Machine Learning
10-701/15-781, Fall 2008

Introduction to Regression

Eric Xing

Lecture 4, January 28, 2006

Reading: Chap. 3, CB

Functional Approximation, cont.
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 …

<table>
<thead>
<tr>
<th>Living area (ft²)</th>
<th># bedroom</th>
<th>Rent ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>230</td>
<td>1</td>
<td>600</td>
</tr>
<tr>
<td>506</td>
<td>2</td>
<td>1000</td>
</tr>
<tr>
<td>433</td>
<td>2</td>
<td>1100</td>
</tr>
<tr>
<td>109</td>
<td>1</td>
<td>500</td>
</tr>
<tr>
<td>…</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>270</td>
<td>1.5</td>
<td>?</td>
</tr>
</tbody>
</table>

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:

```
\begin{align*}
\mathbf{x} &= \begin{bmatrix}
    x_1 & \ldots & x_k
  \end{bmatrix}^T \\
\mathbf{y} &= \begin{bmatrix}
    y_1 \\
    y_2 \\
    \vdots \\
    y_s
  \end{bmatrix}
\end{align*}
```

```
\text{predicted} y \\
\text{(predicted price of lease)}
```

Our goal:

Training set

Learning algorithm
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:
    $$\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$
  - then we have the following general representation of the linear function:
    $$\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \theta$$

- Our goal is to pick the optimal $\theta$. How!
  - We seek $\theta$ that minimize the following cost function:
    $$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

The Least-Mean-Square (LMS) method

- The Cost Function:
  $$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (x_i^T \theta - y_i)^2$$
- Consider a gradient descent algorithm:
  $$\theta_j^{i+1} = \theta_j^i - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$
The Least-Mean-Square (LMS) method

- Now we have the following descent rule:

\[ \theta_j^{t+1} = \theta_j^t + \alpha \sum_{i=1}^{n} (y_n - x_n^\top \theta^t) x_n,j \]

- 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:

\[ \nabla_{\theta} J = \begin{bmatrix} \frac{\partial}{\partial \theta_1} J, \ldots, \frac{\partial}{\partial \theta_k} J \end{bmatrix}^\top = -\sum_{i=1}^{n} (y_n - x_n^\top \theta) x_n \]

\[ \theta^{t+1} = \theta^t + \alpha \sum_{i=1}^{n} (y_n - x_n^\top \theta^t) x_n \]

- This is as a batch gradient descent algorithm
Some matrix derivatives

- For $f : \mathbb{R}^{m \times n} \mapsto \mathbb{R}$, define:
  $$\nabla_A f(A) = \begin{bmatrix}
\frac{\partial}{\partial A_{11}} f & \cdots & \frac{\partial}{\partial A_{1n}} f \\
\vdots & \ddots & \vdots \\
\frac{\partial}{\partial A_{m1}} f & \cdots & \frac{\partial}{\partial A_{mn}} f
\end{bmatrix}$$

- Trace:
  $$\text{tr} A = \sum_{i=1}^{n} A_{ii}, \quad \text{tr} a = a, \quad \text{tr} ABC = \text{tr} CAB = \text{tr} BCA$$

- Some fact of matrix derivatives (without proof)
  $$\nabla_A \text{tr} AB = B^T, \quad \nabla_A \text{tr} A^T C = CAB + C^T AB^T, \quad \nabla_A [A] = [A^{-1}]^T$$

The normal equations

- Write the cost function in matrix form:
  $$J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2$$
  $$= \frac{1}{2} (x_\theta - y)^T (x_\theta - y)$$
  $$= \frac{1}{2} (\theta^T x_\theta - \theta^T y) - \frac{1}{2} (\theta^T y - \theta^T y)$$

- To minimize $J(\theta)$, take derivative and set to zero:
  $$\nabla_{\theta} J = \frac{1}{2} \nabla_{\theta} (\theta^T x_\theta - \theta^T y - \theta^T y_\theta + y^T y_\theta)$$
  $$= \frac{1}{2} (\nabla_{\theta} \theta^T x_\theta - 2 \nabla_{\theta} \theta^T y_\theta + \nabla_{\theta} y^T y_\theta)$$
  $$= \frac{1}{2} (X^T x_\theta + X^T y_\theta - 2 X^T y)$$
  $$= X^T x_\theta - X^T y_\theta = 0$$

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

The normal equations
A recap:

- **LMS update rule**
  \[ \theta_{i+1} = \theta_i + \alpha (y_n - x_n^T \theta) x_n \]
  - **Pros**: on-line, low per-step cost
  - **Cons**: coordinate, maybe slow-converging

- **Steepest descent**
  \[ \theta^{i+1} = \theta^i + \alpha \sum_{n=1}^{n} (y_n - x_n^T \theta) x_n \]
  - **Pros**: fast-converging, easy to implement
  - **Cons**: a batch,

- **Normal equations**
  \[ \theta^* = \left(X^T X\right)^{-1} X^T \tilde{y} \]
  - **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 ..)

Geometric Interpretation of LMS

- The predictions on the training data are:
  \[ \hat{y} = X \theta^* = X \left(X^T X\right)^{-1} X^T \tilde{y} \]

- Note that
  \[ \hat{y} - \tilde{y} = \left(x (x^T x)^{-1} x^T - I\right) \tilde{y} \]
  and
  \[ x^T (\hat{y} - \tilde{y}) = x^T \left(x (x^T x)^{-1} x^T - I\right) \tilde{y} \]
  \[ = \left(x^T x(x^T x)^{-1} x^T - x^T\right) \tilde{y} \]
  \[ = 0 \] !!

\( \hat{y} \) is the orthogonal projection of \( \tilde{y} \) 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:
  \[ y_i = \theta^T x_i + \varepsilon_i \]
  where \( \varepsilon \) is an error term of unmodeled effects or random noise

- Now assume that \( \varepsilon \) follows a Gaussian \( \mathcal{N}(0, \sigma^2) \), then we have:
  \[ p(y_i | x_i; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \theta^T x_i)^2}{2\sigma^2}\right) \]

- By independence assumption:
  \[ L(\theta) = \prod_{i=1}^{n} p(y_i | x_i; \theta) = \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\frac{\sum_{i=1}^{n} (y_i - \theta^T x_i)^2}{2\sigma^2}\right) \]

Probabilistic Interpretation of LMS, cont.

- Hence the log-likelihood is:
  \[ l(\theta) = n \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{\sigma^2} \frac{1}{2} \sum_{i=1}^{n} (y_i - \theta^T x_i)^2 \]

- Do you recognize the last term?
  Yes it is: 
  \[ J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (x_i^T \theta - y_i)^2 \]

- Thus under independence assumption, LMS is equivalent to MLE of \( \theta \)!
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
  \[ y = \theta_0 + \sum_{j=1}^{m} \theta_j \phi_j(x) = \theta^T \phi(x) \]
  where the \( \phi_j(x) \) are fixed basis functions (and we define \( \phi_0(x) = 1 \)).
- Example: polynomial regression:
  \[ \phi(x) := [1, x, x^2, x^3] \]
- 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 \( \phi_j(x) = \exp \left( -\frac{(x - \mu_j)^2}{2\sigma^2} \right) \)
  - Sigmoidal \( \phi_j(x) = \sigma \left( \frac{x - \mu_j}{s} \right) \)
  - Splines, Fourier, Wavelets, etc

1D and 2D RBFs

- 1D RBF

\[ y_{\text{fit}} = \beta_1 \phi_j(x) + \beta_2 \phi_j(x) + \beta_3 \phi_j(x) \]

- After fit:

\[ y_{\text{fit}} = 3\phi_j(x) + 0.2\phi_j(x) + 0.5\phi_j(x) \]
Good and Bad RBFs

- A good 2D RBF

- Two bad 2D RBFs

Locally weighted linear regression

- Overfitting and underfitting

\[
\begin{align*}
\theta_0 &\quad \theta_1 \quad x \\
y & = \theta_0 + \theta_1 x \\
& = \theta_0 + \theta_1 x + \theta_2 x^2 \\
& = \sum_{j=0}^{5} \theta_j x^j
\end{align*}
\]
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 $J(\theta) = \frac{1}{2} \sum_{i=1}^{n} (x_i^T \theta - y_i)^2$
  now we fit $\theta$ to minimize $J(\theta) = \frac{1}{2} \sum_{i=1}^{n} w_i (x_i^T \theta - y_i)^2$
  Where do $w_i$'s come from? $w_i = \exp\left(-\frac{(x_i - x)^2}{2\tau^2}\right)$

- Where $x$ is the query point for which we'd like to know its corresponding $y$

→ 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 $\theta$), 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"

Robust regression

- For $k = 1$ to $R$...
  - Let $(x_k, y_k)$ be the $k$th 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:
    $$w_k = \phi((y_k - y_{est_k})^2)$$

- 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:

\[
y_k = \theta^T x_k + \mathcal{N}(0, \sigma^2)
\]

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

- What LOESS robust regression does:

Assume \( y_k \) was originally generated using the following recipe:

with probability \( p \):

\[
y_k = \theta^T x_k + \mathcal{N}(0, \sigma^2)
\]

but otherwise

\[
y_k \sim \mathcal{N}(\mu, \sigma_{\text{huge}}^2)
\]

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

<table>
<thead>
<tr>
<th>Gender</th>
<th>Rich?</th>
<th>Num. Children</th>
<th># travel per yr.</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>No</td>
<td>2</td>
<td>5</td>
<td>38</td>
</tr>
<tr>
<td>M</td>
<td>No</td>
<td>0</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>M</td>
<td>Yes</td>
<td>1</td>
<td>0</td>
<td>72</td>
</tr>
</tbody>
</table>

Gender?

- Female
- Male

Predicted age=39

Predicted age=36

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