Supervised Learning and Regression

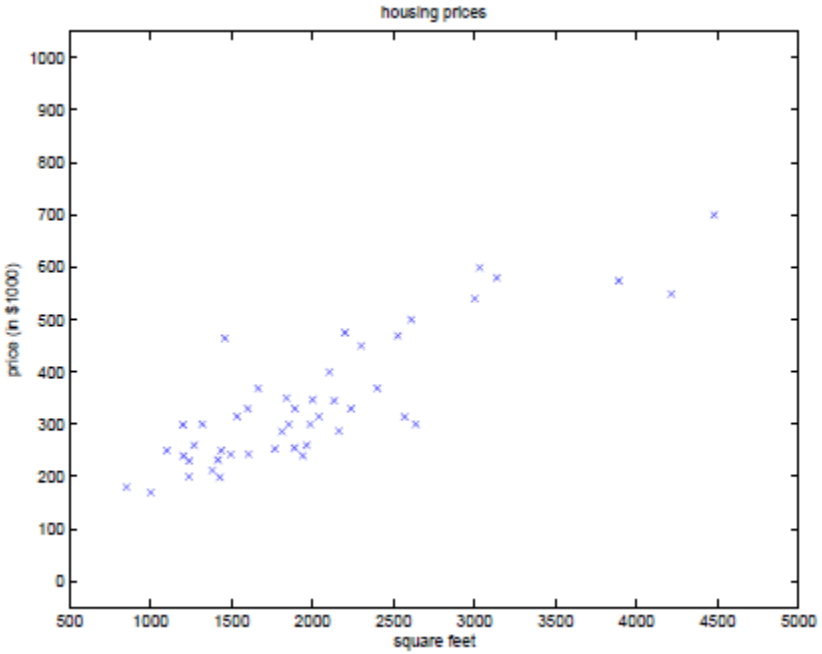
ECE 457B tutorial

The material on these slides is extracted mainly from Stanford Univ's Dr. Andrew Ng's course notes.

Supervised learning

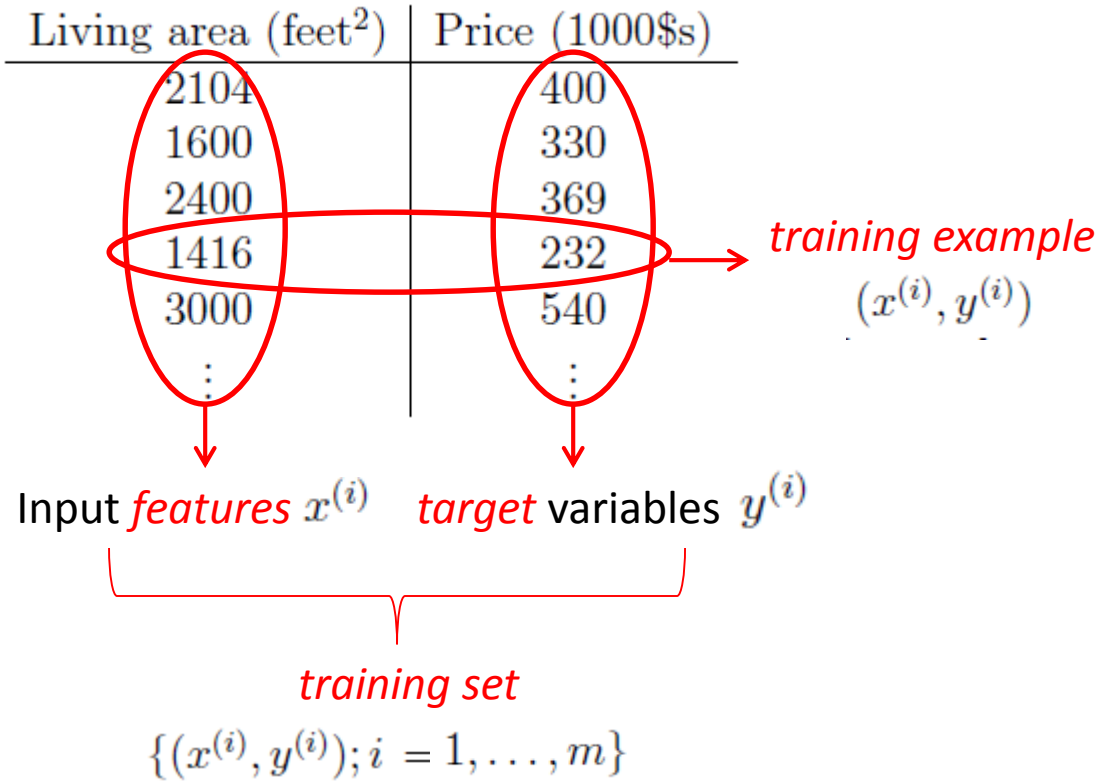
Suppose that we have a dataset giving the living areas and prices of 47 houses from Kitchener:

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



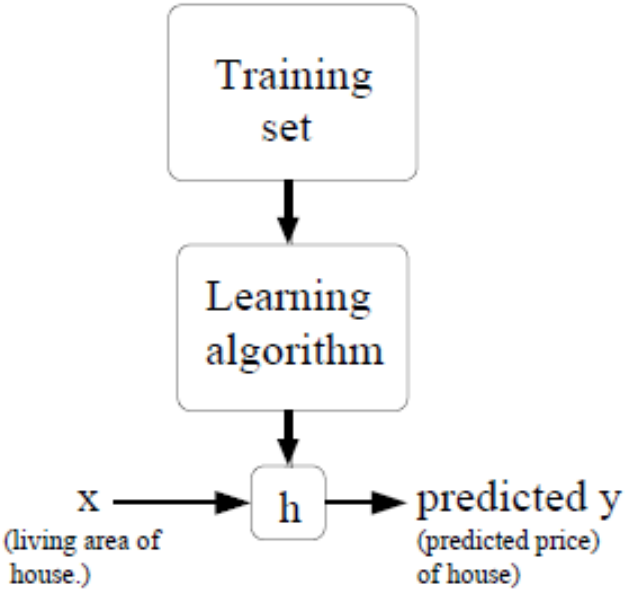
- Given data like this, how can we learn to predict the prices of other houses in Kitchener as a function of the size of their living areas?

Supervised learning



Supervised learning

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$ so that $h(x)$ is a "good" predictor for the corresponding value of y. For historical reasons, this function h is called a **hypothesis**. Seen pictorially, the process is therefore like this:



Regression vs. Classification

- When the target variable that we are trying to predict is continuous, such as in our housing example, we call the learning problem a *regression* problem.
- When y can take on only a small number of discrete values (such as if given the living area, we wanted to predict if a dwelling is house or an apartment, say), we call it a *classification* problem.

Linear Regression

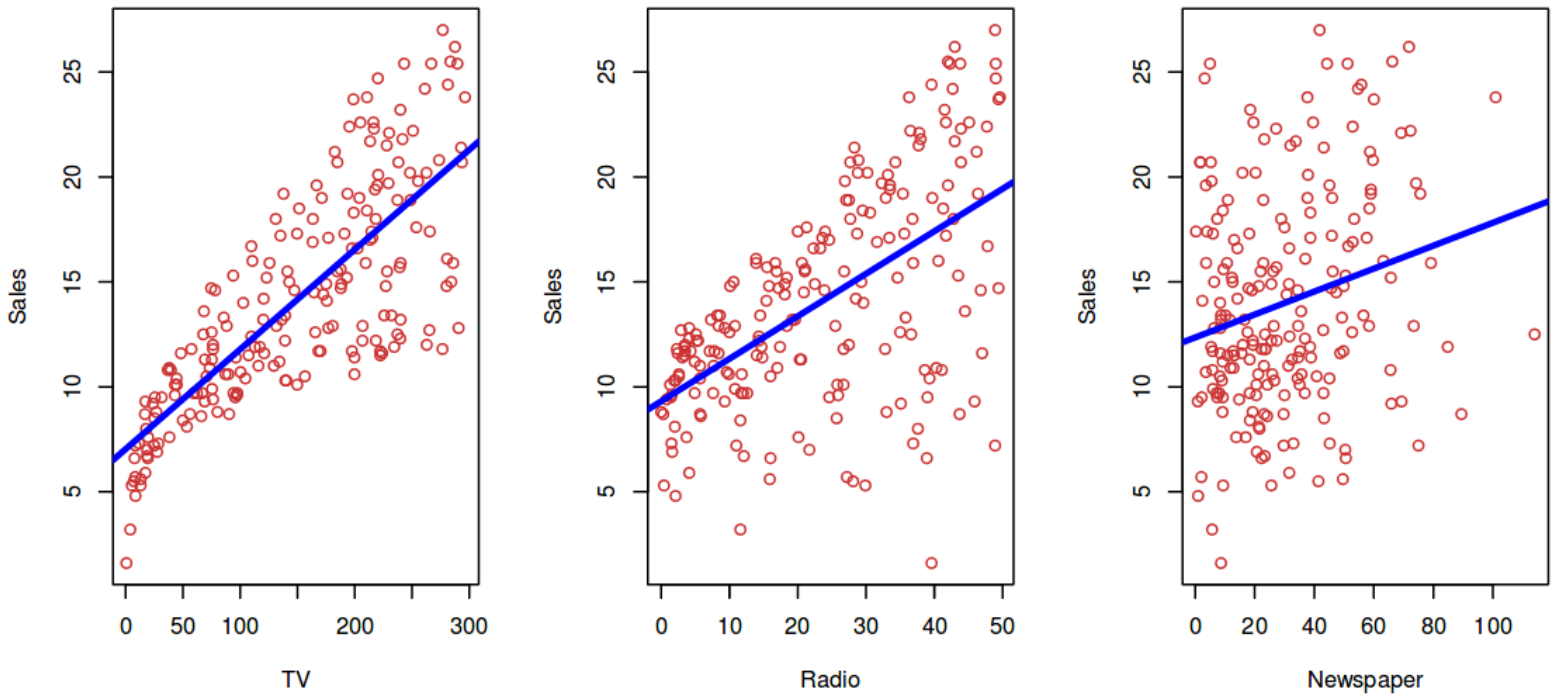
Let's add another feature $x_2^{(i)}$: number of bedrooms:

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

Here, the x's are two-dimensional vectors in $\mathbb{R}^2$.

Note: You may choose to have hundreds of features in designing a learning problem, which depends on the problem you are solving. Selecting powerful features is very important in any learning system and will help achieving a more accurate and robust system.

Example: Advertisement



From https://lagunita.stanford.edu/c4x/HumanitiesScience/StatLearning/asset/linear_regression.pdf

Linear Regression

To perform supervised learning, we must decide how we're going to represent functions/hypotheses h in a computer. As an initial choice, lets say we decide to approximate y as a linear function of x:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

θ_i's are the *parameters* (also called *weights*).

For simplicity, we drop the subscript θ and introduce the intercept term $x_0 = 1$:

$$h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x$$

❖ Now, given a training set, how do we pick, or learn, the parameters θ ?

Linear Regression

- One reasonable method: make $h(x)$ close to y for the training examples.

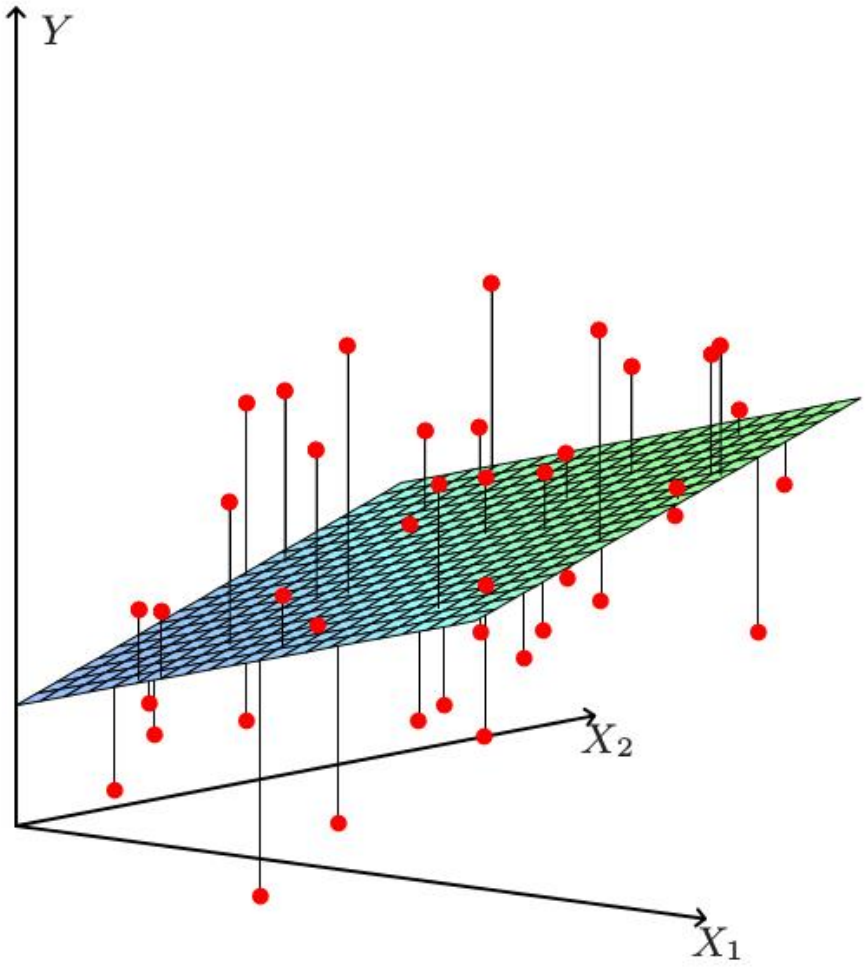
We define the ***cost function***:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

This is called *Least-Squares (LS) cost function* .

** This is a convex quadratic function and therefore there is only one global optima.

Mean Squared Error



From T. Hastie et al., *Elements of Statistical Learning*, 2009.

LMS algorithm

Let's consider ***gradient descent*** algorithm, which starts with some initial θ , and repeatedly performs update, until hopefully we converge to a value of θ that minimizes $J(\theta)$.
Where the update rule is:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Here, α is called the ***learning rate***. This is a very natural algorithm that repeatedly takes a step in the direction of steepest decrease of J.

(This update is simultaneously performed for all values of j= 0, …,n.)

LMS algorithm

In order to implement this algorithm, we have to work out what is the partial derivative term on the right hand side. Lets first work it out for the case of if we have only one training example (x, y), so that we can neglect the sum in the definition of J. We have:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y\right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta(x) - y\right) \\
&= \left(h_\theta(x) - y\right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y\right) x_j
\end{aligned}$$

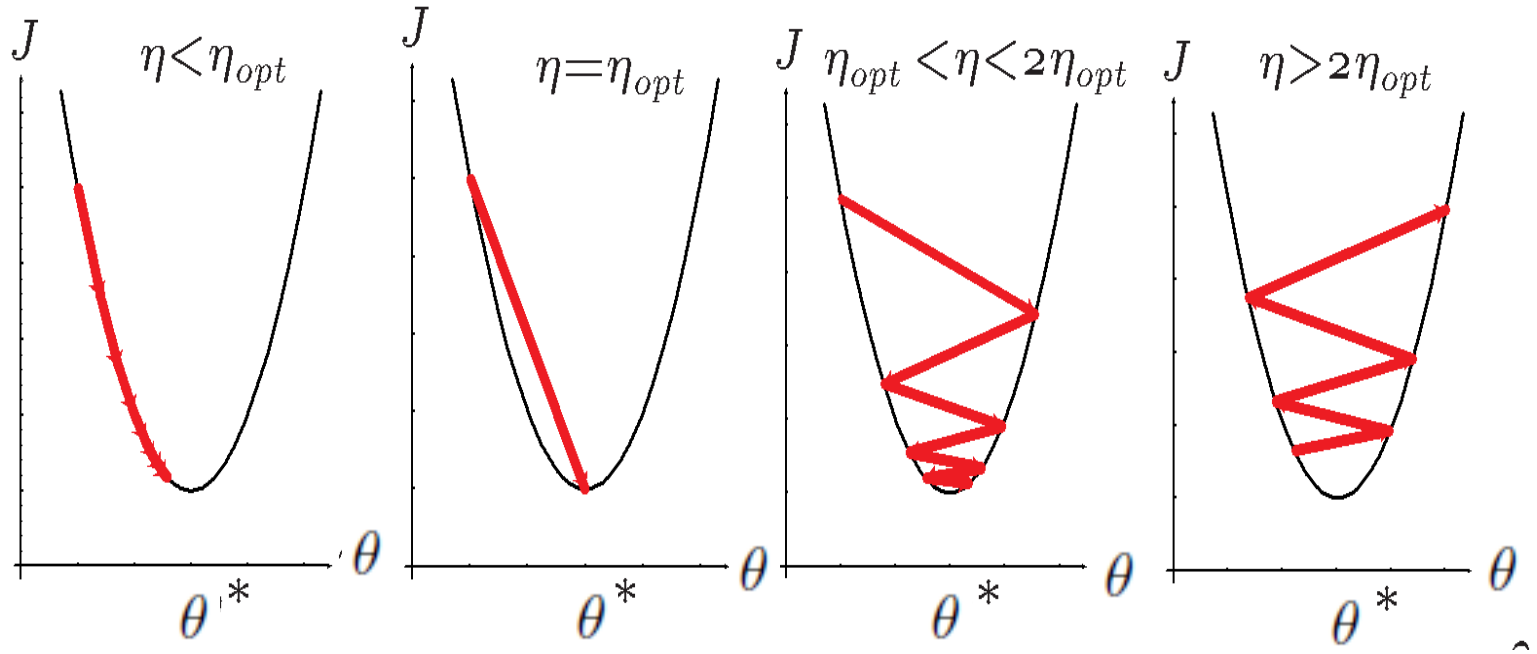
For a single training example, this gives the update rule:[1]

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

This rule is called least mean squares (*LMS*) update rule, also known as *Widrow-Hoff* learning rule.

LMS algorithm

Optimization with different learning rates:



Example: Prostate Cancer

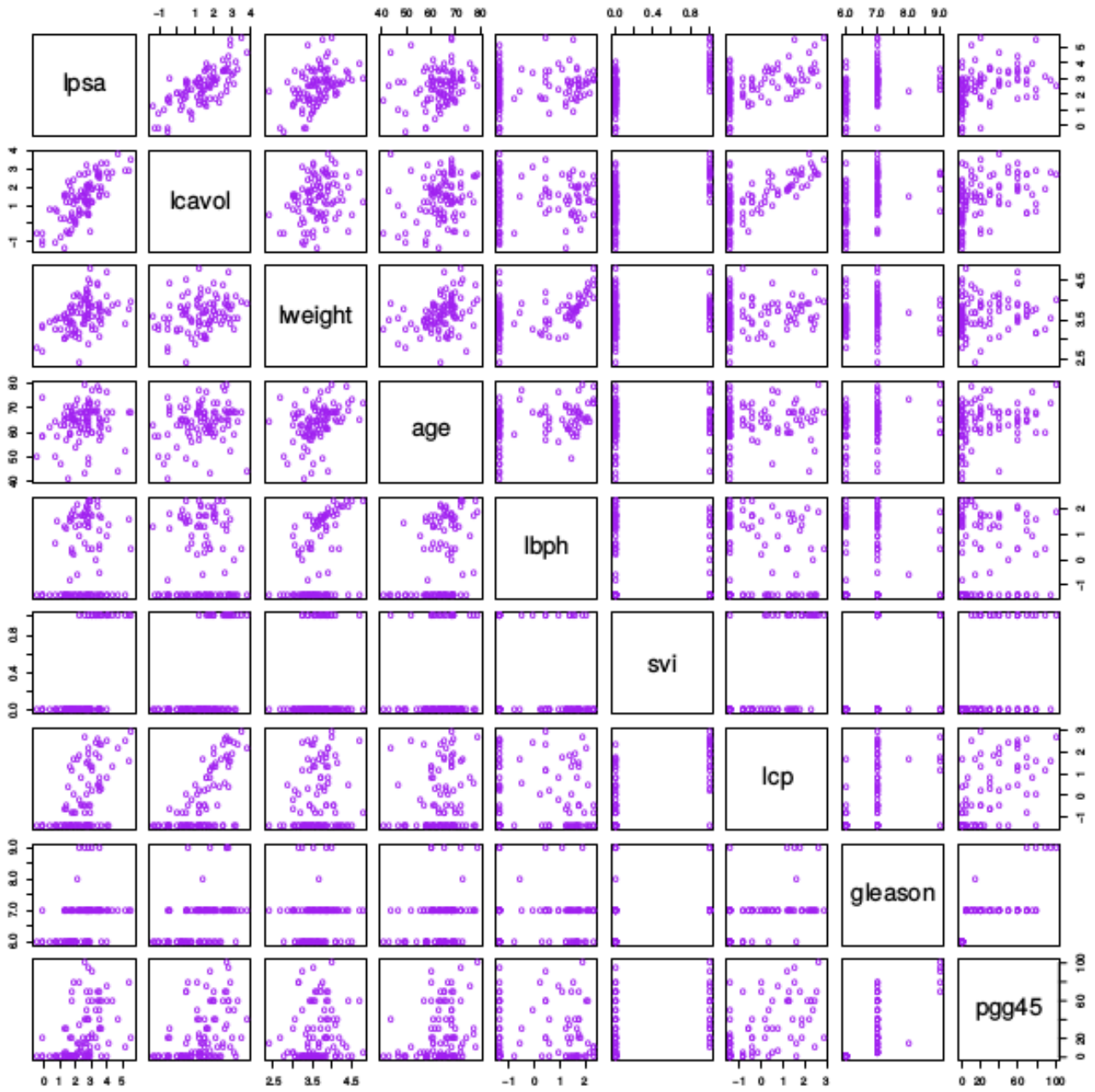
Learn a mapping between a number of clinical measures and the level of prostate-specific antigen using linear regression.

Term	Coefficient
Intercept	2.46
lcavol	0.68
lweight	0.26
age	−0.14
lbph	0.21
svi	0.31
lcp	−0.29
gleason	−0.02
pgg45	0.27

Base error rate: 1.057

LR error rate: 0.521

Example from T. Hastie et al., *Elements of Statistical Learning*, 2009.


lpsa
lcavol
lweight
age
lbph
svi
lcp
gleason
pgg45

LMS algorithm

We derived the LMS rule for when there was only a single training example. There are two ways to modify this method for a training set of more than one example:

1) *Batch gradient descent:*

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

2) *Stochastic gradient descent (or incremental gradient descent):*

Loop {

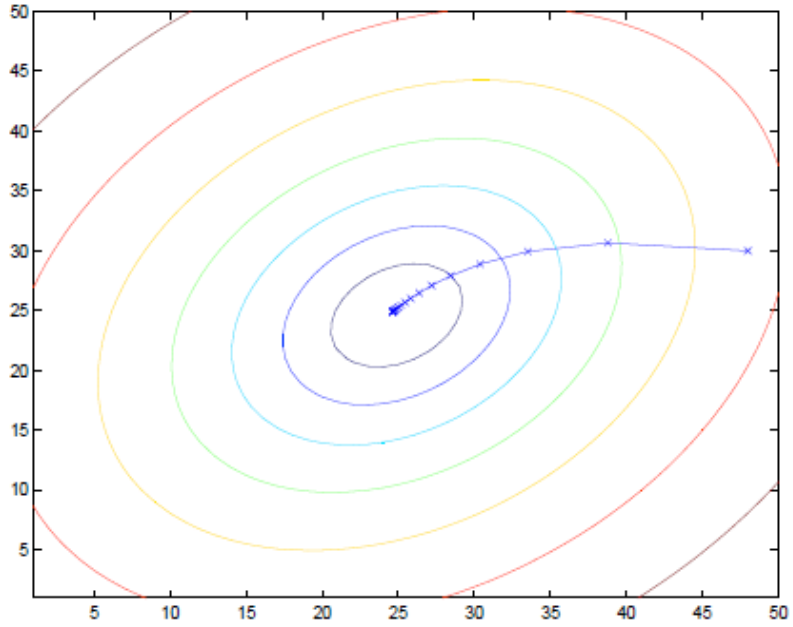
for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}$$

}

}

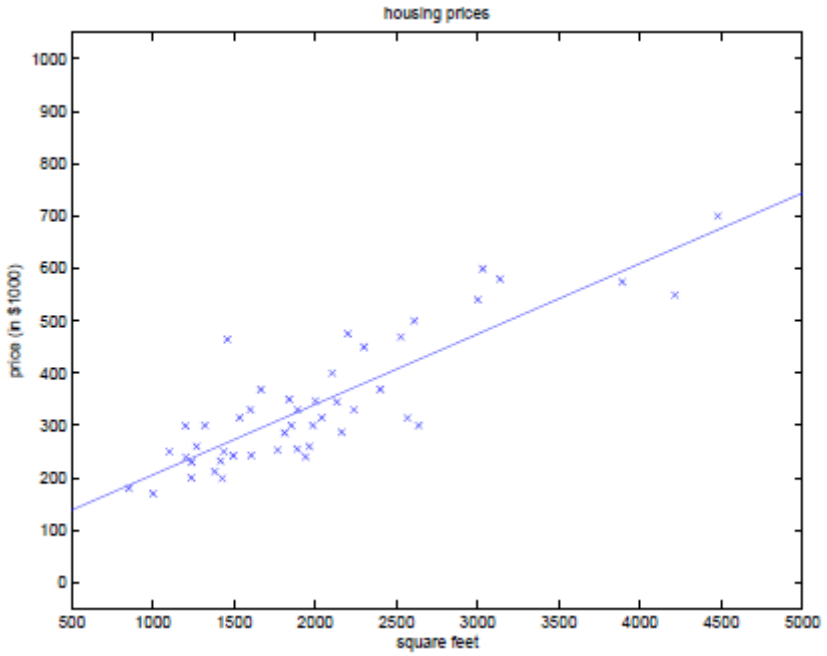
LMS algorithm



The ellipses shown above are the contours of a quadratic function. Also shown is the trajectory taken by gradient descent, with was initialized at (48,30). The x's in the figure (joined by straight lines) mark the successive values of θ that gradient descent went through.

LMS algorithm

When we run batch gradient descent to fit θ on our previous dataset, to learn to predict housing price as a function of living area, we obtain $\theta_0 = 71.27$, $\theta_1 = 0.1345$. If we plot $h_\theta(x)$ as a function of x (area), along with the training data, we obtain the following figure:



If the number of bedrooms were included as one of the input features as well, we get $\theta_0 = 89.60, \theta_1 = 0.1392$, $\theta_2 = -8.738$.

Probabilistic interpretation

We will give a set of probabilistic assumptions, under which least-squares regression is derived as a very natural algorithm:

Let's assume that the target variables and the inputs are related via the equation:

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

Where $\epsilon^{(i)}$ is an error term.
Let's further assume that $\epsilon^{(i)}$ are distributed i.i.d according to a Gaussian distribution:

$$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$$

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

Probabilistic interpretation

This implies that: $$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right)$$

❖Now, given X (the matrix which contains all the $x^{(i)}$'s) and θ , what is the Distribution of the $y^{(i)}$'s ?

The *likelihood* function is defined as: $$L(\theta) = L(\theta; X, \vec{y}) = p(\vec{y}|X;\theta)$$

Note that by the independence assumption, this can also be written as:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{(y^{(i)}-\theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

Probabilistic interpretation

- Now, given this probabilistic model relating the $y^{(i)}$'s and $x^{(i)}$'s, what is a reasonable way of choosing our best guess of the parameters θ ?

The principal of *maximum likelihood* says that we should choose θ to maximize $L(\theta)$.
Instead of maximizing $L(\theta)$ we can also maximize any strictly increasingly Function of $L(\theta)$ In particular, the derivatives will be simpler if we instead Maximize the log likelihood:

$$\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \log \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= \sum_{i=1}^{m} \log \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\
&= m \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^{m} (y^{(i)} - \theta^T x^{(i)})^2
\end{aligned}$$

Probabilistic interpretation

Therefore, maximizing $\ell(\theta)$ gives the same answer as minimizing

$$\frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})^2.$$

Which is in fact the same as our original least-squares cost function $J(\theta)$.

This is thus one set of assumptions under which least-squares regression Can be justified as a very natural method that's just doing maximum likelihood estimation.

Note also that in our previous discussion, the final choice of θ did not depend on σ^2, and indeed we'd have arrived at the same result even if σ^2 was unknown.