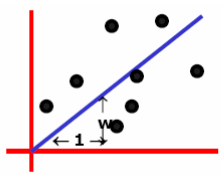
Machine Learning

10-701/15-781, Fall 2006

Introduction to Regression

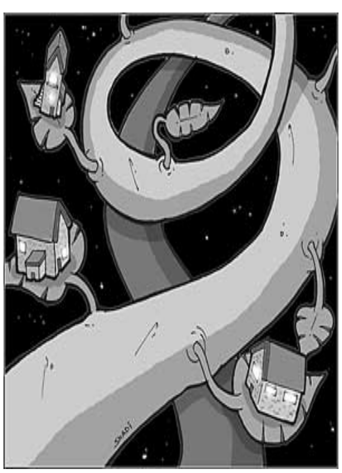
Eric Xing



Lecture 3, September 19, 2006

Reading: Chap. 3, CB

Machine learning for apartment hunting



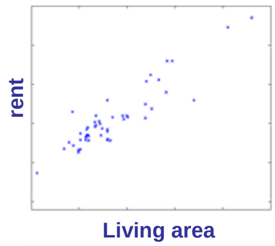
- Now you've moved to Pittsburgh!!

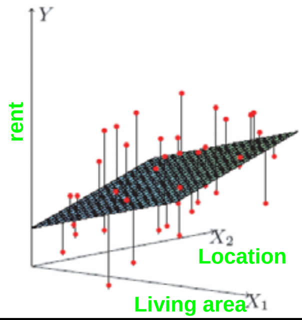
  And you want to find the **most reasonably priced** apartment satisfying your **needs:**

  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



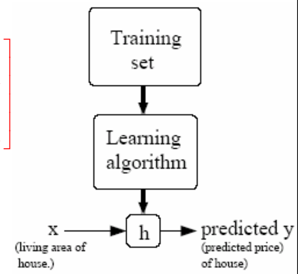


- Features:
  - Living area, distance to campus, # bedroom …
  - Denote as $\mathbf{x}=[x_1, x_2, \ldots x_k]$
- Target:
  - Rent
  - Denoted as y
- Training set:

$$\mathbf{X}=\begin{bmatrix} -- & \mathbf{x}_1 & -- \\ -- & \mathbf{x}_2 & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n & -- \end{bmatrix}=\begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,k} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,k} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n,1} & x_{n,2} & \ldots & x_{n,k} \end{bmatrix}$$

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

Our goal:



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 θ . How!
  - We seek θ 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:

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

- For a single training point, we have:

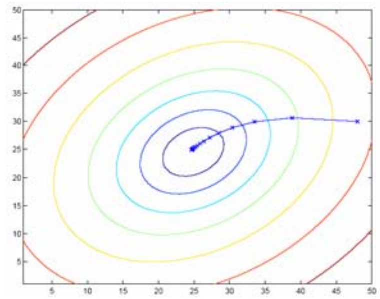
  - This is known as the LMS update rule, or the Widrow-Hoff learning rule
  - This is actually a "**stochastic**", "**coordinate**" descent algorithm
  - This can be used as an **on-line** algorithm

The Least-Mean-Square (LMS) method

- Steepest descent
  - Note that:

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

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



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

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)

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

The normal equations

- Write the cost function in matrix form:

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

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

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

$$
\begin{aligned}
\nabla_\theta J &= \frac{1}{2}\nabla_\theta \mathrm{tr}\left(\theta^T X^T X\theta - \theta^T X^T\vec{y} - \vec{y}^T X\theta + \vec{y}^T\vec{y}\right) \\
&= \frac{1}{2}\left(\nabla_\theta \mathrm{tr}\theta^T X^T X\theta - 2\nabla_\theta \mathrm{tr}\vec{y}^T X\theta + \nabla_\theta \mathrm{tr}\vec{y}^T\vec{y}\right) \\
&= \frac{1}{2}\left(X^T X\theta + X^T X\theta - 2X^T\vec{y}\right) \\
&= X^T X\theta - X^T\vec{y} = 0
\end{aligned}
\quad \Rightarrow \quad \boxed{X^T X\theta = X^T\vec{y}}
$$

The normal equations

$$\Downarrow$$

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

A recap:

- LMS update rule

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

  - Pros: on-line, low per-step cost
  - Cons: coordinate, maybe slow-converging
- Steepest descent

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

  - Pros: fast-converging, easy to implement
  - Cons: a batch,
- 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 $(X^TX)^{-1}$, expensive, numerical issues (e.g., matrix is singular ..)

Geometric Interpretation of LMS

- The predictions on the training data are:

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

$$\mathbf{X} = \begin{bmatrix} -- & \mathbf{x}_1 & -- \\ -- & \mathbf{x}_2 & -- \\ \vdots & \vdots & \vdots \\ -- & \mathbf{x}_n & -- \end{bmatrix}$$

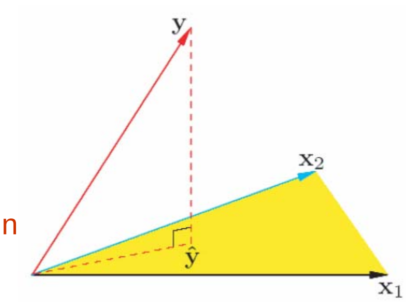
- Note that

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

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

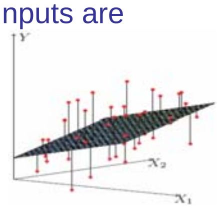


Probabilistic Interpretation of LMS

- 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 ε is an error term of unmodeled effects or random noise



- Now assume that ε 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:

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

Probabilistic Interpretation of LMS, cont.

- Hence the log-likelihood is:

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

- 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 θ !

Beyond basic LR

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

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 $\phi_0(x) = 1$).

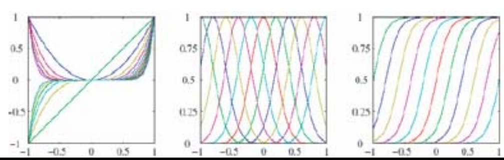
- Example: polynomial regression:

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

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

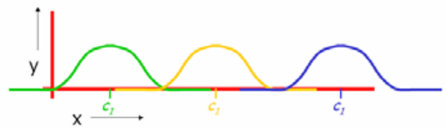
Basis functions

- 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



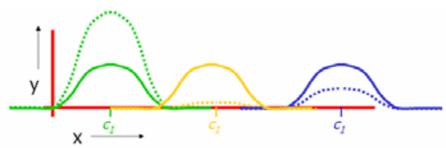
1D and 2D RBFs

- 1D RBF



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

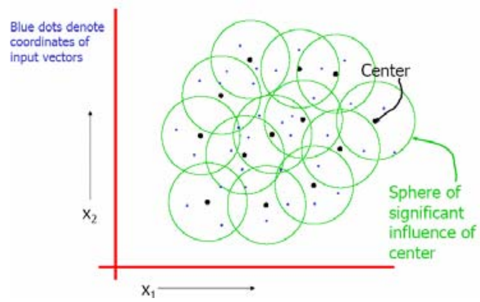
- After fit:



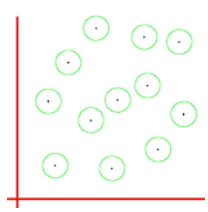
$y^{est} = 2\phi_1(x) + 0.05\phi_2(x) + 0.5\phi_3(x)$

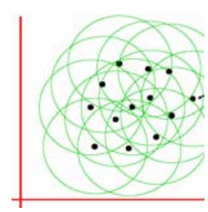
Good and Bad RBFs

- A good 2D RBF



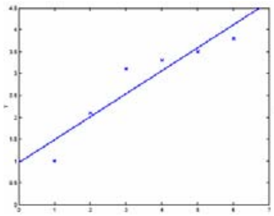
- Two bad 2D RBFs



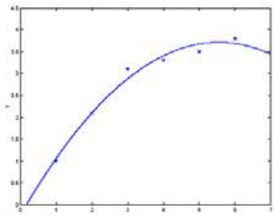


Locally weighted linear regression

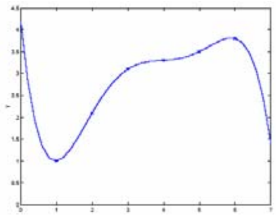
- Overfitting and underfitting



$y = \theta_0 + \theta_1 x$



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



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

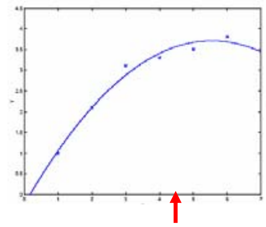
Locally weighted linear regression

- 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 θ 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)

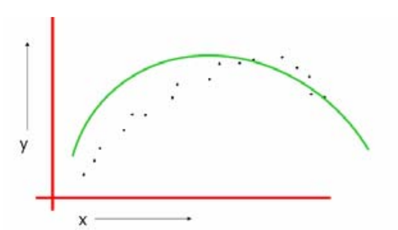
- Do we also have a probabilistic interpretation here (as we did for LR)?

Parametric vs. non-parametric

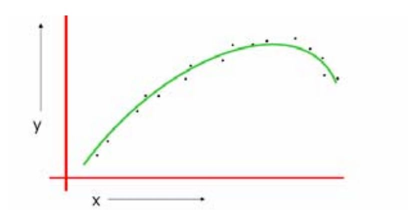
- Locally weighted linear regression is the first example we are running into of a **non-parametric** algorithm.
- The (unweighted) linear regression algorithm that we saw earlier is known as a **parametric** learning algorithm
  - because it has a fixed, finite number of parameters (the θ), which are fit to the data;
  - Once we've fit the θ 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



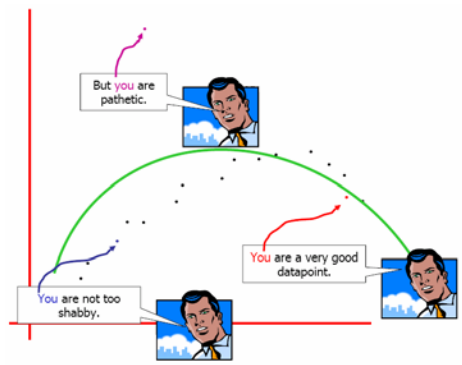
- But this is probably better …



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"

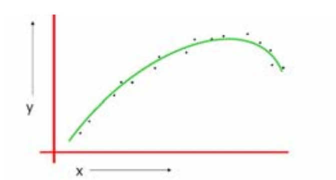


(Courtesy to Andrew Moor)

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 θ

Robust regression—probabilistic interpretation

- **What LOESS robust regression does:**

  Assume y_k was originally generated using the following recipe:

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

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

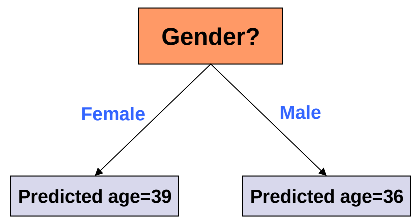
  Computational task is to find the Maximum Likelihood estimates of θ, p, μ and σ_{huge}.

- 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

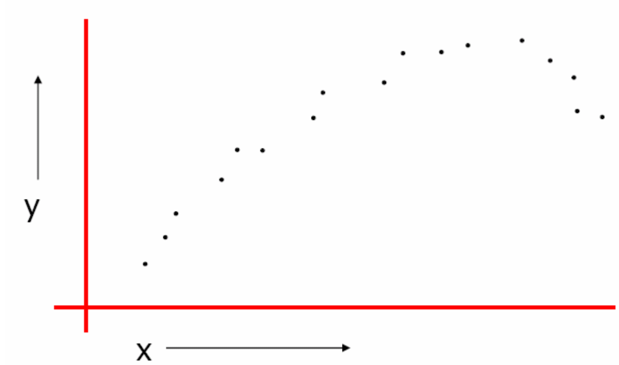
- Decision tree for regression

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



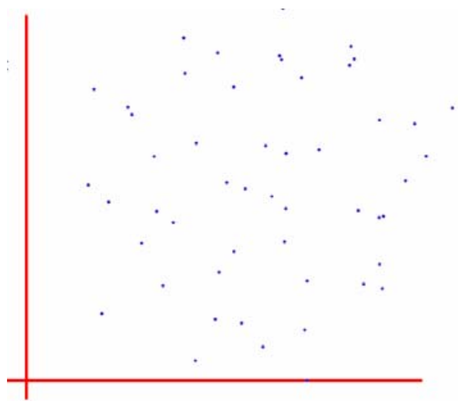
A conceptual picture

- 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 combination or basis functions (that can be non-linear)
- Weighting points by importance versus by fitness