CS 613 Machine Learning

Most Machine Learning algorithms require some idea of similarity and/or distance between observations.

Which function you choose to use often depends on the nature of the data, the equation you're using it in, etc...

As a reference here's some of the most common similarity/distance functions:

Cosine Similarity

Aims to measure similarity based on the angle between two vectors relative to some origin. The function uses the dot product definition of cosine. Due to the nature of cosine the similarity value will be in the range of [-1, 1].

$$similarity(A, B) = \cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{D} A_i B_i}{\sqrt{\sum_{i=1}^{D} A_i^2} \sqrt{\sum_{i=1}^{D} B_i^2}}$$

Gaussian Kernel

The Gaussian Kernel, sometimes known as the *radial basis function kernel* measures the similarity between two objects. The numerator of the exponent is the squared Euclidean distance so when this distance is zero, the similarity is one. Likewise when the squared Euclidean distance is infinite, the similarity is zero.

We can control the fall-off rate by specifying the σ value; a larger sigma will result is slower fall-off.

$$similarity(A, B) = e^{-\frac{\sum_{i=1}^{D}(A_i - B_i)^2}{2\sigma^2}} = e^{-\frac{||A-B||^2}{2\sigma^2}}$$

Histogram Intersection

As the name implied, histogram intersection is often used to compute the intersection of two histograms, resulting in a similarity measurement:

$$similarity(A, B) = \sum_{i=1}^{D} \min(A_i, B_,)$$

Euclidean Distance

The Euclidean Distance, sometimes referred to as the L2-Norm, is the straight line distance between two points.

$$distance(A, B) = \sqrt{\sum_{i=1}^{D} (A_i - B_i)^2}$$

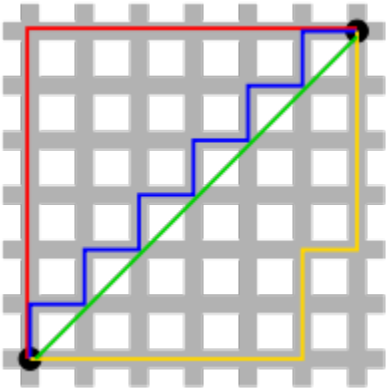
This is typically a good choice if doing things like rotation in the space has some meaning.

Manhattan Distance

The Manhattan Distance, also referred to as the City Block Distance, Taxicab Distance, or L1 Norm, measures the distance between two points as the sum of their difference along each axis.

$$distance(A, B) = \sum_{i=1}^{D} |A_i - B_i|$$

This is typically a good choice if doing things like rotation in the space does **not** make sense.



CS 613 – Machine Learning

Introduction to Machine Learning

Some slides adapted from Matt Burlick, Drexel

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

Types of Machine Learning

- Supervised Learning
  - Learning by example
- Unsupervised Learning
  - Learning by exposure to the raw data distribution
- Reinforcement Learning
  - Learning from right/wrong feedback
- Transfer Learning
  - Learning one task helping you to learn a related task

least important future

Percentage Use

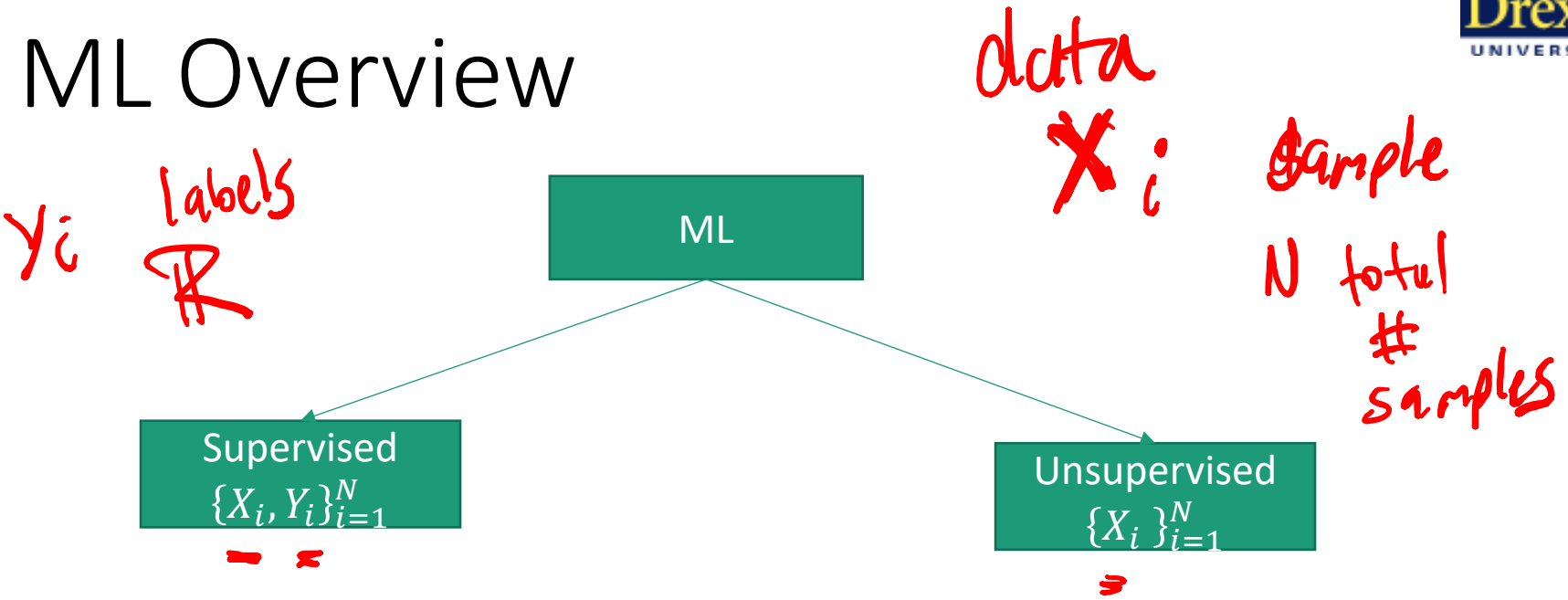
96%

3%

1%

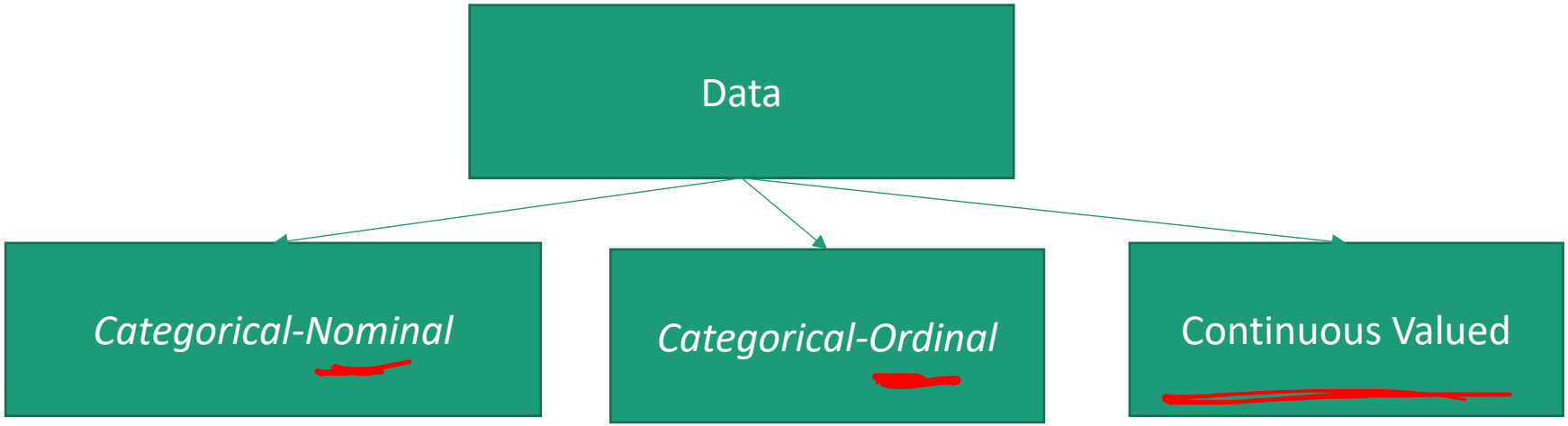
2%

ML Overview



- We can basically break machine learning tasks into two categories
  1. Supervised Learning
  2. Unsupervised Learning
- *Supervised learning*
  - Data X_i and correct answer (label) Y_i given for each example $i \in \{1, \dots, N\}$
- *Unsupervised learning*
  - Only data given for each example, X_i

Types of Data



- Each piece of information pertaining to an observation can fall into one of three categories:
  - *Categorical-Nominal (unordered)*
    - Examples: Car Model, School
  - *Categorical-Ordinal (can be ordered)*
    - Examples: Colors, small < medium < large
  - *Continuous Valued*
    - Examples: Blood Pressure, Height

No Free Lunch Theorem

- The "free lunch" refers to the once-common tradition of saloons in the United States providing a "free" lunch to patrons who had purchased at least one drink. Many foods on offer were high in salt (e.g., ham, cheese, and salted crackers), so those who ate them ended up buying a lot of beer.

- No one method dominates all others over all possible datasets. On a particular dataset, one specific method may work best, but some other method may work better on a similar but different dataset
- To get superior flexibility on a specific problem you make assumptions
- This reduces the generability of the algorithm

No Free Lunch Theorem

- Questions we should ask ourselves are:
  - Do we have labels?
  - Do we want to do clustering?
  - Do we know the number of clusters?
  - Do we want to do regression?
  - Do we want to do classification?
  - Is understanding/interpreting the learned system important?
  - What are our memory and/or processing time limitations?
  - Do we have categorical or continuous-valued data?
  - Do I have enough data?
  - Do I have enough features? Too many?
  - Is linear ok or do I need non-linear?
  - And more…

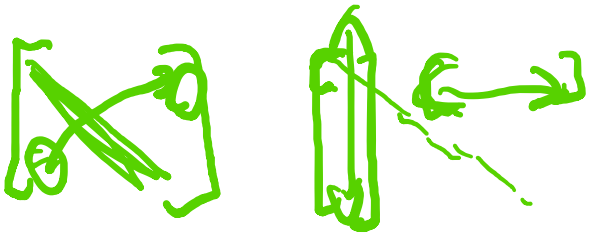
ML Algorithms

- Here's a list of algorithms we'll look at in the class

1. Linear Regression
2. Classification
    a. Probabilistic Decisions (Inference, Bayesian Learning, Decision Trees)
    b. Nearest Neighbors
    c. Support Vector Machines
    d. Logistic Regression
    e. Introduction to Artificial/Deep Networks
3. Feature Reduction (Feature Selection, Feature Projection)
4. Unsupervised Learning
    a. K-Means
    b. Mixture Models
5. Neuro-inspired Machine Learning Data
    a. Neural networks
    b. Sparse Coding
    c. Intro to Recurrent Neural Networks (RNNs)

Line – 60%

- Y = mx + b



dot product

$\theta_1 x_1$ $\theta_0 \cdot 1$

slope intercept

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$\theta_1 x_1 + \theta_2 x_2 \ldots \theta_N x_N$

$ax + by = c$

$by = -ax + c$

$y = -\frac{a}{b}x + \frac{c}{b}$

neural net curves

$2D \to 3D \to 4D \to 5D$ hyperplane

$ax + by + cz = d$

$\theta_1 x_1 + \theta_2 x_2 + \ldots \theta_N x_N$

$\theta \cdot X$

$\theta^T X$

50 [ ] 50 [ ]

50 x 1 · 50 x 1

1 x 50 50 x 1

1 x 1 scalar

$\theta^T x + b$ perceptron

$\theta^T x$ ← linear regression

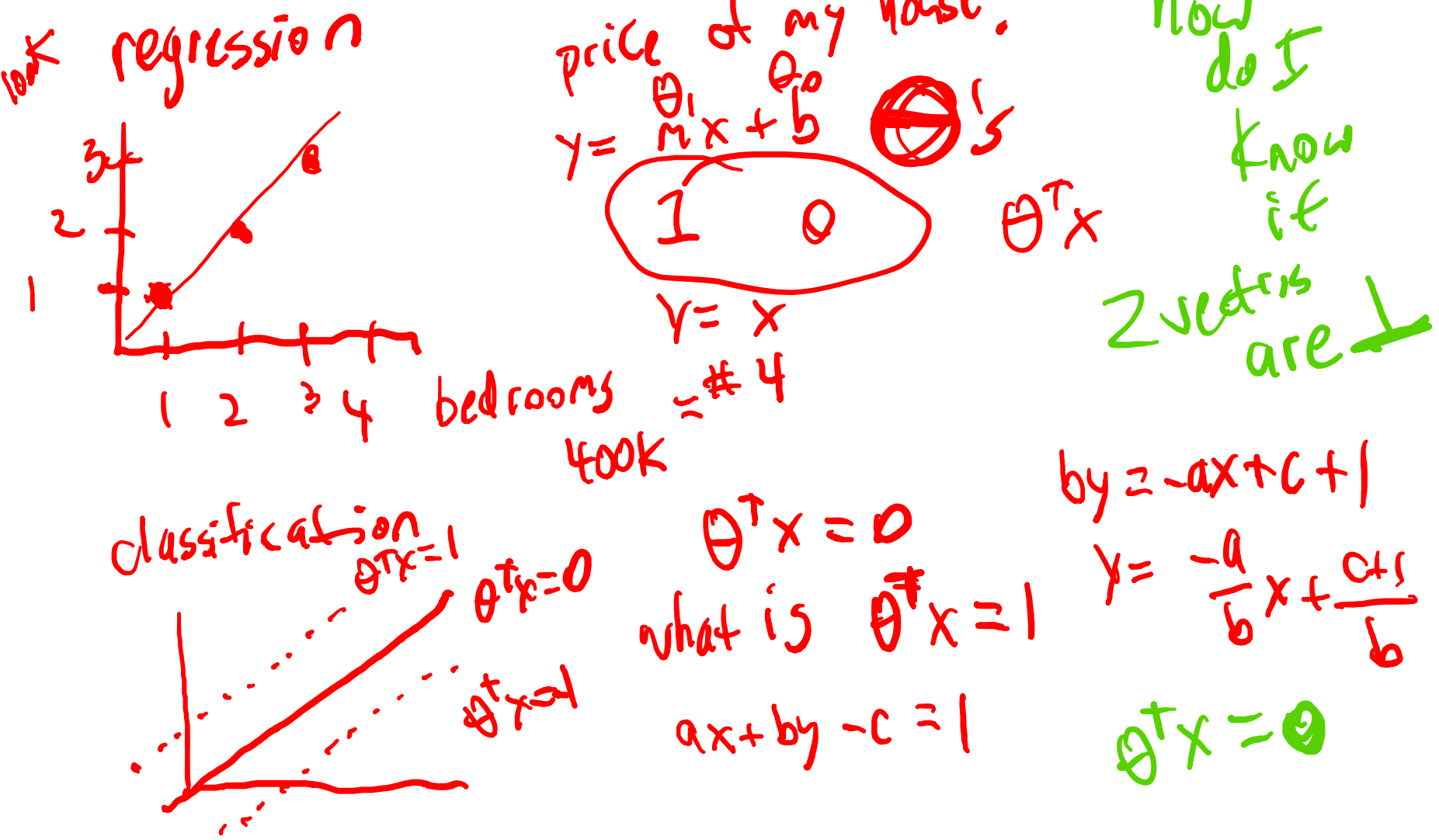
$ax + by - c = 0$

$\theta^T x = 0$ logistic regression

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_N \end{bmatrix} \quad x = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{bmatrix}$$

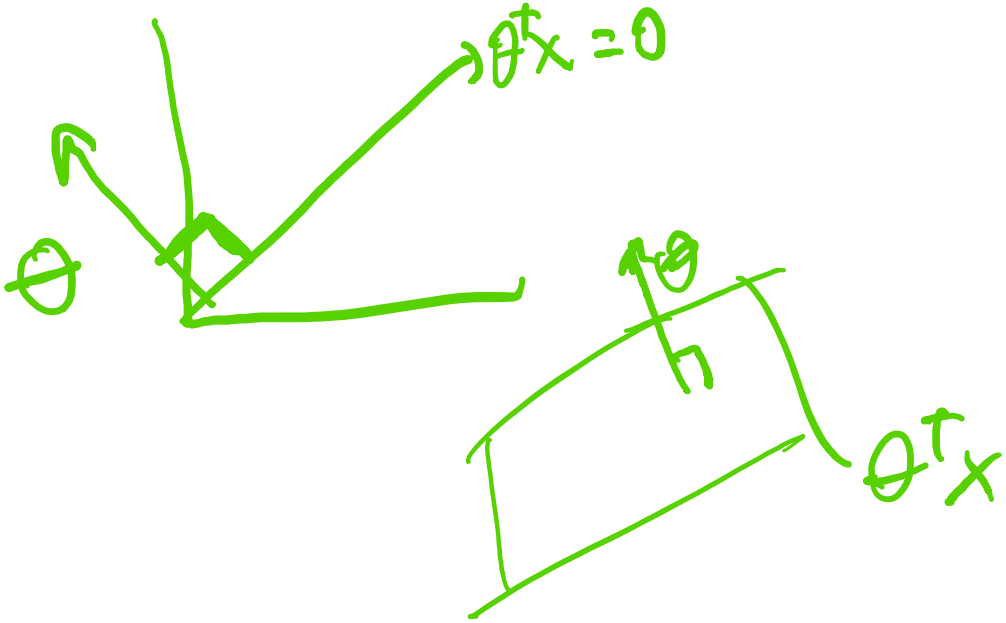
sum

How to use lines



Dot product = 0, when…

- 2 vectors are perpendicular



$b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

θ

$a = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$0 \cdot 1 + 1 \cdot 0 = 0$

$a^T b = \|a\| \|b\| \cos\theta$

$\cos(90)$

θ

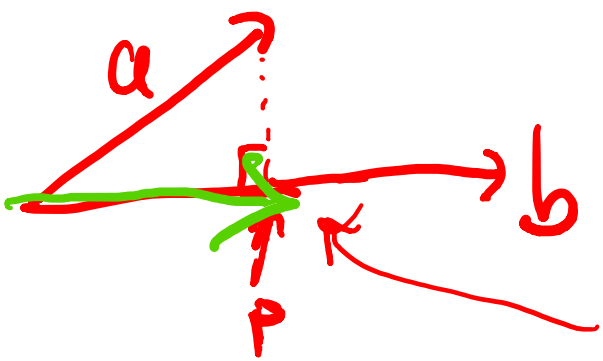
$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$

$\|a\| = \sqrt{a_1^2 + a_2^2 + a_3^2 \dots a_N^2}$

Euclidean distance

L_2 norm

Projections → shadows of one vector onto another



$a \cdot b = \|a\| \|b\| \cos\theta$

$\frac{a \cdot b}{\|b\|} = \|a\| \cos\theta$

scalar projection

vector projection

unit vector

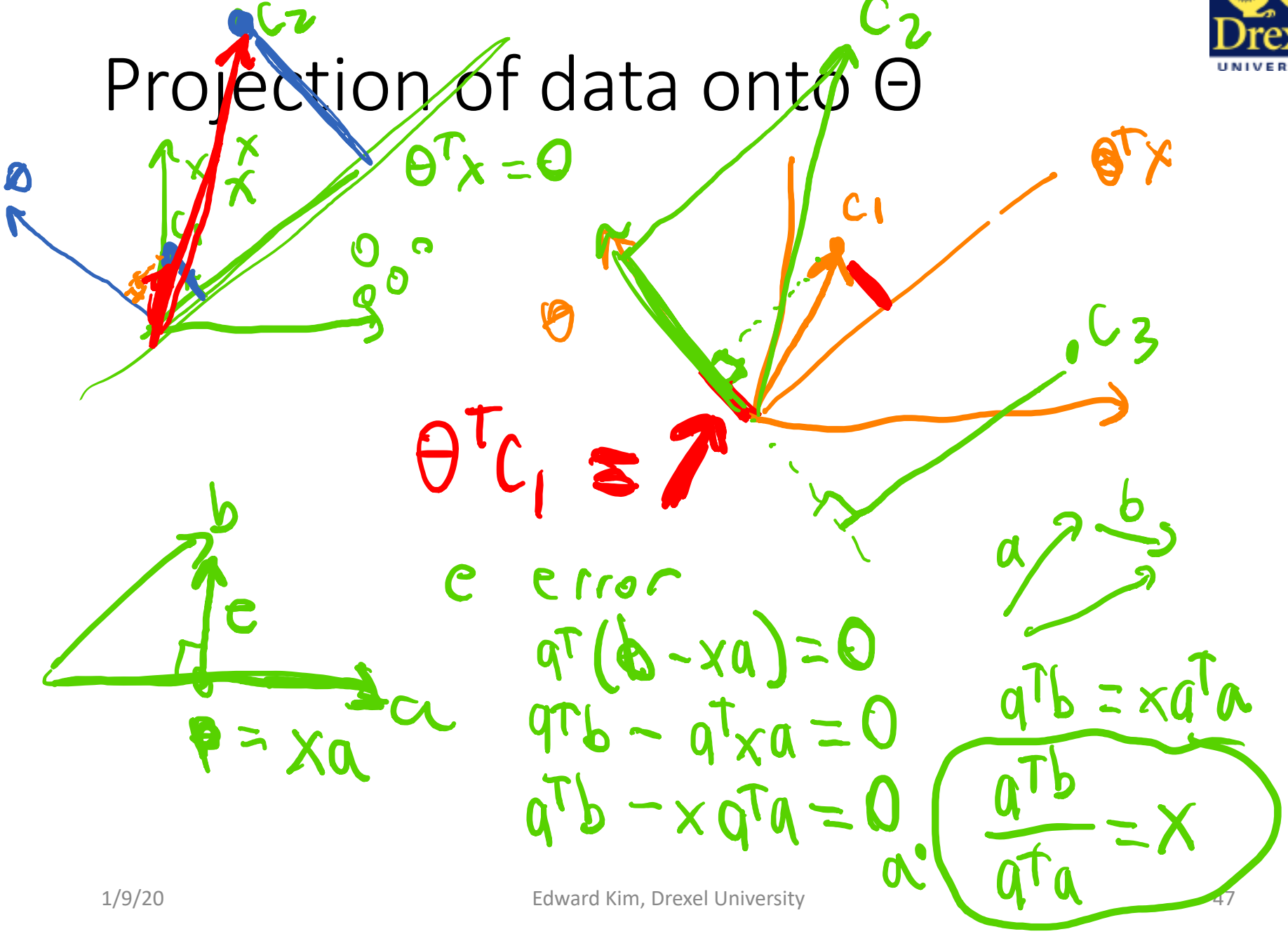
what if b is already unit vector

$\theta^T x$ ~~θx~~ $a \cdot b \Leftrightarrow b \cdot a$

$\frac{a \cdot b}{\|b\|} \cdot \frac{b}{\|b\|} \Rightarrow \frac{\|b\|^2}{b^T b}$ $\|b\|^2 = b_1^2 + \dots b_n^2$

$\frac{a^T b}{b^T b} \cdot b$

Projection of data onto Θ



Linear Regression

Linear Regression Example

line of "best" fit

Linear Regression Example

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

$X =$ $Y =$

$Y = mx + b$

$Y = \theta_0 + \theta_1 x_1$

θ's

x's are the input variables / "features"

y's are the output or target

augment $\theta^T x + b$ by $\theta^T x$

Linear Regression Analysis

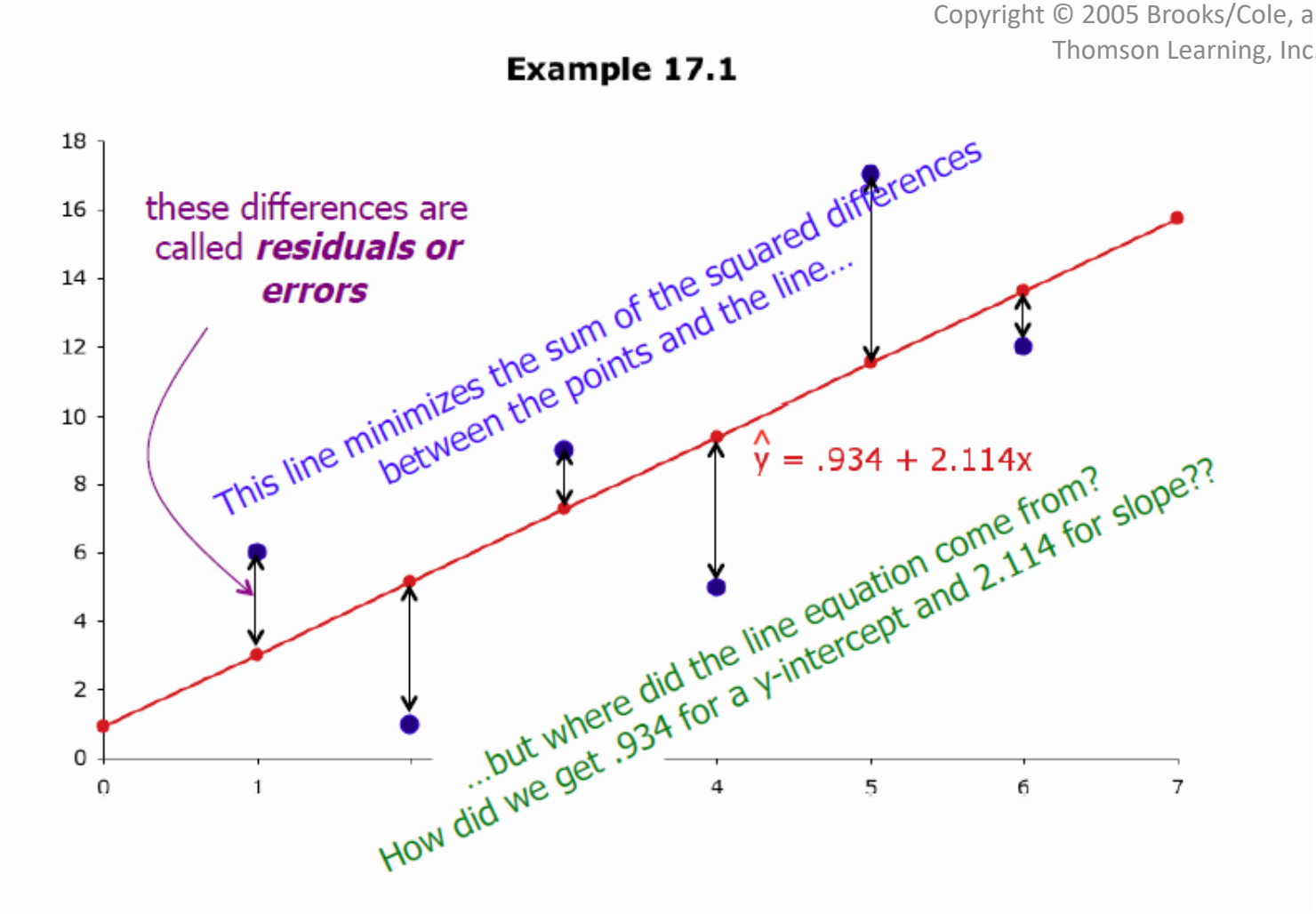
- Let's start off with the simplest version when our data has only one feature

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

- 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*

$$J = \sum_{i=1}^{N} (Y_i - \hat{Y}_i)^2$$

Least Squares Line



Linear Regression Analysis

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

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

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

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* (just to add more confusion!)
- So now we want to minimize (over all observations $(X_i, Y_i) \in (X, Y)$
$$J = \sum_{i=1}^{N} (Y_i - X_i\theta)^2$$

Least Square Estimate

- So now we want to minimize:

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

- Which we can write in matrix form as

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

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

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

Least Square Estimate

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

- Eventually we should arrive at

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

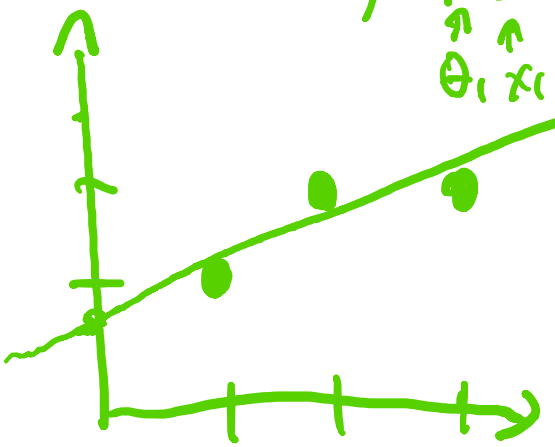
Least Square Estimate

- $J = (Y - X\theta)^T (Y - X\theta)$
- $J = (Y^T - (X\theta)^T)(Y - X\theta)$ //distribution of transpose
- $J = (Y^T - \theta^T X^T)(Y - X\theta)$ //distribution of transpose
- $J = Y^T Y - Y^T X\theta - \theta^T X^T Y + \theta^T X^T X\theta$//distribution
- $\frac{dJ}{d\theta} = -(Y^T X)^T - (X^T Y) + 2X^T X\theta = 0$ //derivative
- $\Rightarrow -X^T Y - X^T Y + 2X^T X\theta = 0$ //simplification
- $\Rightarrow -2X^T Y + 2X^T X\theta = 0$ //simplification
- $\Rightarrow X^T X\theta = X^T Y$ //algebra
- $\theta = (X^T X)^{-1} X^T Y$ //algebra

$\theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

Example 1

(1,1), (2,2), (3,2)



$\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \begin{bmatrix} x_i \end{bmatrix}$

$X^T X \theta = X^T y$

$y = \frac{1}{2}x + \frac{2}{3}$

$X = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \quad y = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$

$(X^T X)^{-1} X^T y$

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$

$\begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$

$3\theta_0 + 6\theta_1 = 5$

$6\theta_0 + 14\theta_1 = 11$

$2\theta_1 = 1$

$\theta_1 = \frac{1}{2}$

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}$

$\theta_0 = \frac{2}{3}$

Example 2

- Model:

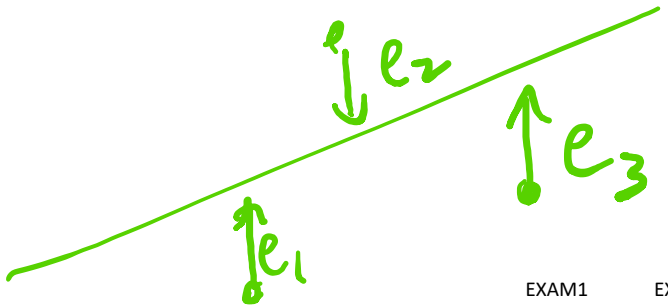
$$Final = \theta_0 + \theta_1 Exam1 + \theta_2 Exam2 + \theta_3 Exam3$$

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



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

- Steps
  1. Add an addition feature with value 1 to the training data
  2. Compute the weights $\theta = (X^T X)^{-1} X^T Y$
  3. Add an addition feature with value 1 to the testing data
  4. Compute predicted values as $x\theta$
  5. 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)

$$\sqrt{\frac{e_1^2 + e_2^2 + e_3^2}{\#\text{ of test samples}}}$$

Example

- Training

$$\theta = \begin{bmatrix} -9.9596 \\ 0.3821 \\ 0.5660 \\ 1.1900 \end{bmatrix}$$

  - $FinalExam = -9.9596 + 0.3821 * Exam1 + 0.5660 * Exam2 + 1.1900 * Exam3$
- Testing
  - Project each point
  - Compute the average error
    - MSE: 7.9566
    - RMSE: $\sqrt{7.9566} = 2.8207$

Ytest =	$\widehat{Ytest}$ =
152	152.4652
185	186.0558
180	182.6552
196	201.1921
142	138.5914
101	101.7790
149	149.9209
115	111.0962

CS 613 – Machine Learning

Introduction to Machine Learning

Some slides adapted from Matt Burlick, Drexel

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

$\theta^T x$

$(X^T X)^{-1} X^T y$ → vector projection

linear regression

$\frac{a^T b}{q^T a}$

Preparing Data

$Ax = b$

A^{-1} doesn't exist

A is not square

↑

$(A^T A)^{-1} (A^T A)x = A^T b$

pseudo inverse

square invertible

$(A^T A)^{-1} A^T$

Parsing Text Files

np.genfromtext
pandas

- The first thing we'll need to do in this class is get data!
- We'll get data from text files
- But unfortunately they may be in all different formats ☹
  - If we're lucky it may be in .csv format and we can use a library to read this (like python's `csv` library)
  - But even then there may be mixed data types making it impossible.
- So one of the first things you'll want to do is write a file parser.

Feature Types

isCat $\in [0,1]$ 0.8
isDog $\in [0,1]$ 0.7
isMouse $\in [0,1]$ 0.3
| Binary

- Let's assume that we now have our data in our system.
- The next thing to consider is the nature of our data.
- As previously mentioned, we categorize each feature as one of three types:
  1. Continuous valued → floating
  2. Categorical-Nominal → "name" → {cat, dog, mouse} → 0 1 2
  3. Categorical-Ordinal → "ordered" → {small, medium, large} → 0 1 2
- Depending on the algorithm being used we may need to
  - Convert categorical features into a set of binary features
  - "Standardize" features so that they have the same range

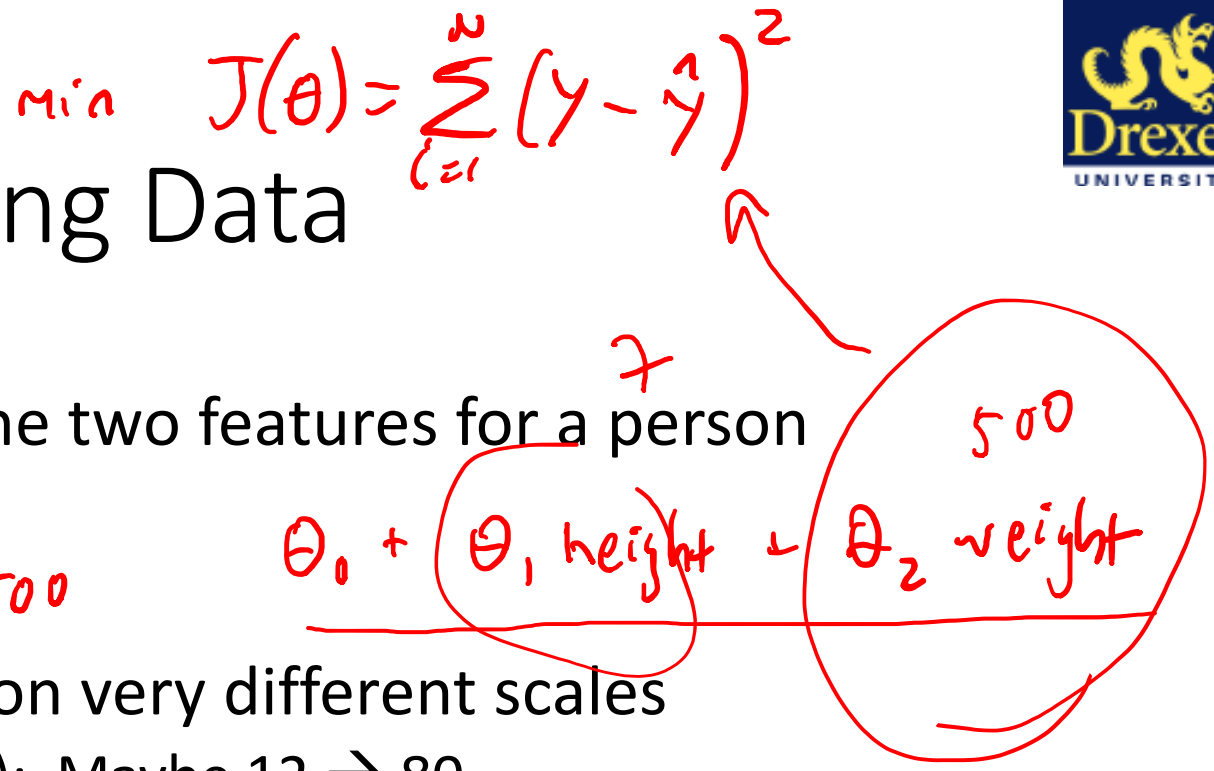
1.5 ?

1.5

Prepping Data

- Categorical → Binary
  - If our algorithm computes some floating value/location of a feature then it wouldn't make sense to have a floating point value of a categorical feature
    - Imagine if a feature is car make $x_1 \in \{Honda, Ford, Toyota\} \rightarrow \{0,1,2\}$
    - What would the meaning of 0.7 be?
  - Instead we'll want to create a set of binary features from our enumerations
    - $isHonda \in [0,1], isFord \in [0,1], isToyota \in [0,1]$
- Standardizing Features
  - If our algorithm incrementally updates parameters, then we may want all features to have the same variance.
  - Otherwise ones with grader range will have greater influence than ones with a smaller range

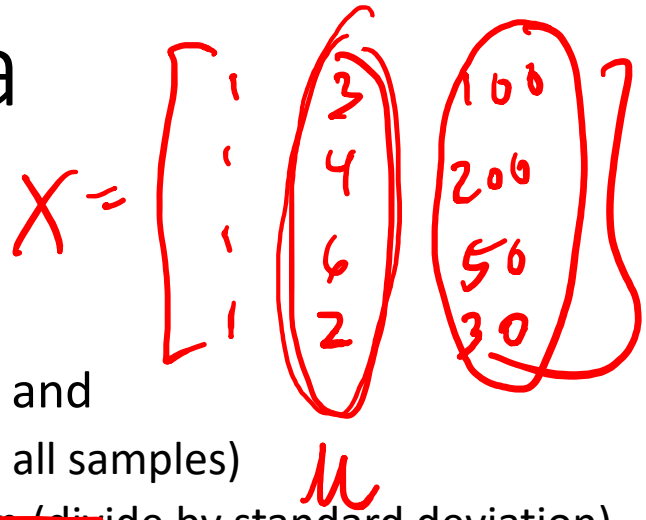
Standardizing Data



- Example: Imagine two features for a person
  - Height 0 – 7
  - Weight 0 – 500
- These are both on very different scales
  - Height (inches): Maybe 12 → 80
  - Weight (lbs): Maybe 5→400
- If we used the data as-is, then one feature may have more influence than the other
  - Depends on the algorithm

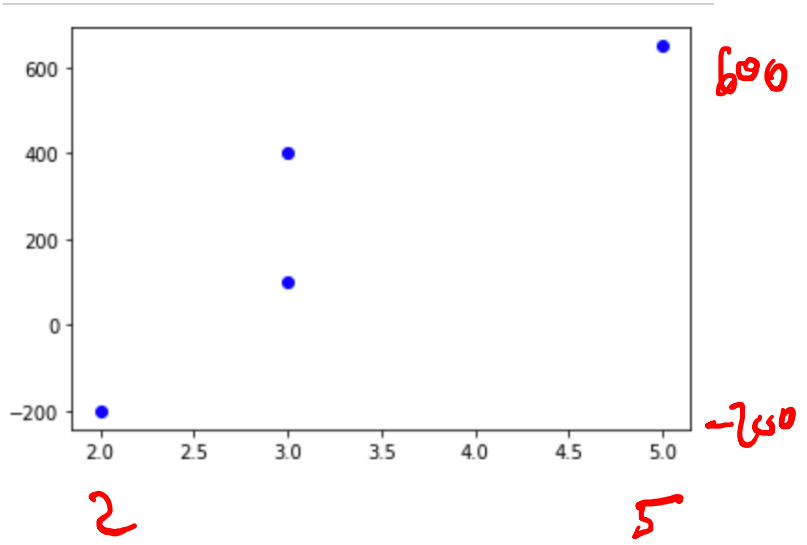
Standardizing Data

- Standardized data has
  - Zero mean
  - Unit deviation
- We treat each feature independently and
  1. Center it (subtract the mean from all samples)
  2. Make them all have the same span (divide by standard deviation).
- Example



```
import numpy as np
import matplotlib.pyplot as plt

X = [[3,400],
     [2,-200],
     [3, 100],
     [5,650]]
X = np.array(X)
plt.plot(X[:,0], X[:,1],'bo')
plt.show()
```

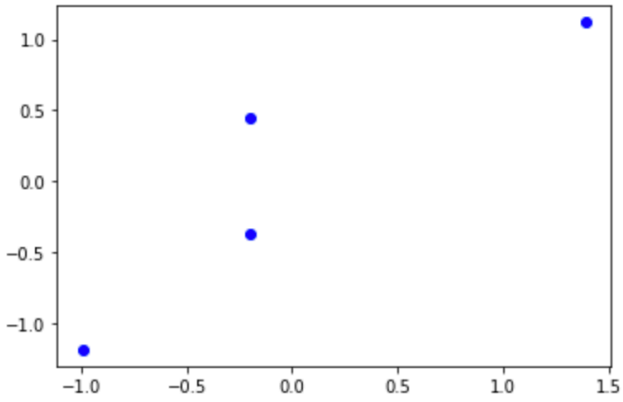


Standardizing Data

- We can compute the mean and standard deviation of each feature easily and then subtract the means from each observation and divide each (centered) observation by the standard deviation of each feature.
- Would it make sense to do this with categorical data?
- How about if it was made into a binary feature set?

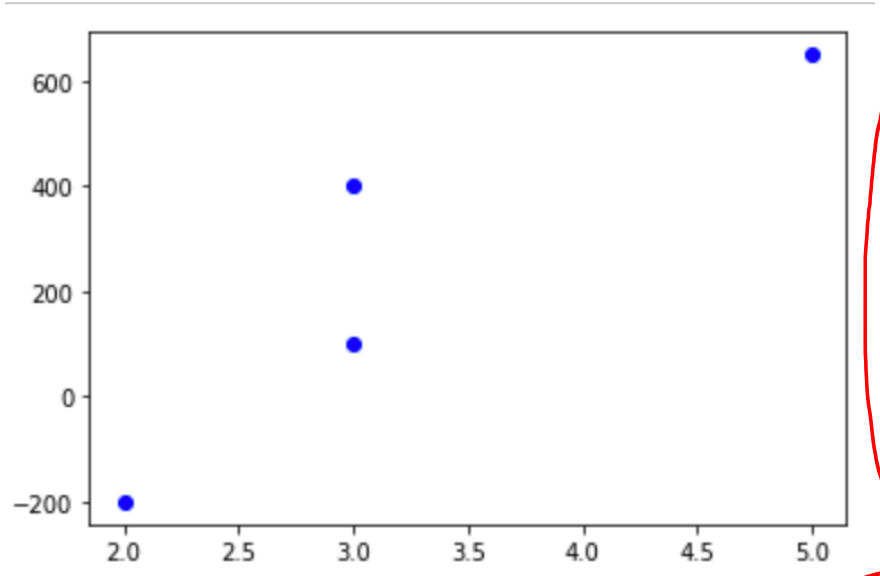
```
mean = np.mean(X, axis=0)
std = np.std(X, axis=0, ddof=1)
# print(mean, std)
# [ 3.25 237.5 ] [ 1.25830574
368.27299657]

sX = (X-mean)/std
plt.plot(sX[:,0], sX[:,1],'bo')
plt.show()
```

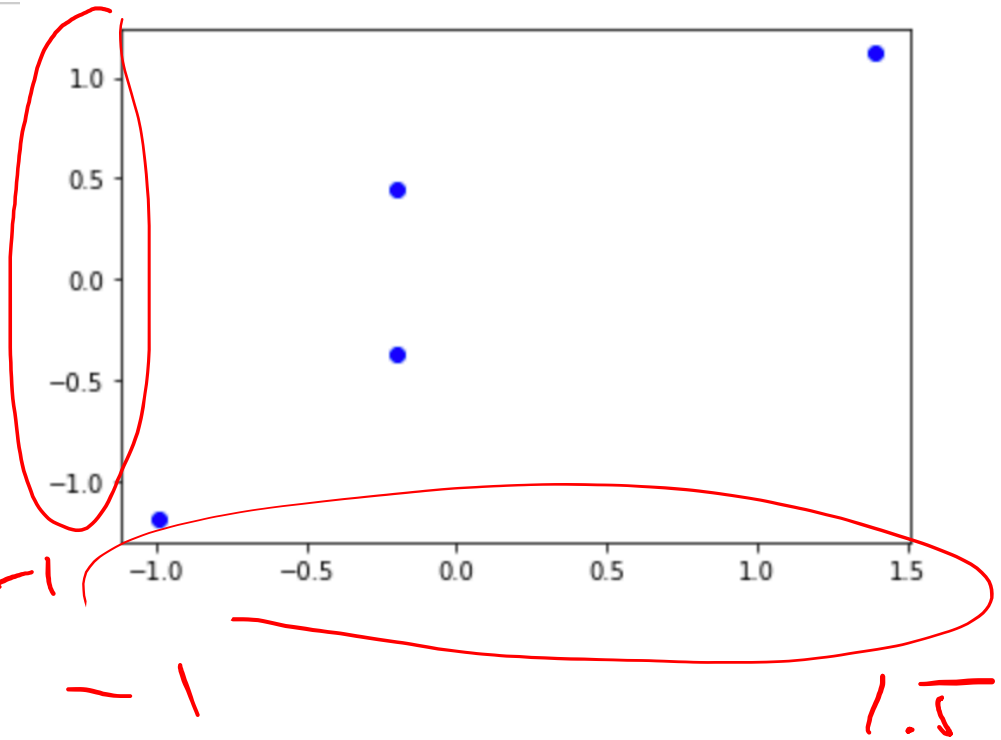


Standardizing Data

Original



Standardized



Note on Standard Deviation

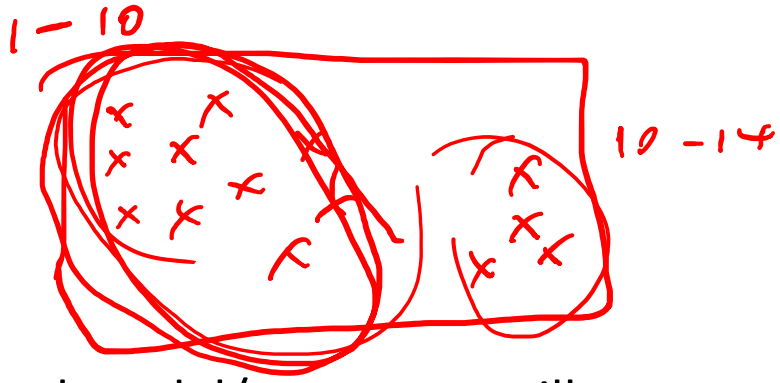
- There are two commonly used, slightly different computations for standard deviation:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

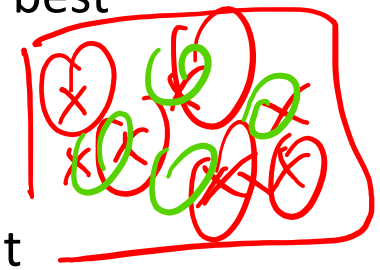
$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$$

- Typically the first version is to be used if we got the mean, μ, somehow other than computing it from the data itself.
- The second version adjusts for this lack of independence of σ and , u if in fact μ was computed as $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$
- For this course, we will typically want to use the version that divides by $N - 1$

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

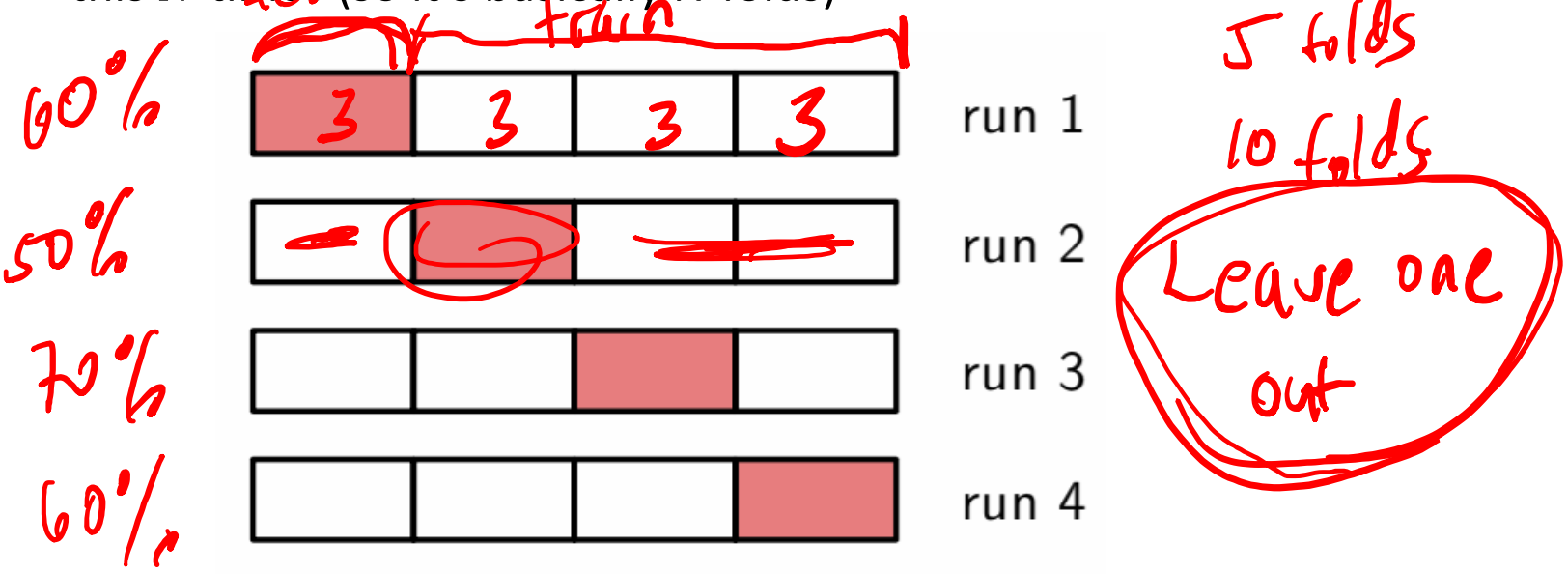
randomly shuffle data

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.

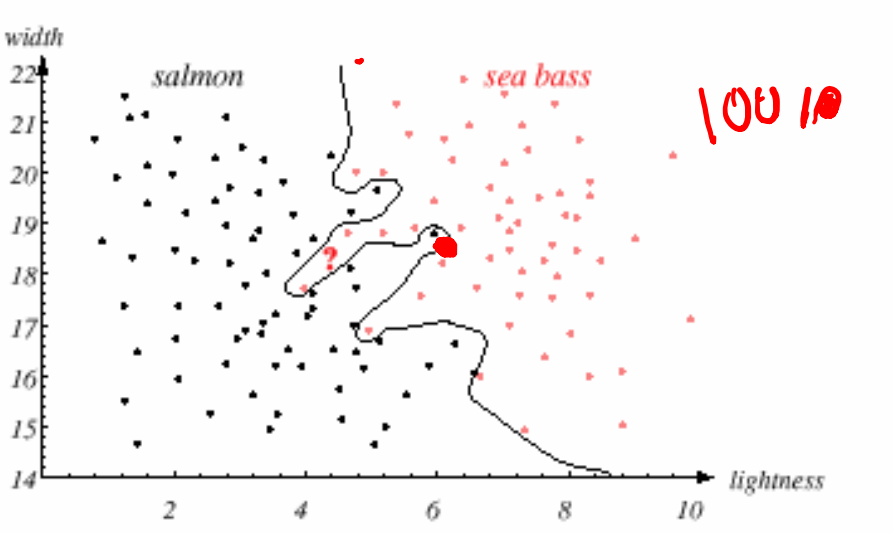
S-Folds Cross Validation

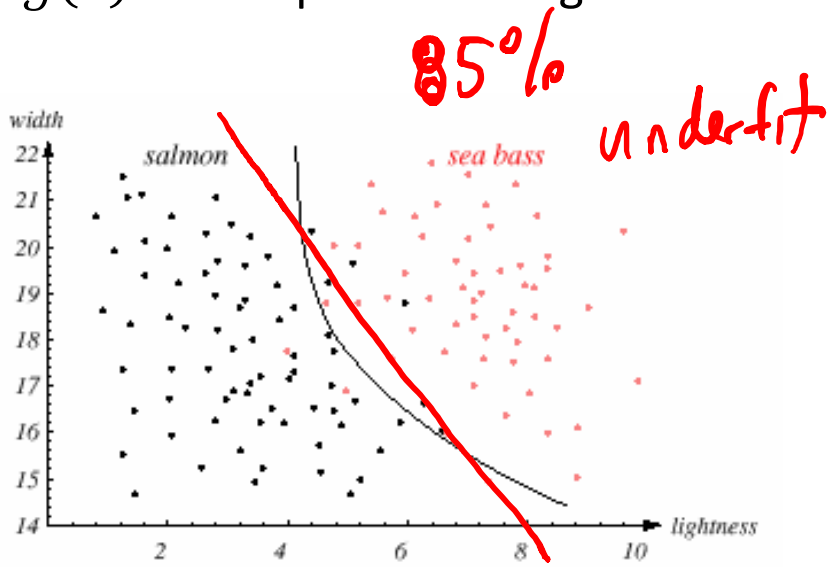
- There are a few types of cross-validation
  - S-Folds: Here we'll divide our data up into S parts, train on $S-1$ of them and test the remaining part. Do this S times
  - Leave-one-out: If our data set is really small we may want to built our system on $N-1$ samples and test on just one sample. And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



Over/Underfitting

- The biggest problem with supervised learning is **over** or **under-fitting**
- If we over-fit our training data, then we're finding a function for the training data, not the function of the entire system.
  - 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 $f(z)$.

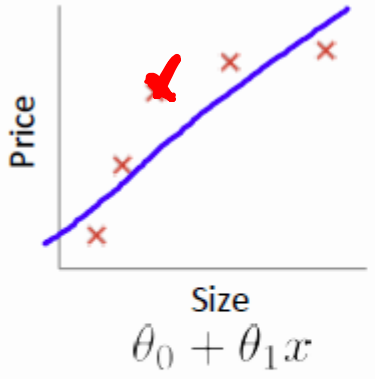




Bias vs Variance

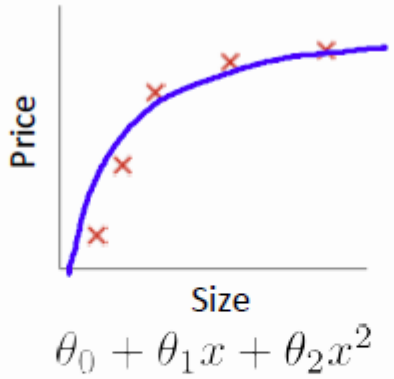
- Sometimes instead of talking about over-fitting and under-fitting we talk about *bias* and *variance*
- If our trained model is very dependent on the training data set (that is it fits it very well), then there will be a large ***variance*** in the models given different training sets.
  - Therefore 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**.
  - Which is not necessarily due to underfitting. But it could be…

Over/Under Fitting



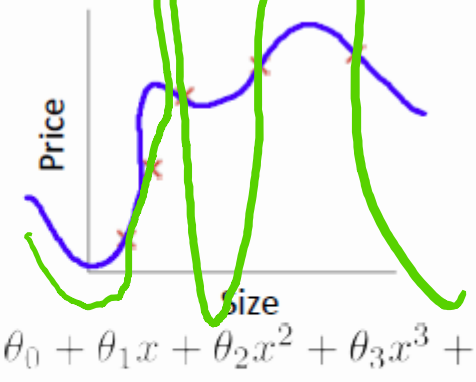
$\theta_0 + \theta_1 x$

High bias
(underfit)



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

"Just right"



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

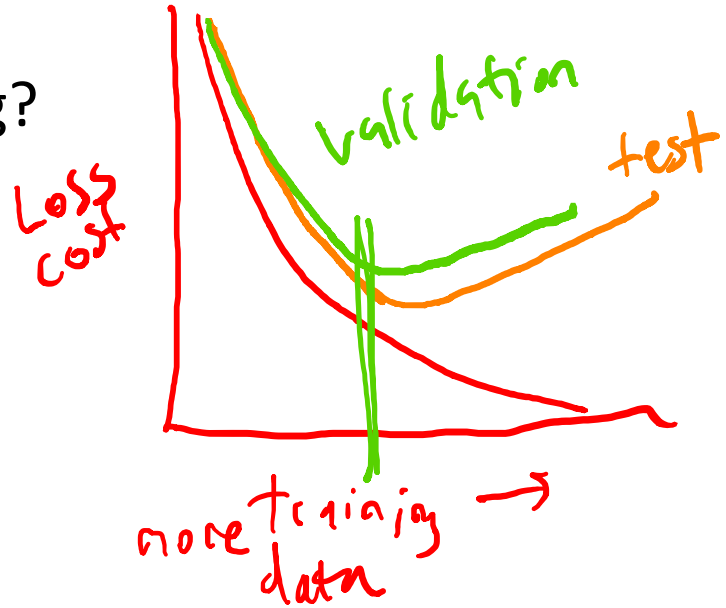
High variance
(overfit)

Detecting 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.

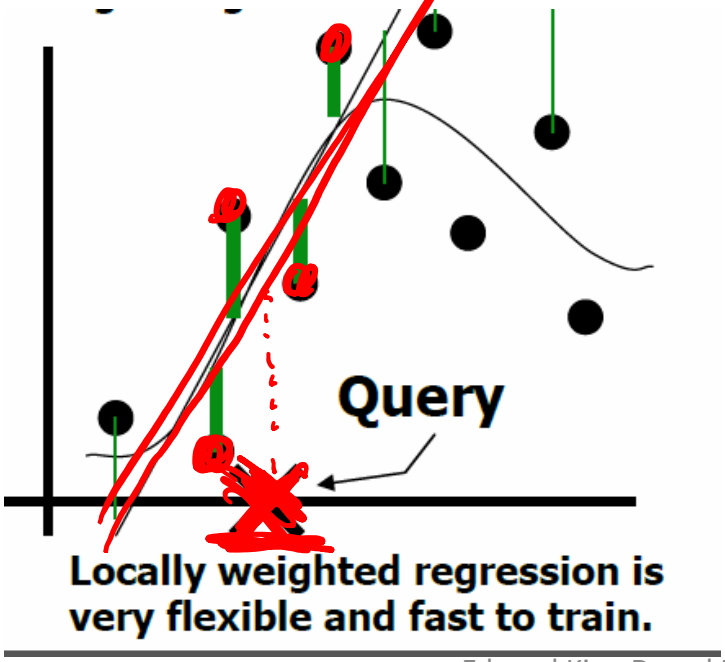
Dealing with Over/Under-Fitting

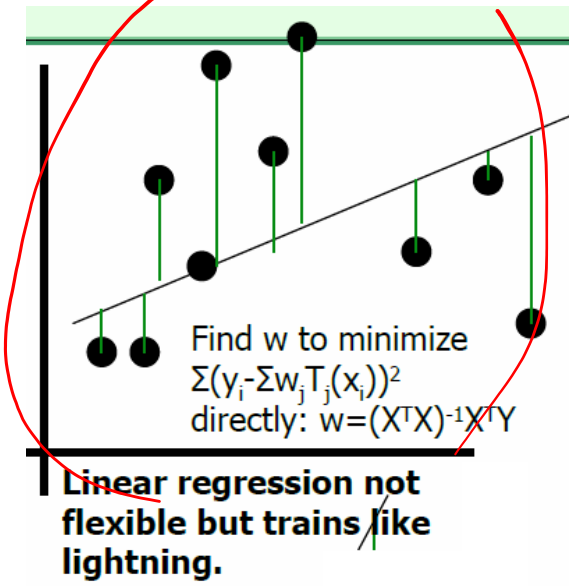
- How can we deal with under-fitting?
  - Make a more complex model
    - May involve need more features
  - Trying a different algorithm
- How can we deal with over-fitting?
  - Use a less complex model
    - May involve using less features
  - Try a different algorithm.
  - Get more data
  - Use a third set to choose between hypothesis (called a *validation set)*.
  - Add a penalization (or regularization) term to the equations to penalize model complexity.



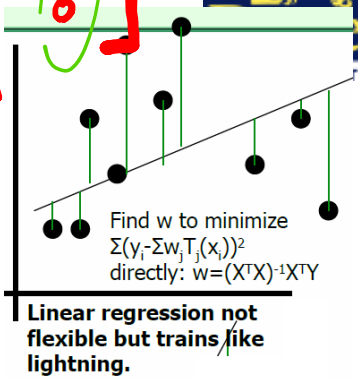
Locally Weighted Regression

- What we just did was *globally* linear regression
  - Since it's linear, it's not very flexible
- Locally weighted regression helps!



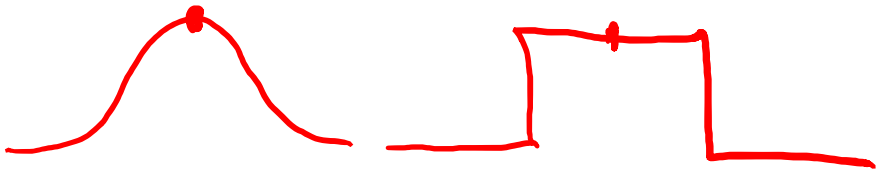


Locally Weighted Regression



- We need to make a few choices:

1. How many neighbors to allow to influence a test point?
   - All of them?



2. A way to give larger weights to closer points
   - A common way is to use the Gaussian similarity metric:

$$\beta(a,b) = e^{\frac{-d(a,b)}{k^2}}$$

   - Where $d(a,b)$ is the distance between a and b and k is some parameter affecting the drop-off rate.

$\sqrt{(a_1-b_1)^2+(a_2-b_2)^2\ldots(a_n-b_n)^2}$

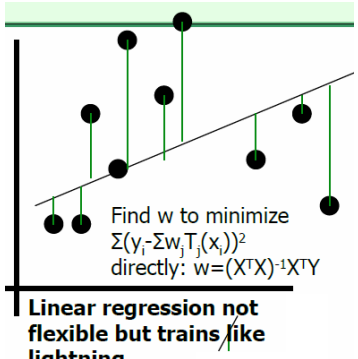
$|a_1-b_1|+|a_2-b_2|\ldots$

Locally Weighted Regression

- Then we need to find a local model:
  - Find θ that minimizes the locally weighted sum of squared residuals relative to sample x:

$$\theta = \underset{\theta}{\operatorname{argmin}} \sum_{k \in training\ set} \beta(x, X_k)(Y_k - X_k\theta)^2$$

  - Where $\beta(x, X_k)$ is the **weight** of training point k to our query/test point $query$



Locally Weighted Regression

$$\theta = \underset{\theta}{\operatorname{argmin}} \sum_{k \in training\ set} \beta(x, X_k)(Y_k - X_k\theta)^2$$

- To solve this, we can first put this equation in matrix format by creating a diagonal matrix of the weights:

$$W = Diag\big(\beta(x, X_1), \ldots, \beta(x, X_N)\big)$$

- Then we can write this as:

$$\theta = argmin_\theta\big((Y - X\theta)^T W (Y - X\theta)\big)$$

- Again taking the derivative, setting equal to zero, and solving for θ we get:

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

Local Regression

- Let's use rows [1,3, 5, 6] for training and the rest for testing
- First I'll standardize the feature data and add a bias feature:
  - $X_{train} = \begin{bmatrix} 1 & -1.2402 \\ 1 & -0.3382 \\ 1 & 0.5637 \\ 1 & 1.0147 \end{bmatrix}$ $Y_{train} = \begin{bmatrix} 6 \\ 9 \\ 17 \\ 12 \end{bmatrix}$
  - $X_{test} = \begin{bmatrix} 1 & -0.7892 \\ 1 & 0.1127 \end{bmatrix}$ $Y_{test} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$

Data Points:

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

Local Regression

$$X_{train} = \begin{bmatrix} 1 & -1.2402 \\ 1 & -0.3382 \\ 1 & 0.5637 \\ 1 & 1.0147 \end{bmatrix} \quad Y_{train} = \begin{bmatrix} 6 \\ 9 \\ 17 \\ 12 \end{bmatrix}$$

$$X_{test} = \begin{bmatrix} 1 & -0.7892 \\ 1 & 0.1127 \end{bmatrix} \quad Y_{test} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

- Let's compute a local model for the first test sample using Gaussian similarity, a squared distance function, all the training data, and a fall-off factor of $k = 1$
- $x = [1, -0.7892]$
- $W = Diag\left(\begin{bmatrix} e^{-0.2034} \\ e^{-0.2034} \\ e^{-1.8305} \\ e^{-3.2542} \end{bmatrix}\right) = \begin{bmatrix} 0.8160 & 0 & 0 & 0 \\ 0 & 0.8160 & 0 & 0 \\ 0 & 0 & 0.1603 & 0 \\ 0 & 0 & 0 & 0.0386 \end{bmatrix}$
- $\theta = \left(X_{train}^T W X_{train}\right)^{-1} X_{train}^T W Y_{train} = \begin{bmatrix} 11.3051 \\ 4.5491 \end{bmatrix}$
- $\hat{y} = [1, -0.7892] \cdot \theta = 7.7148$
- $SE = (y - \hat{y}) = (1 - 7.718)^2 = 45.0884$

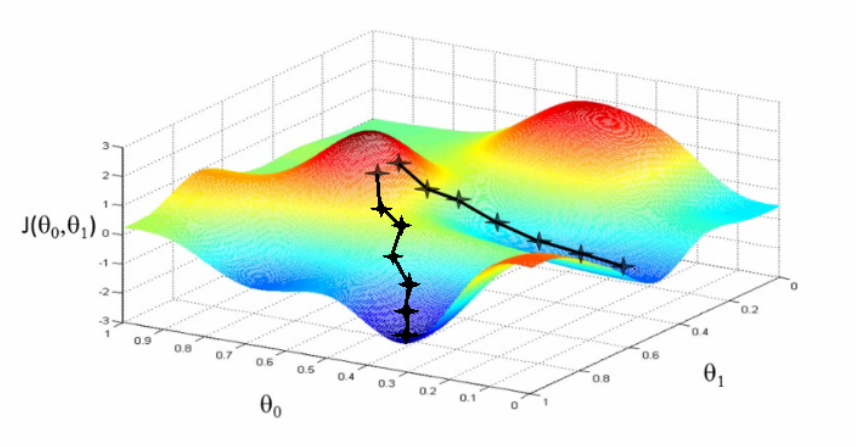
Local Regression

$$X_{train} = \begin{bmatrix} 1 & -1.2402 \\ 1 & -0.3382 \\ 1 & 0.5637 \\ 1 & 1.0147 \end{bmatrix} \quad Y_{train} = \begin{bmatrix} 6 \\ 9 \\ 17 \\ 12 \end{bmatrix}$$

$$X_{test} = \begin{bmatrix} 1 & -0.7892 \\ 1 & 0.1127 \end{bmatrix} \quad Y_{test} = \begin{bmatrix} 1 \\ 5 \end{bmatrix}$$

- Let's compute a local model for the **second** test sample
- $x = [1, 0.1127]$
- $W = \begin{bmatrix} 0.1603 & 0 & 0 & 0 \\ 0 & 0.8160 & 0 & 0 \\ 0 & 0 & 0.8160 & 0 \\ 0 & 0 & 0 & 0.4433 \end{bmatrix}$
- $\theta = \left(X_{train}^T W X_{train}\right)^{-1} X_{train}^T W Y_{train} = \begin{bmatrix} 11.46 \\ 4.3155 \end{bmatrix}$
- $\hat{y} = [1\ , 0.1127] \cdot \theta = 11.9466$
- $SE = 48.2552$
- RMSE (over these two test samples)$= 6.8317$

Gradient Ascent/Descent



Gradient Ascent/Descent

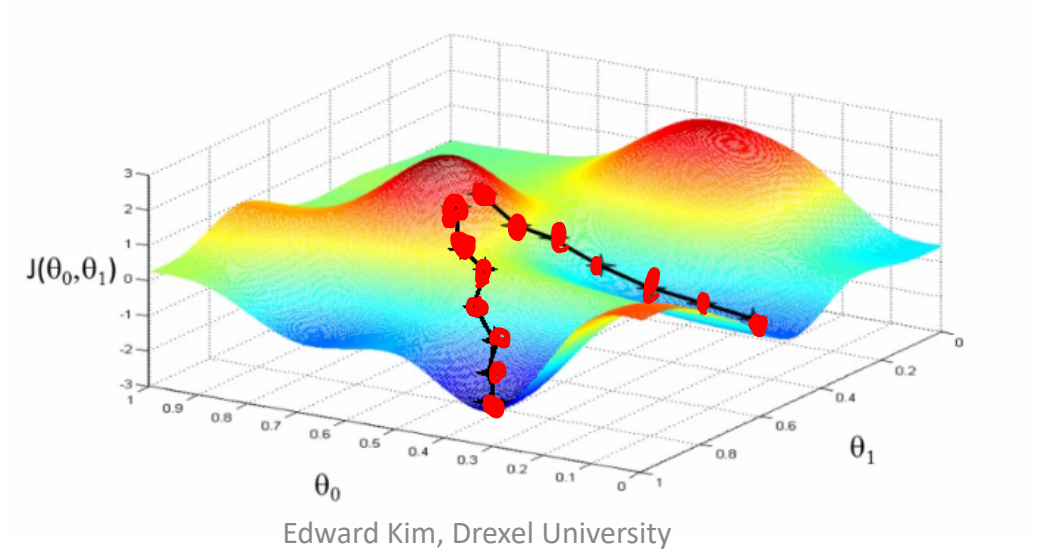
- The solution to linear regression that we just provided, $\theta = (X^T X)^{-1} X^T Y$, is called the *closed-form* solution to the problem.
  - We are able to use mathematics to come up with a direct solution to the minima/maxima problem.
- However, for some problems a closed-form solution/equation 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

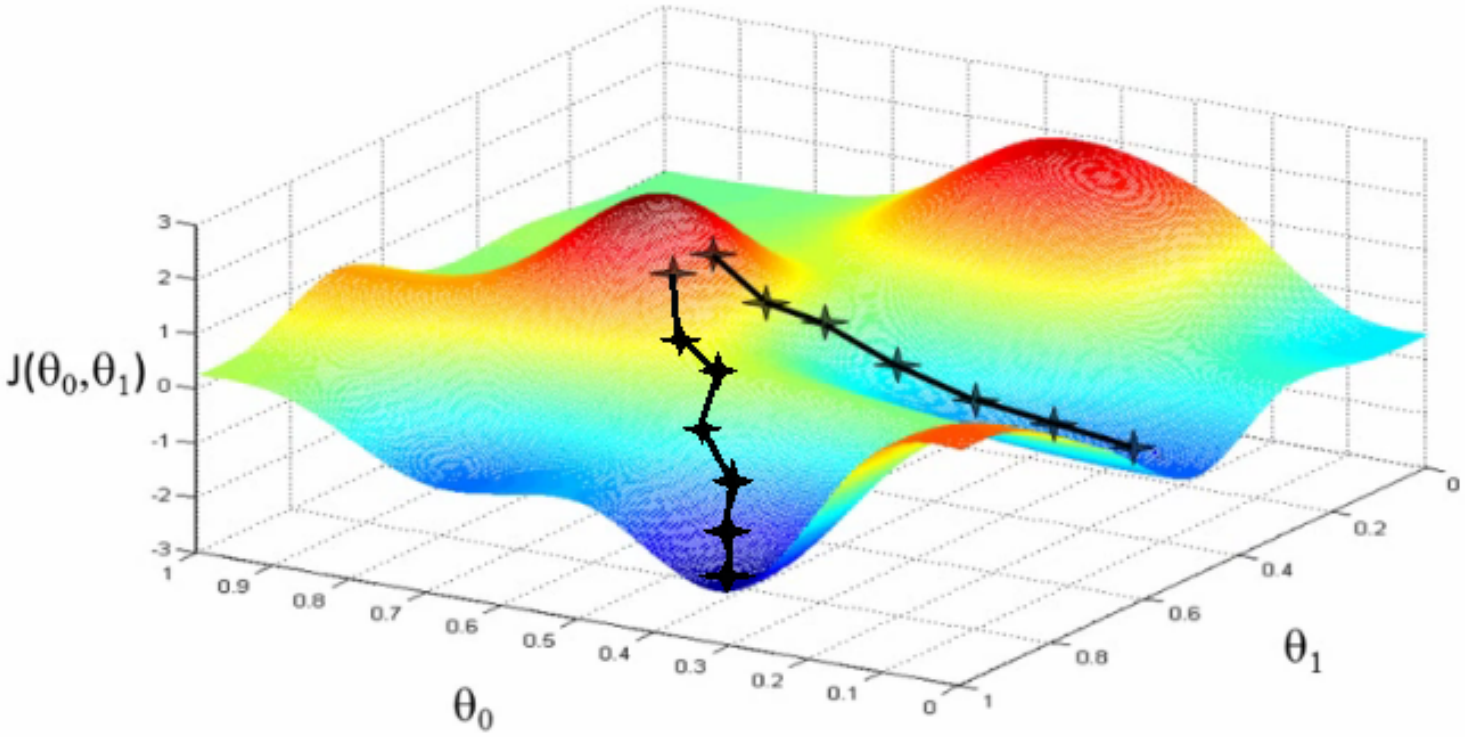
- Another approach to solving a minima/maxima problem is to compute the *gradient* of the equation with regards to each parameter in order to move our current parameter's value in that direction
  - And iteratively do this until we converge to a minima/maxima
- This approach is called ***gradient descent***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

Gradient Descent/Assent

- The gradient of J can be thought of the way (vector) to go towards the maxima/minima
  - The gradient is synonymous with the slope.
- 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/Assent



The graphic depicts gradient assent with two different initial values of (θ_0, θ_1) and updating each parameter simultaneously

Gradient Descent

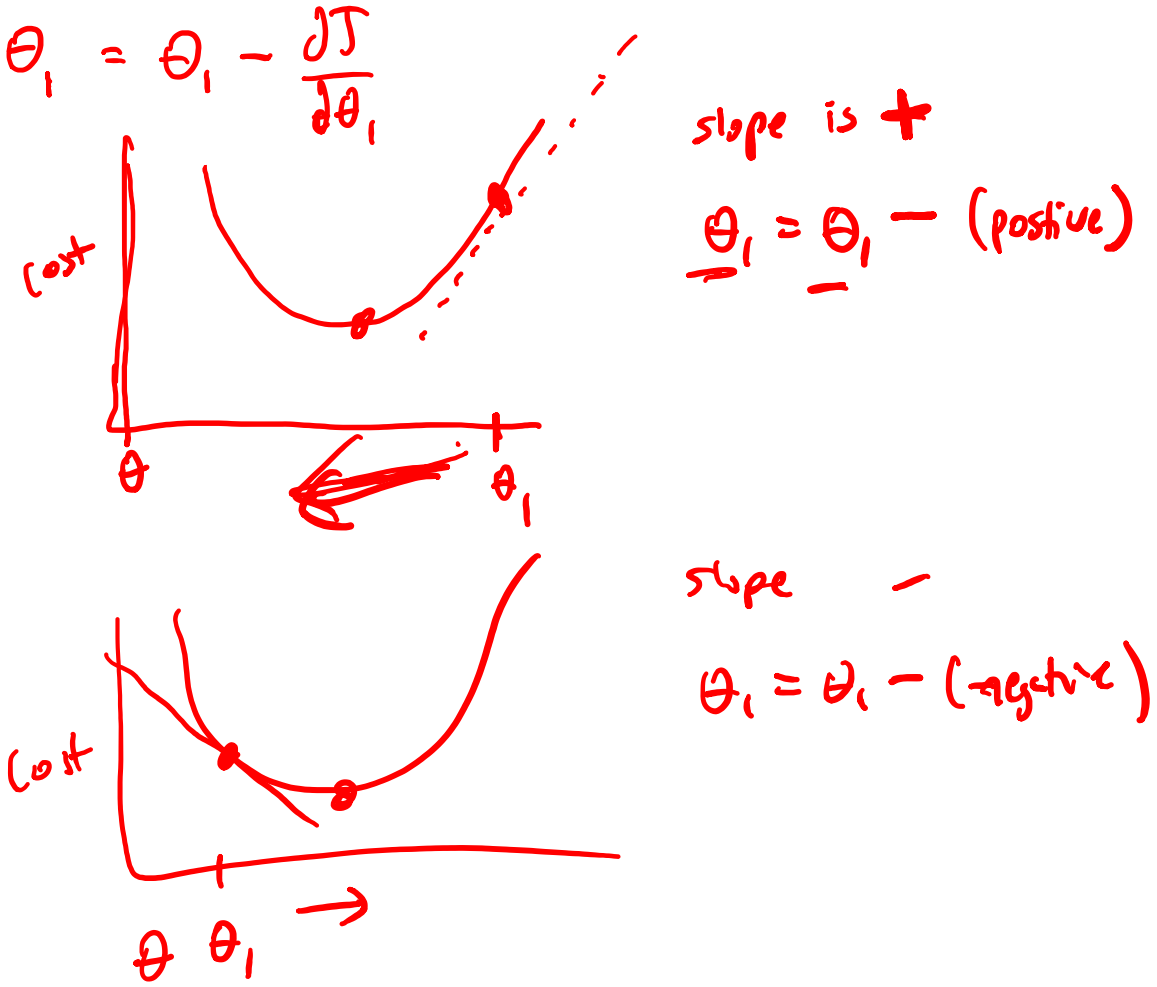
- To find the gradient of J we just take its derivative with respect to the variable we are solving for.
- We want to find the gradient with respect to each of our parameters, $\theta_0, \theta_1, \ldots, \theta_D$
- So the overall gradient, $\frac{\partial J}{\partial \theta}$, can be written as:

$$\frac{\partial J}{\partial \theta} = \begin{bmatrix} \frac{\partial J}{\partial \theta_0} \\ \frac{\partial J}{\partial \theta_1} \\ \cdots \\ \frac{\partial J}{\partial \theta_D} \end{bmatrix}$$

- Then we move our parameters in the direction of this gradient.

$$\theta = \theta - \frac{\partial J}{\partial \theta}$$

Intuition behind gradient update



Least Squares Gradient Descent

- Let's first compute the gradient with respect to a single parameter, θ_i and a single observation, (x, y) and then *vectorize* our solution to update all parameters simultaneously.
- This approach, where we just use one observation at a time is called *iterative,* or *online* gradient learning.
- Returning to our least square linear regression problem, for a single observation the error is:

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

$\frac{\partial}{\partial \theta_1}$ ~~$\theta_0 + \theta_1 x_1 + \theta_2 x_2 \ldots \theta_n x_n$~~

Derive the update rule

$$\frac{\partial J}{\partial \theta_1} = (y - X\theta)^2 \Rightarrow 2(y - X\theta)(0 - x_1)$$

$$-2x_1(y - X\theta)$$

$$\frac{\partial J}{\partial \theta_1} = 2x_1(X\theta - y)$$

$$\frac{\partial J}{\partial \theta_2} = 2x_2(X\theta - y)$$

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

$$\frac{\partial J}{\partial \theta} = \begin{bmatrix} 2x_0(X\theta - y) \\ 2x_1(X\theta - y) \\ 2x_2(X\theta - y) \\ \vdots \end{bmatrix}$$

Least Squares Gradient Descent

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

- Now let's take the gradient of this with respect to one of the parameters, θ_i

$$\frac{\partial J}{\partial \theta_i} = -2x_i y + 2x_i x\theta = 2x_i(x\theta - y)$$

- We can then vectorize this to get all the gradients at once for this observation:

$$\frac{\partial J}{\partial \theta} = 2x^T(x\theta - y) \Rightarrow 2x^T x\theta - 2x^T y = 0$$

$$2x^T x\theta = 2x^T y$$

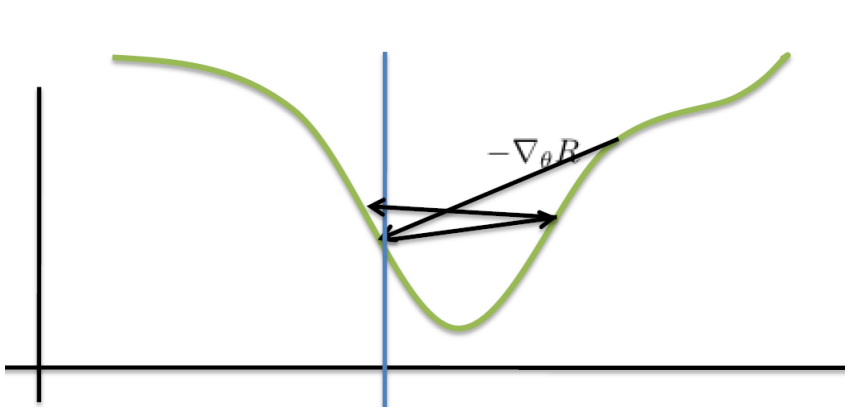
$$\theta = (x^T x)^{-1} x^T y$$

- And update your parameters as:

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

The Learning Rate

- If we just move our parameters according to $\frac{\partial J}{\partial \theta}$ we may either:
  - Go too slowly
  - Or *overjump* the desired minima/maxima.
- Therefore, we typically add a *hyperparameter*, η, called the *learning rate* that controls *how much* to go in the direction of the gradient.
- So now $\theta = \theta - \eta \frac{\partial J}{\partial \theta}$



Iterative Gradient Descent Pseudocode

- Given:
  - Standardized data matrix X of size $N \times D$
  - Target value matrix Y of size $N \times 1$
- Add a bias column to X so that $X = [1, X]$ and has size $N \times (D + 1)$
- Initialize parameters θ_j for $j = 1, \ldots, D + 1$ to some random value. Note that θ will be a column vector of size $(D + 1) \times 1$
- Until convergence
  - $\forall x \in X$
    - Compute gradient $\frac{\partial J}{\partial \theta} = 2x^T(x\theta - y)$
    - Update θ such that $\theta = \theta - \eta \frac{\partial J}{\partial \theta}$

Gradient Descent

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

Variants of Gradient Descent

- This version of gradient descent, where we update the parameters one observation at a time, is called *iterative* or *online* gradient descent.
- The issue with iterative gradient descent is that it may take online to converge and/or be more likely to converge to some local solution.
- Instead we can update the parameters using the *average* of the gradients created by *all* the observations. This is called *batch gradient descent* and can be written as:

$$\theta = \theta - \frac{\eta}{N} X^T (X\theta - Y)$$

  - *Note: Here we let η absorb the other scalar, 2.*
- Of course batch gradient descent is that it requires all the data to be in memory at once
  - Which can be in issue with big data
- Therefore a variant called *stochastic mini-batch* gradient descent is more common
  - Each (outer) iteration, shuffle all the data
  - Break full data into smaller batches and use each to update parameters.

Example 2 (again)

- Model:

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

Note: Final exam out of 200

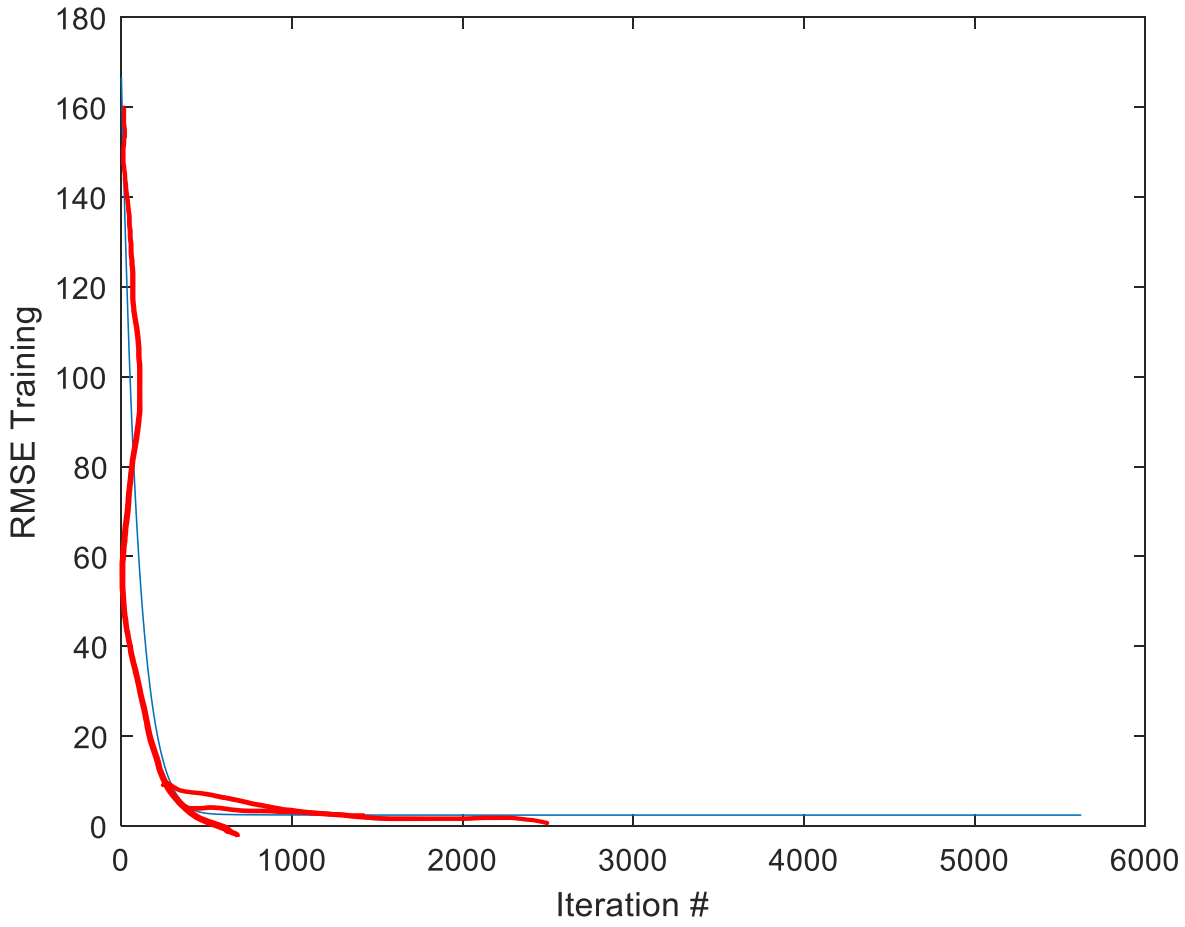
- Testing Set: The first 8 samples
- Training Set: The rest (next 17 samples)
- Let's do this with full-batch gradient descent
- Settings:
  - Standardize data using training data
  - Seed the rng to zero
  - Initialize each parameter to some random number
  - Use batch gradient descent
  - $\eta = 0.01$
  - Terminate when change in training RMSE < 2^{-32}

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

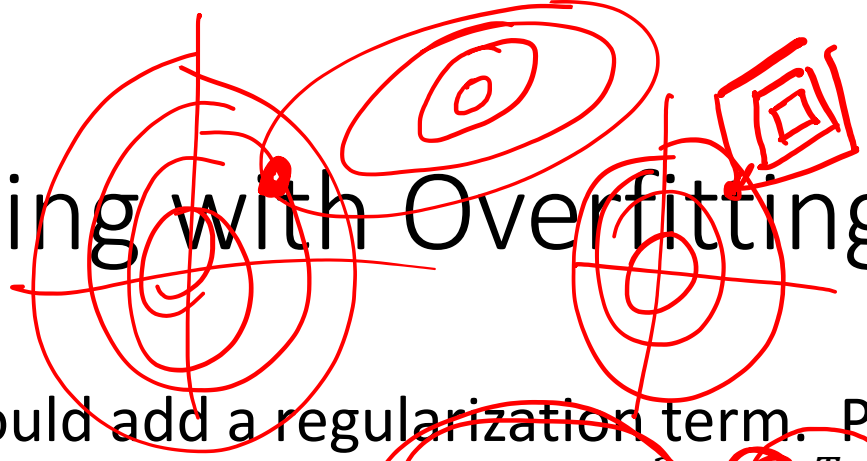
Approaches Compared

- Closed Form Linear Regression
  - Model: $\theta = [166.5294, 3.1218, 4.9823, 10.9629]^T$
  - RMSE: 2.8207
- Gradient Descent
  - Model: $\theta = [166.5294, 3.1231, 4.9815, 10.9623]^T$
  - RMSE Testing: 2.8207
  - Number of iterations: 5625

Approaches Compared



Dealing with Overfitting



- We could add a regularization term. Perhaps:

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

  - 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.

Beyond Linear Regression

- What if we want to find the parameters for a quadratic equation like $y = ax^2 + bx + c$?
- Imagine our observation just has a single feature, ie. $x = [x_1]$.
- We can write our quadratic equation as

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

- If we re-write x to be $[1, x_1, x_1^2]$ then we can write this equation as $y = x\theta$ and we're right back at linear regression!

$(X^T X)^{-1} X^T y$

Logistic Regression

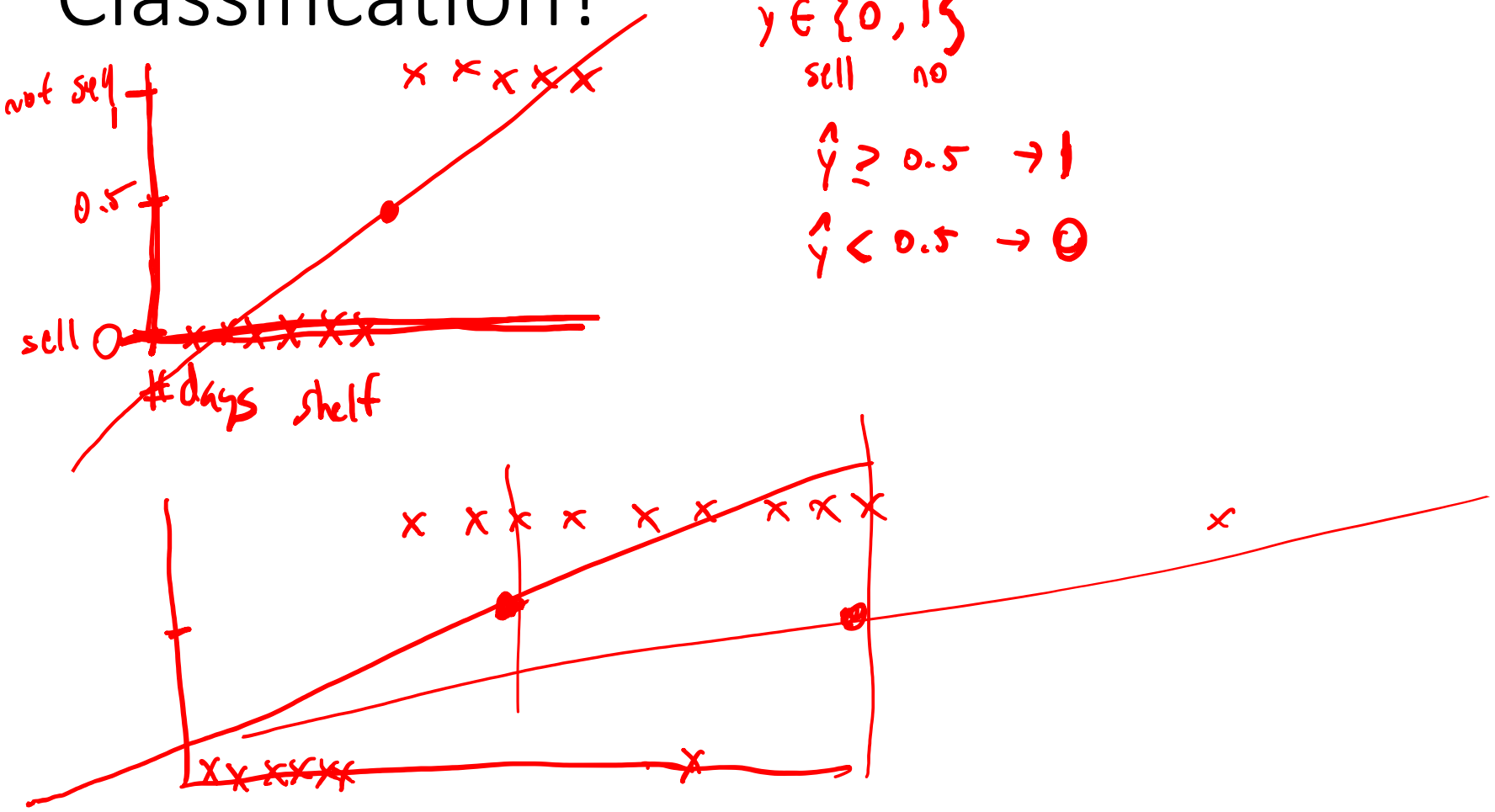
Logistic Regression

independent dependent

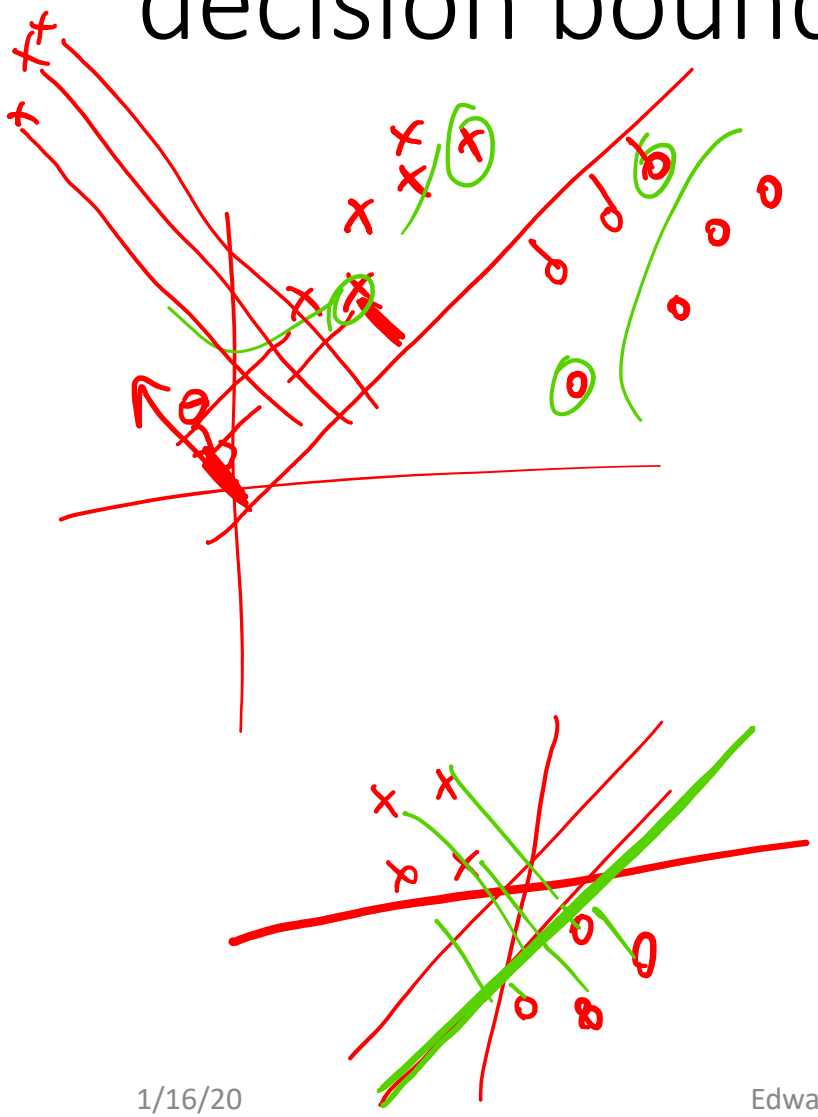
$y = mx + b$

- Logistic Regression is a terrible name!
  - It's not regression at all!
  - It's classification
- But as you'll see, how we do it is extremely similar to *linear* regression

Linear Regression for Classification?



Logistic Regression – Line as decision boundary



$y \in \{0, 1\}$

$ax + by - c = 0$

$\theta^T x = 0$

$\theta \cdot x_i$ = distance to boundary

$\theta \cdot P_i = 1{,}000$ units?

$\theta x_i \Rightarrow$ Prob cat $[0 \rightarrow 1]$

Logistic Regression

- With logistic regression we assume binary classification and want to provide a probability for the positive class:

$$0 \leq P(y = 1) \leq 1$$

- Recall from *linear* regression we computed

$$g(x, \theta) = x\theta$$

- We can alter this for use in computing $P(y = 1)$ as:

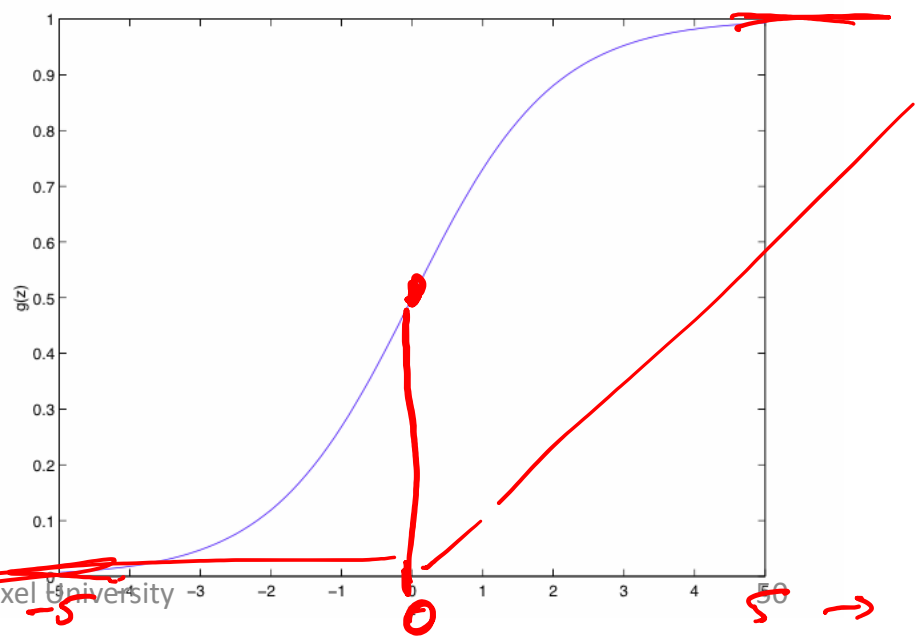
$$P(y = 1) = g(x, \theta) = \frac{1}{1 + e^{-x\theta}}$$

sigmoid
logistic
squashing

Logistic Regression

$$P(y = 1|x) = g(x, \theta) = \frac{1}{1 + e^{-x\theta}}$$

- This function, Let $g(z) = \frac{1}{1+e^{-z}}$ is called the *sigmoid* or *logistic* function
  - Tends to 0 as z decreases
  - Tends to 1 as z increases
- This has the nice characteristic in that it's differentiable
  - Why might that be important?!



Logistic Regression

- If we consider
  - $P(y = 1|x, \theta) = \frac{1}{1+e^{-x\theta}} = g(x, \theta)$
- Then we can compute the probability of being from the negative class as:
  - $P(y = 0|x, \theta) = 1 - g(x, \theta)$
- Ultimately we want to find the parameters θ to minimize the classification error
  - Or conversely, to find the parameters θ to maximize the correct class likelihood

$P(y|x,\theta) = g(x,\theta)^{y}(1-g(x,\theta))^{(1-y)}$

$y=1 = g(x,\theta)$

$y=0 (1-g(x,\theta))$

Fit Parameters Based on Maximum Likelihood

- Given a supervised observation (x, y), we can compute the **likelihood** that we are correct as

$$\ell(y|x,\theta) = \big(g(x,\theta)\big)^{y}\big(1-g(x,\theta)\big)^{(1-y)}$$

- Doing this for the entire dataset, since the observations are *conditionally independent* of one another we have:

$$\ell(Y|X,\theta) = \prod_{t=1}^{N} \ell(Y_t,|X_t,\theta) = \prod_{t=1}^{N} (g(X_t,\theta)^{Y_t}\big(1-g(X_t,\theta)\big)^{(1-Y_t)}$$

Log Likelihood

$\log(ab) = \log a + \log b$

$\log(a^b) = b \log a$

- So what do we do with this likelihood $\ell(Y|X,\theta)$?
  - We want to maximize it!
  - Or minimize $-\ell(Y|X,\theta)$
- So we're going to want to take the derivative
- But taking the derivative of a product of a lot of things involves a very long expansion
- Let's instead first take the log of this
  - Doing so will result in a sum which is easier to take the derivative of.
  - So now we want to *maximize the log likelihood*

Log Likelihood Rules

$\log(a^b) = b \log a$

$\log(ab) = \log a + \log b$

$$\prod_{i=1}^{N} g(x_i, \theta)^{y_i} \cdot (1 - g(x_i, \theta))^{(1-y_i)}$$

$$\sum_{i=1}^{N} y_i \ln g(x_i, \theta) + (1 - y_i) \ln (1 - g(x_i, \theta))$$

Log Likelihood

- From the properties of logarithms
  - $\log_b(mn) = \log_b(m) + \log_b(n)$
  - $\log_b(m^n) = n \cdot \log_b(m)$
- Returning to our likelihood for a single observation, $\ell(y, |x, \theta)$, we get
  - $\ell(y|x,\theta) = ln\left(g(x,\theta)^y\left(1 - g(x,\theta)\right)^{(1-y)}\right)$
  - $= \ln(g(x,\theta)^y) + \ln\left(\left(1 - g(x,\theta)\right)^{1-y}\right)$
  - $= y\, ln\big(g(x,\theta)\big) + (1-y)\ln\big(1 - g(x,\theta)\big)$

Log Likelihood

- Since we're taking the log of product of this for each instance we get a sum!

$$\ell(Y|X,\theta) = \log P(Y|X,\theta) = \sum_{t=1}^{N} Y_t \, ln\big(g(X_t,\theta)\big) + (1-Y_t)\ln\big(1-g(X_t,\theta)\big)$$

- Ideally we'd like to take the derivative of this with respect to θ, set it equal to zero, and solve for θ to find the maxima
  - The closed form approach
  - But this doesn't exist ☹
- So what's our other approach?
  - Do partial derivatives on the parameters and use gradient descent! (actually in this case gradient ascent, since we're trying to maximize)
  - First do this for a single observation and single parameter
    - Then vectorize!

Logistic Regression

- Logistic Regression is a terrible name!
  - It's not regression at all!
  - It's classification
- But as you'll see, how we do it is extremely similar to *linear* regression

Logistic Regression

- With logistic regression we assume binary classification and want to provide a probability for the positive class:

$$0 \leq P(y = 1) \leq 1$$

- Recall from *linear* regression we computed

$$g(x, \theta) = x\theta$$

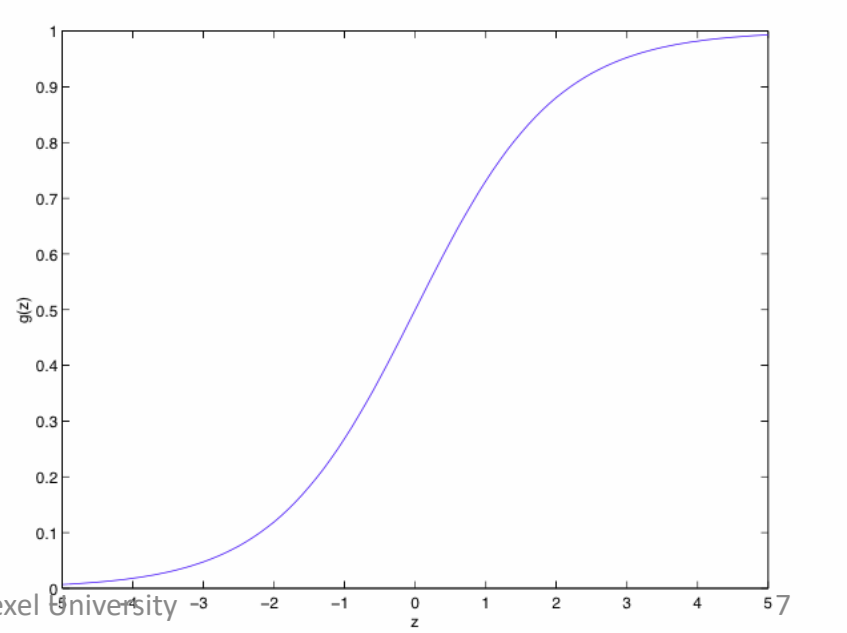
- We can alter this for use in computing $P(y = 1)$ as:

$$P(y = 1) = g(x, \theta) = \frac{1}{1 + e^{-x\theta}}$$

Logistic Regression

$$P(y = 1|x) = g(x, \theta) = \frac{1}{1 + e^{-x\theta}}$$

- This function, Let $g(z) = \frac{1}{1+e^{-z}}$ is called the *sigmoid* or *logistic* function
  - Tends to 0 as z decreases
  - Tends to 1 as z increases
- This has the nice characteristic in that it's differentiable
  - Why might that be important?!



Logistic Regression

- If we consider
  - $P(y = 1|x, \theta) = \frac{1}{1+e^{-x\theta}} = g(x, \theta)$
- Then we can compute the probability of being from the negative class as:
  - $P(y = 0|x, \theta) = 1 - g(x, \theta)$
- Ultimately we want to find the parameters θ to minimize the classification error
  - Or conversely, to find the parameters θ to maximize the correct class likelihood

Fit Parameters Based on Maximum Likelihood

- Given a supervised observation (x, y), we can compute the **likelihood** that we are correct as

$$\ell(y|x,\theta) = \big(g(x,\theta)\big)^{y}\big(1-g(x,\theta)\big)^{(1-y)}$$

- Doing this for the entire dataset, since the observations are *conditionally independent* of one another we have:

$$\ell(Y|X,\theta) = \prod_{t=1}^{N} \ell(Y_t,|X_t,\theta) = \prod_{t=1}^{N} (g(X_t,\theta)^{Y_t}\big(1-g(X_t,\theta)\big)^{(1-Y_t)}$$

Log Likelihood

- So what do we do with this likelihood $\ell(Y|X,\theta)$?
  - We want to maximize it!
  - Or minimize $-\ell(Y|X,\theta)$
- So we're going to want to take the derivative
- But taking the derivative of a product of a lot of things involves a very long expansion
- Let's instead first take the log of this
  - Doing so will result in a sum which is easier to take the derivative of.
  - So now we want to *maximize the log likelihood*

Log Likelihood

- From the properties of logarithms
  - $\log_b(mn) = \log_b(m) + \log_b(n)$
  - $\log_b(m^n) = n \cdot \log_b(m)$
- Returning to our likelihood for a single observation, $\ell(y, |x, \theta)$, we get
  - $\ell(y|x,\theta) = ln\left(g(x,\theta)^y\left(1 - g(x,\theta)\right)^{(1-y)}\right)$
  - $= \ln(g(x,\theta)^y) + \ln\left(\left(1 - g(x,\theta)\right)^{1-y}\right)$
  - $= y\, ln\big(g(x,\theta)\big) + (1-y)\ln\big(1 - g(x,\theta)\big)$

Log Likelihood

- Since we're taking the log of product of this for each instance we get a sum!

$$\ell(Y|X,\theta) = \log P(Y|X,\theta) = \sum_{t=1}^{N} Y_t \, ln\big(g(X_t,\theta)\big) + (1-Y_t)\ln\big(1-g(X_t,\theta)\big)$$

- Ideally we'd like to take the derivative of this with respect to θ, set it equal to zero, and solve for θ to find the maxima
  - The closed form approach
  - But this doesn't exist ☹
- So what's our other approach?
  - Do partial derivatives on the parameters and use gradient descent! (actually in this case gradient ascent, since we're trying to maximize)
  - First do this for a single observation and single parameter
    - Then vectorize!

Log Likelihood Derivation

$$\left(\frac{y}{g(x,\theta)} - \frac{1-y}{1-g(x,\theta)}\right) \cdot x_j \; g(x,\theta)(1-g(x,\theta))$$

$$\theta = \theta - \frac{\eta}{N} X^T (g(x,\theta) - y)$$

$x\theta$

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

$$\frac{(1-g(x,\theta))y}{(1-g(x,\theta))(g(x,\theta))} - \frac{g(x,\theta)(1-y)}{g(x,\theta)(1-g(x,\theta))}$$

$$X = \begin{bmatrix} x_0 \\ x_1 \\ \\ x_N \end{bmatrix} \cdot \begin{bmatrix} (y - g(x,\theta)) \\ (y - g(x,\theta)) \\ \vdots \\ (y - g(x,\theta)) \end{bmatrix}$$

$$y - y g(x,\theta) - g(x,\theta) + y g(x,\theta)$$

$$(1-g(x,\theta))\, g(x,\theta)$$

$$\Theta = \Theta + \frac{\eta}{N} X^T (y - g(x,\theta))$$

Log Likelihood Derivation

$$x_j \cdot g(x,\theta) \cdot \frac{e^{-x\theta}}{1+e^{-x\theta}} \Bigg| \frac{1+e^{-x\theta}}{1+e^{-x\theta}} - \frac{e^{-x\theta}}{1+e^{-x\theta}}$$

thing

$$\frac{1}{1+e^{-x\theta}}$$

$$x_j \cdot g(x,\theta) \cdot (1 - g(x,\theta))$$

$$1 - \text{thing} = g(x,\theta)$$

$$1 - g(x,\theta) = \text{thing}$$

To Maximum Likelihood

$$\frac{\partial}{\partial \theta_j}\ell(y|x,\theta) = \frac{\partial}{\partial \theta_j}\left(y\ln\left(g(x,\theta)\right) + (1-y)\ln\left(1-g(x,\theta)\right)\right)$$

- First off, a reminder…

$$\frac{\partial}{\partial x}(\ln x) = \frac{1}{x}\cdot\frac{\partial}{\partial x}(x)$$

- Therefore

$$\frac{\partial}{\partial \theta_j}\ell(y|x,\theta) = \frac{y}{g(x,\theta)}\frac{\partial}{\partial \theta_j}\left(g(x,\theta)\right) + \frac{(1-y)}{1-g(x,\theta)}\frac{\partial}{\partial \theta_j}\left(1-g(x,\theta)\right)$$

- But what is $\frac{\partial}{\partial \theta_j}(g(x,\theta)$?

To Maximum Likelihood

$$\frac{\partial}{\partial \theta_j}\ell(y|x,\theta) = \frac{y}{g(x,\theta)}\frac{\partial}{\partial \theta_j}\big(g(x,\theta)\big) + \frac{(1-y)}{1-g(x,\theta)}\frac{\partial}{\partial \theta_j}\big(1-g(x,\theta)\big)$$

- $\frac{\partial}{\partial \theta_j} g(x,\theta) = \frac{\partial}{\partial \theta_j}\left(\frac{1}{1+e^{-x\theta}}\right) = \frac{\partial}{\partial \theta_j}\left(1+e^{-x\theta}\right)^{-1}$
- $= -1\left(0 - x_j e^{-x\theta}\right)\left(1+e^{-x\theta}\right)^{-2} = \frac{x_j e^{-x\theta}}{\left(1+e^{-x\theta}\right)^2}$
- $= x_j \frac{1}{1+e^{-x\theta}} \frac{e^{-x\theta}}{1+e^{-x\theta}}$
- $= x_j g(x,\theta)\big(1-g(x,\theta)\big)$

To Maximum Likelihood

$$\frac{\partial}{\partial \theta_j} \ell(y|x,\theta) = \frac{y}{g(x,\theta)} \frac{\partial}{\partial \theta_j} \big(g(x,\theta)\big) + \frac{(1-y)}{1-g(x,\theta)} \frac{\partial}{\partial \theta_j} \big(1-g(x,\theta)\big)$$

- From the previous slide we have

$$\frac{\partial}{\partial \theta_j} g(x,\theta) = x_j g(x,\theta)\big(1-g(x,\theta)\big)$$

- Putting it all together (and simplifying) we get:

$$\frac{\partial}{\partial \theta_j} \ell(y|x,\theta) = x_j \big(y - g(x,\theta)\big)$$

To Maximum Likelihood

$$\frac{\partial}{\partial \theta_j} \ell(y|x,\theta) = x_j\big(y - g(x,\theta)\big)$$

- Vectorizing this for all parameters we have

$$\frac{\partial \ell}{\partial \theta} = x^T\big(y - g(x,\theta)\big)$$

- Vectorizing this to be the mean gradient over all observations we have:

$$\frac{\partial \ell}{\partial \theta} = \frac{1}{N} X^T\big(Y - g(X,\theta)\big)$$

Gradient Ascent Rule

$$\frac{\partial \ell}{\partial \theta} = \frac{1}{N} X^T \big(Y - g(X, \theta)\big)$$

- We want this to go towards a maxima
- So let's update θ as

$$\theta := \theta + \eta \left(\frac{\partial}{\partial \theta} \ell(Y|X, \theta) \right)$$

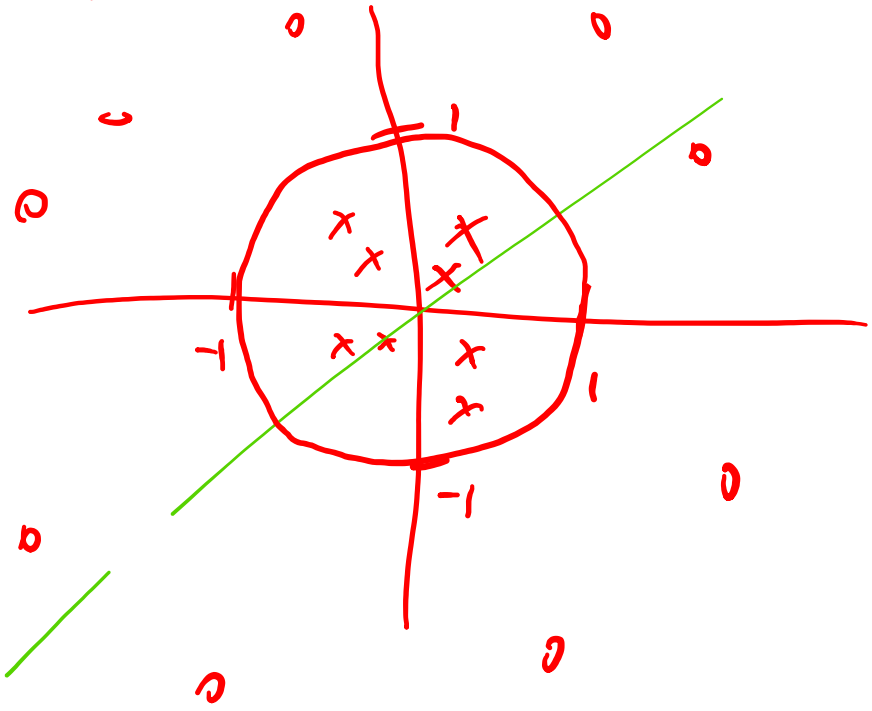
$$\theta = \theta + \frac{\eta}{N} X^T \big(Y - g(X, \theta)\big)$$

- This is (almost) the same form as the least squared error for linear regression!!!!

Non-linear decision boundaries

$\theta_0 + \theta_1 x_1$

$\theta_0 + \theta_1 x_1 + \theta_2 x_1^2$



$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2$

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$ $\Rightarrow$ $-1 + x_1^2 + x_2^2 = 0$

$x_1^2 + x_2^2 = 1$

$x^2 + y^2 = 1$

Evaluating a Classification Algorithm

Evaluation

- How can we tell if our ML algorithm is doing well?
- For estimating values (like in linear regression) we can just compute statistics on the error
- How about with classifiers?
  - We can count how often we predict the correct class
  - We call this *accuracy*.
    - Let Y_i be the true class, and $\widehat{Y}_i$ be the predicted class for observation i.

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(Y_i = \widehat{Y}_i)$$

- In addition, there are two types of classification:
  - Binary classification – Just two classes
  - Multi-Class classification – More than two classes
- Each of these have their own additional evaluations of interest.

Binary Classification Error Types

- For binary classification, often we're focused on attempting to "find" one particular class.
  - We refer to this as the positive class.
  - The other data is called the negative class.
- From this we can describe four different possibilities:
  - True positive = Hit
  - True negative = Correct rejection
  - False positive = False Alarm (Type 1 error)
  - False negative = Miss (Type 2 error)

	Predicted positive	Predicted negative
Positive examples	True positives	False negatives
Negative examples	False positives	True negatives

Evaluating your Classifier

- From the four error types, we can establish some binary-classification-specific measurements:
- *Precision* – percentage of things that were classified as positive and actually were positive
  - $Precision = \frac{TP}{TP+FP}$
- *Recall* – the percentage of true positives (*sensitivity*) correctly identified
  - $Recall = \frac{TP}{TP+FN}$
- *f-measure* – The weighted harmonic mean of precision and recall
  - $F_1 = \frac{2*precision*recall}{precision+recall}$

Using Class Likelihood

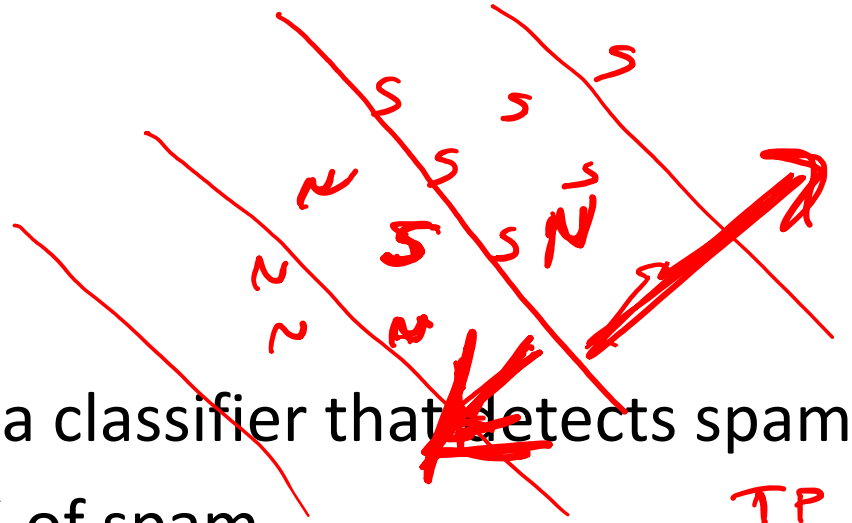
- Some classifiers don't just return what class an observation belongs to, but also return the probability of belonging to that class:

$$P(y = i|x)$$

- In these cases, we can use a *threshold* to determine what class an observation belongs to.
- For instance, for binary classification we can say:

$$\hat{y} = \begin{cases} Positive & P(y = Positive|x) > t \\ Negative & otherwise \end{cases}$$

Spam Example



- Let's imagine creating a classifier that detects spam
- It's easy to catch 100% of spam
  - Throw out ALL the mail
  - Set "threshold" to 0 (or 1)
- It's easy to make no mistakes on good mail
  - Keep ALL the mail
- Perhaps a good starting point is to choose 50% threshold
  - Anything above this is a "positive hit"
  - Anything below this is a "negative rejection"

Precision $\frac{TP}{TP+FP}$ ↓

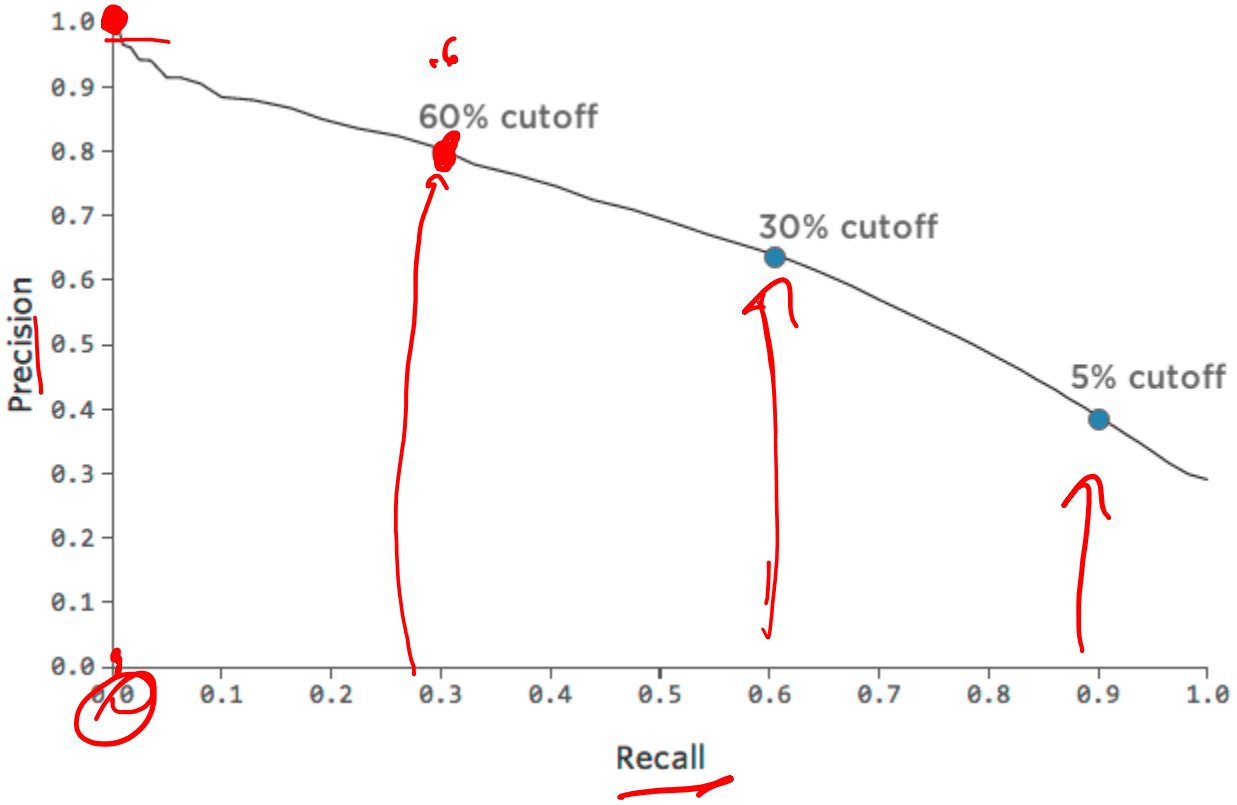
Recall $\frac{TP}{TP+FN}$ ↑

Precision/Recall Tradeoff

- We can explore the effect of this threshold on the precision and recall values.
- The plot of precision vs recall as a function of the threshold creates something called a *precision-recall* curve (PR)

Precision/Recall Curve

t [0, 0.1, 0.2 … 0.9, 1]

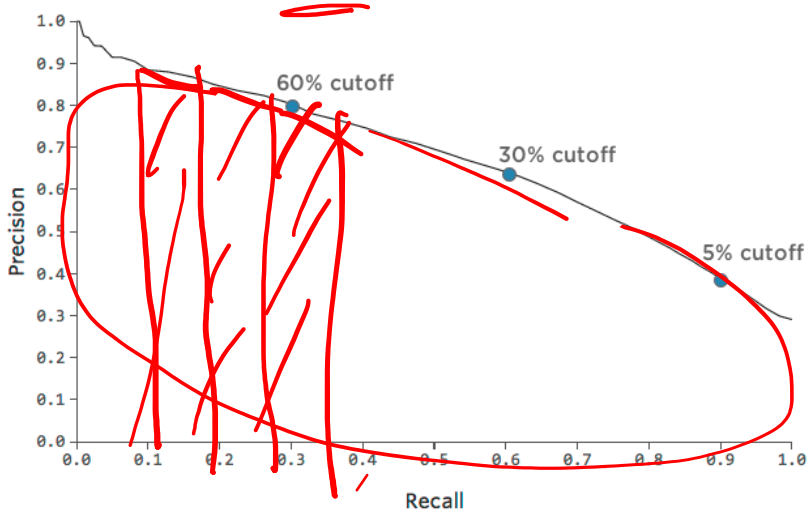


Precision/Recall Curve

- To evaluate a binary classifier, we can also compute the *area under the curve (AUC)* of a PR curve
- Given points on the curve, (R_k, P_k) we can approximate the AUC as:

$$AUC = 1 - \frac{1}{2}\sum_{k=1}^{n}(P_k + P_{k-1})(R_k - R_{k-1})$$

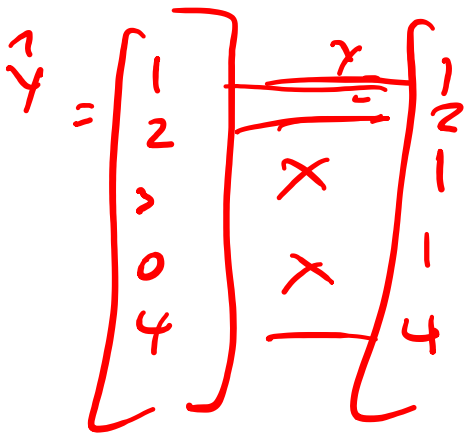
- An ideal PR curve will have an AUC of 1.0



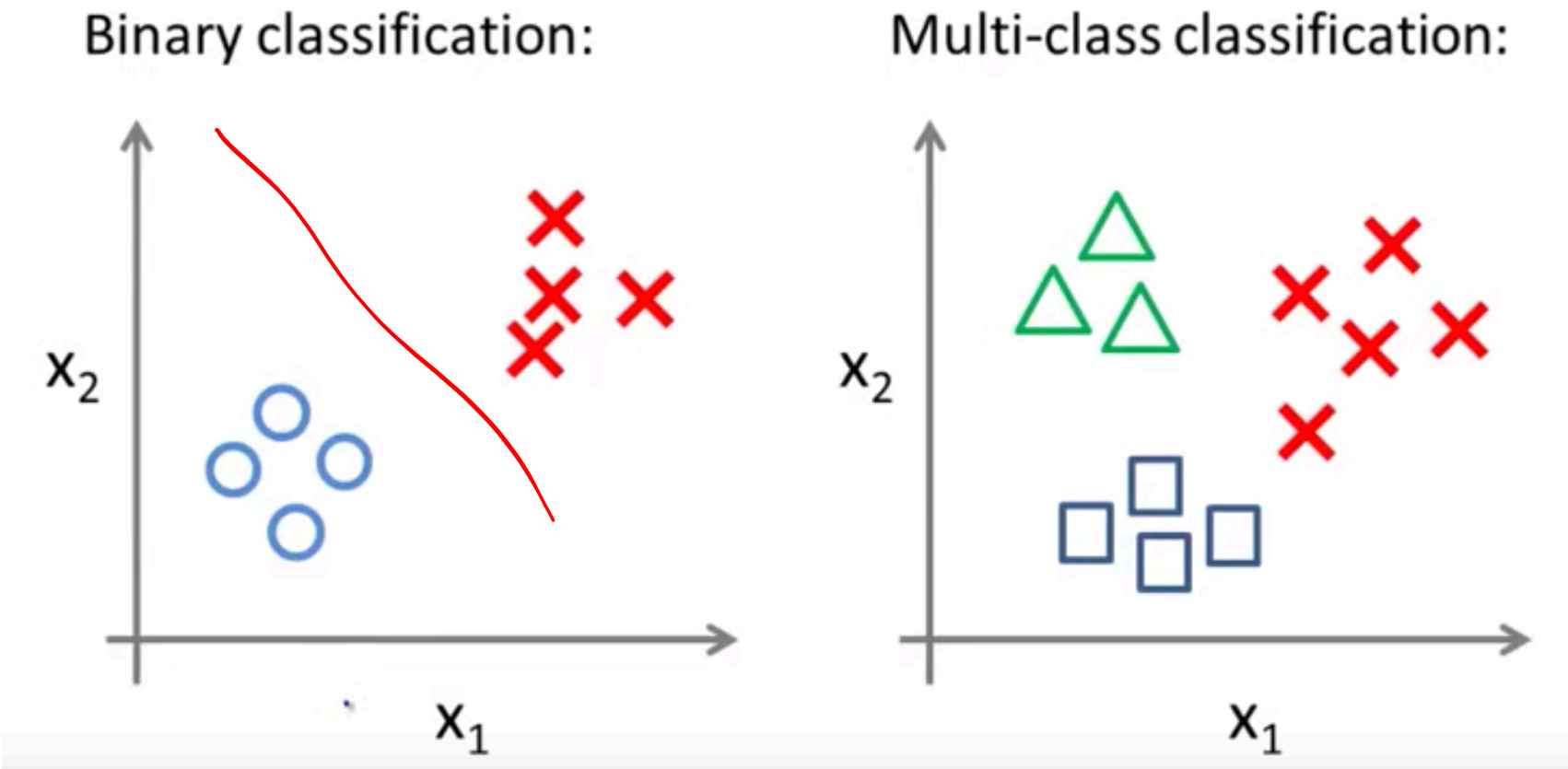
Multi-Class Evaluation

- Just like binary classification, we can evaluate the accuracy of a multi-class classifier:

$$accuracy = \frac{1}{N}\sum_{i=1}^{N}(Y_i = \hat{Y}_i)$$

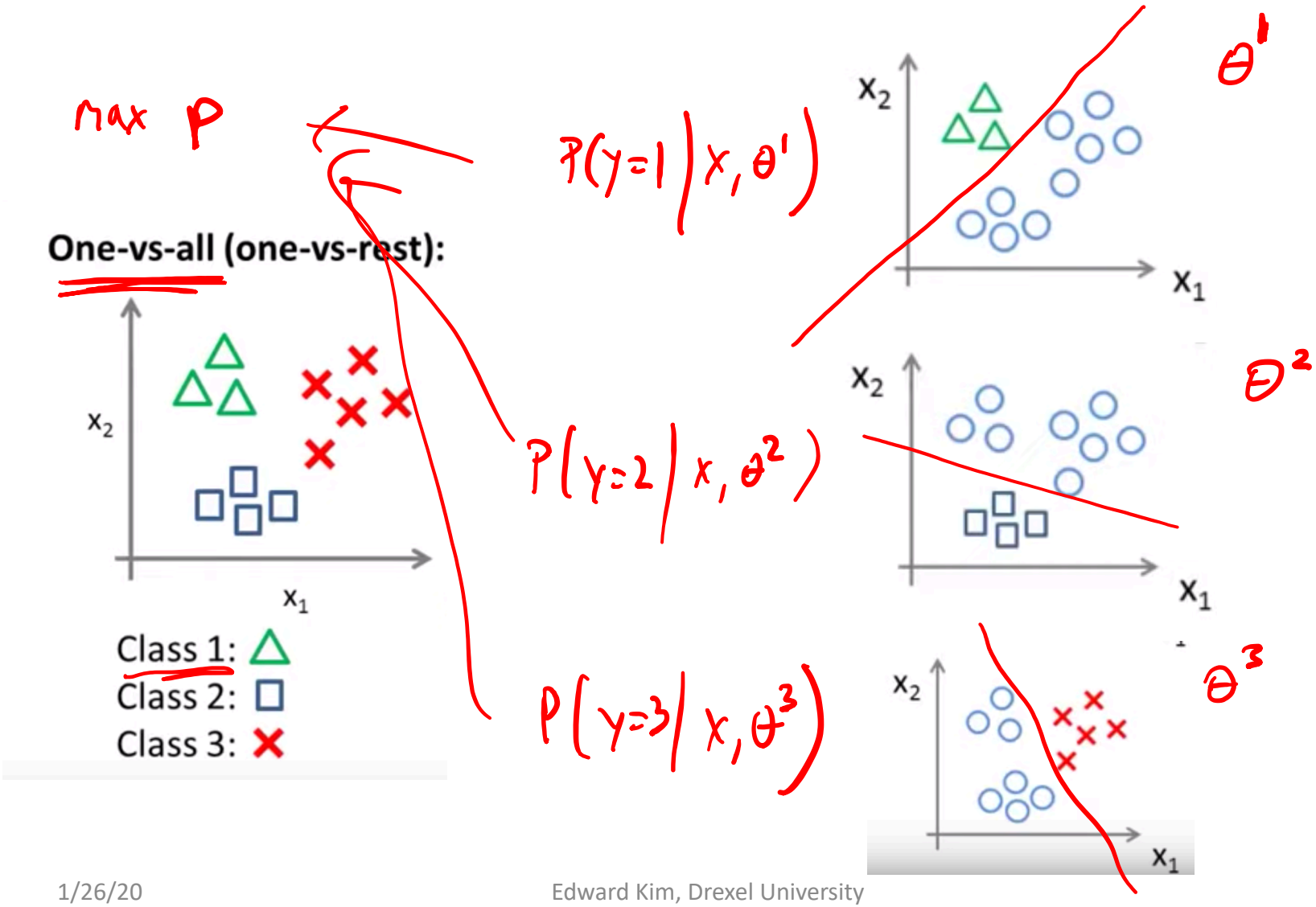


Multiclass classification



From Andrew Ng

Multiclass classification



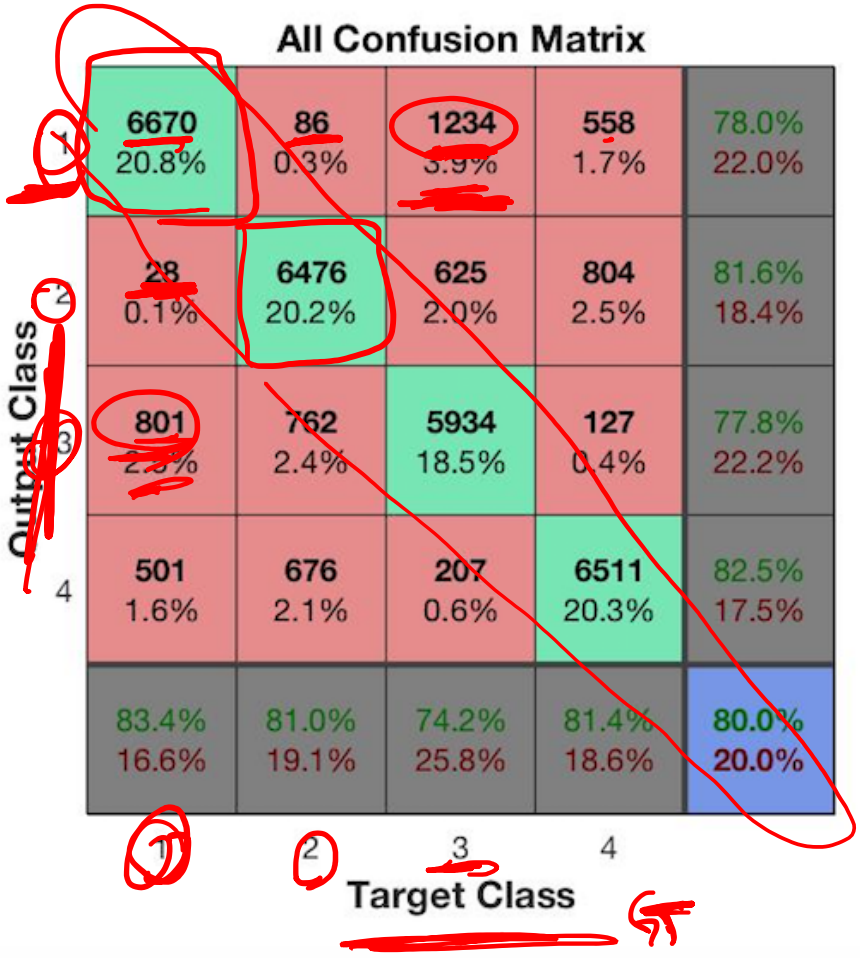
Multi-Class Evaluation

- Just like binary classification, we can evaluate the accuracy of a multi-class classifier:

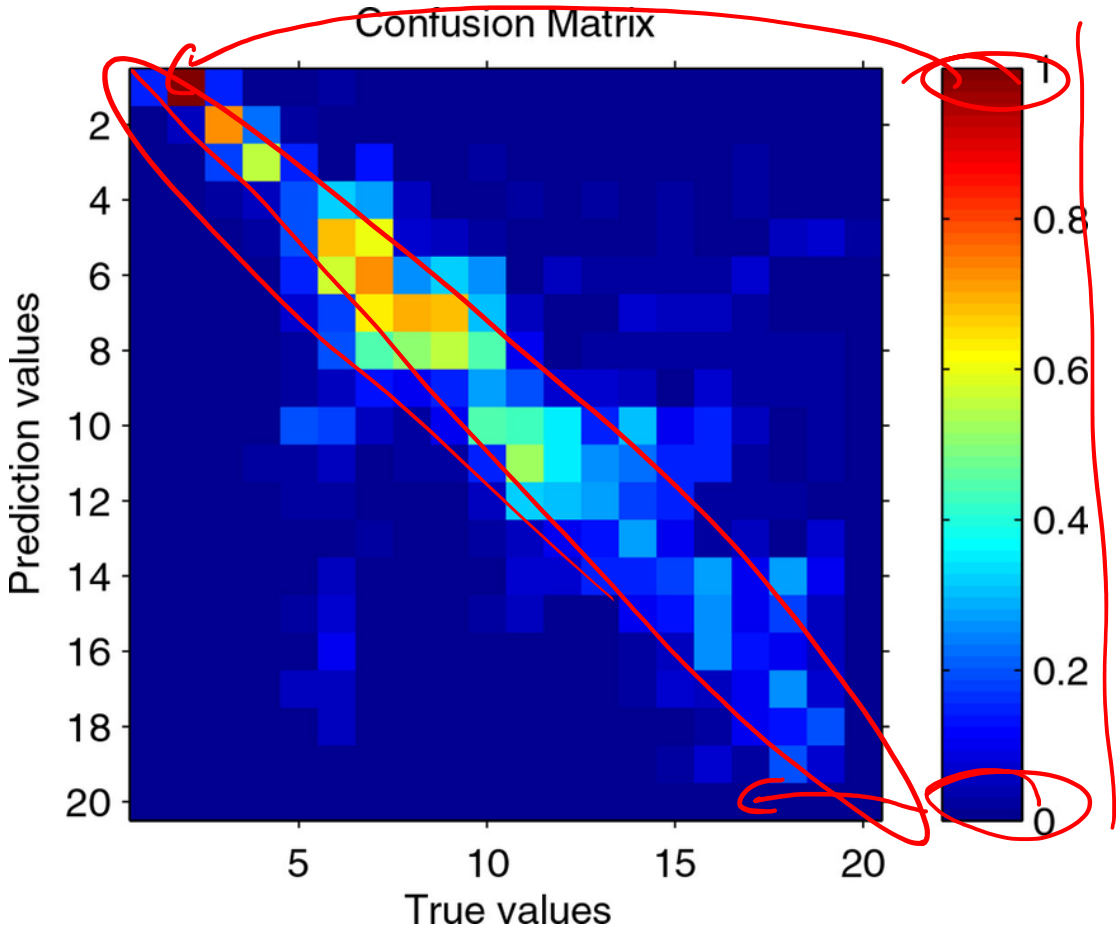
$$accuracy = \frac{1}{N}\sum_{i=1}^{N}\left(Y_i = \widehat{Y}_i\right)$$

- In addition, particular to multi-class classification, we may be interested in investigating which classes get confused with which other classes
- To observe this, we can look at a *confusion matrix*

Confusion Matrix



Confusion Matrix



Statistical Learning

Statistical Learning

- For another approach to classification we will look at the probability and statistics of our labeled data to make predictions on new data
- Review the Prob/Stats Week 0 slides!
- Hopefully these methods matches some of your intuition about data

Inference

training → θ's

testing → new unseen → inference

- Our statistical learning will start with the concept of *inference*
  - Given distribution of seen data, what can we *infer* about new data?
- Given evidence/features $x = [x_1, x_2, \dots, x_D]$ what is the *likelihood* that our object came from class i?
  - This is written as:
    $P(y = i | feature_1 = x_1, feature_2 = x_2, \dots, feature_D = x_D)$
  - We'll just abbreviate this as:

$$P(y = i | f_1 = x_1, f_2 = x_2, \dots, f_D = x_D) = P(y = i | f = x)$$

$P(y|x)$

- We call this value $P(y = i | f = x)$ that we're trying to compute, the *posterior*

$P(y = i)$ prior

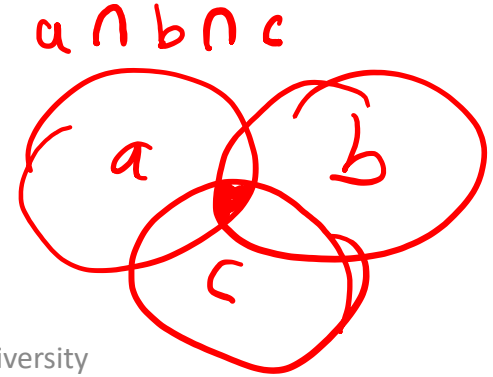
Inference

$$P(y = i | f_1 = x_1, f_2 = x_2, \ldots, f_D = x_D)$$

- Recall from probability that this is a *conditional probability:*

  "Given the first feature has value x_1, the second has x_2, etc.. what is the probability that our class was i?"

- Also recall that $P(a, b, c) = P(a \wedge b \wedge c)$ is called the *joint probability.*



Inference

- From the rules of probability

$$P(y|x) = \frac{P(y \wedge x)}{P(x)} = \frac{P(y,x)}{P(x)}$$

  - Here we call $P(x)$ the *evidence*.

- So we can solve this inference problem as:

$$P(y = i|f_1 = x_1, .., f_D = x_D) = \frac{P(y = i, f_1 = x_1, \dots, f_D = x_D)}{P(f_1 = x_1, \dots f_D = x_D)}$$

- Our final probability is defined purely in terms of the joints
  - And given enough data we be able get the joints easily directly from our data!

The Joint Distribution

$2^3 = 8$

- How to make a joint distribution:

1. Make a truth table listing all combinations of values of your variables (if there are M Boolean variables, then the table will have 2^M rows)
2. Count how many times in your data each combination occurs
3. Normalize those counts by the total data size in order to arrive a probabilities.

Note: The sum of joints must be equal to one

Learning a Joint Distribution

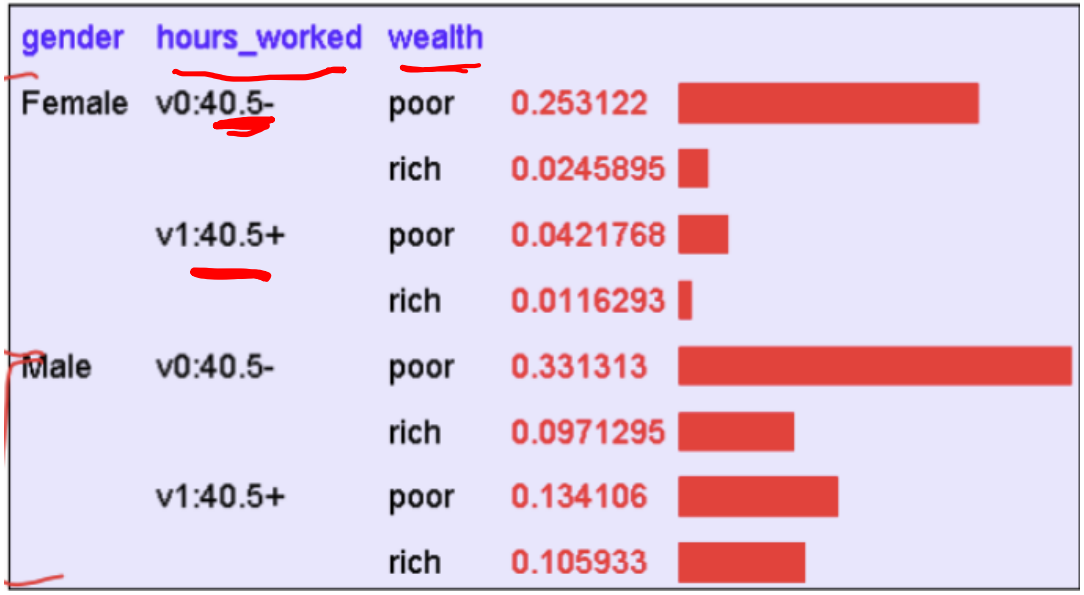
- To Build a JD (joint distribution) table for your attributes in which the probabilities are unspecified just fill in each row with

$$P(row) = \frac{records\ matching\ row}{total\ number\ of\ records}$$

A	B	C	Prob
0	0	0	0.30
0	0	1	0.05
0	1	0	0.10
0	1	1	0.05
1	0	0	0.05
1	0	1	0.10
1	1	0	0.25
1	1	1	0.10

Example

- This JD was obtained by learning from three attributes in the UCI “Adult” Census database



gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

Using the Joint

- Once you have the JD you can easily compute the probability of any logical expression involving your attributes
- Using the *law of total probability* we can compute $P(Y)$ as the sum of all probabilities jointed with Y:

$$P(Y) = \sum_i P(Y \cap x_i) = \sum_{rows\ with\ Y} P(row)$$

- Examples:
  - What is P(Poor)?
  - What is P(Poor Male)?

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

Inference with the Joint

- As mentioned, now we can also easily compute joint/conditional probabilities using our definition of the joint:

$$P(y|x) = \frac{P(y \wedge x)}{P(x)} = \frac{\sum_{rows\ with\ y\ and\ x} P(row)}{\sum_{rows\ with\ x} P(row)}$$

- What is $P(Male|Poor)$?

$$P(M|P) = \frac{P(M,P)}{P(P)}$$

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

$$\frac{0.33 + 0.13}{0.46 + 0.04 + 0.25} = \frac{0.46}{0.75}$$

$P(R|F,-) = \frac{P(R,F,-)}{P(F,-)}$ $\frac{0.024}{0.024+0.253} = 0.086$

Example

- Suppose we want to figure out given gender and hours worked, what is the wealth?
  - We can also write this as $P(W|G,H)$
- What is $P(W = rich|G = female, H = 40.5-)$?

gender	hours_worked	wealth	
Female	v0:40.5-	poor	0.253122
		rich	0.0245895
	v1:40.5+	poor	0.0421768
		rich	0.0116293
Male	v0:40.5-	poor	0.331313
		rich	0.0971295
	v1:40.5+	poor	0.134106
		rich	0.105933

Inference is a big deal!

- There's tons of times you use it:
  - I've got this evidence. What's the chance that my conclusion is true?
  - I've got a sore neck. How likely am I to have Meningitis?
  - The lights are out and it's 9pm. What is the likelihood that my spouse is asleep?

Using Inference for Classification

- How can we use this for classification?
- Consider x, a set of D features
- To figure out which class a set of features should belong to we can just choose the class that maximizes the posterior probability

$$\hat{y} = \underset{i}{argmax}\, P(y = i | f = x)$$

$$= \underset{i}{argmax} \left(\frac{P(y = i, f = x)}{P(f = x)} \right)$$

$\frac{P(dog, pixels)}{P(pixels)}$

$P(cat, pixels)$ / $P(pixels)$

$P(mouse | pixels)$ / $P(pixels)$

Using Inference for Classification

$$\hat{y} = \underset{i}{argmax}\left(\frac{P(y = i, f = x)}{P(f = x)}\right)$$

- But since $P(f = x)$ is the same for all classes we can just do:

$$\hat{y} = \underset{i}{argmax}\, P(y = i, f = x)$$

- And if we have $P(y = i, f = x)$ for all classes i then you can compute the actual probabilities, $P(y = i | f = x)$ by dividing by the sum of the joint probabilities:

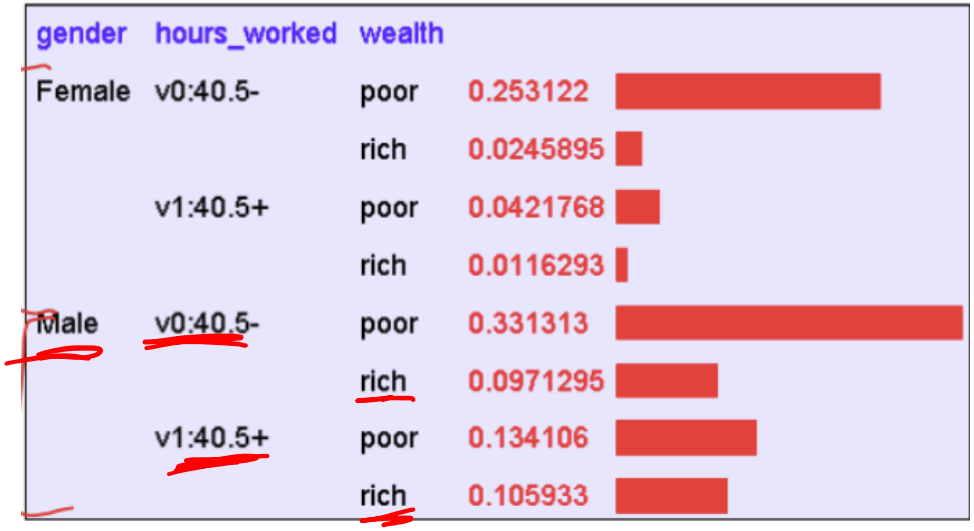
$$P(y = i|x) = \frac{P(y = i, f = x)}{\sum_j P(y = j, f = x)}$$

Inference for Classification Example

- Given a rich male let's classify them as having worked more or less than 40.5 hours per week
  - $P(40.5+|male, rich) \propto P(40.5+, male, rich)$
  - $P(40.5-|male, rich) \propto P(40.5-, male, rich)$

0.105

0.097



$$\frac{0.105}{0.105 + 0.097}$$

2^D 50 features

Naïve Bayesian Inference/Classification

Bayes Rule

- If you recall from probability, Bayes' provided another way to write the posterior $P(y = i|f = x)$:

$$P(y = i|f = x) = \frac{P(y = i)P(f = x|y = i)}{P(f = x)}$$

- In Bayes' Rule we call
  - $P(y = i|f = x)$ the posterior (what we want)
  - $P(y = i)$ the prior (probability that $y = i$)
  - $P(f = x|y = i)$ the likelihood (likelihood of generating x given y)
  - $P(f = x)$ the evidence

Bayes Rule

$$P(y = i|f = x) = \frac{P(y = i)P(f = x|y = i)}{P(f = x)}$$

- The prior is easy to come by if we have data
  - The prior $P(y = i)$ is just the percentage of samples that have their class $y = i$
- How about the likelihood $P(f = x|y = i)$?
- Obviously we can get it if we have a robust joint distribution table: $P(f = x|y = i) = \frac{P(f=x,y=i)}{P(y=i)}$
- But what if we don't have enough data for a robust joint distribution table?

Naïve Bayes Probability

- If we make the assumption that the features are *conditionally independent* (that is, that given one feature conditioned on our class we can't say anything about the other features), then we can re-write $P(f = x|y = i)$ as:

$$P(f = x|y = i) \approx \prod_{k=1}^{D} P(f_k = x_k|y = i)$$

- This is a large assumption, but one that is often made.
- So we can now approximate our posterior as:

$$P(y = i|f = x) = \frac{P(y = i)P(x|y = i)}{P(f = x)} \approx P(y = i)\prod_{k=1}^{D} P(f_k = x_k|y = i)$$

- We call this approach, *Naïve Bayes*

Naïve Bayes Probability

$$P(y = i|f = x) \approx P(y = i) \prod_{k=1}^{D} P(f_k = x_k|y = i)$$

- Note that this formula doesn't have the evidence $P(x)$ in it!
  - That's because since we're approximating the numerator based on the conditional independent assumption, the overall Bayes rule doesn't hold.
- Fortunately since we know that since the sum of the posteriors over all classes should be one, if we need an actual probability we can again just divide by the sum of this equation over all classes:

$$P(y = i|f = x) \approx \frac{P(y = i) \prod_{k=1}^{D} P(f_k = x_k|y = i)}{\sum_j P(y = j) \prod_{k=1}^{D} P(f_k = x_k|y = j)}$$

Example

- Let's try to determine if an object is a banana, an orange, or something else based on its length, sweetness, and color.
  - Where length, sweetness and color are all binary features (isLong?, isSweet?, isYellow?)
- Below is a table showing observations of 1000 pieces of fruit
  - For instance, there are 500 Bananas, and 400/500 of them are considered "Long"

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

Example

- How about with Naïve Bayes?

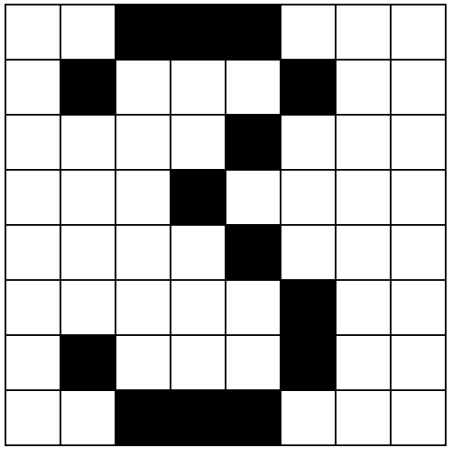
$$P(B|L, \neg S, Y) \approx P(B)P(L|B)P(\neg S|B)P(Y|B)$$

- But if I want an actual probability we also need to compute this for the Orange and Other class than divide by their sum…

Fruit	Long	Sweet	Yellow	Total
Banana	400	350	450	500
Orange	0	150	300	300
Other	100	150	50	200
Total	**500**	**650**	**800**	**1000**

Example – Digit classification

- Input: pixel grids

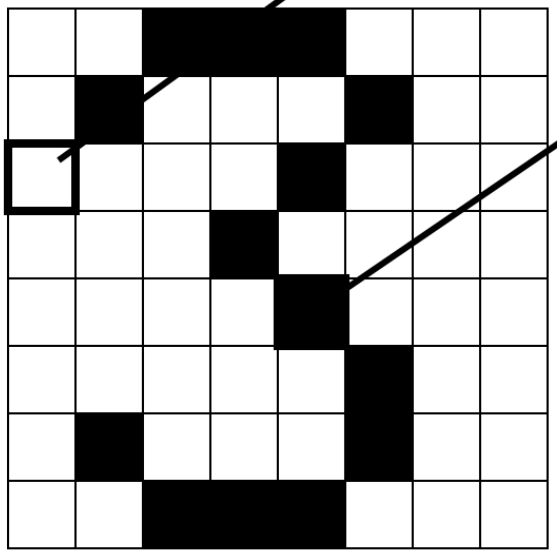


- Output: a digit 0-9

Digit Classification

$P(Y)$

1	0.1
2	0.1
3	0.1
4	0.1
5	0.1
6	0.1
7	0.1
8	0.1
9	0.1
0	0.1



$P(F_{3,1} = on|Y)$

1	0.01
2	0.05
3	0.05
4	0.30
5	0.80
6	0.90
7	0.05
8	0.60
9	0.50
0	0.80

$P(F_{5,5} = on|Y)$

1	0.05
2	0.01
3	0.90
4	0.80
5	0.90
6	0.90
7	0.25
8	0.85
9	0.60
0	0.80

Careful…

$P(\text{features}, C = 2)$

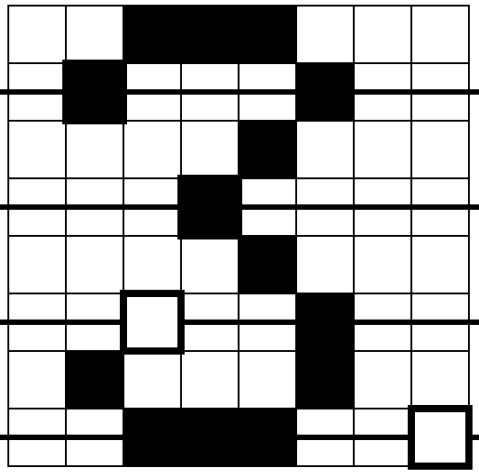
$P(C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.8$

$P(\text{on}|C = 2) = 0.1$

$P(\text{off}|C = 2) = 0.1$

$P(\text{on}|C = 2) = 0.01$



$P(\text{features}, C = 3)$

$P(C = 3) = 0.1$

$P(\text{on}|C = 3) = 0.8$

$P(\text{on}|C = 3) = 0.9$

$P(\text{off}|C = 3) = 0.7$

$P(\text{on}|C = 3) = 0.0$

2 wins!!

Inference vs Naïve Bayes

- Which should we use?
- Depends on what you have
  - Ideally use regular inference
  - If we have even less joint information and we can make an independence assumption then we can try naïve inference

Continuous Valued Data

Categorical vs Continuous Valued Data

- Each feature can typically fall into one of two categories:
  1. Categorical– The feature can have one of M possible values (categories, enumerations, finite possible values)
  2. Continuous – The features can have any value!
- In all our inference work we had to essentially count how many times something occurred in order to compute its probability
  - This is only feasible for categorical data
- What if our data is continuous?

What if we have continuous x_k?

- The general form of Naïve Bayes is:

$$P(y = i | f = x) \propto P(y = i) \prod_{j=1}^{D} P(f_k = x_k | y = i)$$

- If a feature is enumerated, then extract the individual probabilities $P(f_k = x_k | y = i)$ from the observed data.
- Alternatively maybe we can decide on an equation for $P(f_k | y = i)$!

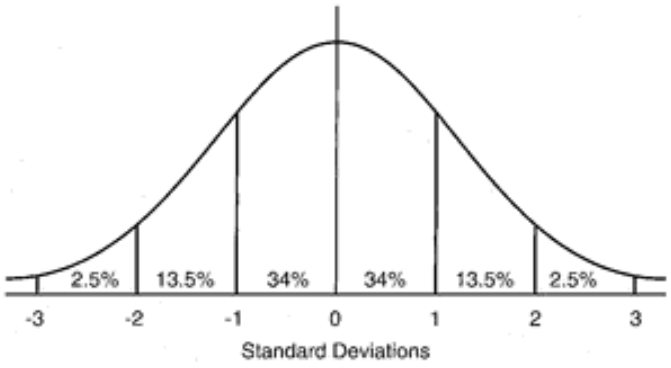
Gaussian Distributions

- If we decide that the feature is normally distributed (i.e a Gaussian, bell-shaped curve, etc..) then we can get something proportional to the probability $P(f_k = x_k | y = i)$ as:

$$P(f_k = x_k | y = i) \propto \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(x_k - \mu_i)^2}{2\sigma_i^2}}$$

  - Where μ_i is the expected or mean value of feature f_k for data from class $y = i$
  - And σ_i is the standard deviation of the feature f_k for data from class $y = i$

- Note: This equation is referred to as the in probability as the probability density function (pdf)



Gaussian Naïve Bayes: Continuous x, Discrete y

- How do we train a Naïve Bayes classifier with continuous features?
- For each discrete class C_i
  - First estimate the prior $P(y = i)$
  - For each attribute k, *estimate* $P(f_k = x_k | y = i)$ as $p(f_k = x_k | y = i)$by computing the attribute's mean and variance μ_{ik}, σ_{ik} from samples from that class C_i

Continuous Example

- Distinguish children from adults based on size
  - Classes $C = \{a, c\}$
  - Attributes: height[cm], weight[kg]
  - Training examples $\{h, w, y\}$, 4 adults, 12 children
- Class probabilities: $P(y = a)$?, $P(y = c)$?
- Model for adults:
  - Height ~Gaussian with
    - $\mu_{a,h} = \frac{1}{4}\sum_{i:y_i=a}(x_{i,h})$
    - $\sigma_{a,h}^2 = \frac{1}{4}\sum_{i:y_i=a}(x_{i,h} - \mu_{a,h})^2$
  - Weight ~ Gaussian $\mathcal{N}(\mu_{a,w}, \sigma_{a,w})$
- Model for children…
  - Height ~ $\mathcal{N}(\mu_{c,h}, \sigma_{c,h})$
  - Weight ~ $\mathcal{N}(\mu_{c,w}, \sigma_{c,w})$

Continuous Example

- Now given a test sample $x = (w, h)$ we want to compute $P(y = a|f = x)$ and $P(y = c|f = x)$
- To get our posteriors we want:
  - $P(y = a|f = x) = \frac{P(y=a)P(f = x|y = a)}{P(f=x)}$
  - $P(y = c|f = x) = \frac{P(y=c)P(f = x|y = c)}{P(f=x)}$
- Then if we make a naïve independence assumption we arrive at
  - $P(y = a|f = x) \propto P(y = a)P(f_1 = w|y = a)P(f_2 = h|y = a)$
  - $P(y = c|f = x) \propto P(y = c)P(f_1 = w|y = c)P(f_2 = h|y = c)$