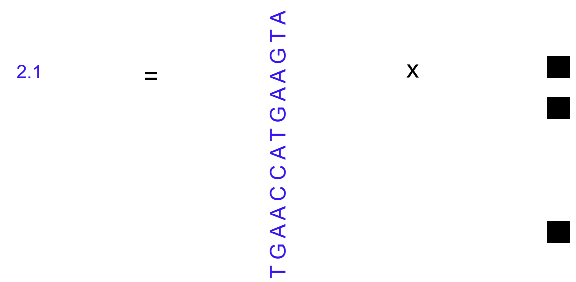
$$\hat{\boldsymbol{\beta}} = \text{argmin}_{\beta} \|\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1$$

- Still enforces sparsity!



# Lasso for Reducing False Positives

Trait | Genotype | Association Strength




**Lasso Penalty** for sparsity

$$\beta^* = \arg\min_{\beta} (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta) + \lambda \sum_{j=1}^{J} |\beta_j|$$

Many zero associations (**sparse** results), but what if there are multiple related traits?



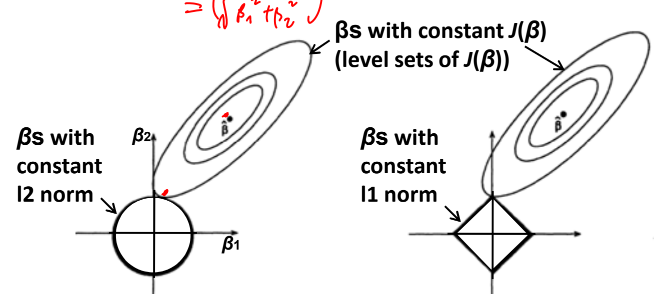
# Ridge Regression vs Lasso

$$\min_{\beta} (\mathbf{X}\beta - \mathbf{Y})^T (\mathbf{X}\beta - \mathbf{Y}) + \lambda \text{pen}(\beta) = \min_{\beta} J(\beta) + \lambda \text{pen}(\beta)$$

small

**Ridge Regression:**
$\text{pen}(\beta) = \|\beta\|_2^2$

**Lasso:** **HOT !**
$\text{pen}(\beta) = \|\beta\|_1$




**Lasso (l1 penalty) results in sparse solutions – vector with more zero coordinates**
**Good for high-dimensional problems – don't have to store all coordinates!**

# Bayesian Interpretation

- Treat the distribution parameters $\theta$ also as a *random variable*
- The *a posteriori* distribution of $\theta$ after seem the data is:

$$p(\theta \mid D) = \frac{p(D \mid \theta) p(\theta)}{p(D)} = \frac{p(D \mid \theta) p(\theta)}{\int p(D \mid \theta) p(\theta) d\theta}$$

This is Bayes Rule

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{marginal likelihood}}$$

**Bayes, Thomas (1763)** An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London,* **53:370-418**

**The prior p(.) encodes our prior knowledge about the domain**





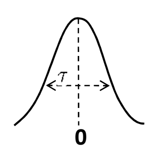
# Regularized Least Squares and MAP

**What if $(X^TX)$ is not invertible ?**

$$\hat{\beta}_{\text{MAP}} = \arg\max_{\beta} \underbrace{\log p(\{(X_i, Y_i)\}_{i=1}^{n} \mid \beta, \sigma^2)}_{\text{log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

**I) Gaussian Prior**

$$\beta \sim \mathcal{N}(0, \tau^2 \mathrm{I}) \qquad p(\beta) \propto e^{-\beta^T \beta / 2\tau^2}$$




$$\hat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i \beta)^2 + \lambda \|\beta\|_2^2$$

$\lambda$ ↓ constant$(\sigma^2, \tau^2)$

**Ridge Regression**

**Closed form: HW**

**Prior belief that β is Gaussian with zero-mean biases solution to "small" β**

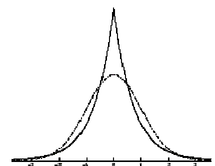
# Regularized Least Squares and MAP

**What if ($X^TX$) is not invertible ?**

$$\widehat{\beta}_{\text{MAP}} = \arg\max_{\beta} \underbrace{\log p(\{(X_i,Y_i)\}_{i=1}^n | \beta, \sigma^2)}_{\text{log likelihood}} + \underbrace{\log p(\beta)}_{\text{log prior}}$$

**II) Laplace Prior**

$$\beta_i \stackrel{iid}{\sim} \text{Laplace}(0,t) \qquad p(\beta_i) \propto e^{-|\beta_i|/t}$$



$$\widehat{\beta}_{\text{MAP}} = \arg\min_{\beta} \sum_{i=1}^{n} (Y_i - X_i\beta)^2 + \lambda \|\beta\|_1$$

$\lambda \rightarrow \text{constant}(\sigma^2, t)$

**Lasso**

**Closed form: HW**

**Prior belief that β is Laplace with zero-mean biases solution to "small" β**





# Beyond basic LR

- LR with non-linear basis functions
- Locally weighted linear regression
- Regression trees and Multilinear Interpolation

## Non-linear functions:



## LR with non-linear basis functions

- LR does not mean we can only deal with linear relationships
- We are free to design (non-linear) features under LR

$$y = \theta_0 + \sum_{j=1}^{m} \theta_j \phi(x) = \theta^T \phi(x)$$

where the $\phi_j(x)$ are fixed basis functions (and we define $\phi_0(x) = 1$).

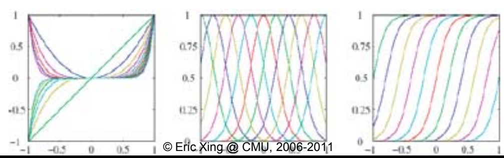
- Example: polynomial regression:

$$\phi(x) := \left[1, x, x^2, x^3\right]$$

- We will be concerned with estimating (distributions over) the weights $\theta$ and choosing the model order $M$.

# Basis functions

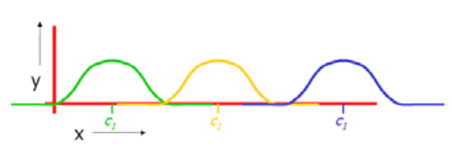
- There are many basis functions, e.g.:
  - Polynomial $\phi_j(x) = x^{j-1}$
  - Radial basis functions $\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right)$
  - Sigmoidal $\phi_j(x) = \sigma\left(\frac{x-\mu_j}{s}\right)$
  - Splines, Fourier, Wavelets, etc





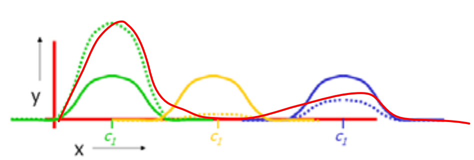

# 1D and 2D RBFs

- 1D RBF




$y^{est} = \beta_1\phi_1(x) + \beta_2\phi_2(x) + \beta_3\phi_3(x)$

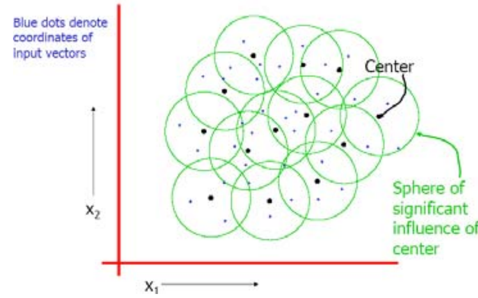
- After fit:



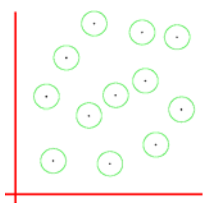

$y^{est} = 2\phi_1(x) + 0.05\phi_2(x) + 0.5\phi_3(x)$

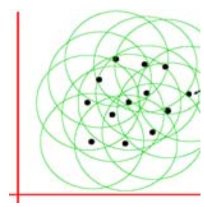
# Good and Bad RBFs

- A good 2D RBF



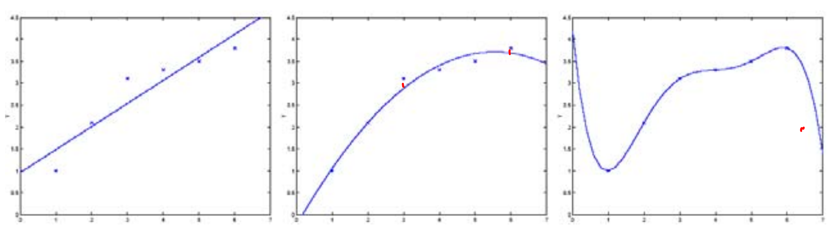

- Two bad 2D RBFs








# Overfitting and underfitting



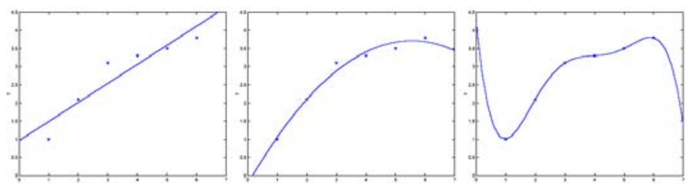
$y = \theta_0 + \theta_1 x$

$y = \theta_0 + \theta_1 x + \theta_2 x^2$

$y = \sum_{j=0}^{5} \theta_j x^j$

# Bias and variance

- We define the bias of a model to be the expected generalization error even if we were to fit it to a very (say, infinitely) large training set.
- By fitting "spurious" patterns in the training set, we might again obtain a model with large generalization error. In this case, we say the model has large variance.






# Locally weighted linear regression

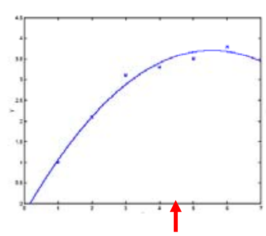
- The algorithm:

  Instead of minimizing $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(\mathbf{x}_i^T\theta - y_i)^2$

  now we fit $\theta$ to minimize $J(\theta) = \frac{1}{2}\sum_{i=1}^{n}w_i(\mathbf{x}_i^T\theta - y_i)^2$

  Where do $w_i$'s come from? $w_i = \exp\left(-\frac{(\mathbf{x}_i - \mathbf{x})^2}{2\tau^2}\right)$

  - where $\mathbf{x}$ is the query point for which we'd like to know its corresponding $\mathbf{y}$



→ Essentially we put higher weights on (errors on) training examples that are close to the query point (than those that are further away from the query)

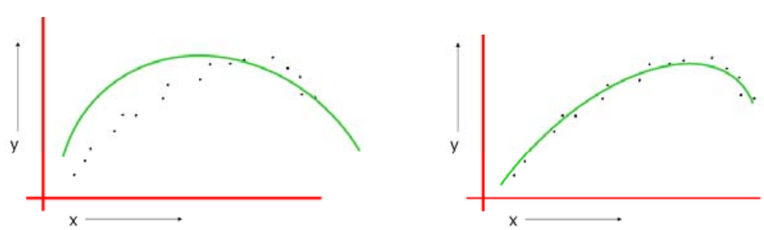
## Parametric vs. non-parametric

- Locally weighted linear regression is the second example we are running into of a **non-parametric** algorithm. (what is the first?)
- The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm
  - because it has a fixed, finite number of parameters (the $\theta$), which are fit to the data;
  - Once we've fit the $\theta$ and stored them away, we no longer need to keep the training data around to make future predictions.
  - In contrast, to make predictions using locally weighted linear regression, we need to keep the entire training set around.
- The term "non-parametric" (roughly) refers to the fact that the amount of stuff we need to keep in order to represent the hypothesis grows linearly with the size of the training set.

## Robust Regression

- The best fit from a quadratic regression
- But this is probably better …

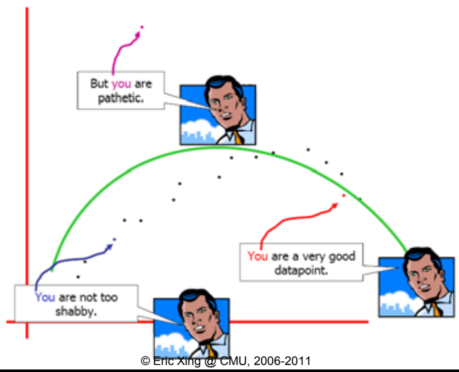



**How can we do this?**

## LOESS-based Robust Regression

- Remember what we do in "locally weighted linear regression"?
  → we "score" each point for its impotence
- Now we score each point according to its "fitness"



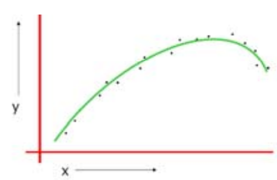

(Courtesy to Andrew Moor)




## Robust regression

- For k = 1 to R…
  - Let $(x_k, y_k)$ be the kth datapoint
  - Let $y^{est}_k$ be predicted value of $y_k$
  - Let $w_k$ be a weight for data point $k$ that is large if the data point fits well and small if it fits badly:

$$w_k = \phi\left((y_k - y_k^{est})^2\right)$$




- Then redo the regression using weighted data points.
- Repeat whole thing until converged!

## Robust regression—probabilistic interpretation

- **What regular regression does:**

  Assume $y_k$ was originally generated using the following recipe:

$$y_k = \theta^T \mathbf{x}_k + \mathcal{N}(0, \sigma^2)$$

  Computational task is to find the Maximum Likelihood estimation of $\theta$

## Robust regression—probabilistic interpretation

- **What LOESS robust regression does:**

  Assume $y_k$ was originally generated using the following recipe:

  with probability $p$: $$y_k = \theta^T \mathbf{x}_k + \mathcal{N}(0, \sigma^2)$$

  but otherwise $$y_k \sim \mathcal{N}(\mu, \sigma^2_{\text{huge}})$$

  Computational task is to find the Maximum Likelihood estimates of $\theta$, $p$, $\mu$ and $\sigma_{\text{huge}}$.

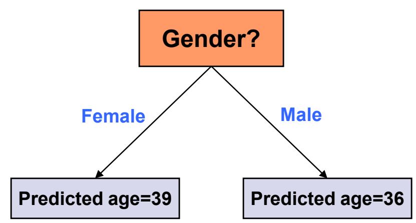
- The algorithm you saw with iterative **reweighting/refitting** does this computation for us. Later you will find that it is an instance of the famous **E.M.** algorithm

# Regression Tree

- Decision tree for regression

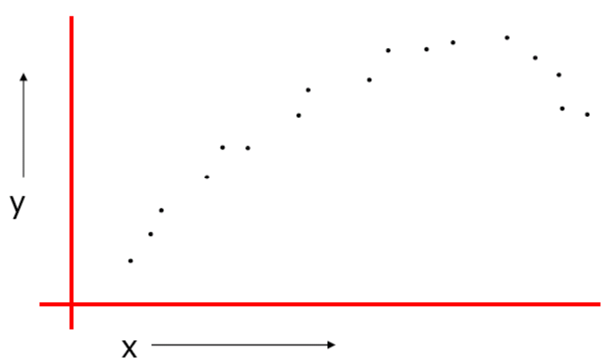
| Gender | Rich? | Num. Children | # travel per yr. | Age |
|---|---|---|---|---|
| F | No | 2 | 5 | 38 |
| M | No | 0 | 2 | 25 |
| M | Yes | 1 | 0 | 72 |
| : | : | : | : | : |





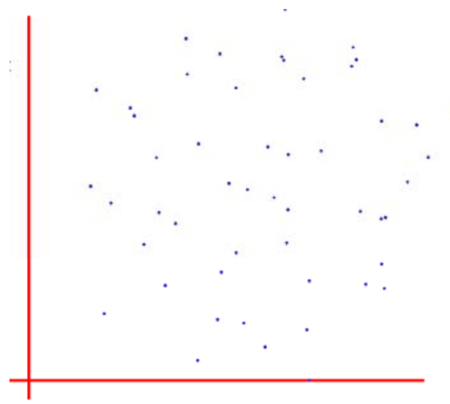

# A conceptual picture

- Assuming regular regression trees, can you sketch a graph of the fitted function y*(x) over this diagram?



# How about this one?

- Multilinear Interpolation



- We wanted to create a continuous and piecewise linear fit to the data




# Take home message

- Gradient descent
  - On-line
  - Batch
- Normal equations
- Equivalence of LMS and MLE
- LR does not mean fitting linear relations, but linear Windows Marketplace combination or basis functions (that can be non-linear)
- Weighting points by importance versus by fitness