CS 383 – Machine Learning

Regression

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Linear Regression
- Reading
  - Springer
    - Sections 3.1-3.2 – Linear Regression
    - Section 5.1 – Cross Validation

Regression

- Ok so in this course we want to:
  - ~~Look at and manipulate data~~
  - ~~Be able to find patterns in unlabeled data (clustering)~~
  - Build a system that, given some data, can predict a values (regression)
    - This system will be built using prior labeled data
  - Build a system that, given some data, can predict a label for that data (classification)
    - This system will be built using prior labeled data

Supervised Learning

- As mentioned earlier, with regression we're trying to predict a value given some data
- These systems are built using prior *labeled* data
  - That is, for each data sample X_t there is an associated value Y_t
  - Together this gives us our dataset: $\{X_t, Y_t\}_{t=1}^{N}$
- Since we have labeled data, this is our first example of *supervised learning.*
  - And there's several things/caveats about supervised learning that we must first discuss

Data Sets

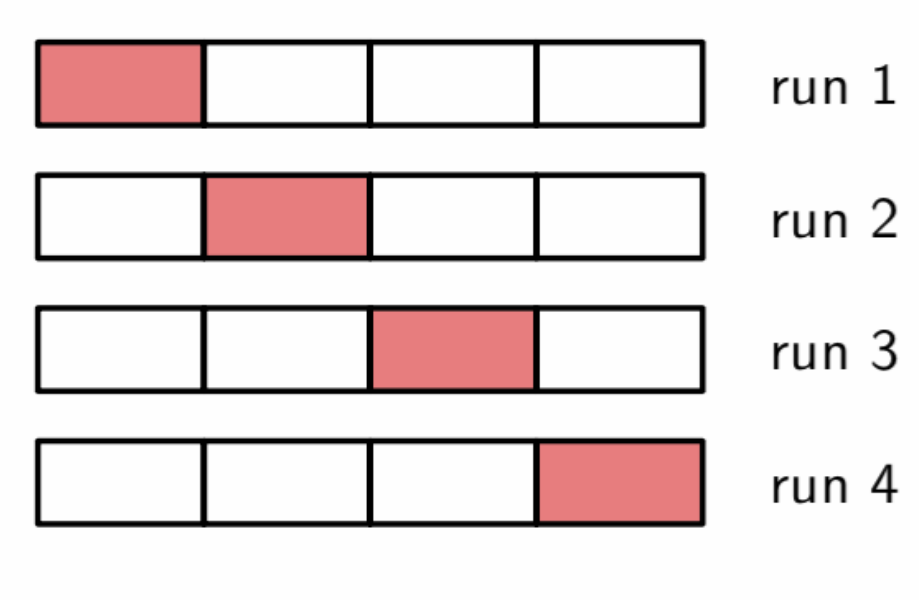
- For constructing our final model/system we will want to use all of our data
- However, often we need to figure out which model is best
- Therefore we will split our data into two groups:
  1. **Training Data**
  2. **Testing Data**
- Typically this is done as a 2/3 training, 1/3 testing split
- We then build/train our system using the training data and test our system using the testing data
- Now we can compare this system against others!
- Once we've chosen our model then we can use all the data and know what the upper-bound on the error should be
- The key is to have the training data and testing data pulled from the same distribution

Cross Validation

- What if we don't have that much data?
- Then we can do something called *cross-validation*
- Here we do several training(and validation, if necessary)/testing runs
  - Keeping track of all the errors
- We can then compute statistics for our this classifier based on the list of errors.
- Again, in the end our final system will be build using all the data.

S-Folds Cross Validation

- There are a few types of cross-validation
  - S-Folds: Here we'll divide our data up into S parts, train on $S-1$ of them and test the remaining part. Do this S times
  - Leave-one-out: If our data set is really small we may want to built our system on $N-1$ samples and test on just one sample. And do this N times (so it's basically N-folds)



Learning Function

- In general with supervised learning, with the *absence* of *noise* we can say there is some function $f(x)$ such that

$$y = f(x)$$

- However, in reality, what we observe may be noisy:

$$y = f(x) + \epsilon$$

- The error ϵ can come from several things:
  - Error in measurements
  - Other variables other than those in x that would account for y

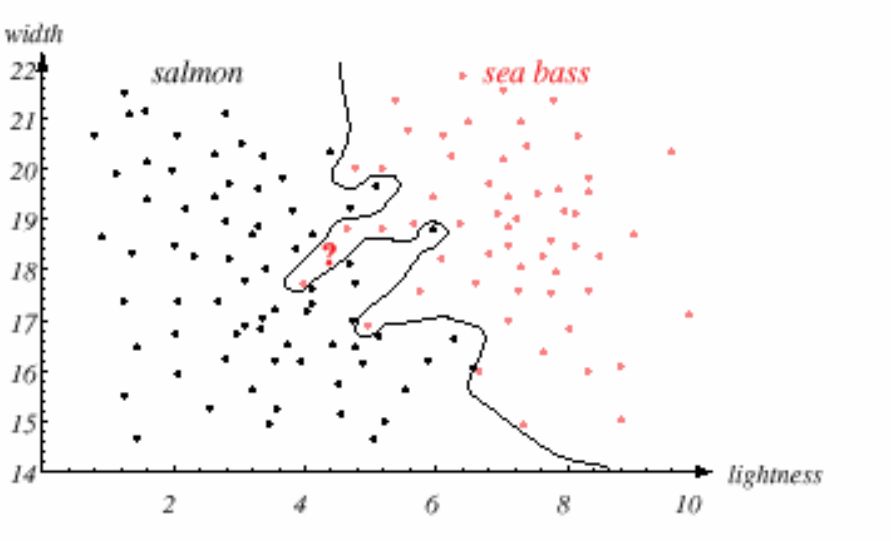
Learning Function

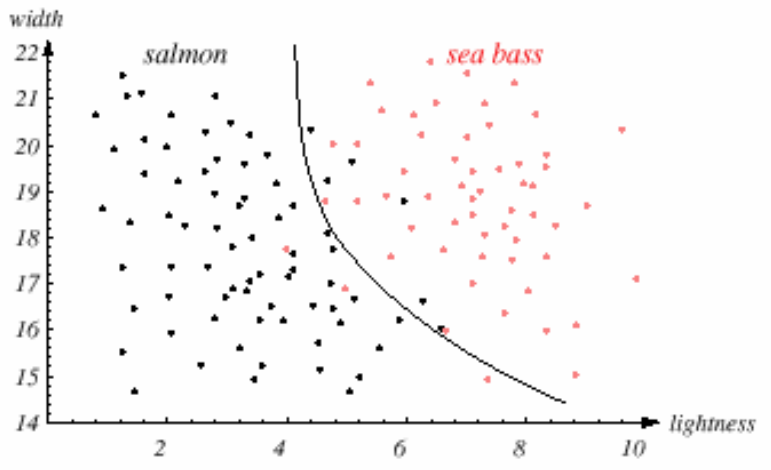
$$y = f(x) + \epsilon$$

- Ultimately what we want to do is to learn a function $g(x)$ that is an approximation of the true underlying $f(x)$
- That is, we want to learn the real underlying model, not the model+error.

Over/Underfitting

- The biggest problem with supervised learning is over or under-fitting
- If we over-fit our training data, then we're fitting the noise $f(x) + \epsilon$ and not $f(x)$, and therefore may not fit the general data
- If we under-fit our data then our model $g(x)$ is not specific enough to be an approximation to $f(x)$.

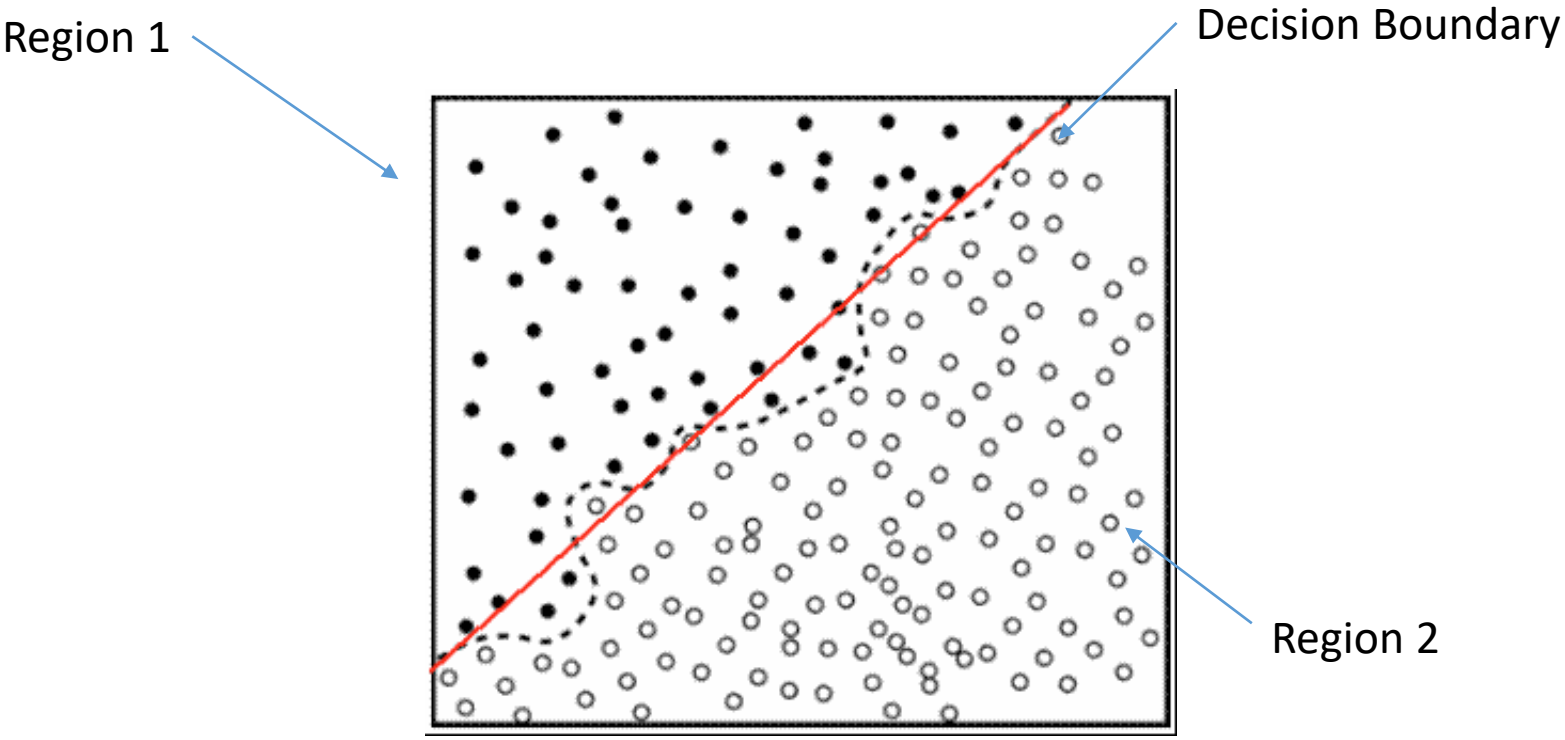




Over/Underfitting

- Here's some examples of how/when over and under fitting can occur
  - Too complex of a model – Such flexibility may allow us to fit the training data very well, but not generalize well. The model is likely learning to noise too (overfitting)
  - Too simple of a model – Our model may not be flexible enough to model what's truly happening (underfitting).
  - Too little data – Easy for a model to fit this data well (overfit)

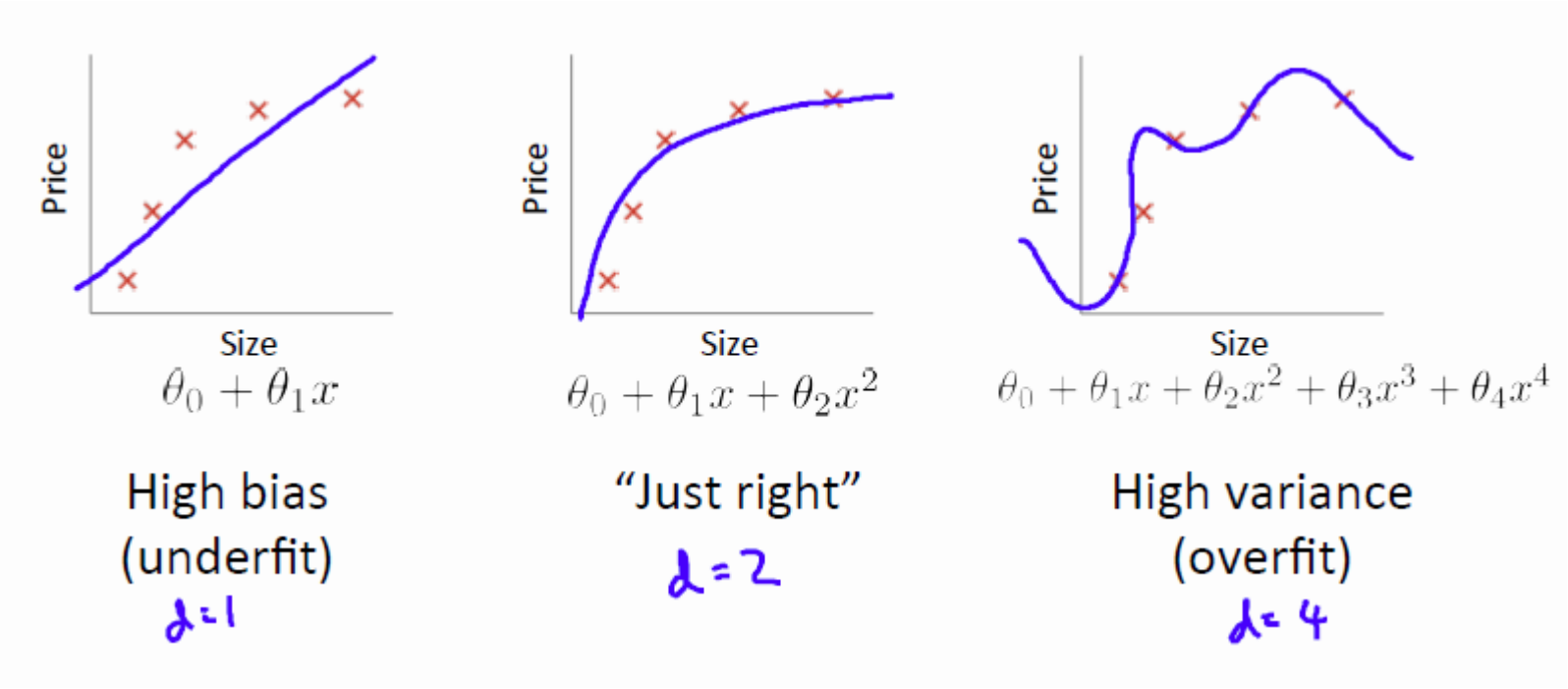
Over/Under Fitting



Bias vs Variance

- Sometimes instead of talking about over-fitting and under-fitting we talk about *bias* and *variance*
- If our trained model is very dependent on the training data set (that is it fits it very well), then there will be a large ***variance*** in the models given different training sets.
  - Therefore we say variance and overfitting are related
- Conversely, if our model barely changes at all when it sees different training data, then we say it is **biased**.
  - Therefore we say bias is related to underfitting.

Over/Under Fitting



Dealing with Over/Under-Fitting

- How can we detect under-fitting?
  - If we don't do well on either the training or the testing sets
- How can we detect over-fitting?
  - If we do well on the training set but poorly on the testing set.
- How can we deal with under-fitting?
  - Make a more complex model
    - May involve need more features
  - Trying a different algorithm
- How can we deal with over-fitting?
  - Use a less complex model
    - May involve using less features
  - Try a different algorithm.
  - Get more data
  - Use a third set to choose between hypothesis (called a *validation set)*.
  - Add a penalization (or regularization) term to the equations to penalize model complexity.

Linear Regression

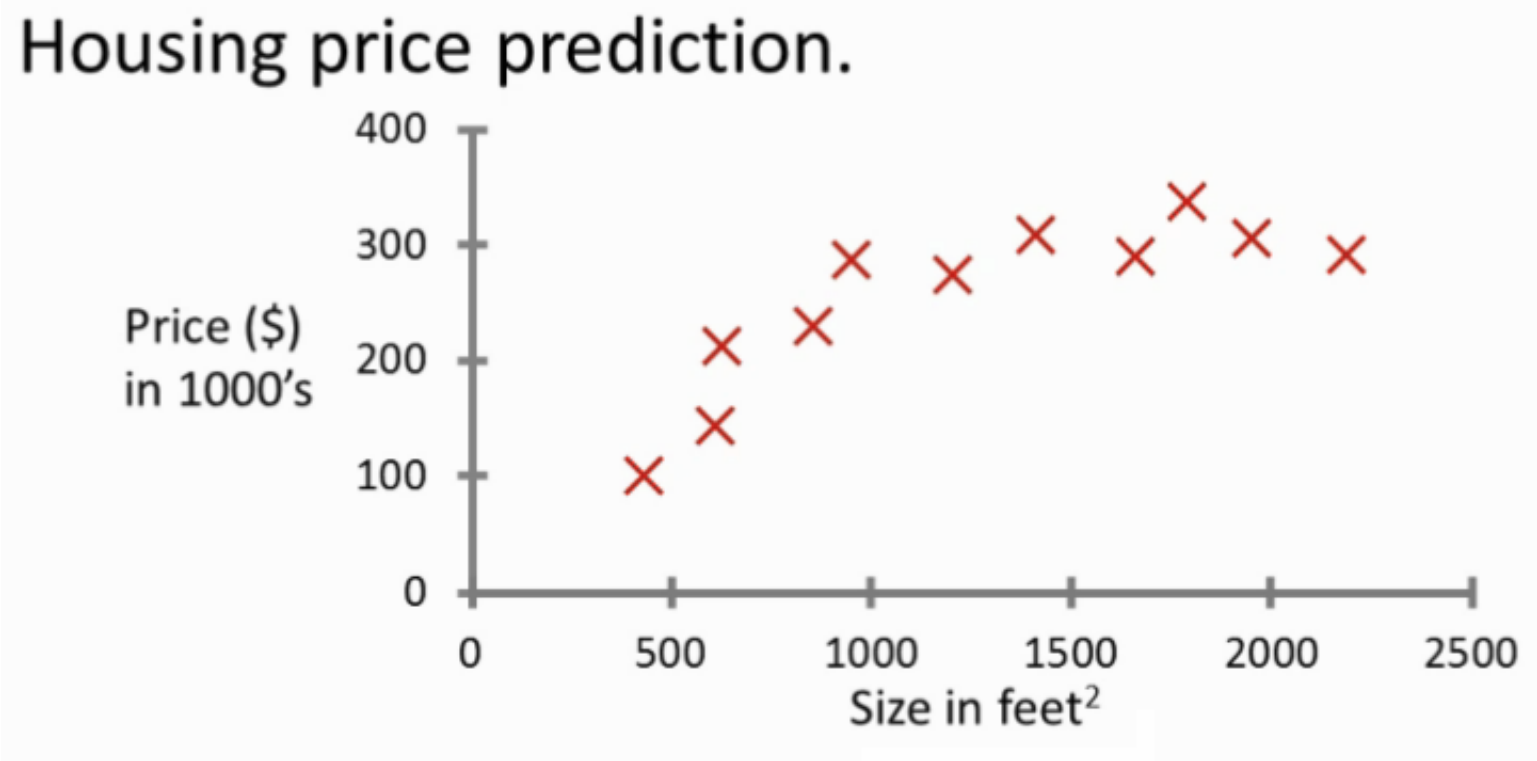
Linear Regression

- For regression $f(x)$, and therefore $g(x)$, return a *continuous-value*
- The simplest regression function, and the one we will study, is called *linear regression* since $g(x)$ is a linear combination of the features.
  - Here we make the assumption that the underlying true function $f(x)$ is near linear.

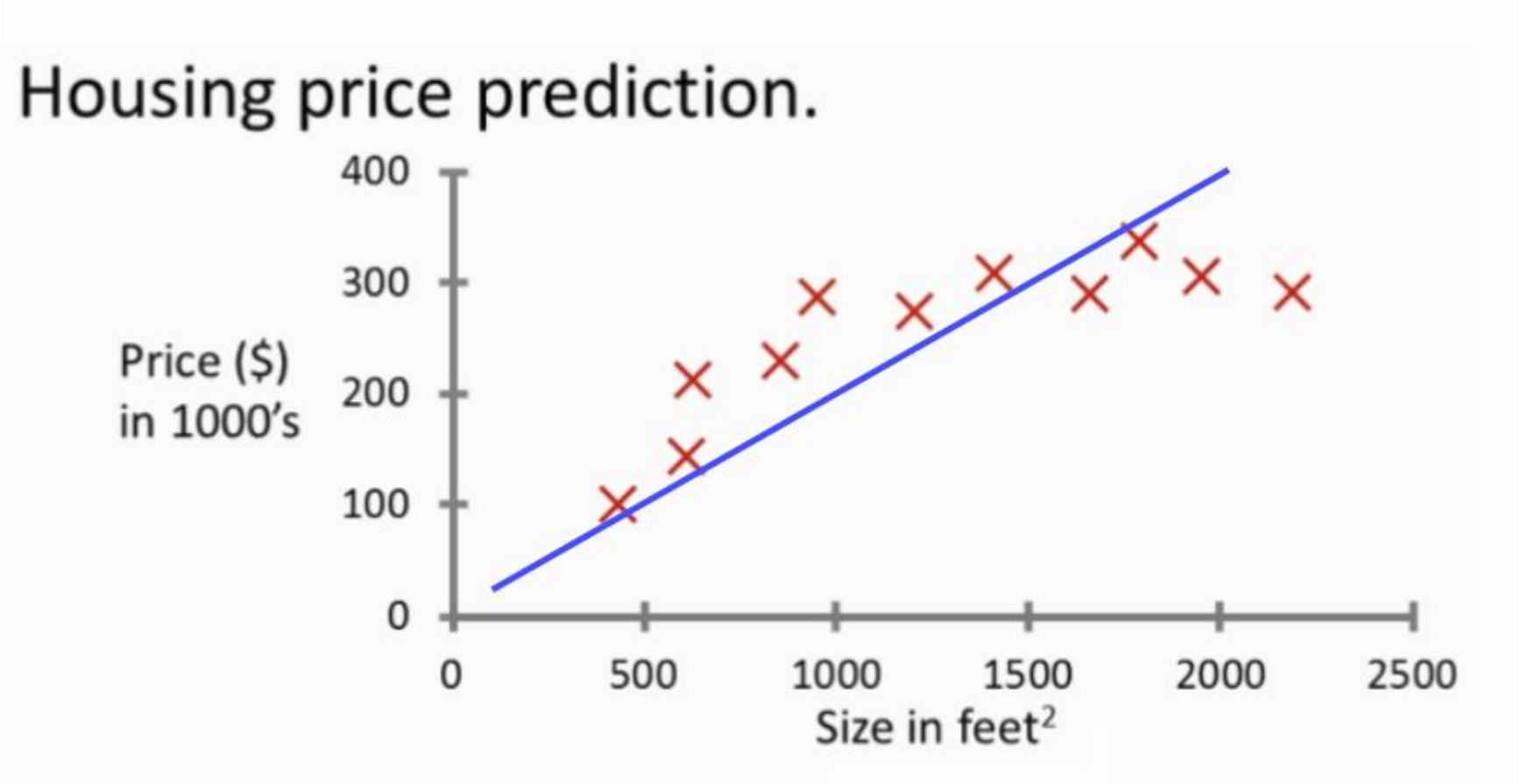
$$g(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_D x_D$$

- Linear regression is a *parameter-based* algorithm since we are trying to learn the parameters $\theta = [\theta_0, \theta_1, \ldots, \theta_D]$ of the underlying model

Linear Regression Example



Linear Regression Example



Linear Regression Analysis

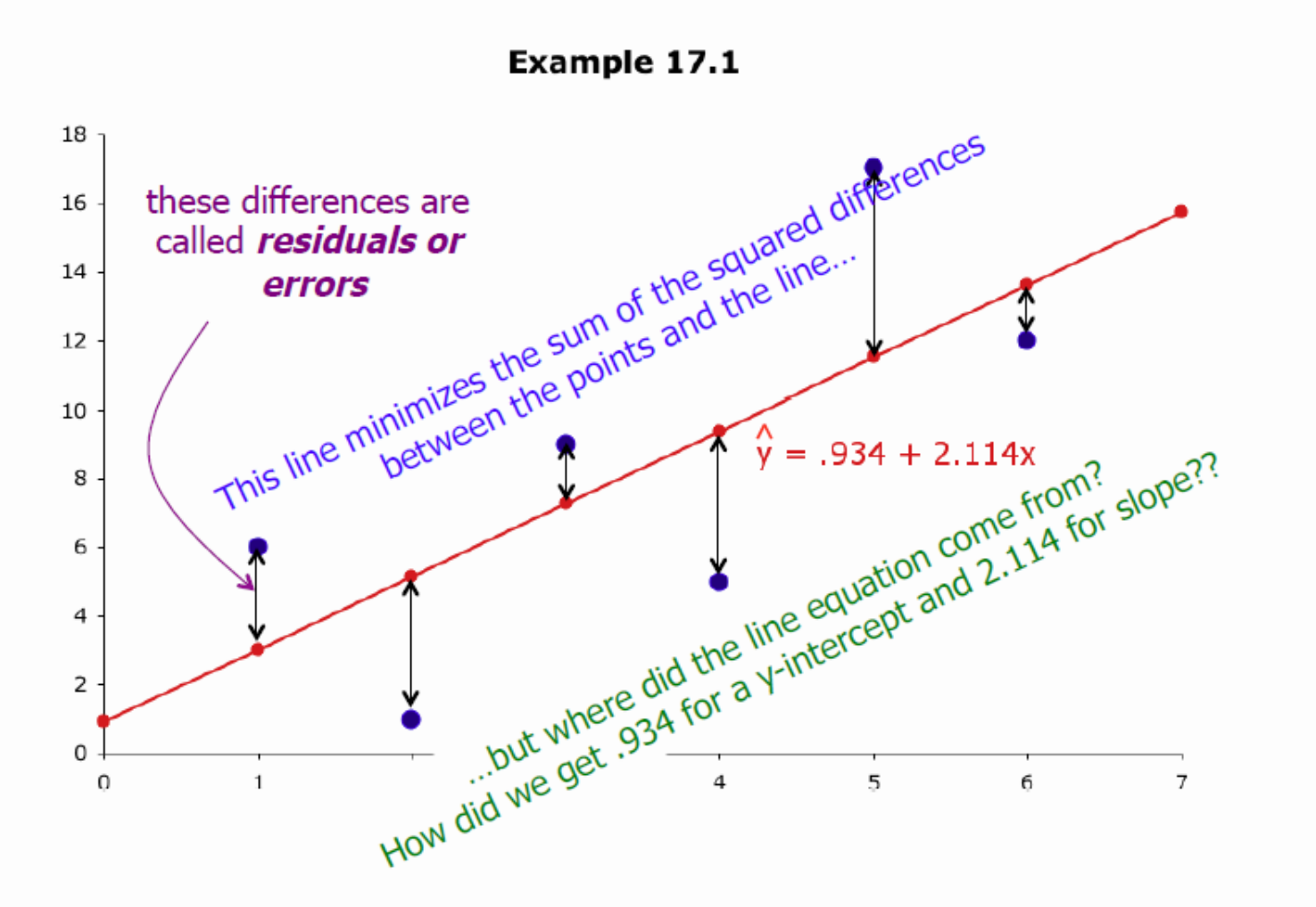
- Let's start off with the simplest version when our data has only one feature

$$\hat{y} = g(x) = \theta_0 + \theta_1 x_1$$

- Recall we want to find our model, in this case $\theta = [\theta_0, \theta_1]^T$ such that $g(X_i) \approx Y_i \; \forall \; (X_i, Y_i)$
- To solve this problem we need to choose some error function to minimize (or some likelihood to maximize).
- One of the most common is called the *least squares*

$$J(\theta) = \sum_{i=1}^{N} \left(Y_i - \widehat{Y}_i\right)^2$$

Least Squares Line



Linear Regression Analysis

- So we want to value θ that minimizes the square of the error

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(Y_i - \widehat{Y}_i\right)^2$$

$$= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \left(Y_i - g(X_i)\right)^2$$

Least Square Estimate

- For a generally observation with D features we can write
$$g(x) = \theta_0 + \theta_1 x_1 + \cdots \theta_D x_D$$
- If we add an extra feature with a value of one to the beginning of all data instances such that $x = [1\ x]$, then we can write this equation as:
$$g(x) = x\theta$$
- We call this additional feature (or more specifically, parameter θ_0), the *bias*
- So now we want to minimize
$$J(\theta) = \sum_{i=1}^{N} (Y_i - X_i\theta)^2$$

Least Square Estimate

- So now we want to minimize:

$$J(\theta) = \sum_{i=1}^{N} (Y_i - X_i\theta)^2$$

- Which we can write in matrix form as

$$J(\theta) = (Y - X\theta)^T (Y - X\theta)$$

- How can we find the minimum of this?
  - Take the derivative with respect to θ, set it equal to zero, and solve for θ!

$$\frac{\partial}{\partial\theta} J(\theta) = 0$$

Least Square Estimate

$$\frac{\partial}{\partial\theta}(Y - X\theta)^T(Y - X\theta) = 0$$

- Eventually we should arrive at

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

Least Square Estimate

- $J(w) = (y - xw)^T(y - xw)$
- $J(w) = (y^T - (xw)^T)(y - xw)$ //distribution of transpose
- $J(w) = (y^T - w^T x^T)(y - xw)$ //distribution of transpose
- $J(w) = y^T y - y^T xw - w^T x^T y + w^T x^T xw$ //distribution
- $\frac{dJ}{dw} = -(y^T x)^T - (x^T y) + 2x^T xw = 0$ //derivative
- $\Rightarrow -x^T y - x^T y + 2x^T xw = 0$ //simplification
- $\Rightarrow -2x^T y + 2x^T xw = 0$ //simplification
- $\Rightarrow x^T xw = x^T y$ //algebra
- $w = (x^T x)^{-1} x^T y$ //algebra

Let's Estimate This Data!

- Let's use all but the (4,5) sample to build our model
  - We'll try to predict that
- Steps
  1. First let's standardize our data
     - Find the mean μ and standard deviation σ of the **training** data
  2. Add an addition feature with value 1 to the data
  3. Compute the weights $\theta = (X^T X)^{-1} X^T Y$
  4. Standardize the testing data using the μ and σ from the training data
  5. Add an addition feature with value 1 to the data
  6. Compute predicted values as $x\theta$
  7. Compute the average error
     - Since our approach was based on the squared error we should take the mean of the squared error, called the *mean squared error (MSE)*
     - Then we can always take the square root of that, called the *root mean squared error* (RMSE)

Data Points:

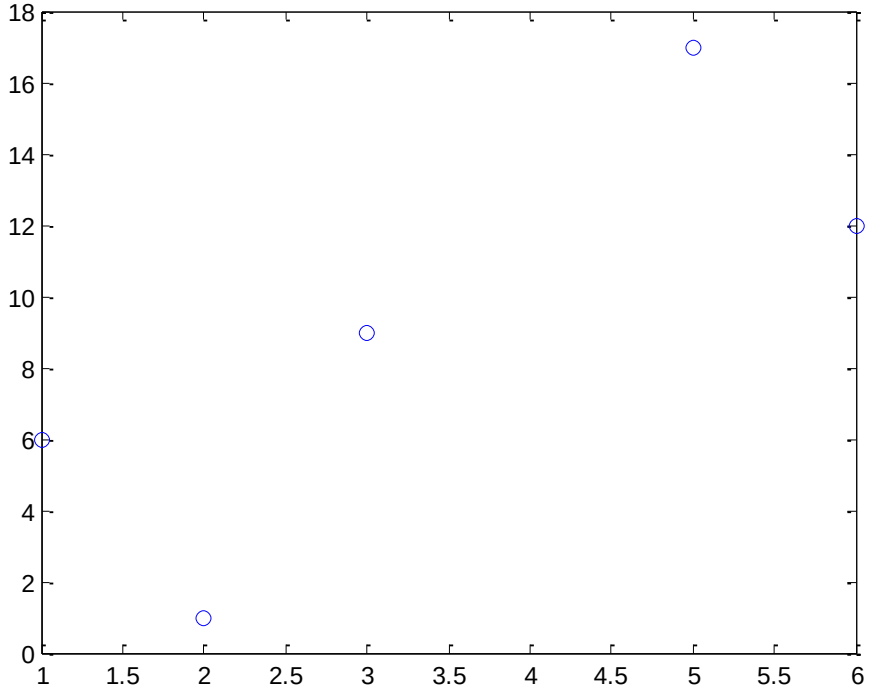
x	y
1	6
2	1
3	9
4	5
5	17
6	12

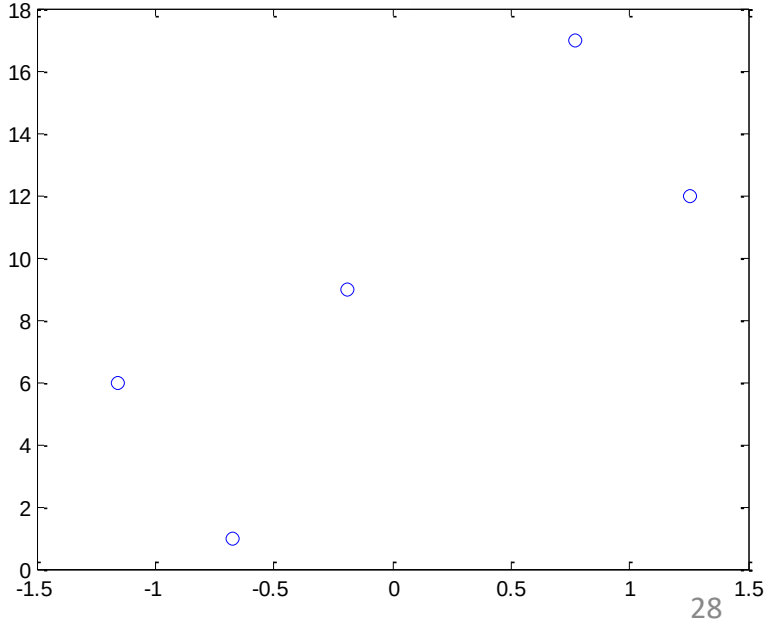
Let's Estimate This Data!

- $\mu = 3.4$
- $\sigma = 2.0736$

Data Points:

x	y
1	6
2	1
3	9
4	5
5	17
6	12





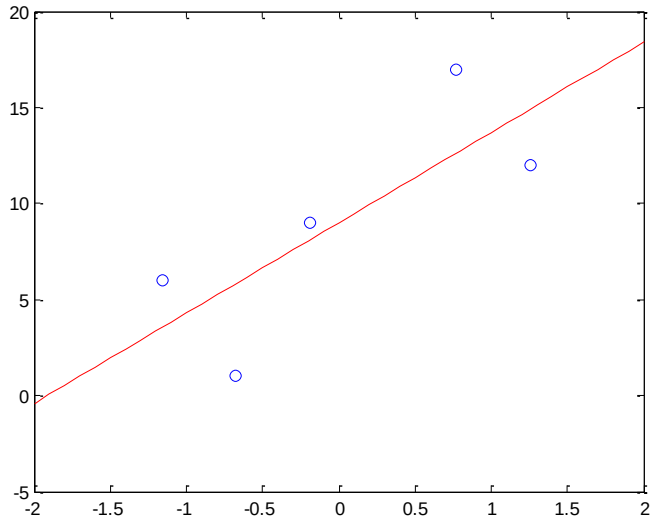
Let's Estimate This Data!

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

- $\theta = [9, 4.7]^T$
- So our model is $\hat{y} = 9 + 4.7x_1$

Data Points:

x	y
1	6
2	1
3	9
4	5
5	17
6	12

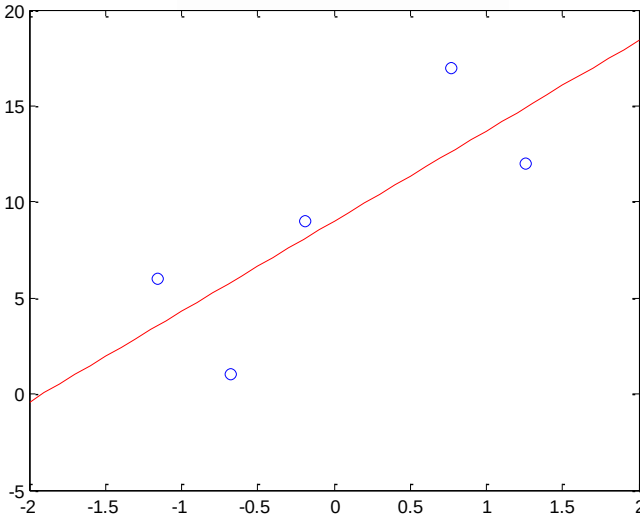


Let's Estimate This Data!

Data Points:

x	y
1	6
2	1
3	9
4	5
5	17
6	12

- So let's predict the value for $x = 4$
- Standardizing it
  - $x \rightarrow (4 - 3.4)/2.0736 = 0.2893$
- $\hat{y} = 9 + 4 * 0.2893 \ = \ 10.1572$
- Error
  - $MSE = (5 - 10.1572)2 \ = \ 26.5967$
  - RMSE=$\sqrt{26.5967} = 5.1572$

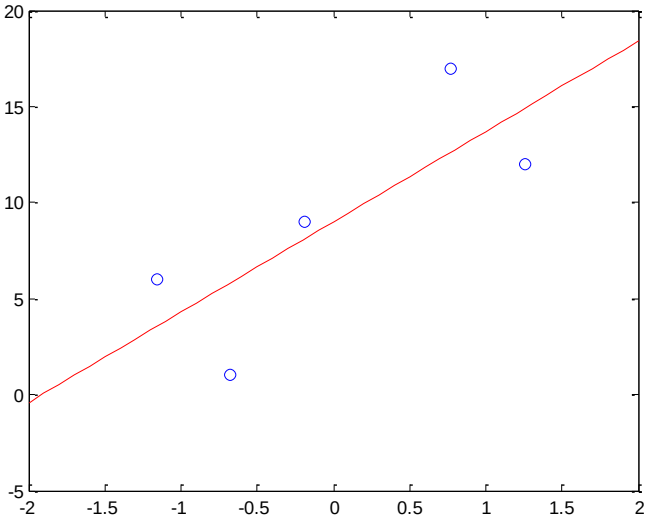


Improvements?

- What's wrong with this?
  - Lots of noise
- What could make it better?
  - More data
  - More complex model
    - Therefore maybe more features?
    - Non-linear regression?
- Was one test enough?
  - Make do leave-one-out cross validation

Data Points:

x	y
1	6
2	1
3	9
4	5
5	17
6	12



Example 2

- Model:

$$Final = \theta_0 + \theta_1 Exam1 + \theta_2 Exam2 + \theta_3 Exam3$$

Note: Final exam out of 200

- Testing Set: The first 8 samples
- Training Set: The rest (next 17 samples)

EXAM1	EXAM2	EXAM3	FINAL
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177
78	83	77	159
82	86	90	177
86	82	89	175
78	83	85	175
76	83	71	149
96	93	95	192

Example 2

- Training
  - Going through the same process…

$$\theta = [166.5294, 3.1218, 4.9834, 10.9629]^T$$

  - $FinalExam = 166.5294 + 3.1218 * Exam1 + 4.9834 * Exam2 + 10.9629 * Exam3$
- Testing
  - Standardize each point using info from the training data
  - Project each point
  - Compute the average error
    - MSE: 7.9566
    - RMSE: $\sqrt{7.9566} = 2.8207$

Ytest =

152
185
180
196
142
101
149
115

Y2 =

152.4652
186.0558
182.6552
201.1921
138.5914
101.7790
149.9209
111.0962

Gradient Ascent/Descent

Gradient Ascent/Descent

- The solution to linear regression that we just provided, $\theta = (X^T X)^{-1} X^T Y$, is called the *closed-form* solution to the problem.
  - We are able to use mathematics to come up with a direct solution to the minima/maxima problem.
- However, for some problems a closed-form solution/equation may not exist and/or if our matrix is large it may become infeasible to compute inverse
  - Or inverse may not exist in some cases

Gradient Ascent/Descent

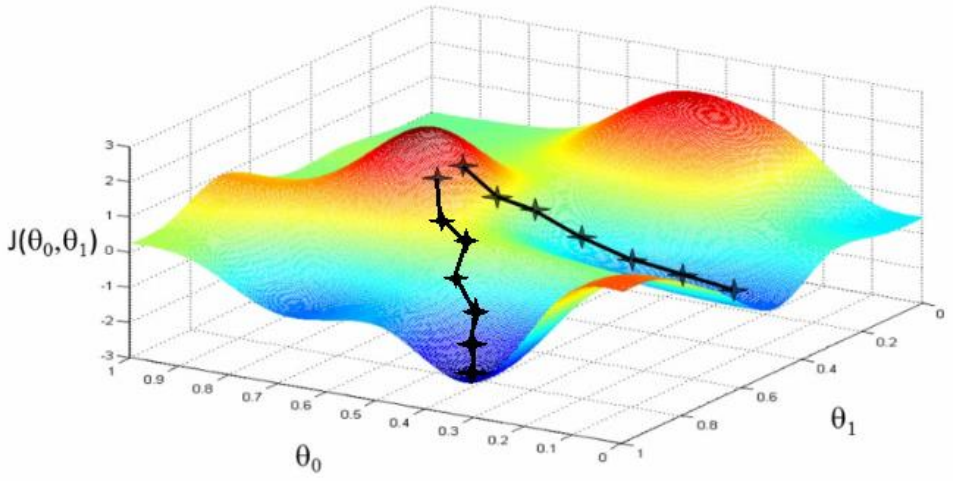
- Another approach to solving a minima/maxima problem is to compute the *gradient* of the equation with regards to each parameter in order to move our current parameter's value in that direction
  - And iteratively do this until we converge to a minima/maxima
- This approach is called ***gradient descent*** and generalizes nicely to lots of applications where we need to find the values of parameters to minimize or maximize some function

Gradient Descent

- The basic idea is that we want to find the values $\theta_0, \ldots, \theta_k$ to minimize some function $J(\theta_0, \ldots, \theta_k)$
- Approach
  1. Start with some values for the parameters
  2. Update them incrementally in the direction of the *gradient* of J
  3. Hopefully we converge on a minima

Gradient Descent/Assent

- The gradient of J can be thought of the way (vector) to go towards the maxima
- If we want to minimize J (if it's an error function), then we want to go "downhill"
  - Opposite direction of the gradient



Gradient Descent

- Typically we use gradient descent if we can't have a closed-form solution to solve all the parameters at once.
- So instead we'll move each parameter, one at a time, in the direction of it's gradient:
  - Initialize parameter θ_i to some value
  - For each data instance (training sample) we update

$$\theta_i = \theta_i - \eta \frac{\partial J}{\partial \theta_i}$$

  - Where η is the *learning rate*
- So obviously we need to know $J(\theta_i)$ and determine $\frac{\partial J}{\partial \theta_i}$

Least Squares Gradient Descent

- Let's return to our least squares example
  - Here $\mathrm{J}(\theta) = (Y - X\theta)^2 = \sum_{i=1}^{N}(Y_i - X_i\theta)^2$
- For a single sample(x, y) this is just:

$$J(\theta) = (y - x\theta)^2$$

- Gradient descent looks to find the gradient (change, derivative) one parameter at a time. Therefore if we're looking for the gradient with respect to parameter θ_j this is:

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}(y - x\theta)^2$$

Least Squares Gradient Descent

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial}{\partial \theta_j}(y - x\theta)^2$$

- Expending this…

$$= \frac{\partial}{\partial \theta_j}(y^2 - 2yx\theta + (x\theta)^2)$$

- What is $\frac{\partial}{\partial \theta_j}(yx\theta)$?
  - Recall that $x\theta = \sum_{i=1}^{D} x_i\theta_i$

$$yx_j$$

- How about $(x\theta)^2$

$$2x\theta x_j$$

Least Squares Gradient Descent

$$\frac{\partial}{\partial \theta_j}(y^2 - 2yx\theta + (x\theta)^2)$$

- So

$$\frac{\partial J}{\partial \theta_j} = -2yx_j + 2(x\theta)x_j$$

$$\frac{\partial J}{\partial \theta_j} = 2(x\theta - y)x_j$$

- Let's drop the scalar so it's:

$$\frac{\partial J}{\partial \theta_j} = 2(x\theta - y)x_j$$

Least Square Gradient Descent

- Therefore our update rule is:

$$\theta_j = \theta_j - \eta(x\theta - y)x_j$$

- You will obviously likely have more than 1 training sample so you can iterate over each sample and then move on to the next parameter etc..

Gradient Descent

- So the overall algorithm is:
  1. Start with some values for the parameters (random?)
  2. Until some termination criteria is met (number of iterations? Lack of change in the parameters?)
     a. For each parameter
        i. For each sample
           a. Update the current parameter using this current sample and the update equation.

Variants of Gradient Descent

- This version of gradient descent (often called *iterative* gradient descent) is susceptible to cycles
  - Since we're going over the same data in the same order over and over
- A variant is called *stochastic gradient descent*
  - Here we either randomize our data each time or select a random sample each time.
- However doing one sample at a time can take a while to converge. Another approach is *batch gradient descent* where we first compute the cumulative error then update:

$$\theta_j = \theta_j - \frac{\eta}{N}\sum_{i=1}^{N}\left((X_i\theta - Y_i)X_{ij}\right)$$

Gradient Descent

- The problem is that to do batch processing we need all the data in memory at once (bad for large datasets)
- In practice quite often a “hybrid” approach is used (mini-batch)

More on the Learning Rate

- The learning rate η can effect gradient descent in several ways:
  - Too small?
    - Takes a long time to converge
  - Too large?
    - May jump back and forth over a/the minima
- Also if you don't standardize your data and you use the same η for all features then you may have difficulty getting to the globally optimal result

Gradient Descent

- Termination Criteria:
  - Max number of iterations reached
  - Parameters change very little
  - Change in the training error is very little.

Example 2

- Model:

$$Final = w_0 + w_1 Exam1 + w_2 Exam2 + w_3 Exam3$$

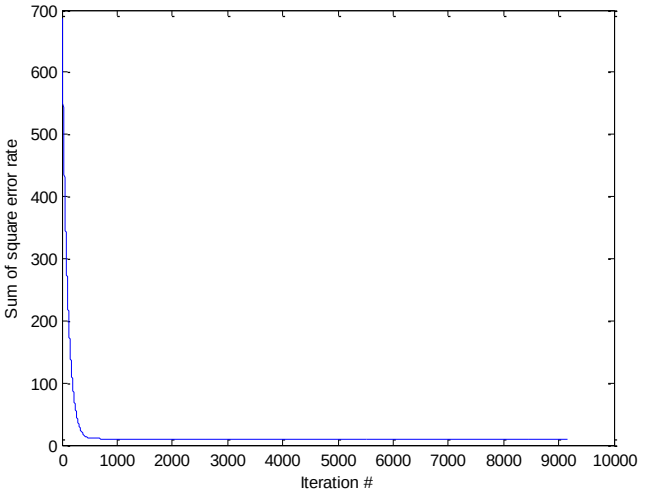
Note: Final exam out of 200

- Testing Set: The first 8 samples
- Training Set: The rest (next 17 samples)
- Let's do this with gradient descent
- Settings:
  - Standardize data using training data
  - Seed the rng to zero
  - Initialize each parameter to some random number
  - Use batch gradient descent
  - $\eta = 0.01$
  - Terminate when change in training error < eps

EXAM1	EXAM2	EXAM3	FINAL
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177
78	83	77	159
82	86	90	177
86	82	89	175
78	83	85	175
76	83	71	149
96	93	95	192

Approaches Compared

- Closed Form Linear Regression
  - Model: $\theta = [166.5294, 3.1218, 4.9834, 10.9629]^T$
  - RMSE: 2.8207
- Gradient Descent
  - Model: $\theta = [166.5294, 3.1218, 4.9834, 10.9629]^T$
  - RMSE: 2.8207
  - Number of iterations: 9167



Dealing with Overfitting

- We could add a regularization term. Perhaps:

$$J(\theta) = (Y - X\theta)^2 + \lambda\theta^T\theta$$

  - Therefore the larger θ is, the more it costs us
  - λ is a blending term to say how much this cost should contribute to the overall cost.
  - Then we can re-compute the closed form and/or the gradient using this equation.
  - Of course we'd have to somehow decide on λ! ☹
- If we're doing gradient descent, then we could try to halt the iterative training process by using a third data set, called a *validation set*
  - On each iteration we'll compute the validation error.
  - We'll stop when the validation error increases.