CS4710: Artificial Intelligence
Intro to Machine Learning

Intelligent beings (e.g., humans) are excellent at learning. How can we try to program systems to learn interesting new things?

Supervised Learning: Regression

Intro to Regression

- Regression is a statistical method for learning and predicting classifications
- Works by minimizing a cost function over all data points
- Can be extended in several important ways
- Is a very good, general, supervised learning technique

Example: Predicting NBA Stats

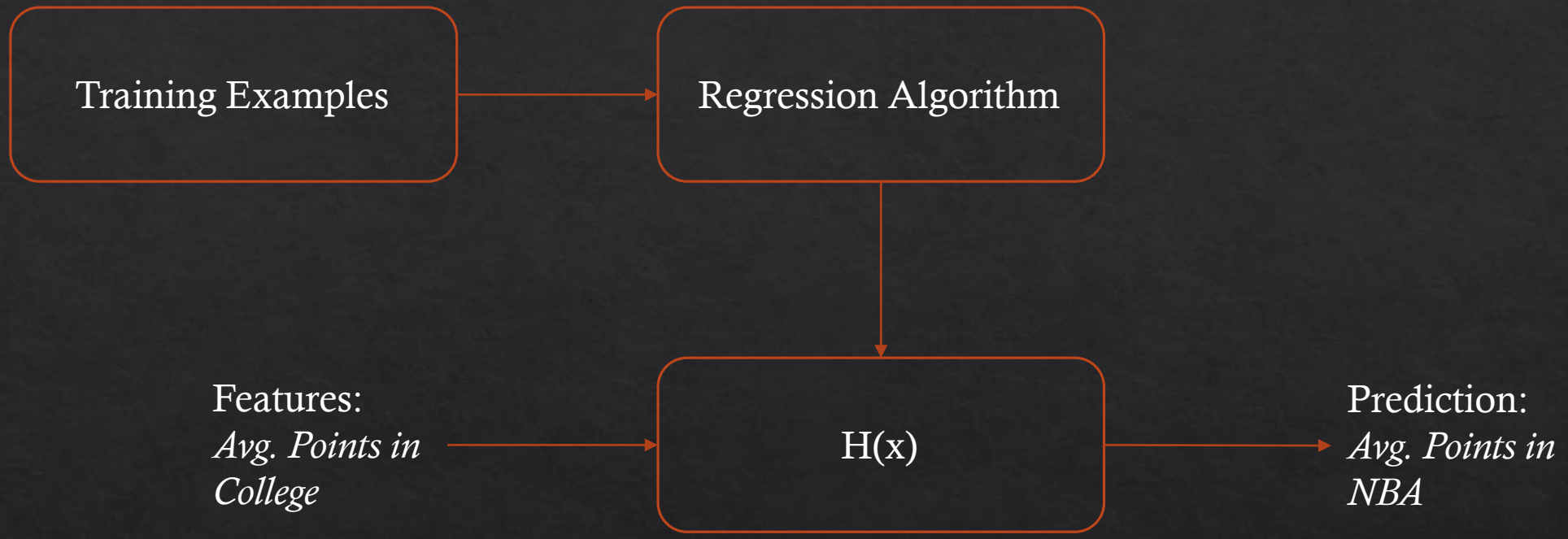
Suppose we know the number of points several NBA players scored in college and currently score in the NBA.

We want to predict how many points current college players will score in the NBA

** Yes, this model is too simplistic so far*

Name	Pts College	Pts in NBA
Joe Harris	12.6	2.5
Sean Singletary	16.9	2.4
Kevin Durant	25.8	25.4
Tony Bennett	19.4	3.5
Chris Paul	15.0	18.9
Anthony Davis	14.2	24.7

Regression Process



H(x) is called the **hypothesis function**

This inputs a data point and outputs a prediction output

Regression algorithm finds the best hypothesis function

Hypothesis Function for Linear Regression

$$H_\theta(x) = \theta_0 + \theta_1 * x$$

- X is the input feature (average points in college)
  - Soon, we will extend this to include multiple features
- θ_0 and θ_1 are real-valued weights
- Our regression algorithm will calculate the best theta values that predict our training set

Hypothesis Function

$$H_\theta(x) = \theta_0 + \theta_1 * x$$

- This is clearly a linear formula, but it doesn't have to be
- We can fit a quadratic by replacing x with x^2 or even something else
- Everything else we see today will be the same regardless, but as programmer, you must settle on a hypothesis function
- How to pick a hypothesis? Look at a plot of your data and see what shape it has

Hypothesis Function

$$H_\theta(x) = \theta_0 + \theta_1 * x$$

- Let's stick with a linear cost function for now (just for simplicity)
- We need to *learn* what the best values the two theta values are, so we need to define some objective function that measures how "good" our hypothesis is
- What makes a hypothesis good? If it does a good job of predicting all of the test data we were given!

Cost Function

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

- The cost function measures how well the hypothesis matches the training examples
- The i superscript denotes the ith training example (NOT raising anything to the ith power)
- The inside of the summation is the difference between the hypothesis prediction and the actual value. This difference is squared (both to give more weight to good guesses, and to get rid of negative values
- m is the number of training examples I was given

Gradient Descent Learning Algorithm

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

- **Goal:** Find theta values that minimize $J(\theta)$

- **Outline:**
  - Start with some arbitrary values of θ_0 and θ_1
  - Adjust theta values to reduce J(θ)
  - Repeat until no more adjustments can be made (or adjustments are doing VERY little)

Gradient Descent Learning Algorithm

- **Problem:** How do I know how to adjust each theta value so my overall cost function goes down?
- Remember, I want to adjust ALL of the theta values at once.
- So…it's not really an option to try different values and see what happens. If we have more features (which we will in a moment), then this will be way too slow.
- **Solution:** let's differentiate the cost function!

Gradient Descent Learning Algorithm

Repeat:

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

- α is called the **learning rate**, this is a value that you choose (higher means you'll change theta values faster.
- θ_j is the jth theta value from our hypothesis function (remember there were two). When we have more features, we will have more theta values to update.
- Partial derivative tells us which direction to change theta to lower the cost function

Gradient Descent Learning Algorithm

Let's differentiate J:

$$\frac{\partial}{\partial \theta_j} J(\theta) = ??$$

← On Board

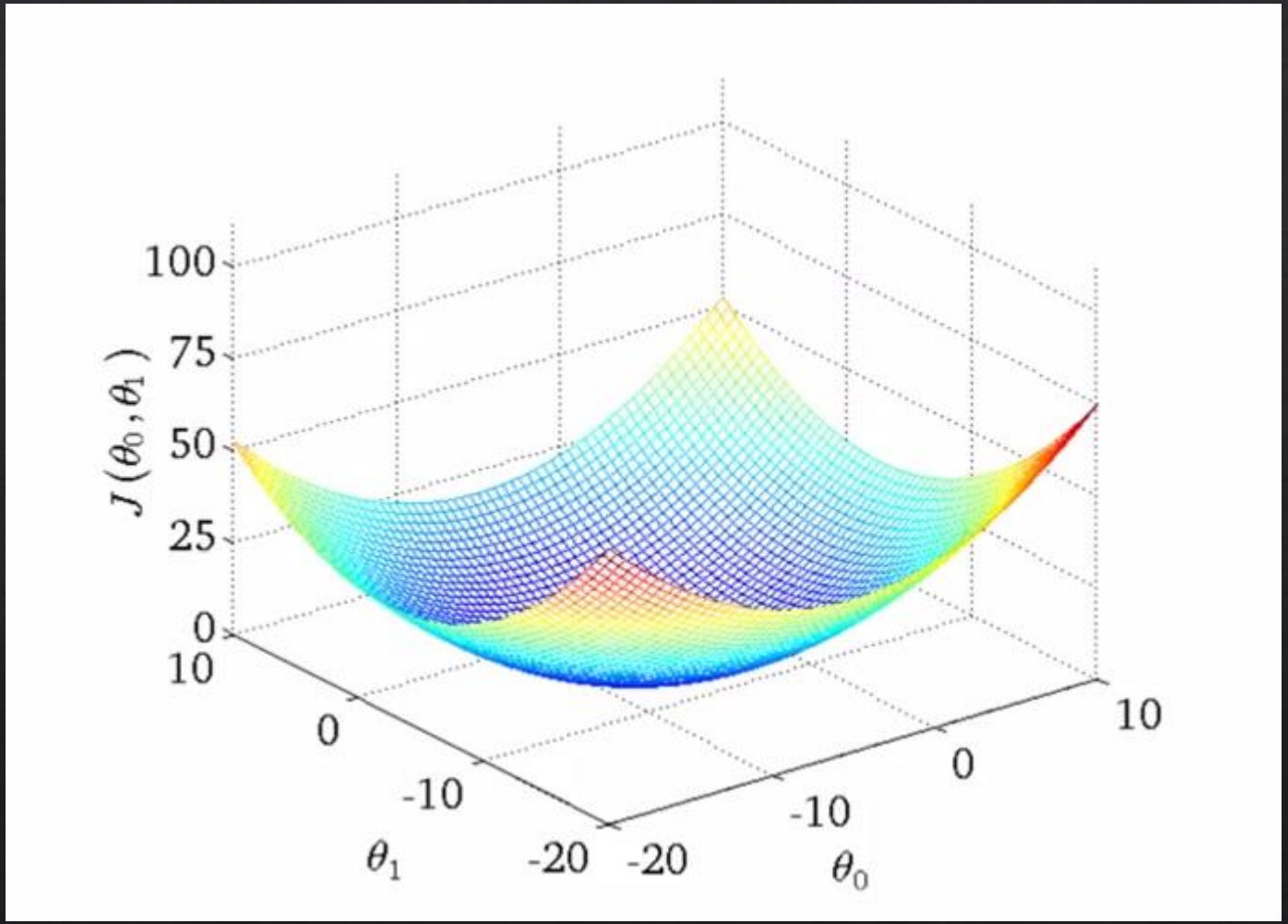
Gradient Descent Learning Algorithm

Let's differentiate J:

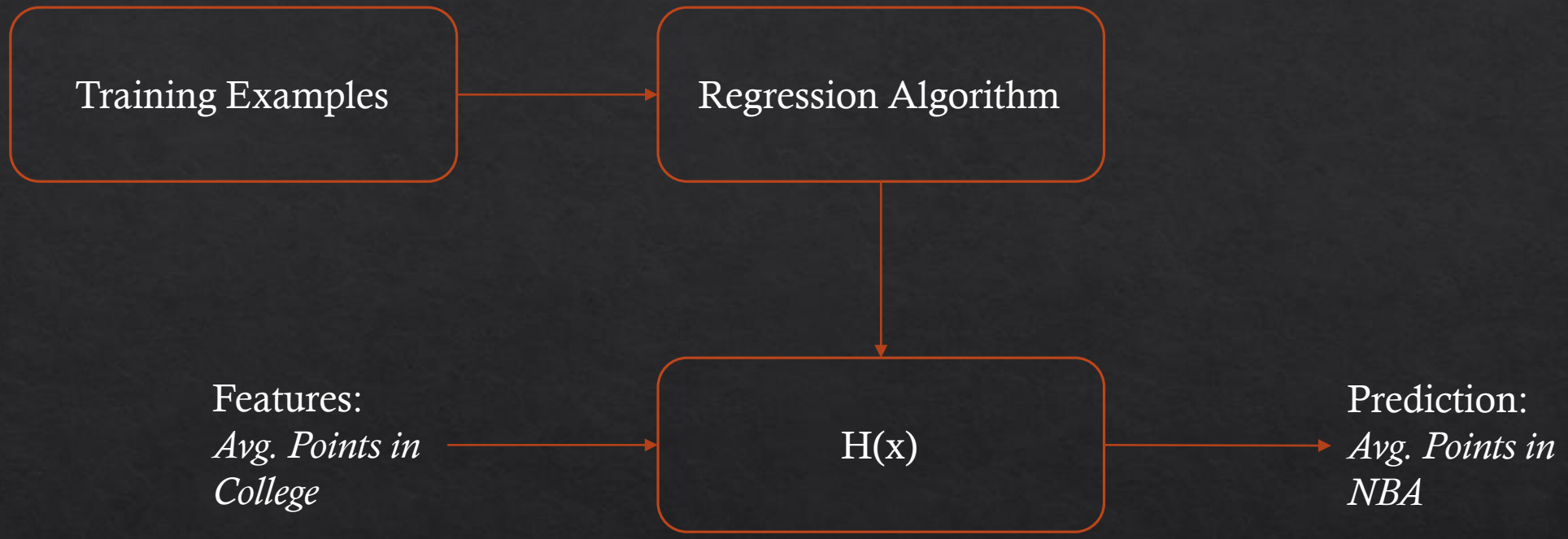
$$\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) * x^i$$

Gradient Descent: Local Minima?



Regression Process



H(x) is called the **hypothesis function**

This inputs a data point and outputs a prediction output

Regression algorithm finds the best hypothesis function

Supervised Learning: Regression

Multiple variables, polynomial regression, feature scaling, and other slight adjustments to linear regression

Example: Predicting NBA Stats

Name	Pts College	Steals/Game	FG%	3pt%	Pts in NBA
Joe Harris	12.6	0.9	.445	.407	2.5
Sean Singletary	16.9	1.6	.406	.362	2.4
Kevin Durant	25.8	1.9	.473	.404	25.4
Tony Bennett	19.4	1.4	.528	.497	3.5
Chris Paul	15.0	2.5	.472	.470	18.9
Anthony Davis	14.2	1.4	.623	.150	24.7

Our previous data set wasn't quite as reasonable. Don't we want more data?

Sure! So what if we had several features for each player, could we more accurately predict NBA performance? Probably!

Hypothesis Function: Multiple Features

$$H_\theta(x) = \theta_0 + \theta_1 * x_1 + \ldots + \theta_n * x_n$$

- X is now a vector of features of size (m x n)
- θ is a vector of weights of size m+1
- Like before, we can use polynomial terms if we'd like.

Hypothesis Function: Polynomial Terms

$$H_\theta(x) = \theta_0 + \theta_1 * x_1 + \ldots + \theta_n * x_n$$

- What is an easy way to deal with a feature that should be fit to a polynomial? Two options:
- 1. You can add $\theta_i * x_i^2 + \theta_{i'} * x_i$ to the cost function
- 2. You can scale every instance of that feature by that polynomial and then use these new values as a new feature

Cost Function

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

- Doesn't change!
- This is great!
- So, gradient descent still works as before.

Gradient Descent Learning Algorithm

Let's differentiate J:

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

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) * \boldsymbol{x_j}^{\boldsymbol{i}}$$

Supervised Learning: Regression

The normal equation

The Normal Equation

- Problem (ish):
  - Gradient descent takes many steps to converge
  - Might take a while
  - Might have to tweak alpha and other parameters to get good performance

- Our cost function is convex. So we should be able to differentiate it and set that equation to 0 and solve

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

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

…for every j

The Normal Equation

- Ok, so we are done. Just calculate this for all values of j:

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

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

…for every j

The Normal Equation

- Ok, how do we actually do this. Turns out there is a really easy way!

Construct a matrix X that comprises your features

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

And another called Y with the ground truth values

$$Y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

The Normal Equation

- Ok, how do we actually do this. Turns out there is a really easy way!

Construct a matrix X that comprises your features

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

And another called Y with the ground truth values

$$Y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Fortunately, the vector containing the theta values for your fitted hypothesis function can be computer directly by:

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

The Normal Equation

Fortunately, the vector containing the theta values for your fitted hypothesis function can be computer directly by:

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

You may need a matrix library (or something like matlab) to make this easy.

Gradient Descent vs. Normal Equation

- Works no matter what (logistic regression and other types)
- Is reasonably fast, guaranteed to converge
- Need to choose alpha (often is slow and needs to be adjusted a couple times
- Needs to iterate, which often isn't instantaneous

- Great for the regressions we've seen so far
- Directly computes minimum theta values for our hypothesis function
- Easy to implement
- Doesn't work for all regression techniques
- VERY slow if n gets large (but admittedly, n has to be quite big…10^5 or so)

Logistic Regression

Linear Regression vs. Logistic Regression

- Typically used when the ground truth contains ***continuous values***

- Used for ***classification problems*** where the ground truth takes the form of ***discrete values***

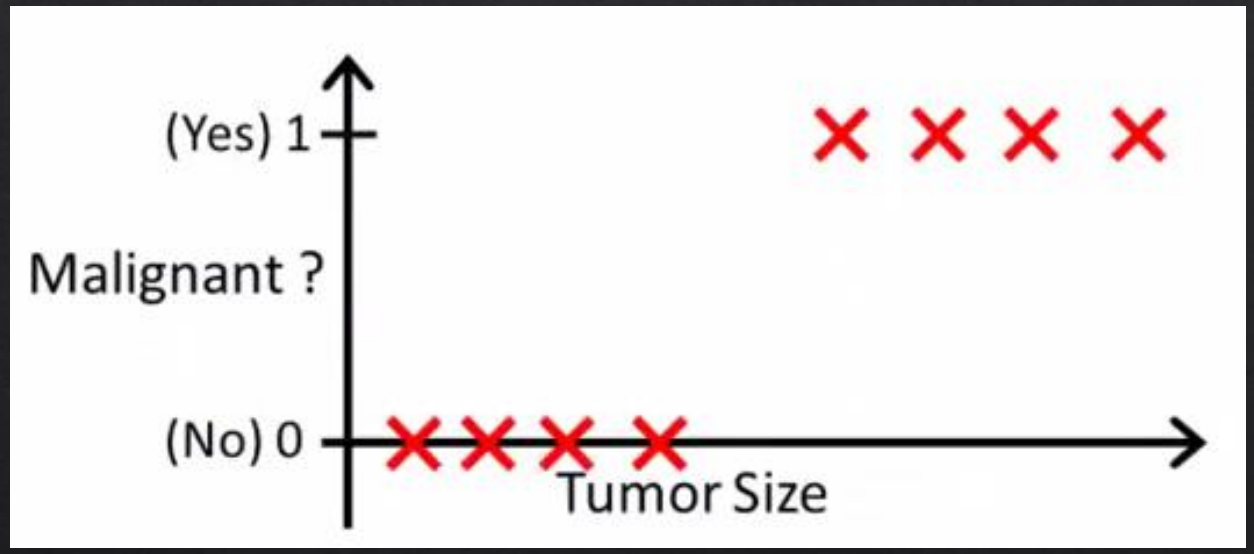
- Very widely used in practice

- Examples:
  - Email spam/not spam
  - Tumor malignant or benign
  - Etc…

Logistic Regression

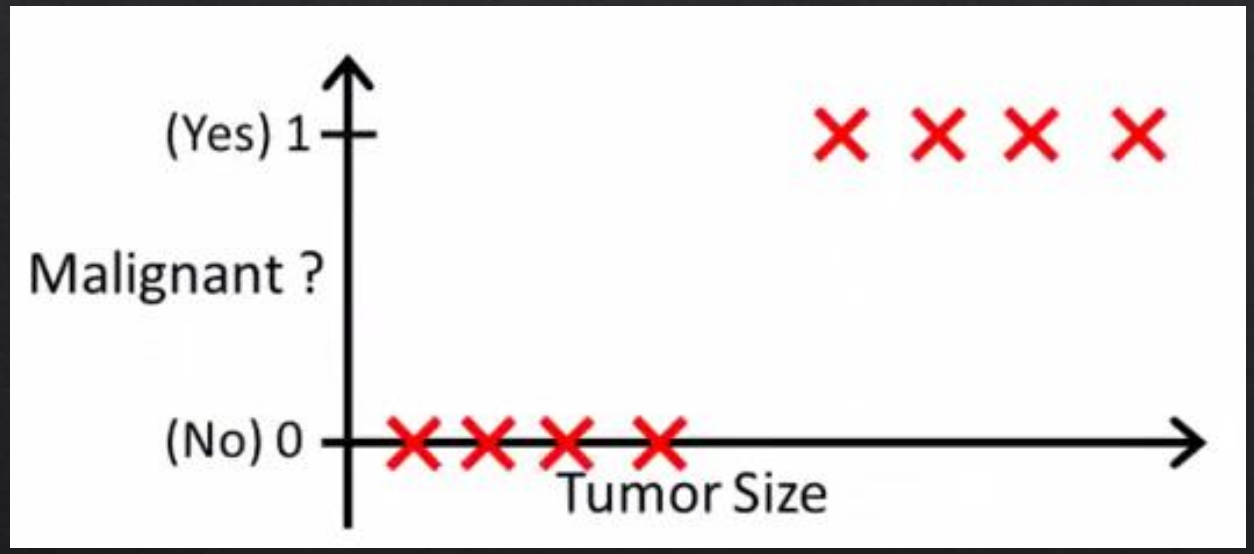
- Training set ***x*** contains ***n features***
  - As before, these features can be anything but should be numeric
- Output variable is one of two classifications
  - $y \in \{0, 1\}$
  - 0 is one class (e.g., email not spam)
  - 1 is other class (e.g., email is spam)
- We will extend this to more than two classifications soon…

Logistic Regression: Example



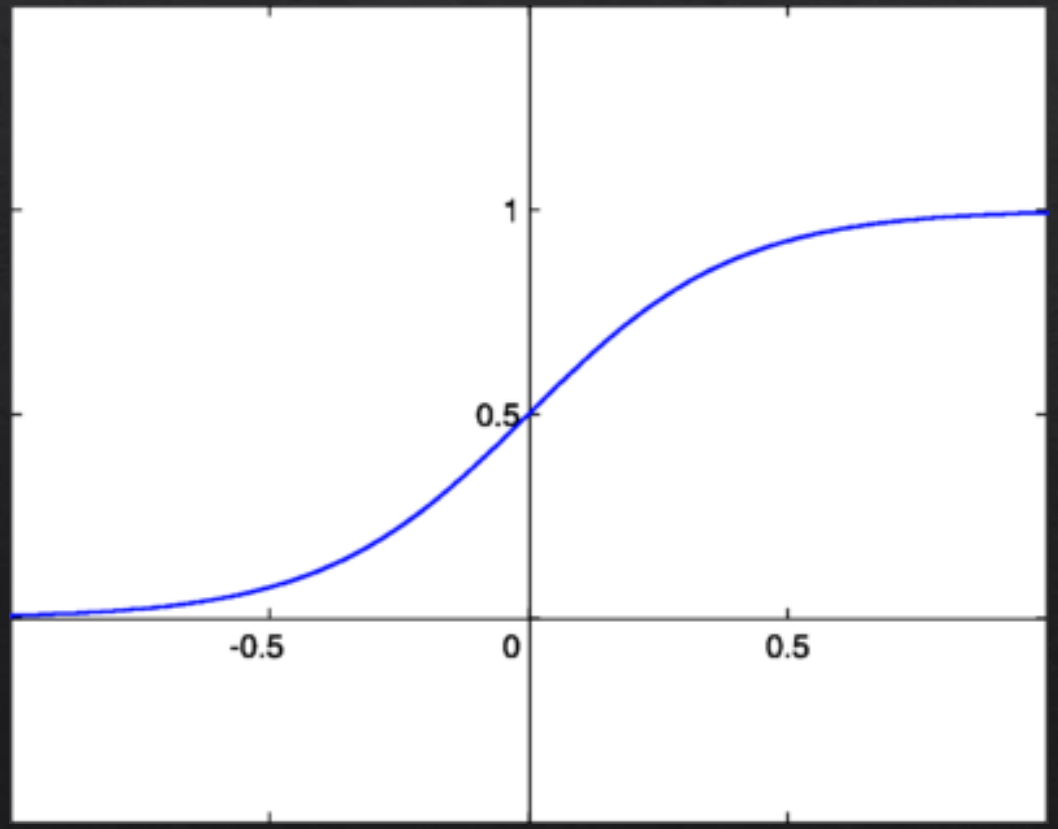
- Training set ***x*** contains ***n features***
  - As before, these features can be anything but should be numeric
- Output variable is one of two classifications
  - $y \in \{0, 1\}$
  - 0 is one class (e.g., tumor NOT malignant)
  - 1 is other class (e.g., tumor is malignant)
- We will extend this to more than two classifications soon…

Logistic Regression: Example



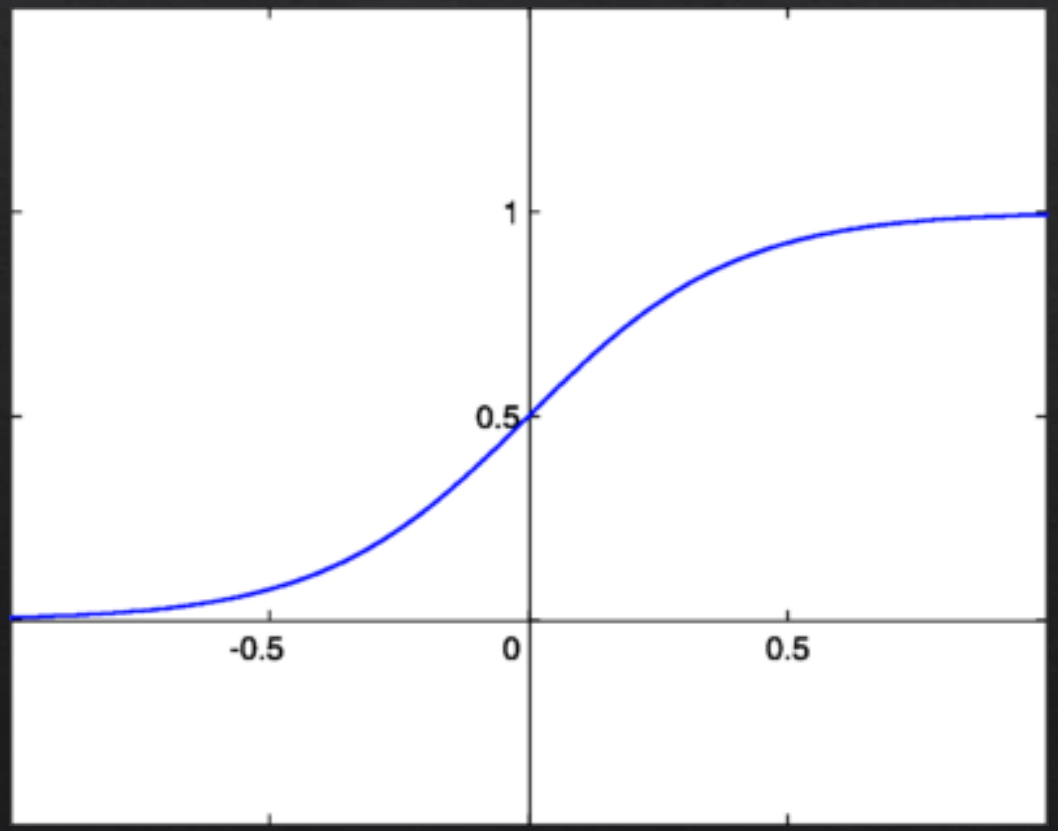
- As before, we want to fit a curve to this data
- However, a linear fit doesn't make any sense, so what curve can we fit here?
- What might happen if we do fit a line to this data? What if there are some outliers?
- *…hint: it is one of our old friends!*

The Sigmoid Function!



- $Y = \frac{1}{1+e^{-x}}$
- So, logistic regression is basically linear regression but we are fitting to a sigmoid instead!

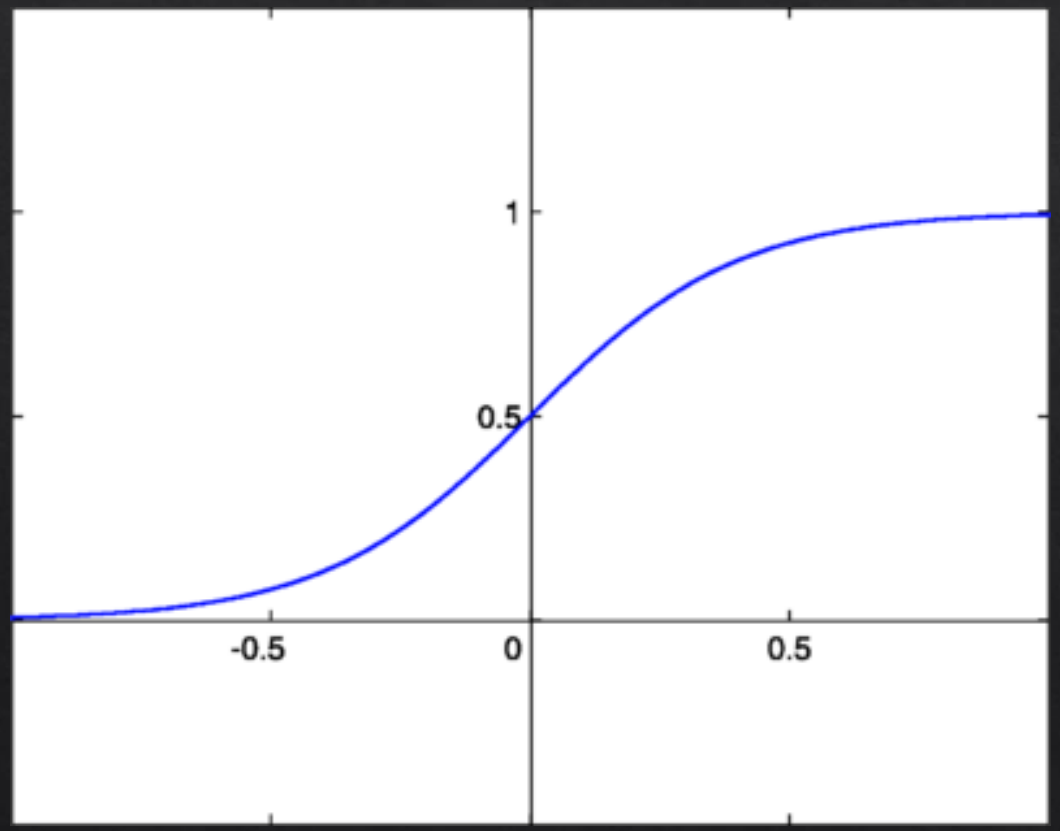
Interpreting the Sigmoid Function!



$$Y = \frac{1}{1 + e^{-x}}$$

- Assume I've fit this curve to my data and I'm given a new instance ***z***
- Plugging the features of ***z*** into my sigmoid will give me ***the probability that z should be classified as a 1***
- I can use that information however I wish

Logistic Regression: Hypothesis Function



- Linear regression was:

$$H_\theta(x) = \theta_0 + \theta_1 * x_1 + \ldots + \theta_m * x_m$$

Which can be written as: $\theta^T x$

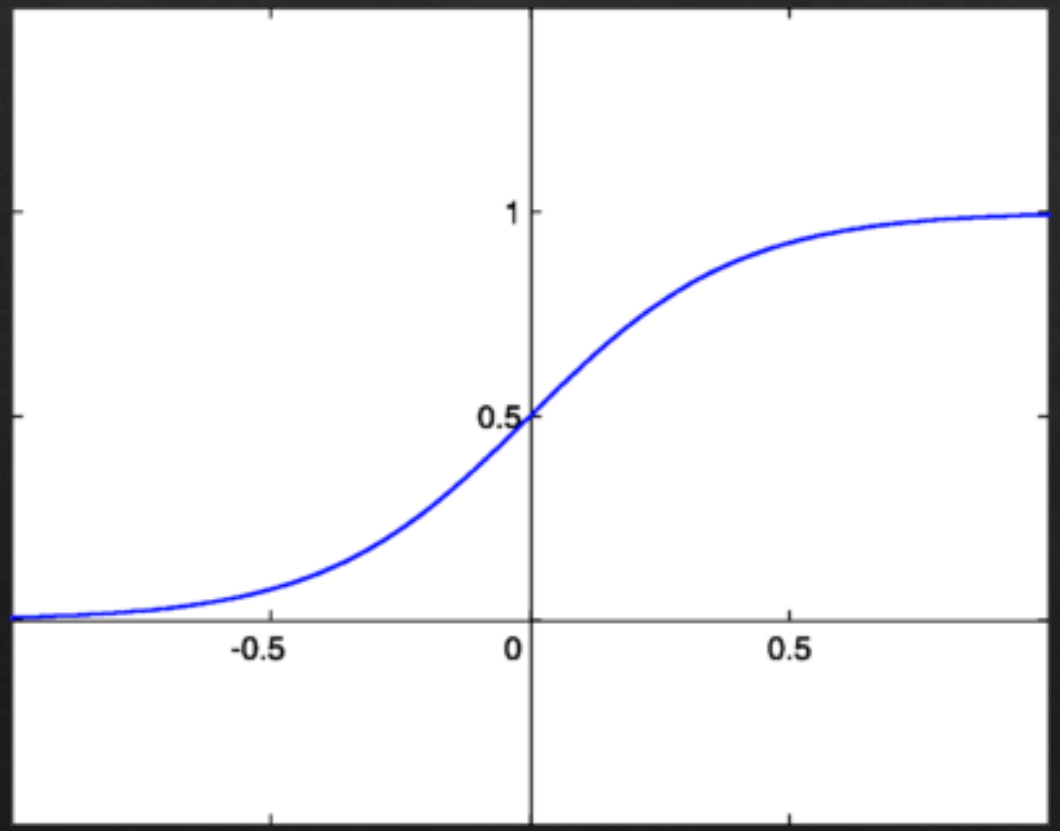
- Now we will use:

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

- Where:

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

Logistic Regression: Hypothesis Function



- Now we will use:

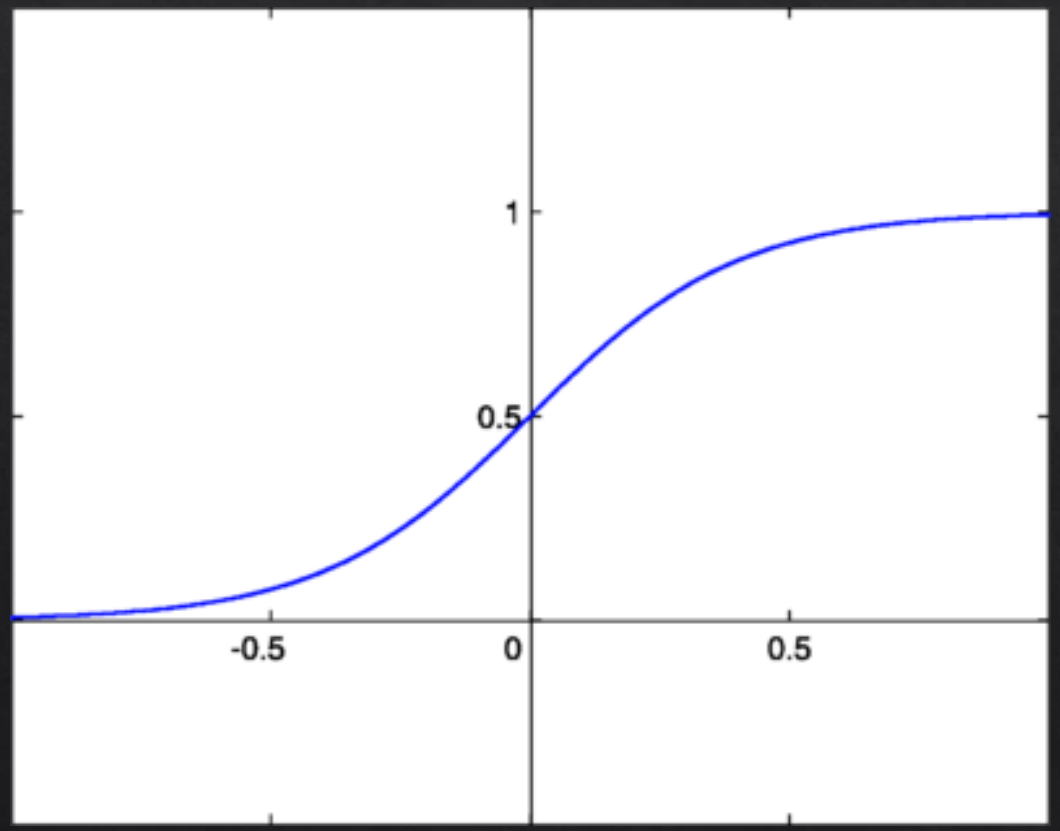
$$H_\theta(x) = g(\theta^T x)$$

- Where:

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

What should I do if I plug a new example into my hypothesis and get 0.6?

Logistic Regression: Decision Boundary



- Basic Idea: Figure out where cutoff should be between predicting a classification of 0 versus a classification of 1

Very Basic Version:

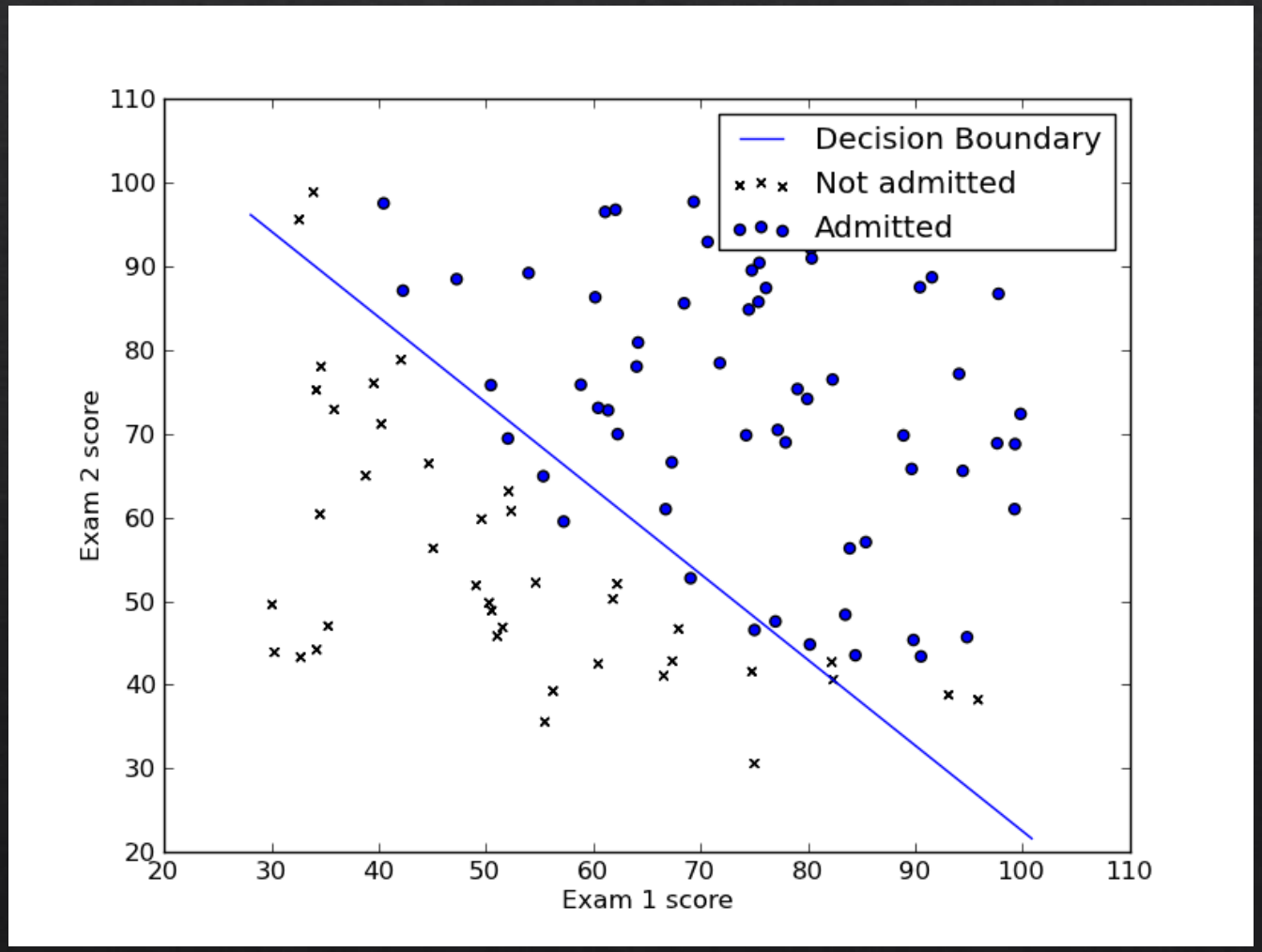
If $h_\theta(x) \geq 0.5$*:*

Output 1

Else If $h_\theta(x) < 0.5$*:*

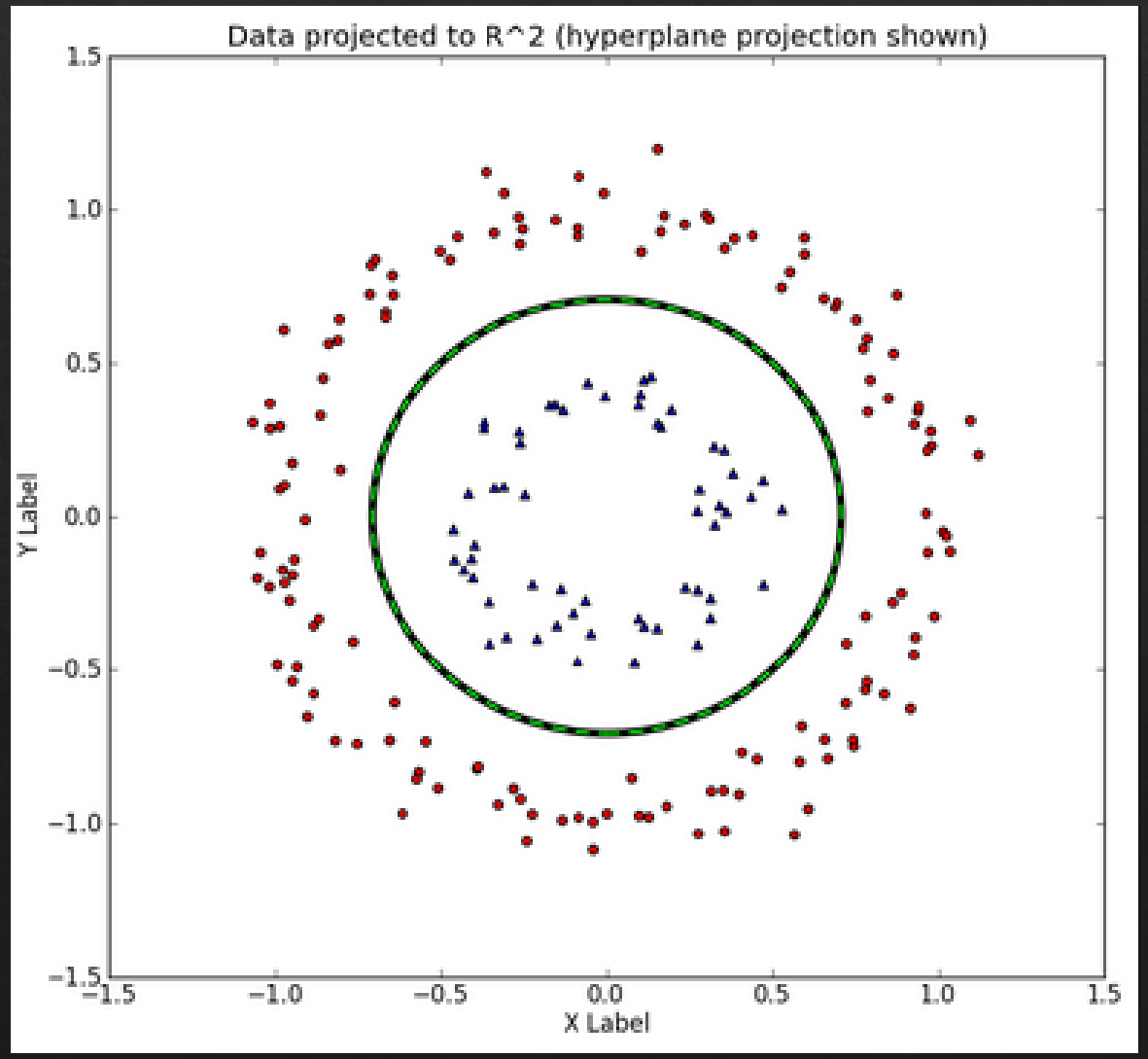
Output 0

Logistic Regression: Decision Boundary



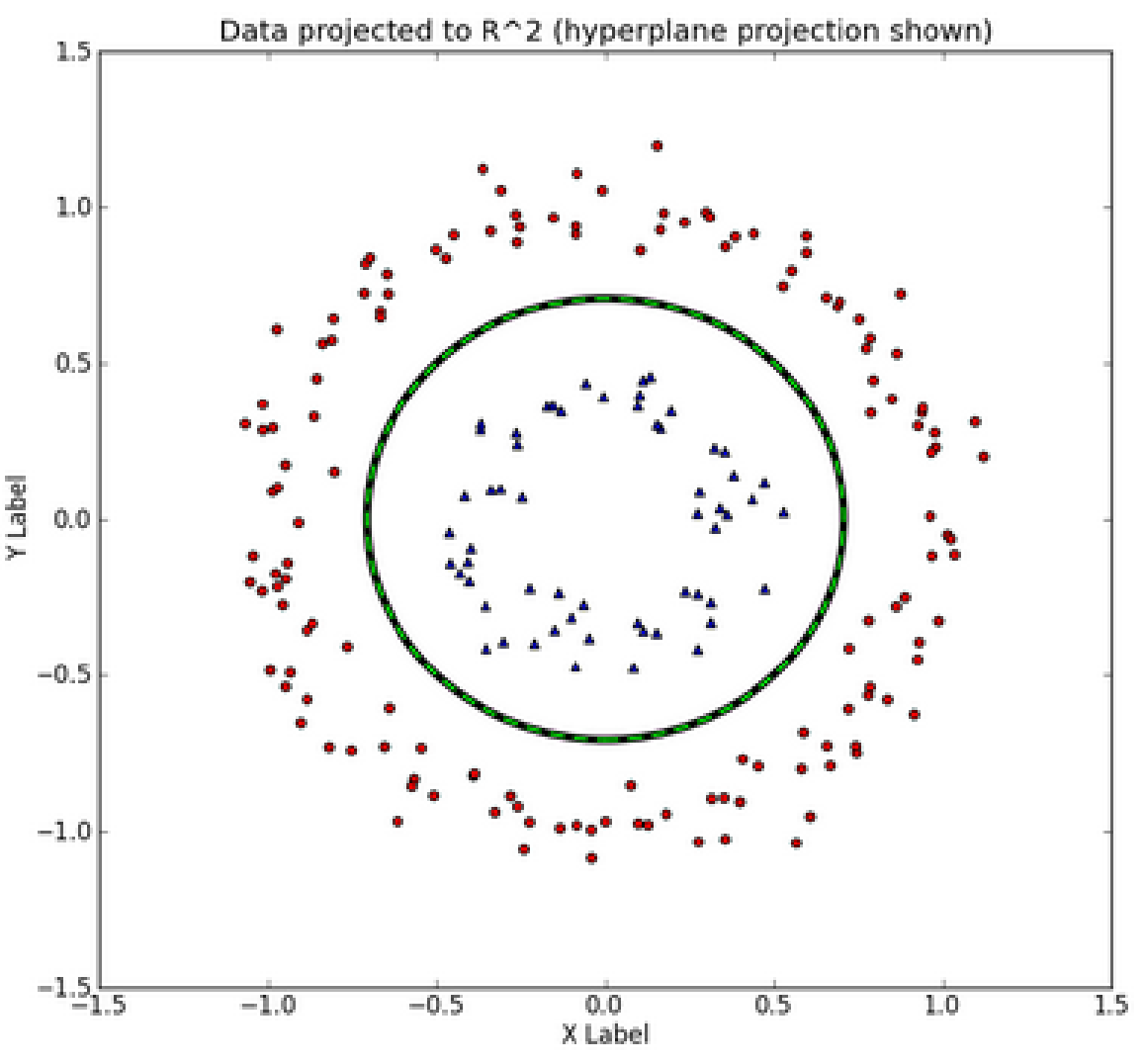
- Because we are passing a linear function into our sigmoid, our decision boundary looks like this
- Assumes two features were used
- So an input that falls directly on this line will result in exactly 0.5 being output from the sigmoid!
- Intuitive, the line separates the two classifications

Logistic Regression: Decision Boundary



- But what if our data points look like this!?
- No line does a good job of splitting these classifications!
- Need a new function to use as input to our sigmoid!!

Logistic Regression: Decision Boundary

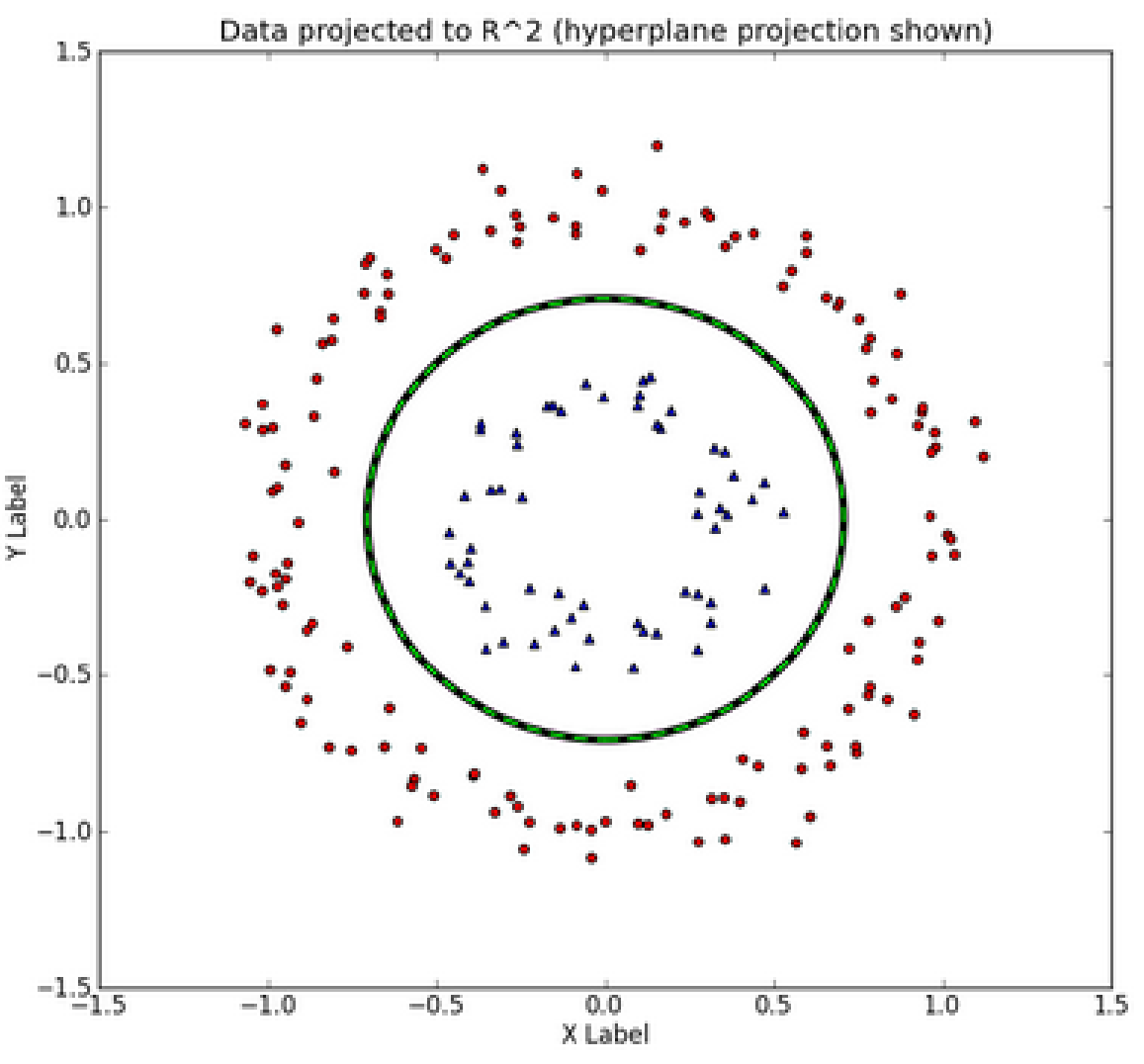


Instead use:

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

Remember that g() is still the sigmoid function

Logistic Regression: Decision Boundary



Instead use:

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

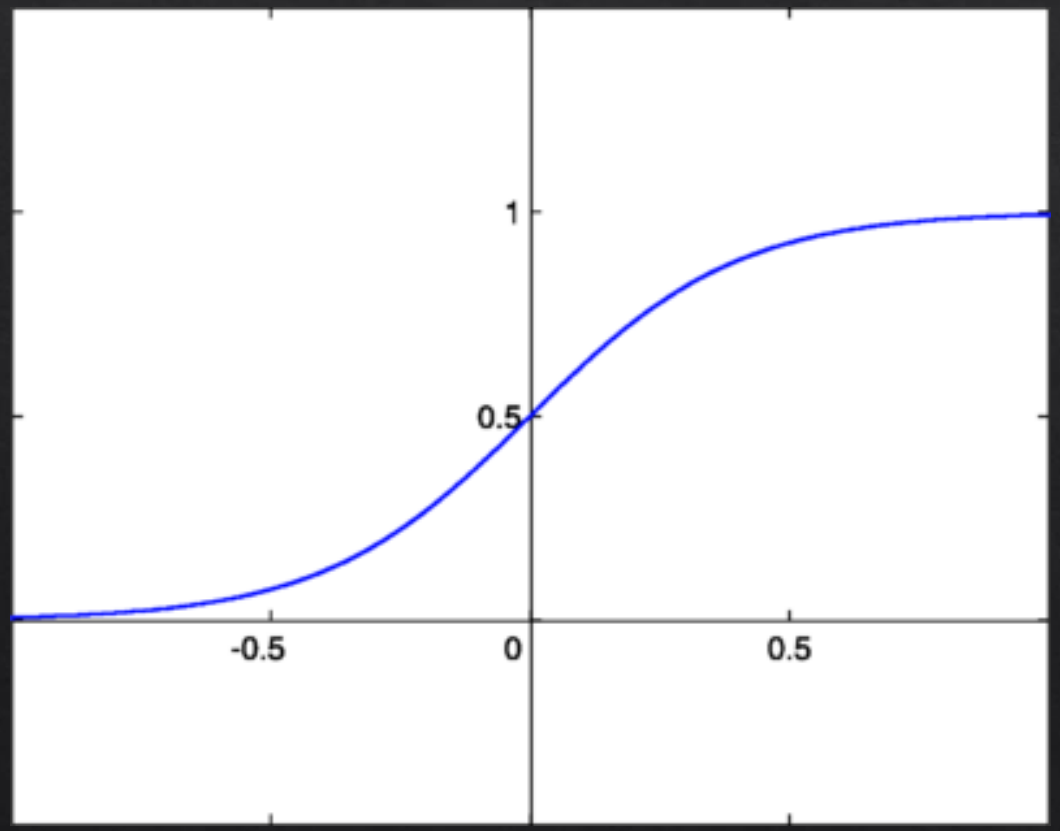
Suppose we learn the following theta vals:

$$-1 + {x_1}^2 + {x_2}^2$$

So $\theta = [-1, 0, 0, 1, 1]$

This is the formula for a circle! Yay!

Logistic Regression: Cost Function

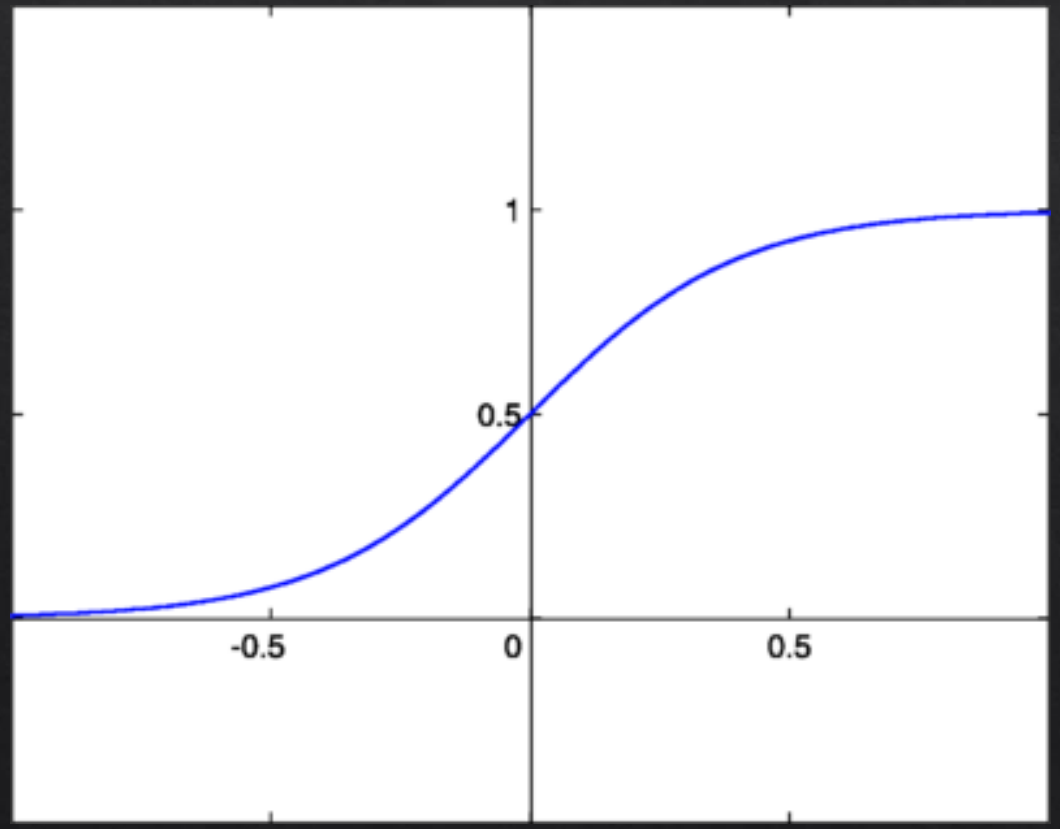


- Need a cost function that measures how well our current theta values predict the training data…just like before:

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

What is the problem with this? Can we use this and just do gradient descent again?

Logistic Regression: Cost Function



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

What is the problem with this? Can we use this and just do gradient descent again?

Because h(x) is a sigmoid, this cost function is not convex, so we aren't guaranteed to find global minimum

Logistic Regression: Better Cost Function

Instead we use:

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

Why does this make sense?

Let's plot it!

Logistic Regression: Better Cost Function

We can write this function:

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

Like this instead:

$$Cost(h_\theta(x), y) = -y * \log\big(h_\theta(x)\big) - (1 - y) * \log(1 - h_\theta(x))$$

These are equivalent. The y or (1-y) part will cancel out one of the two terms but keep the other.

Logistic Regression: Better Cost Function

Phew! So our final cost function for a single training example is:

$$Cost(h_\theta(x), y) = -y * \log\big(h_\theta(x)\big) - (1 - y) * \log\big(1 - h_\theta(x)\big)$$

And thus, for ALL training examples is:

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

Logistic Regression: Gradient Descent

So let's find the values of θ that minimizes this cost function:

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

Reminder:

Loop{

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

//So just need to differentiate J and we are done!

}

Logistic Regression: Gradient Descent

Reminder:

Loop{

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

//So just need to differentiate J and we are done!

}

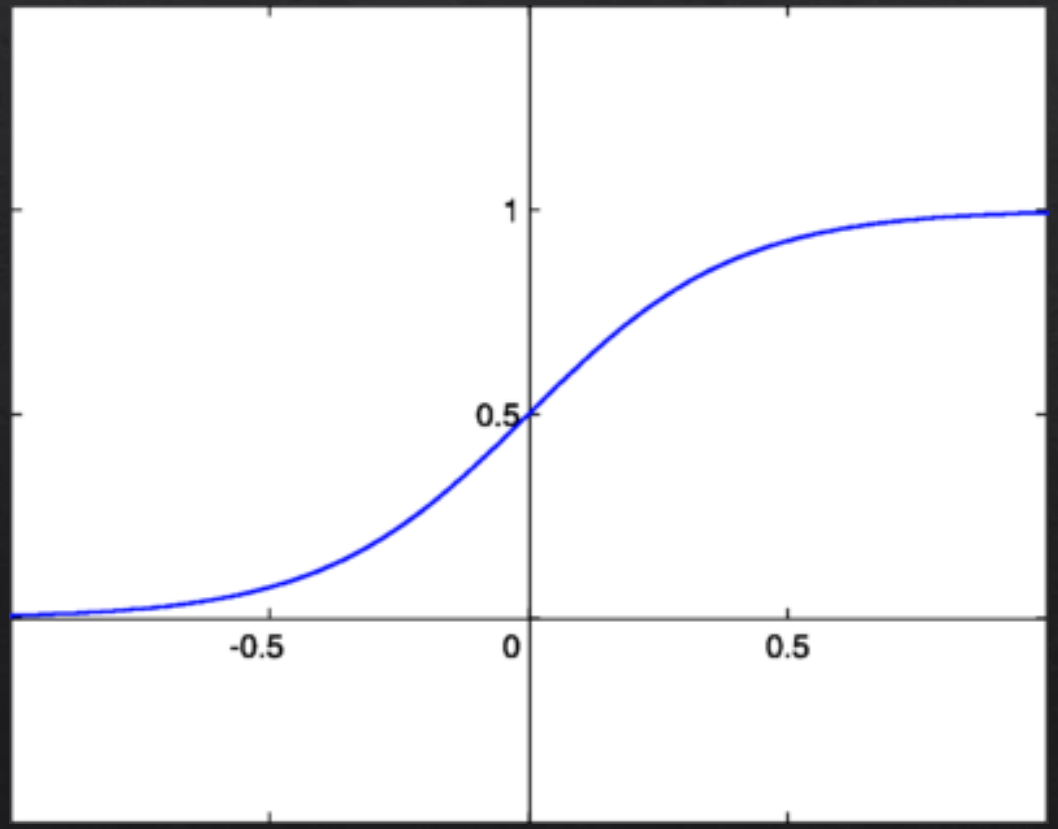
Where:

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

Logistic Regression

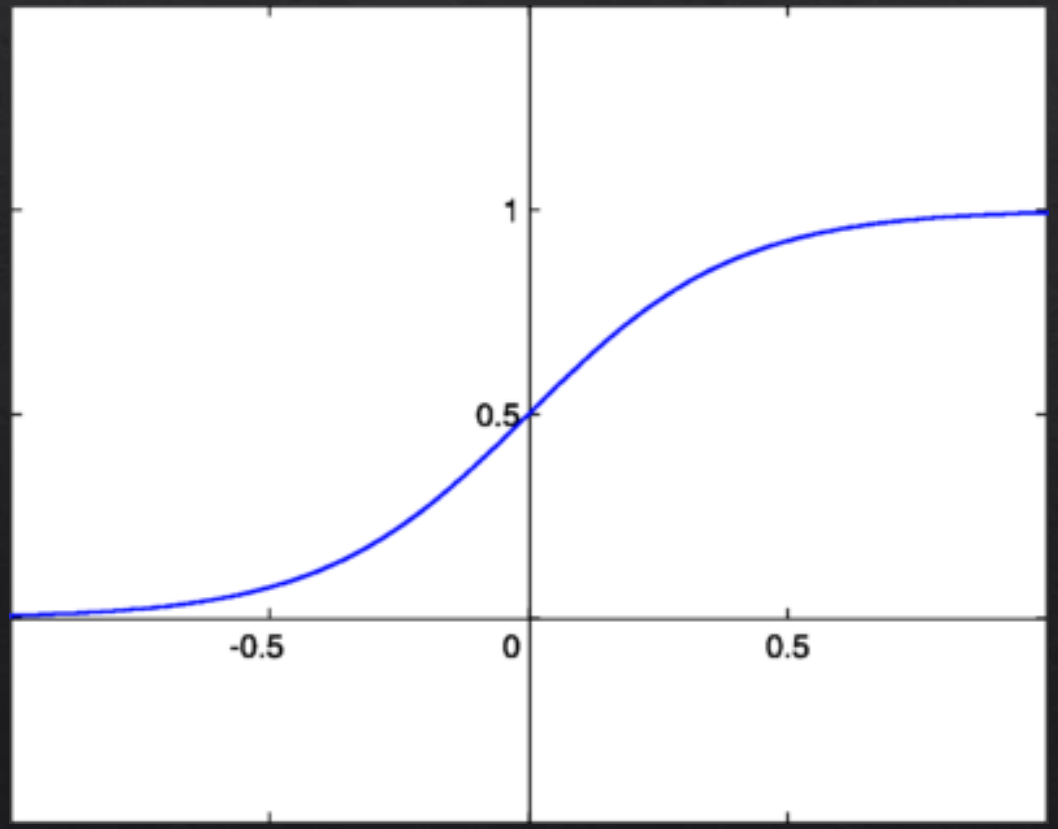
Multi-class classification

Multi-Class Classification!



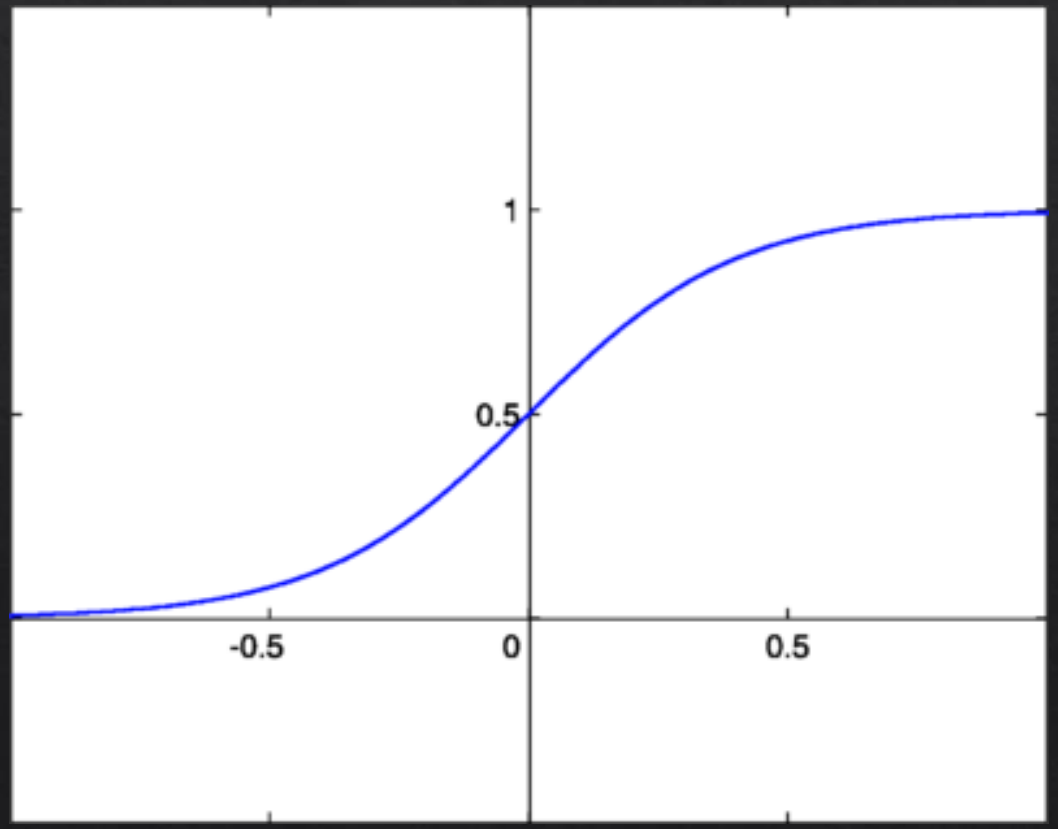
- Problem: Doing classification but there is more than one class!
  - Before we had 0 and 1
  - But we could have 0-5 or anything else!
- How can we adapt logistic regression to handle this?

Many vs. One Algorithm!



- One way to deal with this is called the many versus one algorithm
- Suppose we have 5 classifications (1-5)
- Idea:
  - Treat 1 as a 1 and 2-5 as 0
  - Run LR as we did before
  - Then, Treat 2 as 1 and 1,3,4,5 as 0
  - Run another LR

Many vs. One Algorithm!



- One way to deal with this is called the many versus one algorithm
- Suppose we have 5 classifications (1-5)
- Idea:
  - Treat 1 as a 1 and 2-5 as 0
  - Run LR as we did before
  - Then, Treat 2 as 1 and 1,3,4,5 as 0
  - Run another LR

Many vs. One Algorithm!

- So, we end up with (in this case) five unique logistic regressions!
- 1 vs 2-5 //Will return the percent chance an example is a 1 or something else
- 2 vs 1,3-5 //Will return the percent chance an example is a 2 or something else
- 3 vs 1,2,4,5 //…3 or something else
- 4 vs 1-3,5 //…4 or something else
- 5 vs 1-4 //…5 or something else

Many vs. One Algorithm!

- So, we end up with (in this case) five unique logistic regressions!

- 1 vs 2-5 //Will return the percent chance an example is a 1 or something else
- 2 vs 1,3-5 //Will return the percent chance an example is a 2 or something else
- 3 vs 1,2,4,5 //…3 or something else
- 4 vs 1-3,5 //…4 or something else
- 5 vs 1-4 //…5 or something else

- So, if we send an example through each regression it might return something like:
- 1: 0.2 2: 0.4 3: 0.81 4: 0.46 5: 0.33

Many vs. One Algorithm!

- So, if we send an example through each regression it might return something like:
- 1: 0.2 2: 0.4 3: 0.81 4: 0.46 5: 0.33

- In this case we would return that this example is a 3 because that is the highest percentage we calculated

- Usually, we return the percentage as well (as a degree of confidence measure)

Summary

Conclusions!

- ***Supervised Learning***: Given training examples, learn to recognize patterns and identify new unseen instances of the same kind

- ***Version Space Learning***:
  - Simple, easy to code, good for Boolean data only
  - Not terribly useful unless training set is quite simple

- ***Decision Tree Induction***
  - Simple to understand (CS people love trees!), easy to code, easy to adapt to new datasets
  - May need to tweak how tree is built so that you learn the best tree possible

Conclusions!

- ***Supervised Learning***: Given training examples, learn to recognize patterns and identify new unseen instances of the same kind

- ***Regression:***
  - Really strong, general technique for learning. Good for almost any data set with continuous output variable. Can use Normal Equation or Gradient Descent. Can be adapted to fit polynomials, normalized for simplification, etc. very easily
  - Not as sophisticated as some approaches we'll see later (Neural Networks etc.)

- ***Logistic Regression***:
  - Great for classification tasks (output variable is discreet). Naturally probabilistic. Same cost function (in form) as linear regression. Can handle multiple classes quite easily