WELCOME

Introduction to Machine Learning

DBDA.X408.(33)

Instructor:

Bill Chen

UCSC Silicon Valley Extension
PROFESSIONAL EDUCATION

UCSC Silicon Valley Extension

E: xchen375@ucsc.edu

Week 2

Linear Regression: General linear Models.

Model Fitting: Training/Testing/Validation.

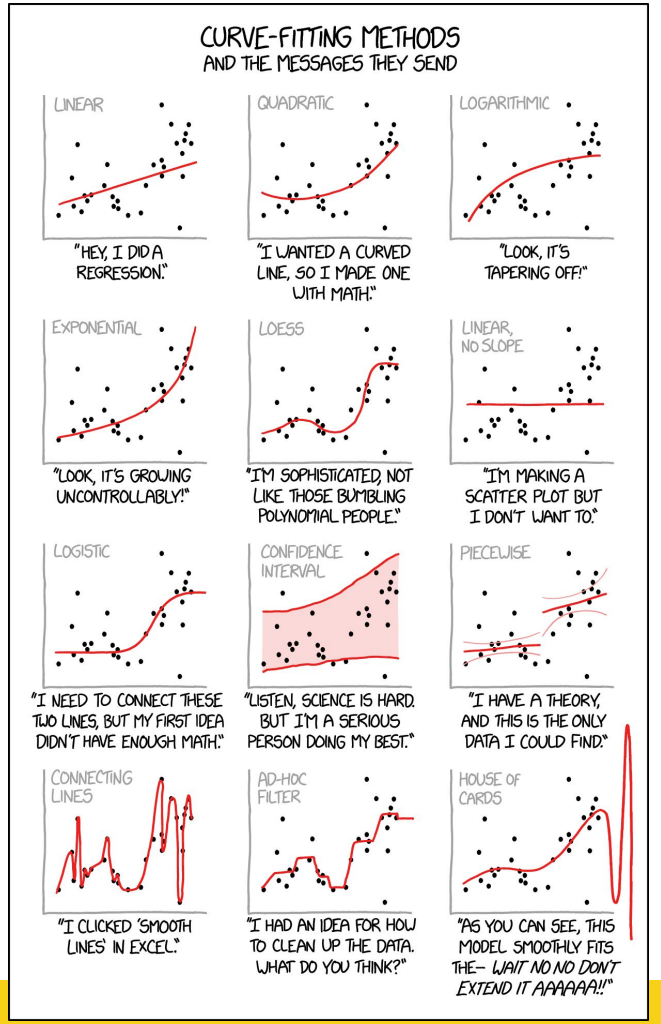
Confusion Matrix. Receiver Operating Characteristic.

Logistic regression.

Newton's Method.

Probability Review.

Weighted Least Squares.



Previously we defined Machine Learning as

Question for student?

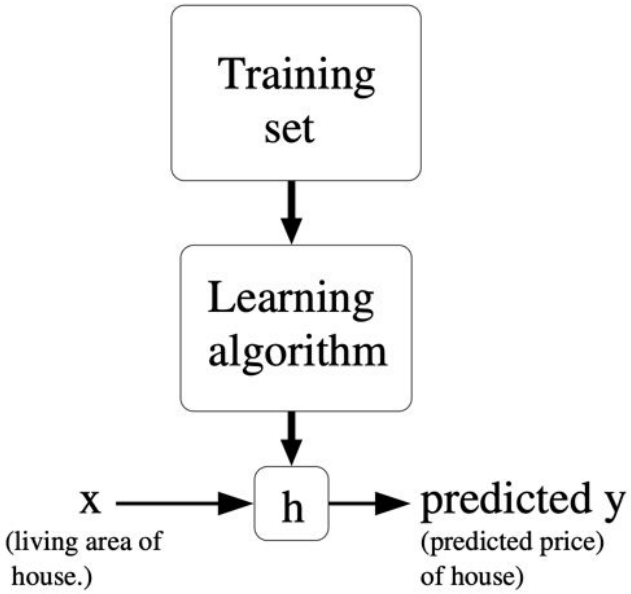
To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function.

$$h: X \rightarrow Y$$

so that $h(x)$ is a “good” predictor for the corresponding value of y, so that $h(x)$ is a “good” predictor for the corresponding value of y.

For historical reason, this function h is called a $hypothesis$.

Hypothesis



Regression Problem:

When target variable is continuous, such as housing price, amount, etc.

Classification Problem:

When target variable is discrete value, such as townhouse vs house.

Linear Regression, Logistic Regression, Newton's Method

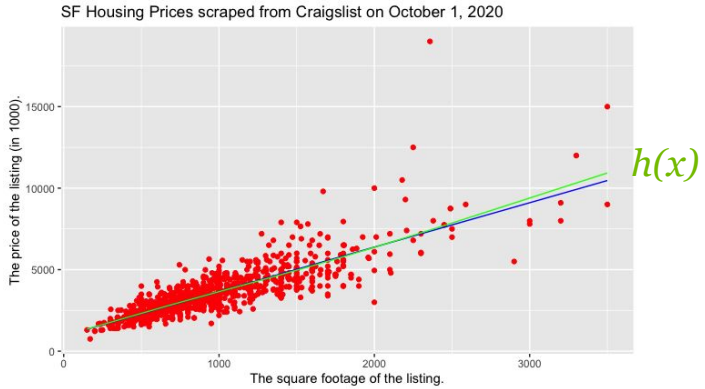
Linear Regression

- To perform supervised learning, we must decide how we're going to represent functions/hypotheses h in a computer. We can approximate y as linear function h of x

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

- Here, the θ_i's are the parameters (also called weights) parameterizing the space of linear functions mapping from X to Y.

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

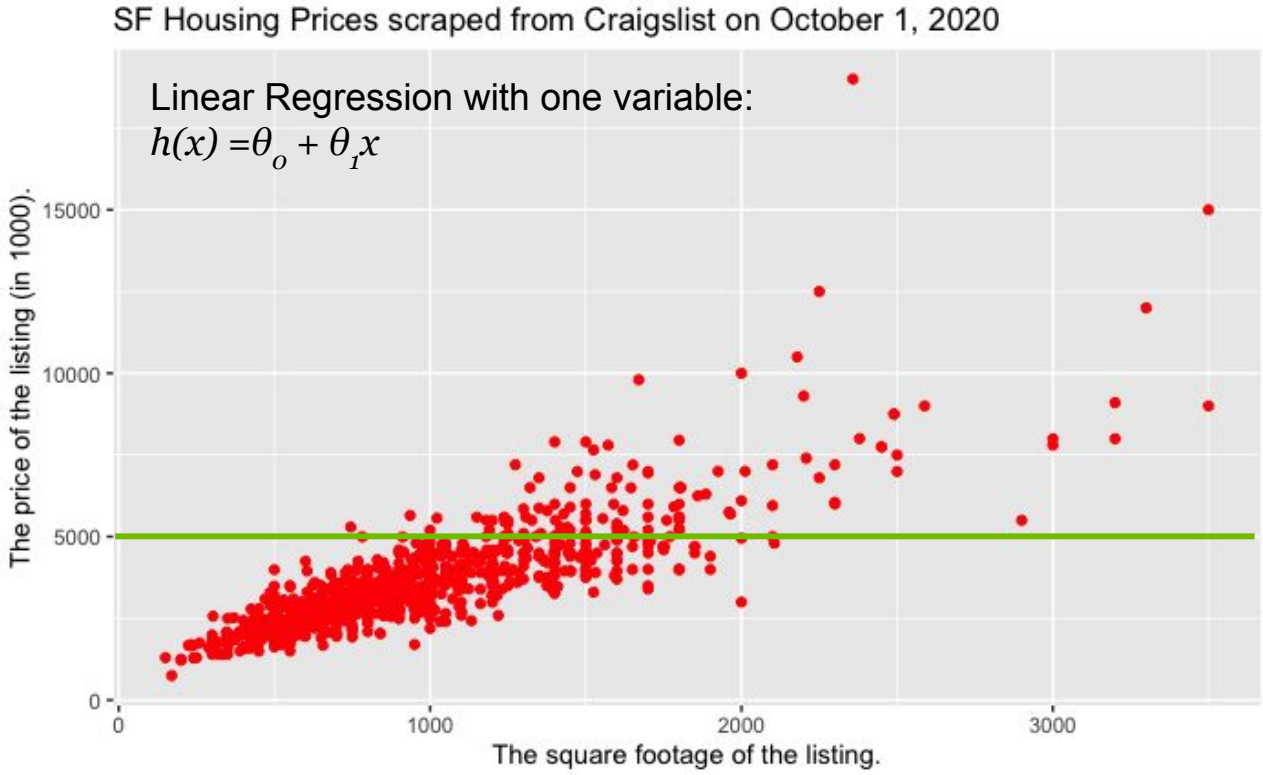


Linear Regression with one variable:
$h(x) = \theta_o + \theta_1 x$

Linear Regression

Idea: choose θ_o , θ_1 so that $h_\theta(x)$ is close to y for our training example:

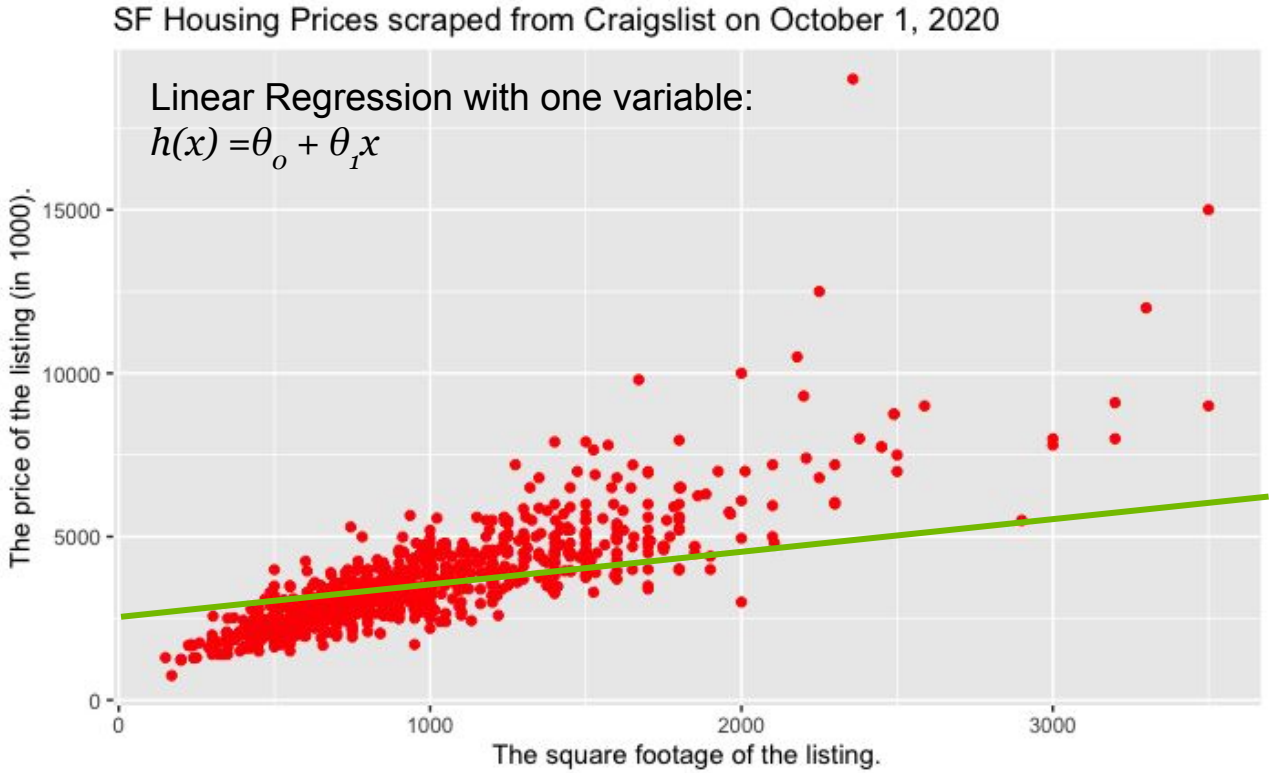
- θ_o = 5000
- θ_1 = 0



Linear Regression

Idea: choose θ_o , θ_1 so that $h_\theta(x)$ is close to y for our training example:

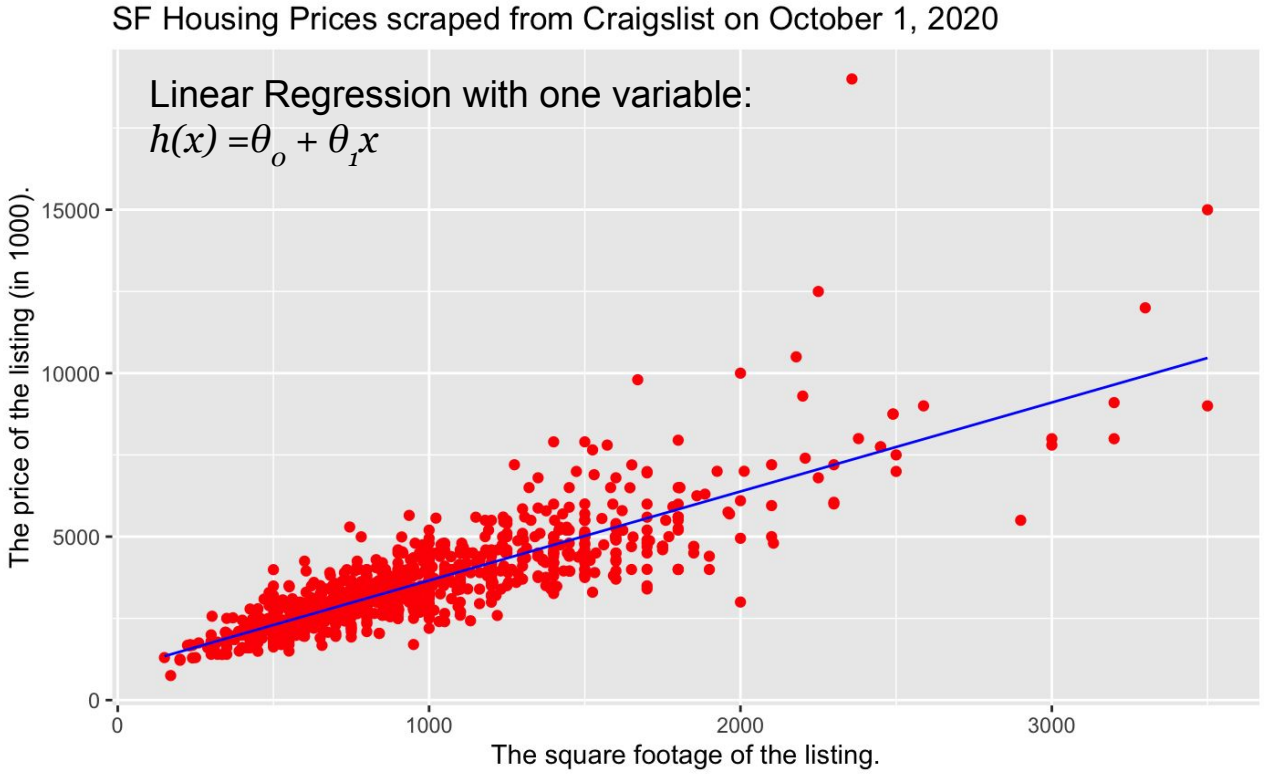
- θ_o = 2500
- θ_1 = 0.5



Linear Regression

Idea: choose θ_o , θ_1 so that $h_\theta(x)$ is close to y for our training example:

- θ_o = 935.36
- θ_1 = 2.72



Linear Regression

We can also write the formula:

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

As:

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

What the Machine is “learning” here?

Linear Regression

One reasonable method seems to be to make $h(x)$ close to y, at least for the training examples we have. To formalize this, we will define a function that measures, for each value of the θ's, how close the $h(x^{(i)})$'s are to the corresponding $y^{(i)}$'s. We define the cost function:

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

We want to choose θ so as to minimize $J(\theta)$.

LMS algorithm

We want to choose θ so as to minimize $J(\theta)$. To do so, let's use a search algorithm that starts with some "initial guess" for θ, and that repeatedly changes θ to make $J(\theta)$ smaller, until hopefully we converge to a value of θ that minimizes $J(\theta)$. Specifically, let's consider the gradient descent algorithm, which starts with some initial θ, and repeatedly performs the update:

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

(This update is simultaneously performed for all values of $j = 0, \ldots, d$.) 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.

LMS update rule

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

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

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

The rule is called the LMS update rule (LMS stands for “least mean squares”), and is also known as the Widrow-Hoff learning rule.

LMS update rule

- The magnitude of the update is proportional to the error term $(y^{(i)} - h(x^{(i)}))$;
  - Aka, the machine is learning from the “mistake”,
  - if we encountering a training example on which our prediction $h(x^{(i)})$ nearly matches the actual value of $y^{(i)}$, then we find that there is little need to change the parameters
  - in contrast, a larger change to the parameters will be made if our prediction $h(x^{(i)})$ has a large error (i.e., if it is very far from $y^{(i)}$)
- Repeat until convergence

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}, (\text{for every } j) \qquad (1.1)$$

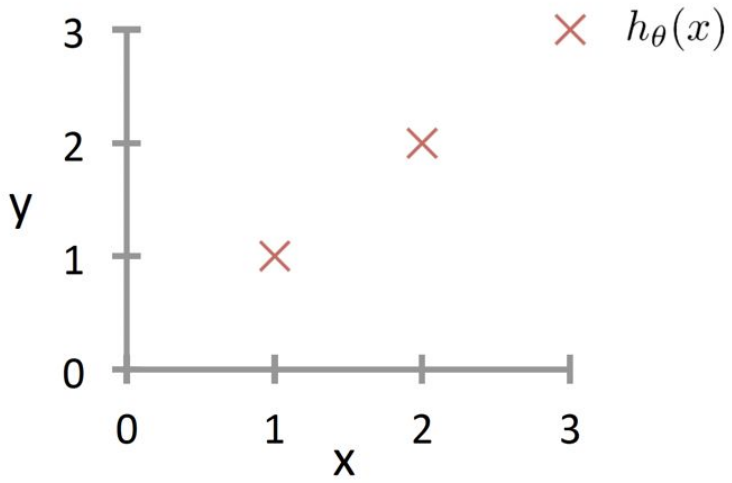
- Rewrite in a vector format: $\theta := \theta + \alpha \sum_{i=1}^{n} \left(y^{(i)} - h_\theta(x^{(i)})\right) x^{(i)}$

LMS update rule

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

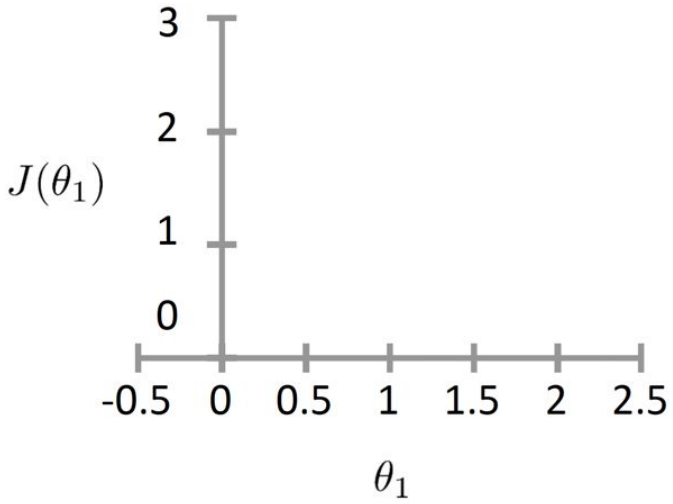
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



LMS update rule

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

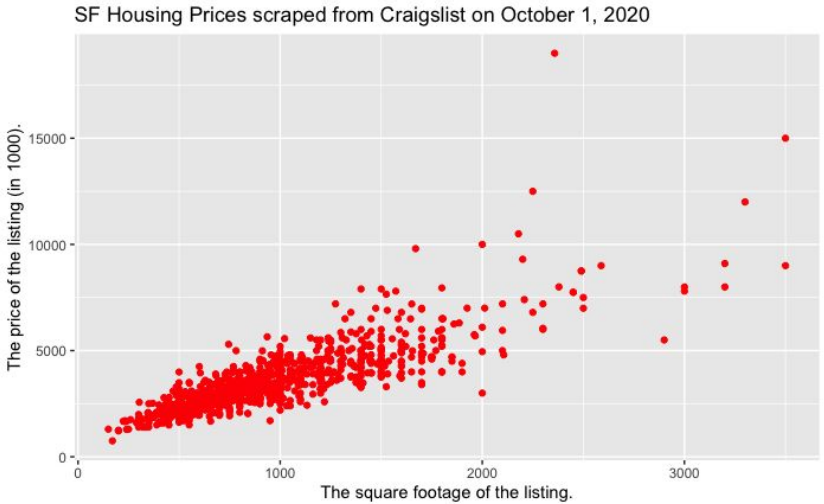
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

LMS update rule

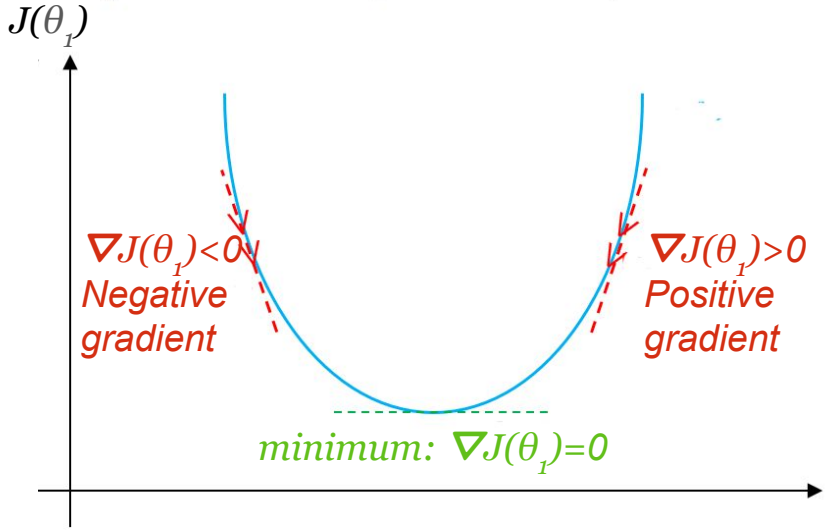
$h_\theta(x)$

(for fixed θ_1, this is a function of x)

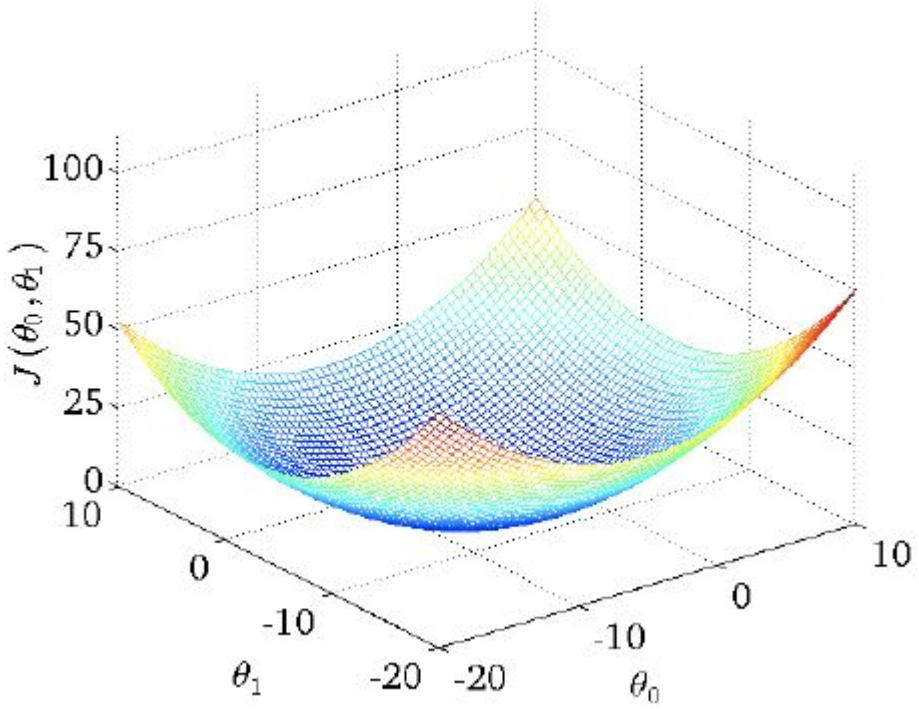


$J(\theta_1)$

(function of the parameter θ_1)



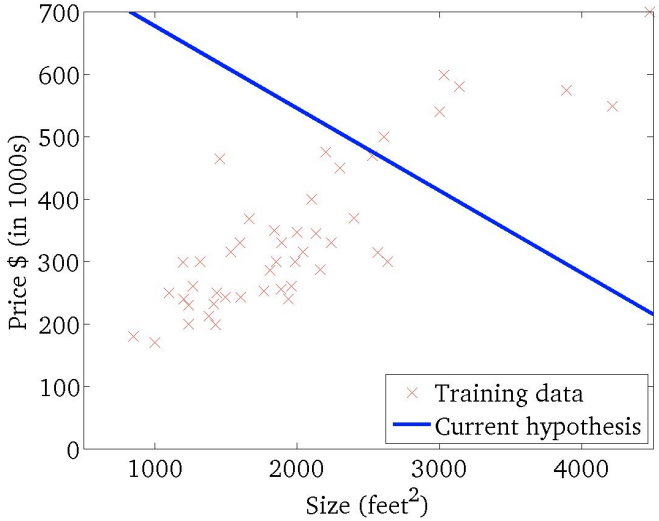
LMS update rule



LMS update rule

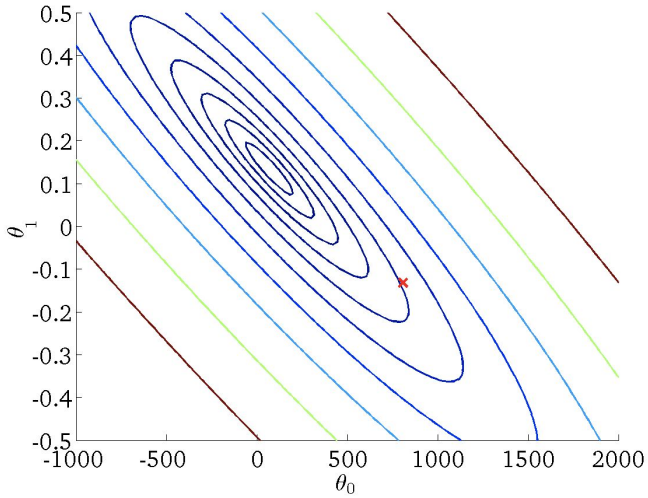
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

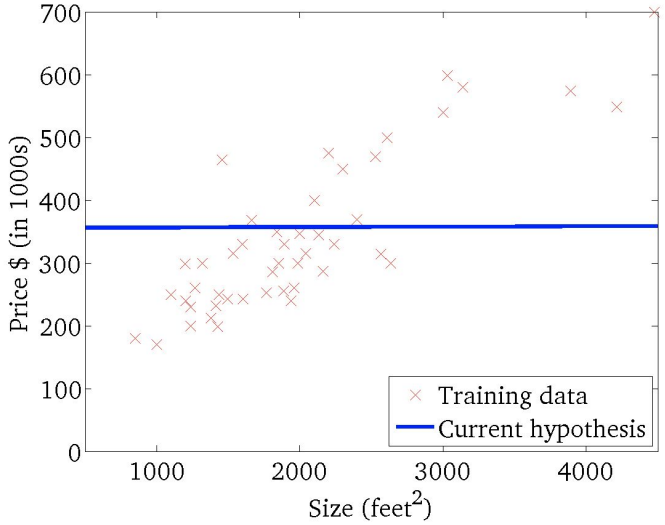
(function of the parameter θ_1)



LMS update rule

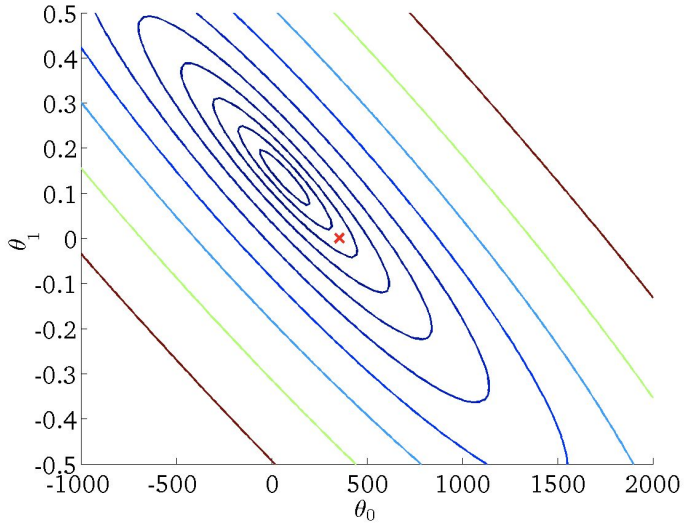
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

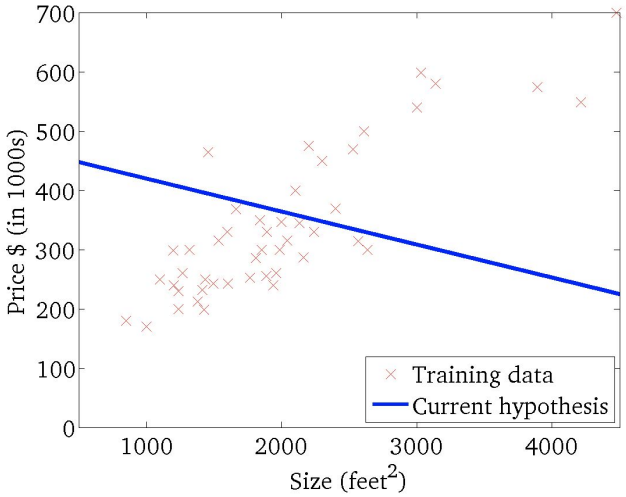
(function of the parameter θ_1)



LMS update rule

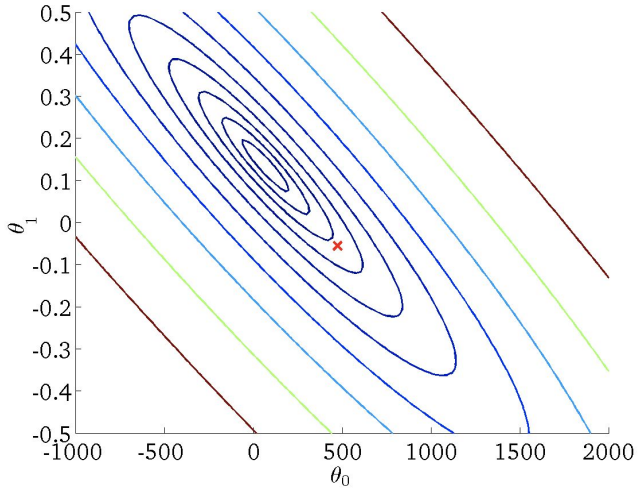
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

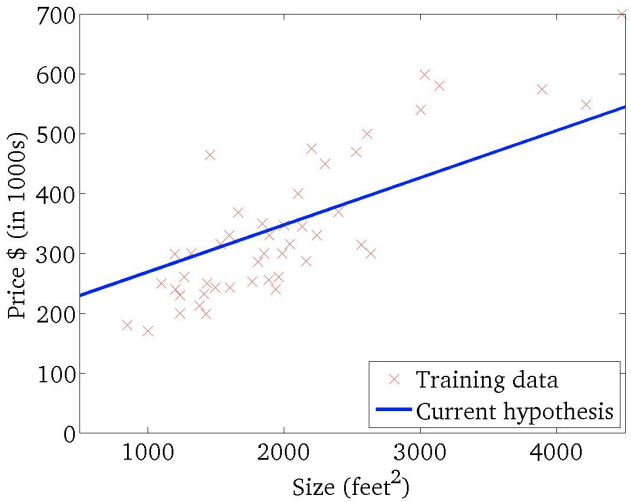
(function of the parameter θ_1)



LMS update rule

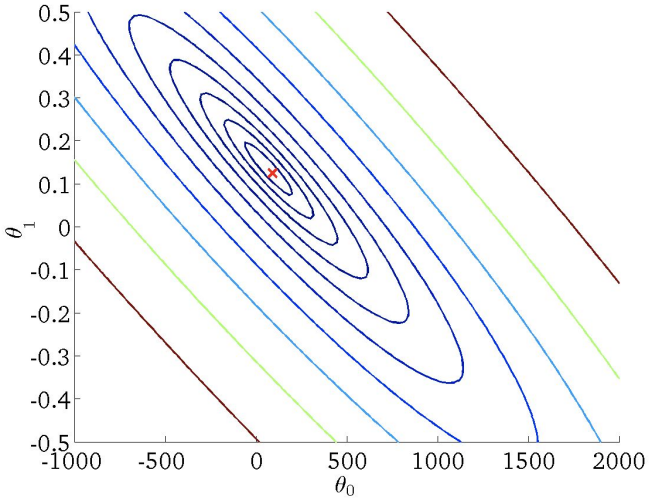
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



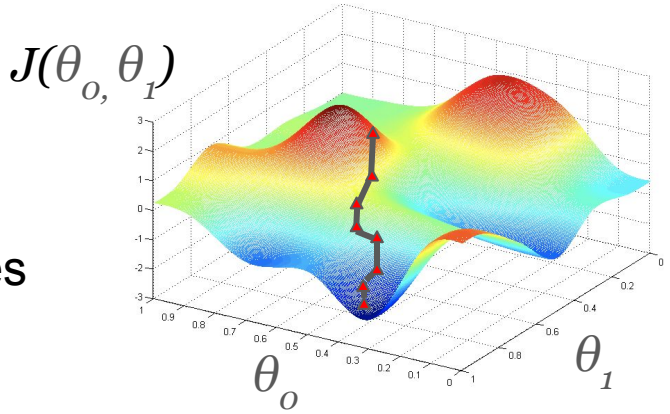
Gradient Descent

Have some function $J(\theta_0, \theta_1)$

Want $min_{\theta 0, \theta 1} J(\theta_0, \theta_1)$

Outline:

- Start with some (θ_0, θ_1), could be random values or just (0,0)
- Keep changing (θ_0, θ_1) to reduce $J(\theta_0, \theta_1)$ until we hopefully end up with at a minimum



Gradient Descent

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

}

Correct: Simultaneous update

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\theta_1 := \text{temp1}$$

Incorrect:

$$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$
$$\theta_0 := \text{temp0}$$
$$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$
$$\theta_1 := \text{temp1}$$

Gradient Descent

repeat until convergence {

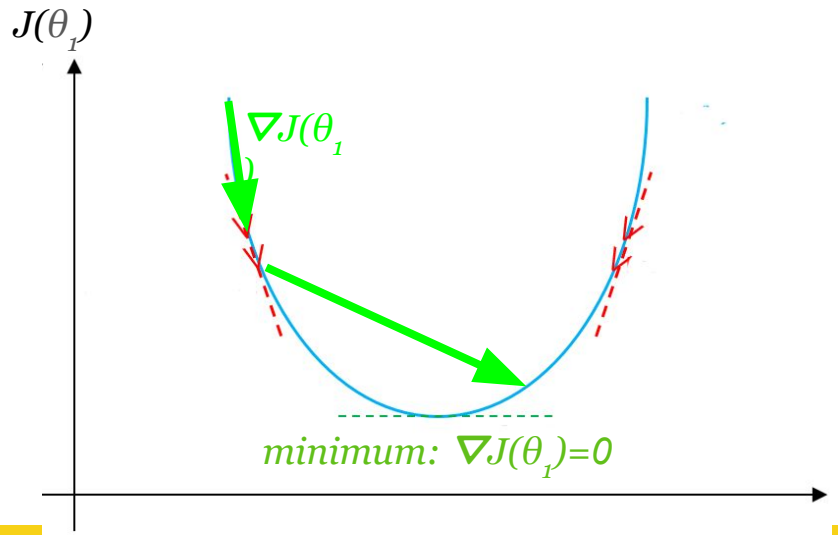
$$\theta_j := \theta_j - \alpha \underbrace{\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)}$$

}

Learning rate

Derivative

(simultaneously update $j = 0$ and $j = 1$)



If α is too small, gradient descent can be slow.

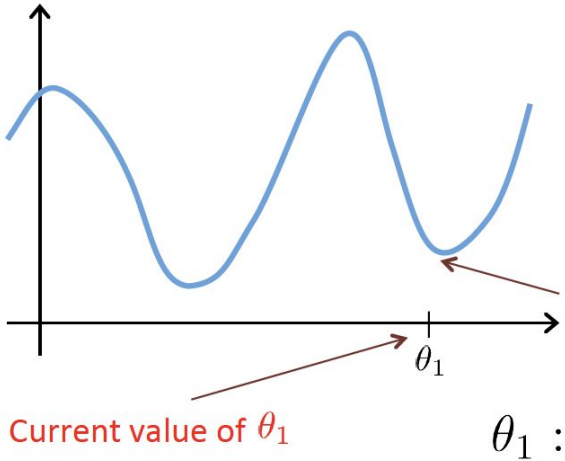
If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

Gradient Descent

Gradient descent can converge to a local minimum, even with the learning rate α fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease α over time.



$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Gradient Descent for Linear regression

Linear Regression Model	Gradient Descent

$h_\theta(x) = \theta_0 + \theta_1 x$
$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
	repeat until convergence {
	$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
	(for $j = 1$ and $j = 0$)
	}

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) =$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) =$$

Gradient Descent Algorithm

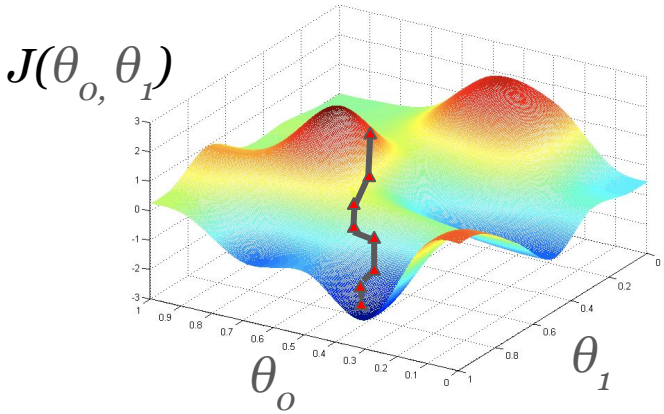
repeat until convergence {

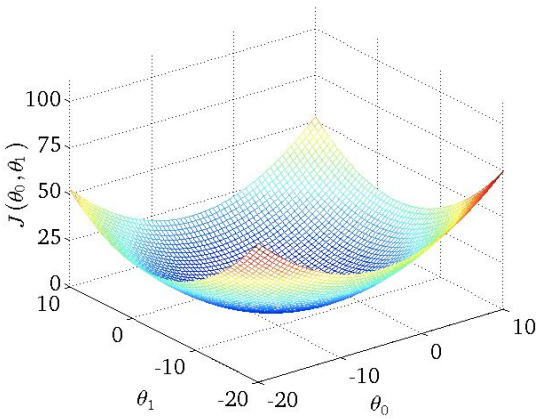
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

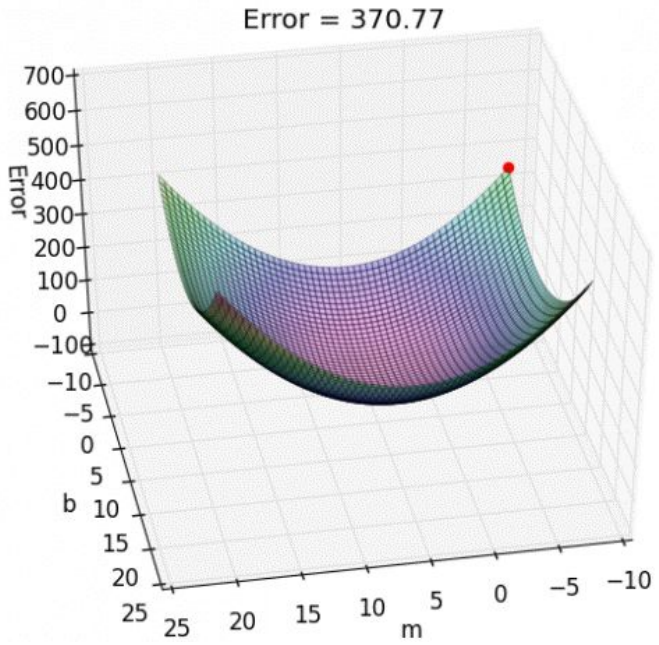
}

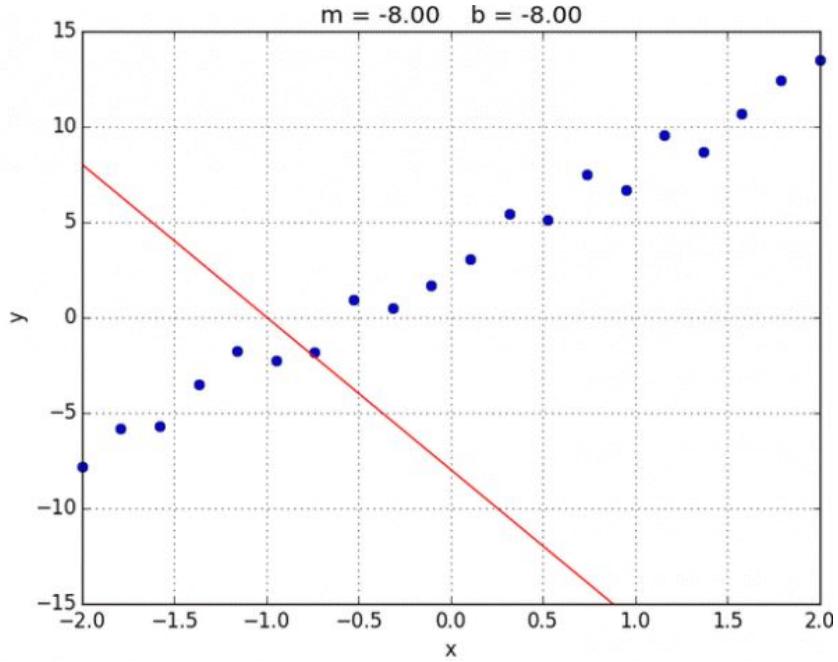
Simultaneous update





Gradient Descent Algorithm





Batch Gradient Descent

"Batch": each step of gradient descent uses all the training examples.

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

m = all the data.

Multiple Feature Linear Regression

Multiple Features (Variables)

price <dbl>	sqft <dbl>	beds <dbl>	bath <dbl>	laundry <chr>	pets <chr>	housing_type <chr>	parking <chr>
6800	1600	2	2.0	(a) in-unit	(d) no pets	(c) multi	(b) protected
3500	550	1	1.0	(a) in-unit	(a) both	(c) multi	(b) protected
5100	1300	2	1.0	(a) in-unit	(a) both	(c) multi	(d) no parking
9000	3500	3	2.5	(a) in-unit	(d) no pets	(c) multi	(b) protected
3100	561	1	1.0	(c) no laundry	(a) both	(c) multi	(d) no parking
3800	800	2	1.0	(b) on-site	(c) cats	(c) multi	(b) protected
3100	750	1	1.0	(a) in-unit	(d) no pets	(c) multi	(b) protected
3000	650	1	1.0	(b) on-site	(a) both	(c) multi	(d) no parking
3000	650	1	1.0	(b) on-site	(a) both	(c) multi	(b) protected
3200	650	1	1.0	(a) in-unit	(c) cats	(c) multi	(b) protected
3100	650	1	1.0	(b) on-site	(a) both	(c) multi	(b) protected
15000	3500	4	3.5	(a) in-unit	(d) no pets	(a) single	(b) protected

y x_1 x_2 x_3 x_4 x_5 x_6 x_7

Notation:

- n = number of features
- $x^{(i)}$ = input (features) of i^{th} training example
- $x_j^{(i)}$ = value of features j of i^{th} training example

Hypothesis

Previously: $h_\theta(x) = \theta_0 + \theta_1 x$

What is the multiple variables one?

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

For convenience of notation, define $x_0 = 1$

Multivariate linear regression.

Gradient Descent for Multivariate linear regression

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function: $\underbrace{J(\theta_0, \theta_1, \ldots, \theta_n)}_{J(\theta)} = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \overbrace{J(\theta_0, \ldots, \theta_n)}^{J(\theta)}$

Simultaneously update for every $j = 0, \ldots, n$

Gradient Descent

n=1: repeat

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

Simultaneously update θ_0, θ_1

n>1: repeat

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

Simultaneously update all θ's

Features

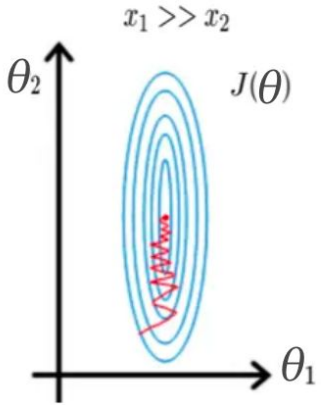
Feature Scaling

sqft <dbl>	beds <dbl>
1600	2
550	1
1300	2
3500	3
561	1
800	2
750	1
650	1
650	1
650	1
650	1
3500	4

Size: range 0-3500

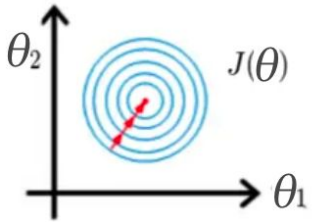
Number of bedrooms: 0-5

Gradient descent without scaling



Gradient descent after scaling variables

$0 \le x_1 \le 1$
$0 \le x_2 \le 1$



Idea: make sure features are on a similar scale.

Mean Normalization: Replace x_i with (x_i - μ_i / range)

$$x_1 = \frac{size-1000}{3500}$$

$$x_2 = \frac{\#bedrooms-3}{5}$$

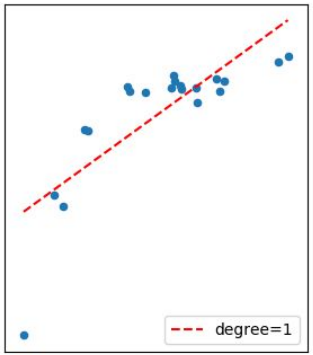
Standardization: statistic

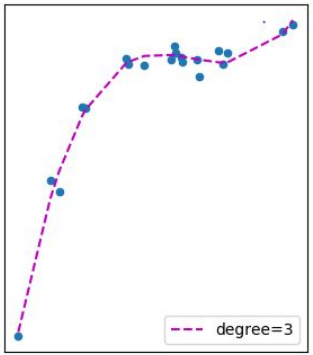
$$z = \frac{x_i - \mu}{\sigma}$$

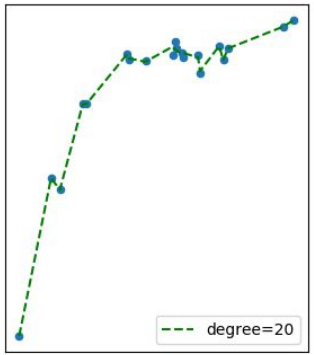
Polynomial Regression

Polynomial Regression

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$







Will this still linear?

What about the following?

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

Normal Equation

Normal Equation

Normal equation is a method to solve for θ analytically.

$$\theta \in \mathbb{R}^{n+1} \quad J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Set $\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0$ Solve $\theta_0, \theta_1, \ldots, \theta_n$

price <dbl>	sqft <dbl>	beds <dbl>	bath <dbl>
6800	1600	2	2.0
3500	550	1	1.0
5100	1300	2	1.0

$$X = \begin{bmatrix} 1 & 1600 & 2 & 2 \\ 1 & 550 & 1 & 1 \\ 1 & 1300 & 2 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 6800 \\ 3500 \\ 5100 \end{bmatrix}$$

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

Gradient Descent vs. Normal Equation

Gradient Descent

- Need choose learning rate
- Needs many iterations
- Works well even when feature size is large

Normal Equation

- No need learning rate
- No iteration
- Need to compute $(X^T X)^{-1}$
- Slow if feature size is very large
- If m << n, non-invertible (singular/degenerate), need to do feature deletion or regularization.

m: sample size, n: feature size

Logistic Regression

Classification

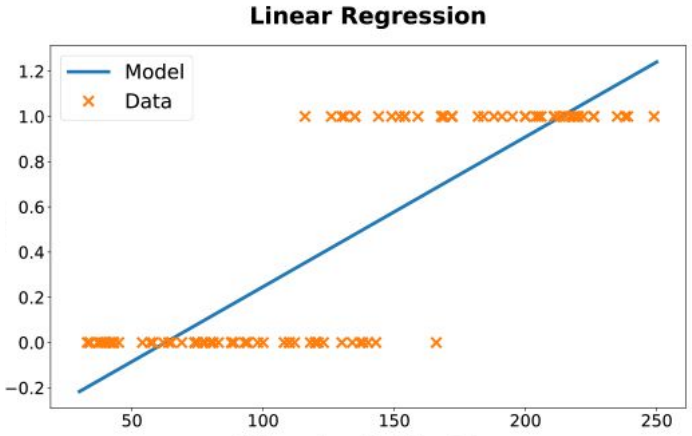
Email: spam vs. not spam

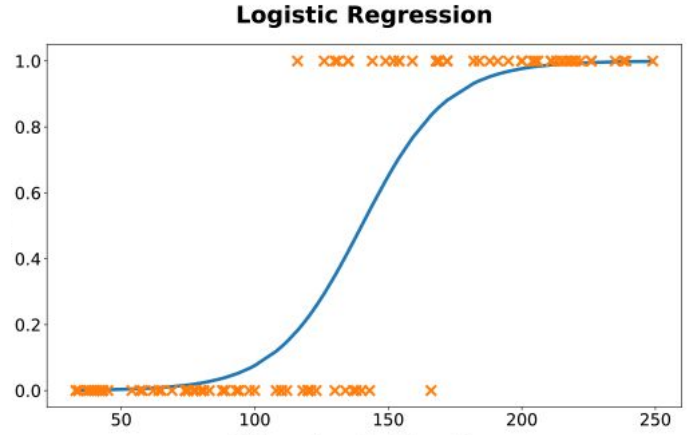
Online: transaction vs. fraudulent

Tumor: malignant vs. benign

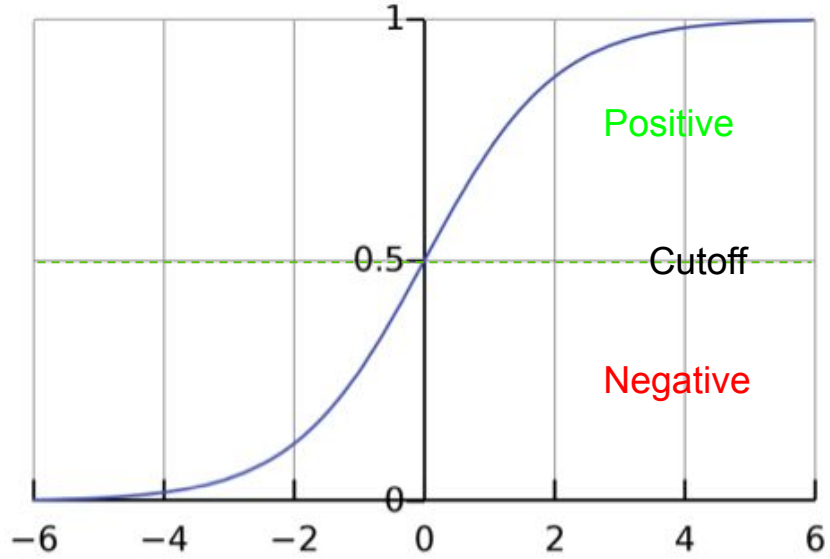
Want $h(x) \in [0,1]$. $h_\theta(x) = g(\theta^T x)$

Logistic Function: $g(z) = \frac{1}{1+e^{-z}}$.





Logistic Regression



$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$

Threshold classifier output $h_\theta(x)$ at 0.5.

- If $h_\theta(x) >= 0.5$, predict "$y=1$"
- If $h_\theta(x) > 0.5$, predict "$y=0$"

Sigmoid function = Logistic function

How do we interpret $h_\theta(x)$?

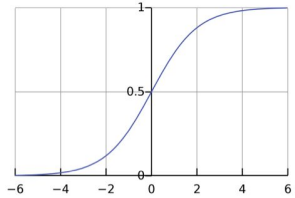
$$h_\theta(x) = 0.7$$

Tell patient that 70% chance of tumor being malignant

Logistic Regression

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}$$



$P(y = 1 \mid x;\theta) = h_\theta(x)$

"Probability that y=1, give x, parameterized by θ"

$P(y = 0 \mid x;\theta) = 1 - h_\theta(x)$

What is the $h(x)$ function? $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

$z = 0$:

$z \to \infty$:

$z \to -\infty$:

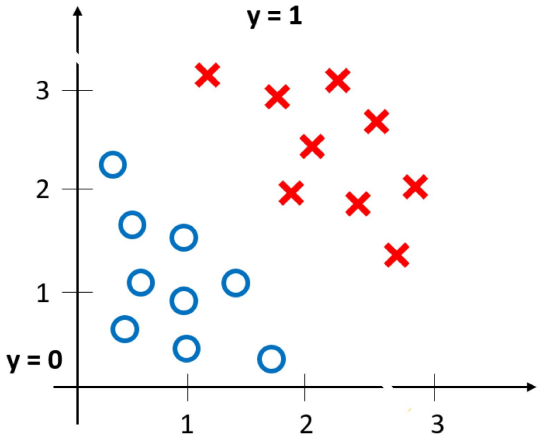
Logistic Regression

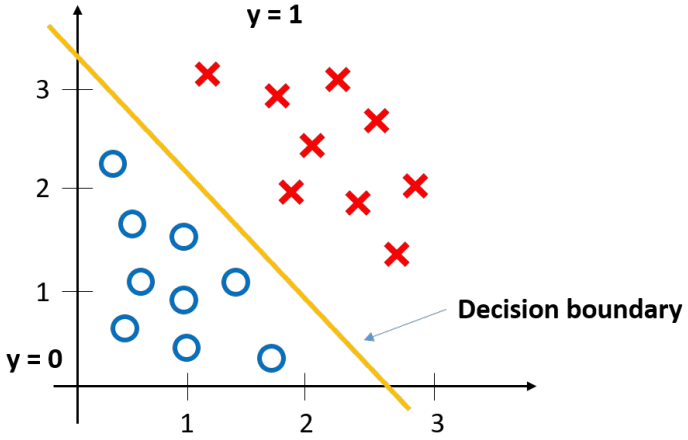
If we define h function as:

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

Logistic regression algo is actually to learn what values for coefficients allow a decision boundary.

Can you find 3 values that allow we can make a decision boundary between the two labels?



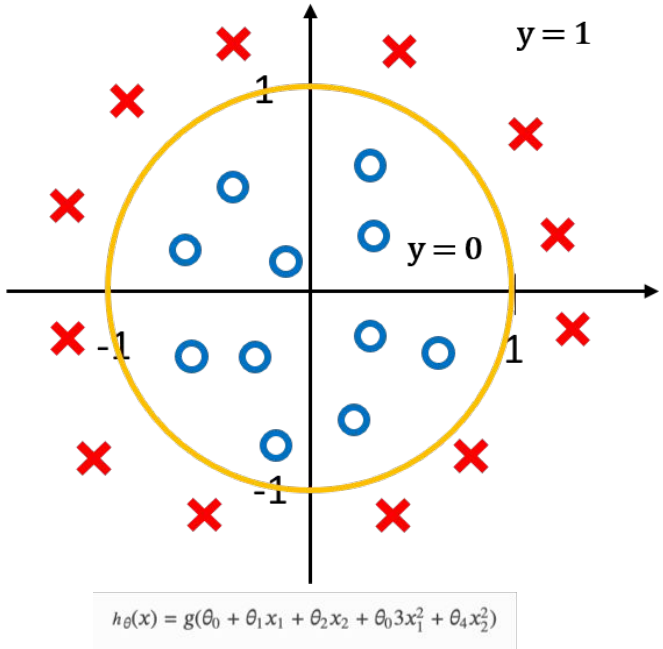


Non-linear Decision Boundaries

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict $y = 1$ if $-1 + x_1^2 + x_2^2 \geq 0$

What other similar formula you can think of?
Make sure they are ‘linear’.



Cost Function of Logistic Regression

Training data: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \qquad x_0 = 1, y \in \{0, 1\}$$

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

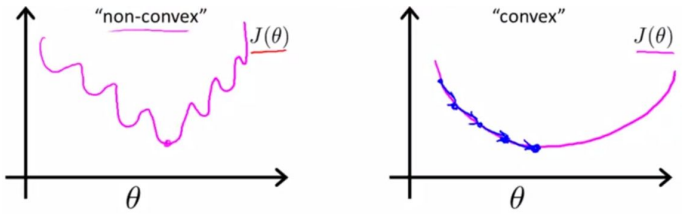
How to choose the parameters?

Cost Function of Logistic Regression

Linear regression cost function:

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

$$\text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$



Logistic regression cost function:

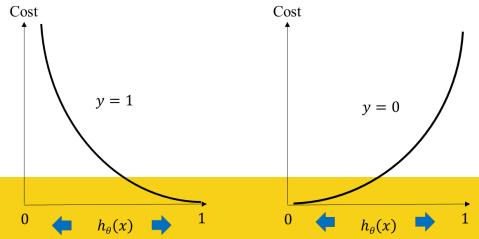
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$

But as $h_\theta(x) \to 0$

$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

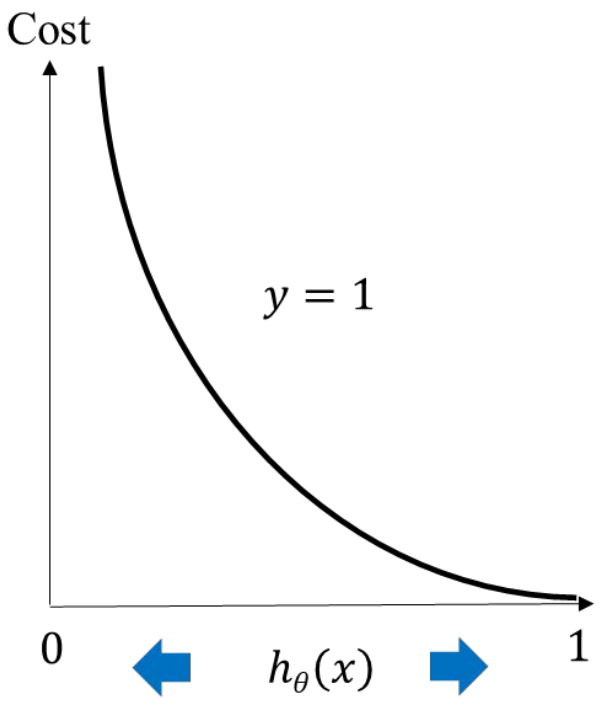


Cost Function of Logistic Regression

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

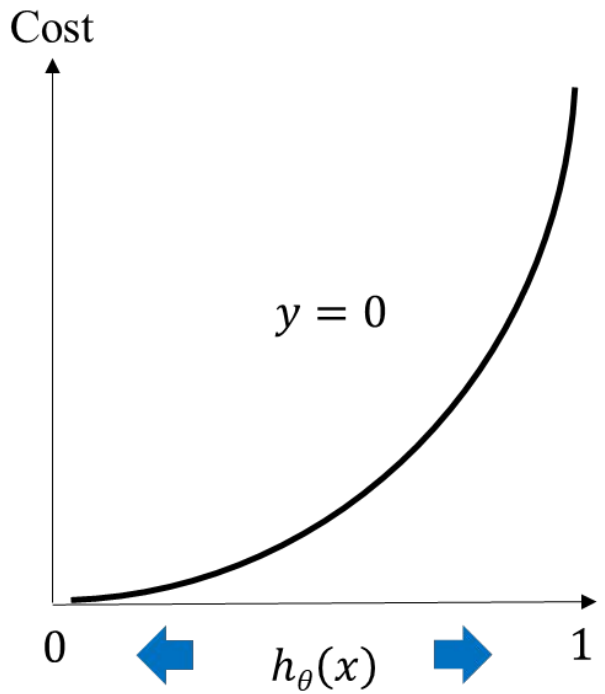
$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.



Cost Function of Logistic Regression

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Simplified Cost Function and Gradient Descent

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

To fit parameters θ:

$$\min_\theta J(\theta)$$

To make a prediction given new x.

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

Gradient descent:

$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$

Want $\min_\theta J(\theta)$

Repeat $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$ (simultaneously update all θ)

Algorithm looks identical to linear regression!

Advanced Optimization

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

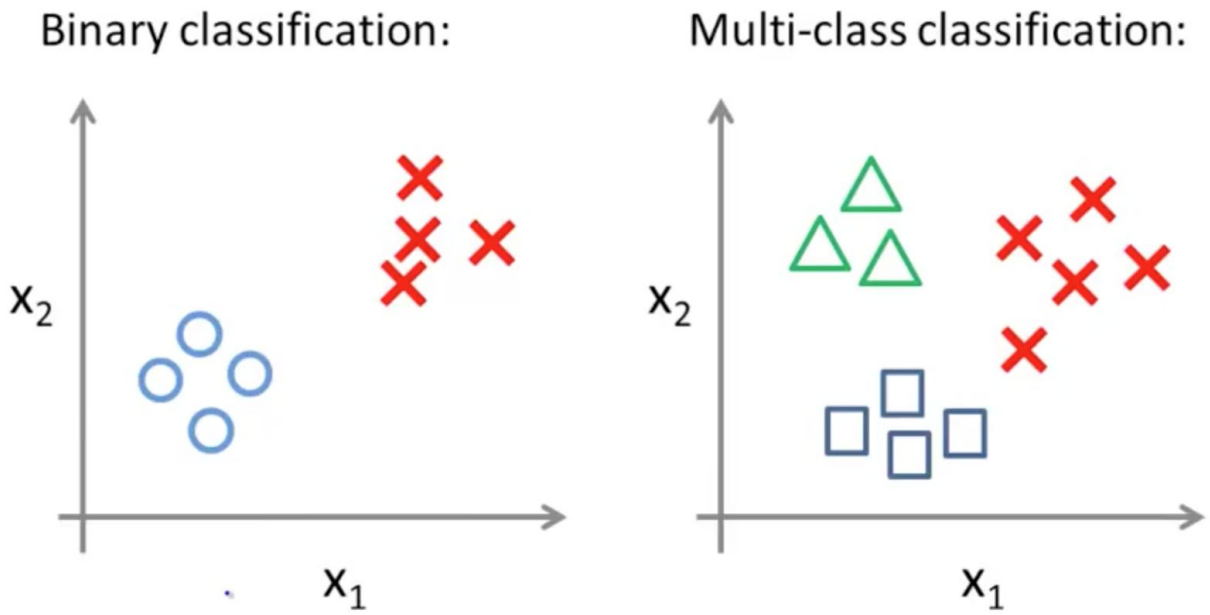
Advantages:

- No need to manually pick learning rate
- Often faster than gradientdescent.

Disadvantages:

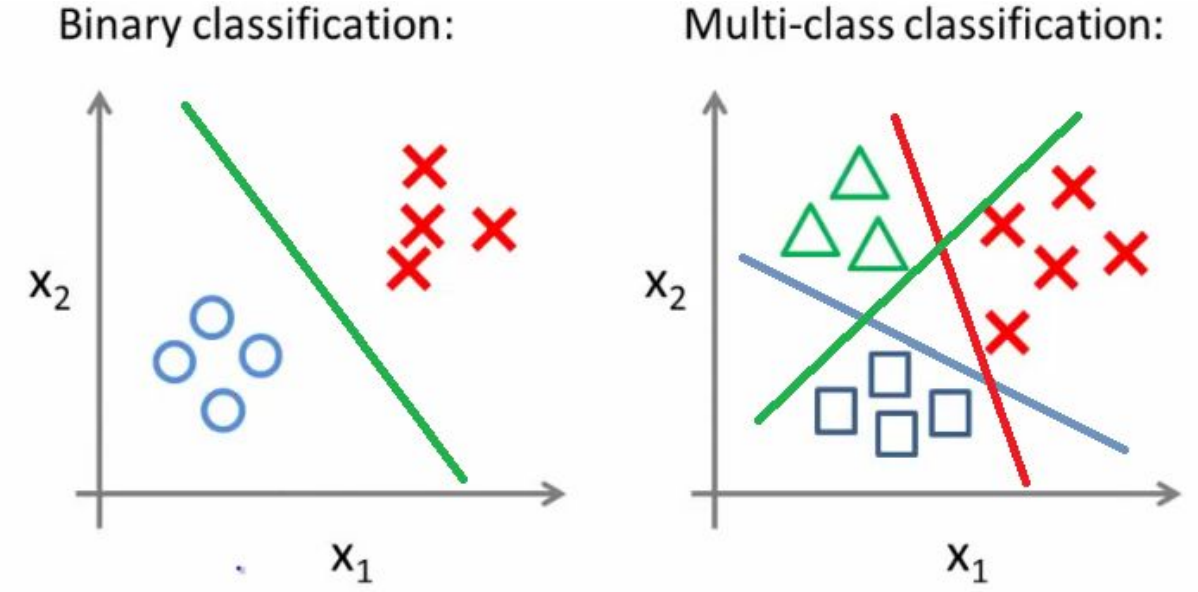
- More complex

Logistic Regression: Multi-class



What do you think we should do?

Logistic Regression: Multi-class



One-vs-all:

- Train a logistic regression classifier for each class i to predict the probability that $y=i$.
- On a new input x, to make a prediction, pick the class i that maximizes

$$\max_i h_\theta^{(i)}(x)$$

Model Fitting and Diagnosis

Measuring Success

Regression Metrics:

- Error: Difference between predicted and actual value.

Classification:

- True Positive: Correctly identified as relevant
- True Negative: Correctly identified as not relevant
- False Positive: Incorrectly labeled as relevant
- False Negative: Incorrectly labeled as not relevant

Example: Image Classification (Identify Cats)

Prediction:      

Image:      

True Positive

True Negative

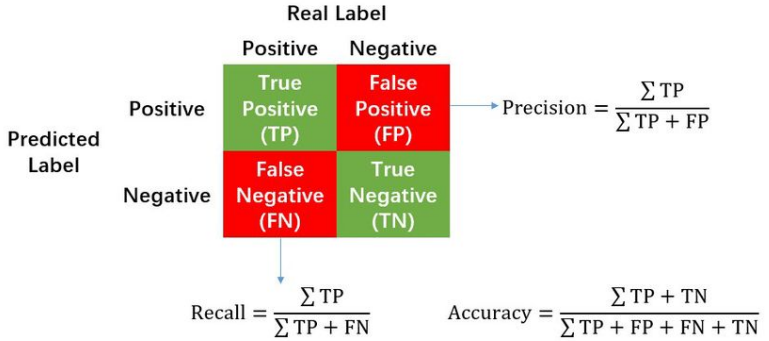
False Negative

False Positive

Images from the STL-10 dataset

Precision, Recall, and Accuracy

- Precision
  - Percentage of positive labels that are correct
  - Precision = (# true positives) / (# true positives + # false positives)
- Recall
  - Percentage of positive examples that are correctly labeled
  - Recall = (# true positives) / (# true positives + # false negatives)
- Accuracy
  - Percentage of correct labels
  - Accuracy = (# true positives + # true negatives) / (# of samples)



Confusion Matrix

F1 Score

How to compare precision/recall numbers

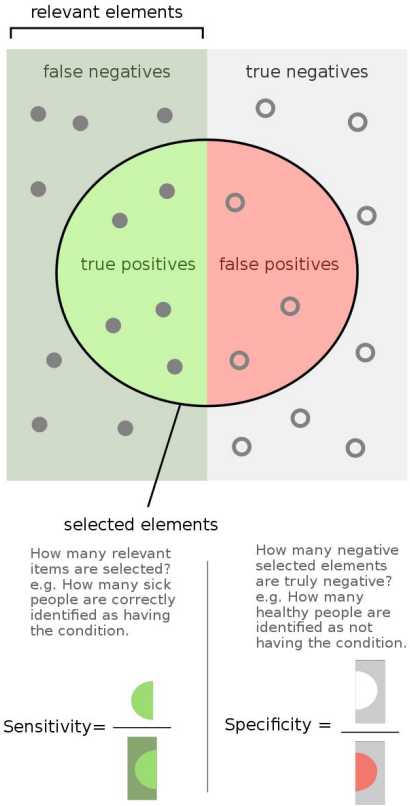
Average $\frac{P+R}{2}$

F1 Score $2\frac{PR}{P+R}$

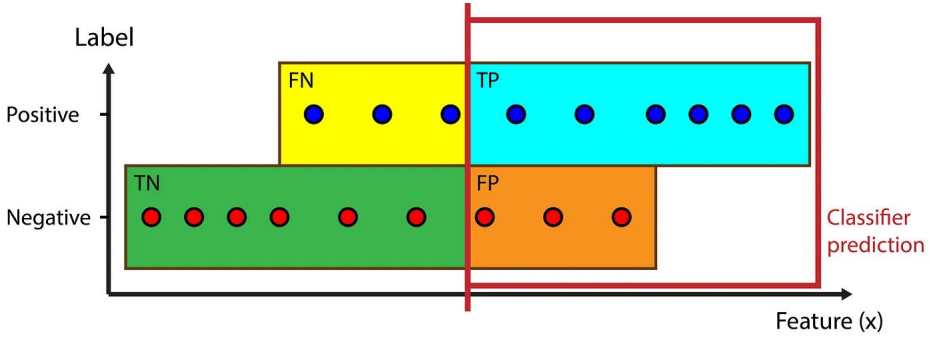
	Precision (P)	Recall (R)
Algo 1	0.5	0.4
Algo 2	0.7	0.2
Algo 3	0.02	1.0

Sensitivity and Specificity

	Actual Classes: Positive	Actual Classes: Negative	Evaluation Metrics
Classification outputs: Positive	True Positive (TP)	False Positive (FP)	Positive Predictive Value (PPV)/Precision TP/(TP+FP)
Classification outputs: Negative	False Negative (FN)	True Negative (TN)	Negative Predictive Value (NPV)/Precision TN/(TN+FN)
	P = TP + FN	N = FP + TN	Accuracy (TP + TN) = (P + N)
Evaluation Metric	Sensitivity (Recall) TP/P	Specificity (TN/N) = 1 - FPR	



Put all together



Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision = TP / (TP + FP)

Specificity or TNR = TN / (TN + FP)

Sensitivity or Recall or TPR = TP / (TP + FN)

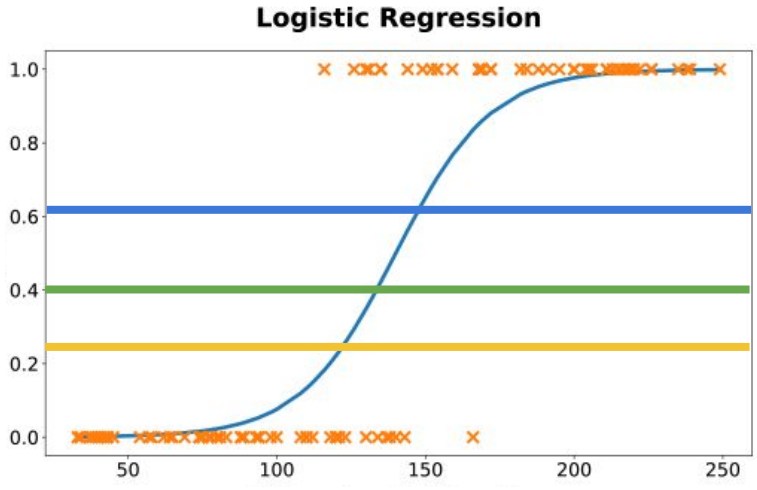
FPR = FP / (TN + FP)

Can you calculate these metrics?

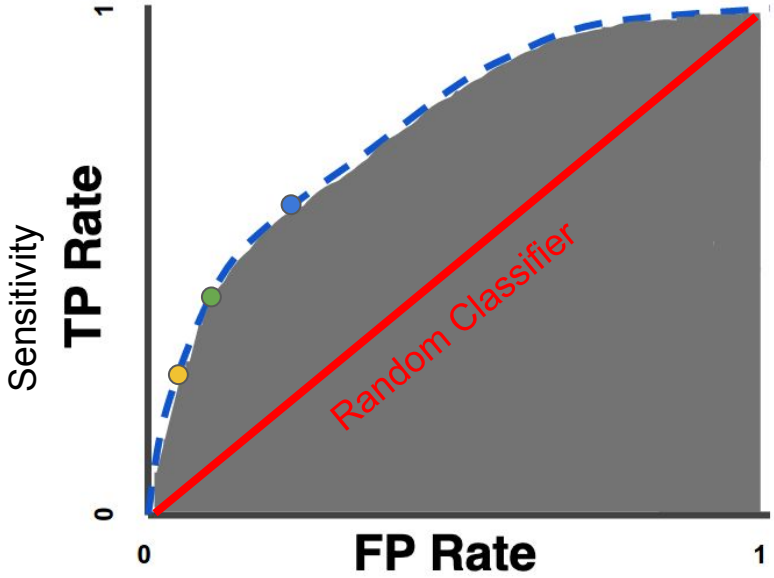
- Accuracy:
- Precision:
- Specificity:
- Sensitivity:
- FPR:

Receiver Operating Characteristic (ROC)

A ROC curve is a graph that shows the performance of a binary classifier, which is a model that classifies objects into two groups based on their features



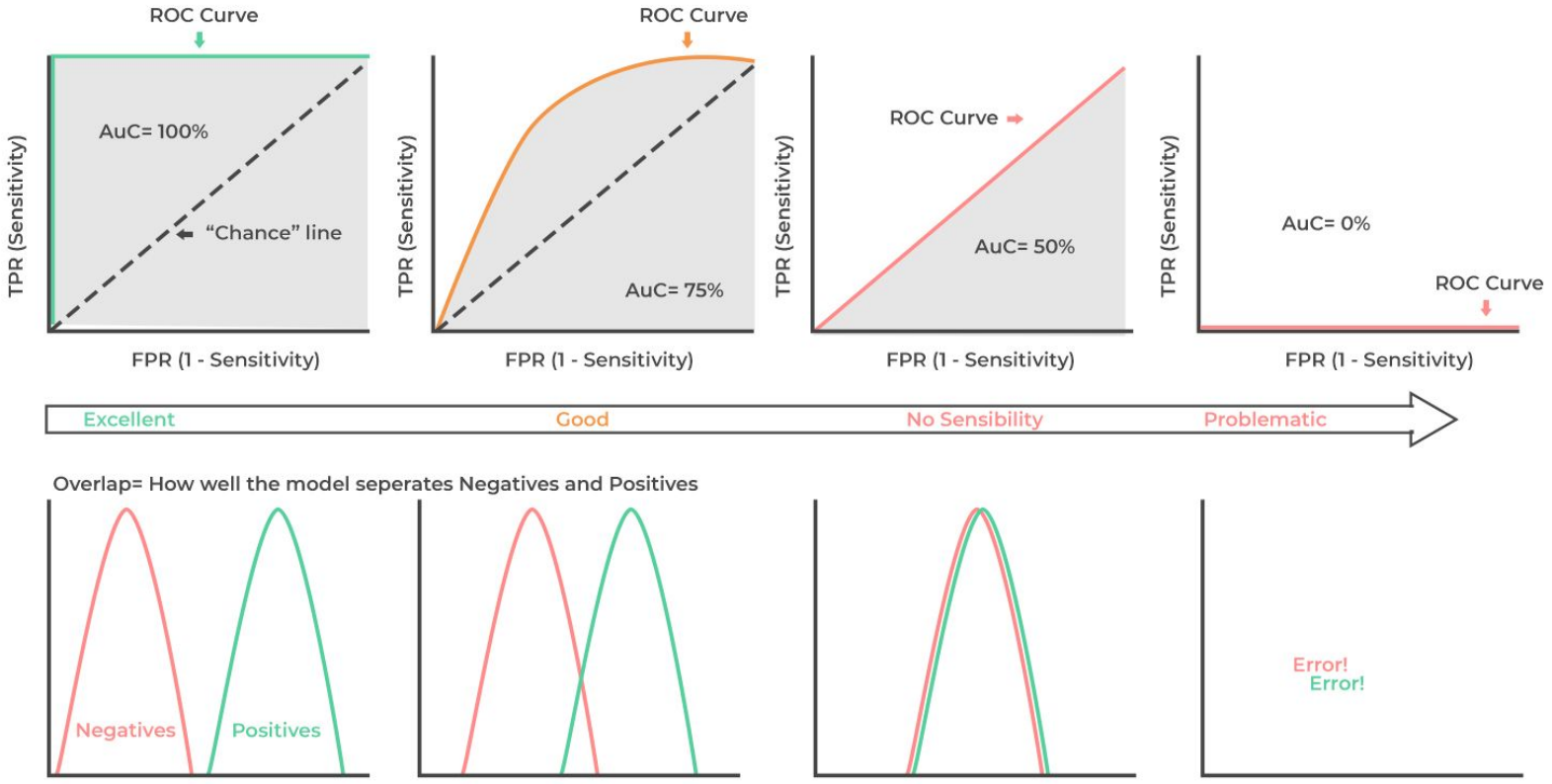
Cutoff



1 - Specificity

http://navan.name/roc/

Receiver Operating Characteristic



Training and Test Data for Linear Regression

Training: Learn parameter θ from training data (minimizing training error $J(\theta)$)

Testing: assess the accuracy of model

Ration:

- Train:test :: 70:30
- Train:test :: 80:20

Why? If we don't have test set,

what will happen?

Overfitting

Fails to generalize to new examples not in training set.

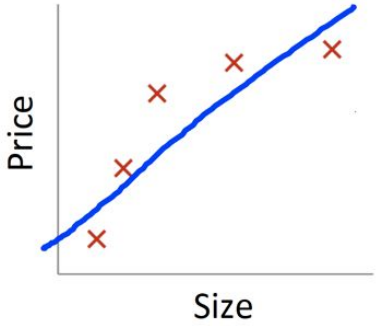
How to make the model generalize? (Perform well or decent on new data)

Testing dataset: will be the judge when to stop training to prevent overfitting, but how can we evaluate the model?

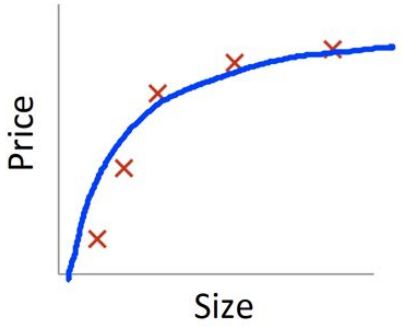
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Source of overfitting

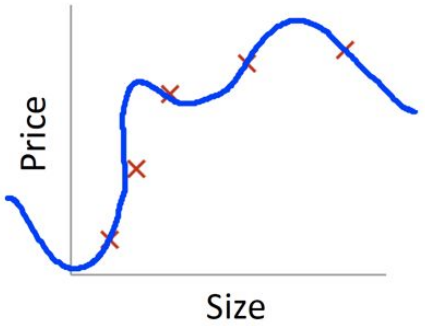
Model too complex:



$\theta_0 + \theta_1 x$



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



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

$$h_\theta(x) = \theta_0 + \theta_1 x$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$$
$$\vdots$$
$$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$$

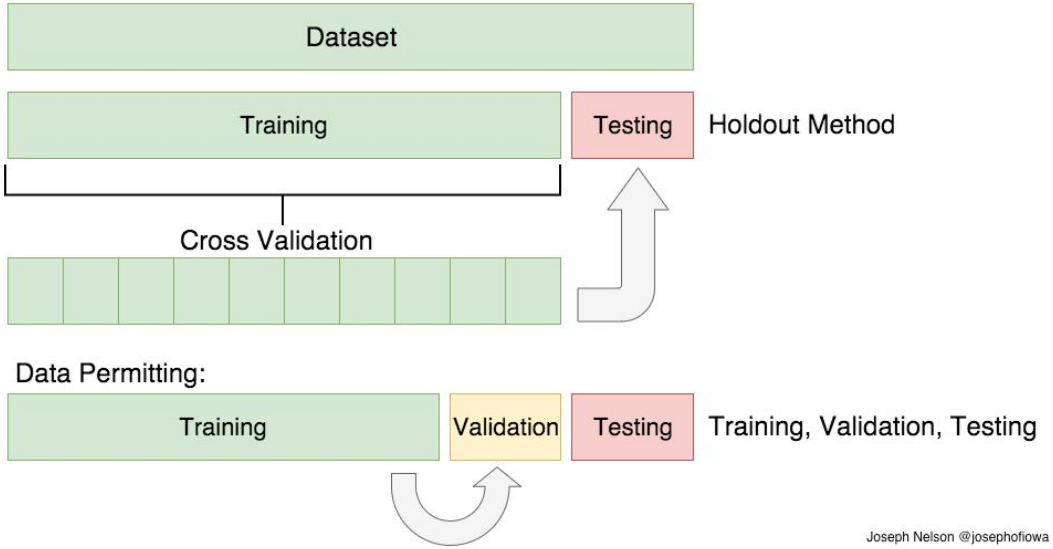
Address Overfitting

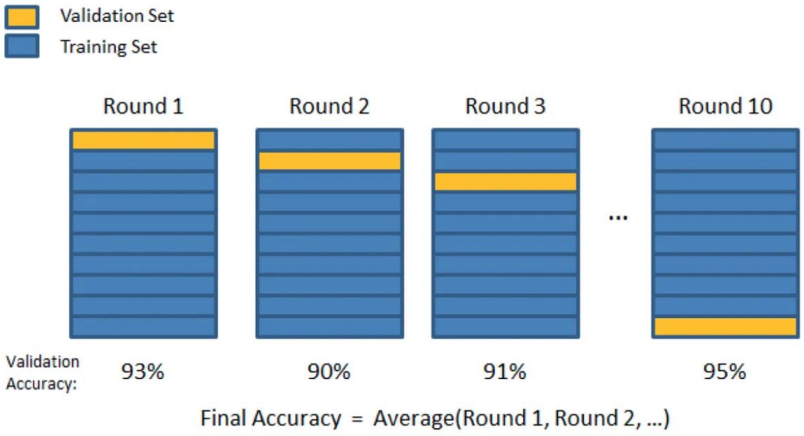
Options:

1. Reduce number of features.
   a. Manually select features to keep.
   b. Model selection algorithm (later in course).
2. Regularization (Will talk more later).
   a. Keep all the features, but reduce magnitude/values of parameters.
   b. We can added into the cost function so it will automatically do the job for us

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right]$$

Train:Validation:Test Set (Data is large enough)





Here testing is the final evaluation for the model.

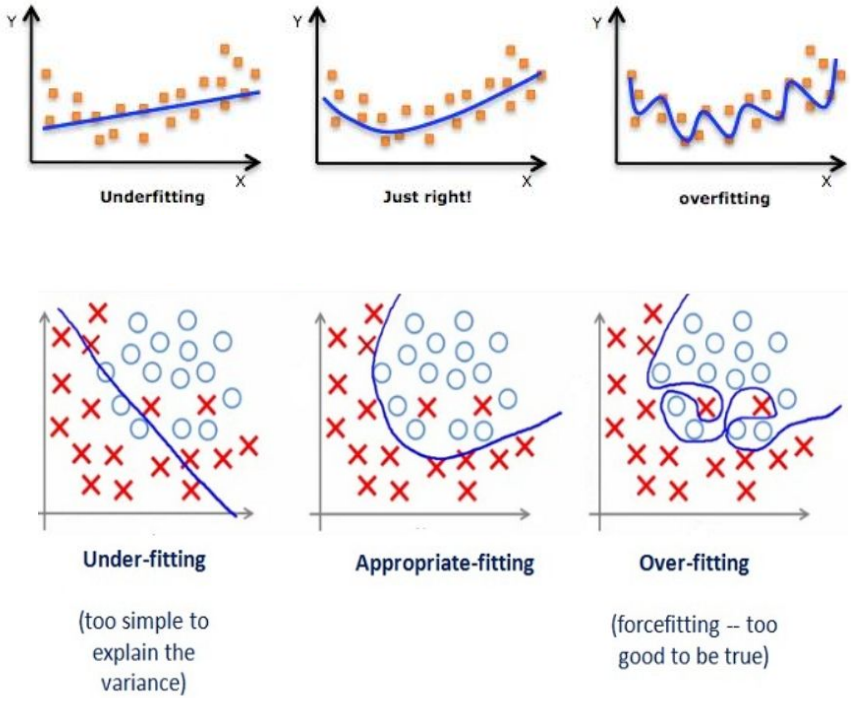
Bias vs. Variance

Bias: expected difference between model's prediction and truth

Variance: how much the model differs among training sets

Model Scenarios:

- High Bias: Model makes inaccurate predictions on training data
- High Variance: Model does not generalize to new datasets
- Low Bias: Model makes accurate predictions on training data
- Low Variance: Model generalizes to new datasets



Probability Review and Weighted Least Squares

Probability Review

We have already seen the linear regression model function:

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

Where did this model come from?

One way to view is via a *probabilistic interpretation* (helpful throughout the course).

Probability vs. Linear Regression

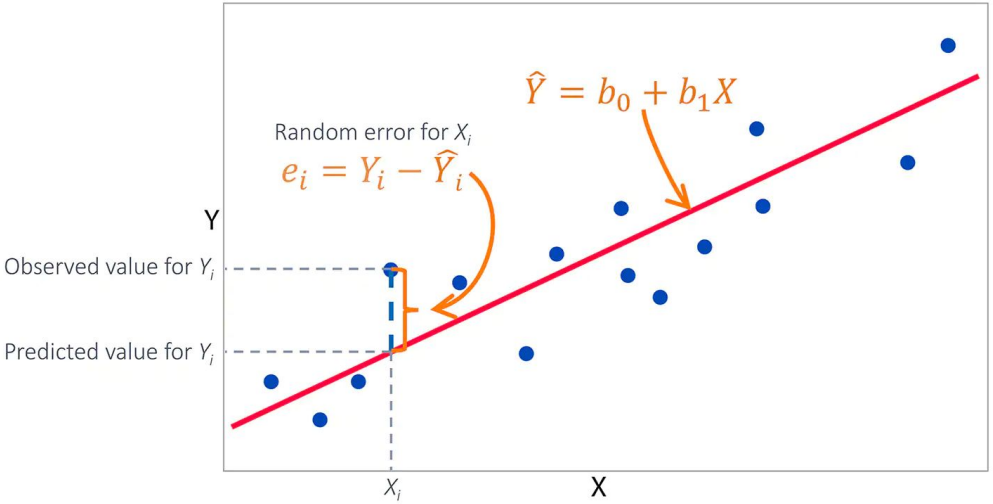
We make an assumption (common in statistics) that the data are *generated* according to some model (that may contain random choices). That is,

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

Here, $\varepsilon^{(i)}$ is a random variable that captures “noise" that is, unmodeled effects, measurement errors, etc.

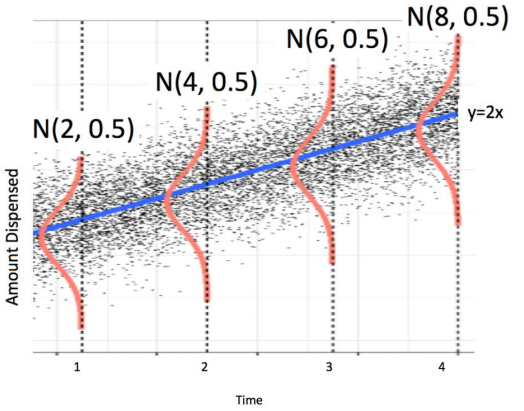
Please keep in mind: this is just a model! As they say, all models are wrong but some models are useful. This model has been shockingly useful.

Properties of Noise



Error follow a normal distribution:

$\varepsilon \sim N(0, \sigma^2)$



Modified from image at https://bookdown.org/roback/bookdown-bysh/ch-poissonreg.html

Weighted Least Squares

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

The method of **ordinary least squares** assumes that there is constant variance in the errors (which is called **homoscedasticity**). The method of **weighted least squares** can be used when the ordinary least squares assumption of constant variance in the errors is violated (which is called heteroscedasticity).

If we define the reciprocal of each variance, σ_i^2, as the weight, $w_i = 1/\sigma_i^2$, then let matrix W be a diagonal matrix containing these weights:

$$\begin{pmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n^2 \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & w_n \end{pmatrix}$$

Weighted Least Squares

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

Normal Equation

Normal equation is a method to solve for θ analytically.

$\theta \in \mathbb{R}^{n+1}$ $J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Set $\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0$ Solve $\theta_0, \theta_1, \ldots, \theta_n$

price <dbl>	sqft <dbl>	beds <dbl>	bath <dbl>
6800	1600	2	2.0
3500	550	1	1.0
5100	1300	2	1.0

$$X = \begin{bmatrix} 1 & 1600 & 2 & 2 \\ 1 & 550 & 1 & 1 \\ 1 & 1300 & 2 & 1 \end{bmatrix} \quad y = \begin{bmatrix} 6800 \\ 3500 \\ 5100 \end{bmatrix}$$

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

Slide 42

$$\begin{pmatrix} \sigma_1^2 & 0 & \ldots & 0 \\ 0 & \sigma_2^2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_n^2 \end{pmatrix}$$

Normal equation

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

$$\mathbf{w} = \begin{pmatrix} w_1 & 0 & \ldots & 0 \\ 0 & w_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & w_n \end{pmatrix}$$

$$\Theta = (X^T W X)^{-1} X^T W y$$

Final Project