Categorical Data

Categorical Data Is Data Based on Classes

- Examples
  - Colors (red, green, blue)
  - Logic (true, false)
  - Objects (car, house, truck)
- This data is non-numeric, but still important
- Question becomes how do we make this data numeric?

One-Hot Encoding

Color	Red	Green	Blue	Pink
Red	1	0	0	0
Red	1	0	0	0
Blue	0	0	1	0
Green	0	1	0	0
Green	0	1	0	0
Pink	0	0	0	1

- In essence we have taken category data and turned it into indicator data, or even more appropriately, vector data. “Red” is now a vector [1,0,0,0], “Green” is [0,1,0,0], etc.
- This works for both data **features** and data **targets**.

Be Careful with “Discrete” Data

- Is the number of rooms in a house categorical?
  - Answer: No
- While often we label things as “Category 1, Category 2,” etc., sometimes (like in the example above), two rooms really represents more “rooms” than one. This is called discrete data, in that it can take on only certain values.
- Discrete data is different than categorical data.
  - Imagine I have two classes of students: class 1 and class 2
    - If I out my data in a class “1” or “2,” this would tell my models that class 2 is “twice” of something than class 1. Here I should treat them as categorical.

Examples of Discrete Data

- Number of rooms in a house
- Number of cars a family owns
- Number of children
- Number of cylinders in a car engine

Do not one-hot encode discrete data!

Categorical Features

- Data that is an input feature, once one-hot encoded, can be put into any model.
- For example, if we were modeling the price of a car using linear regression and one of the features one car color:
  - Change the color column to a group of columns that one-hot encodes the available car colors.
  - Use the one-hot encoded data directly in our model.

DataScience@SMU

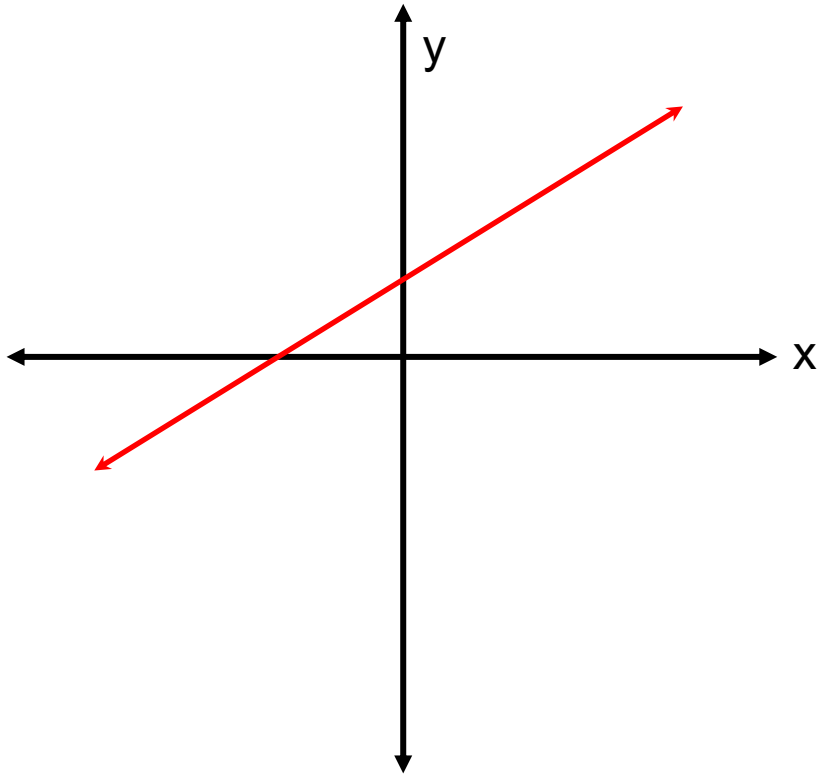
Linear to Logistic

What Is the Target Is One-Hot Encoded?

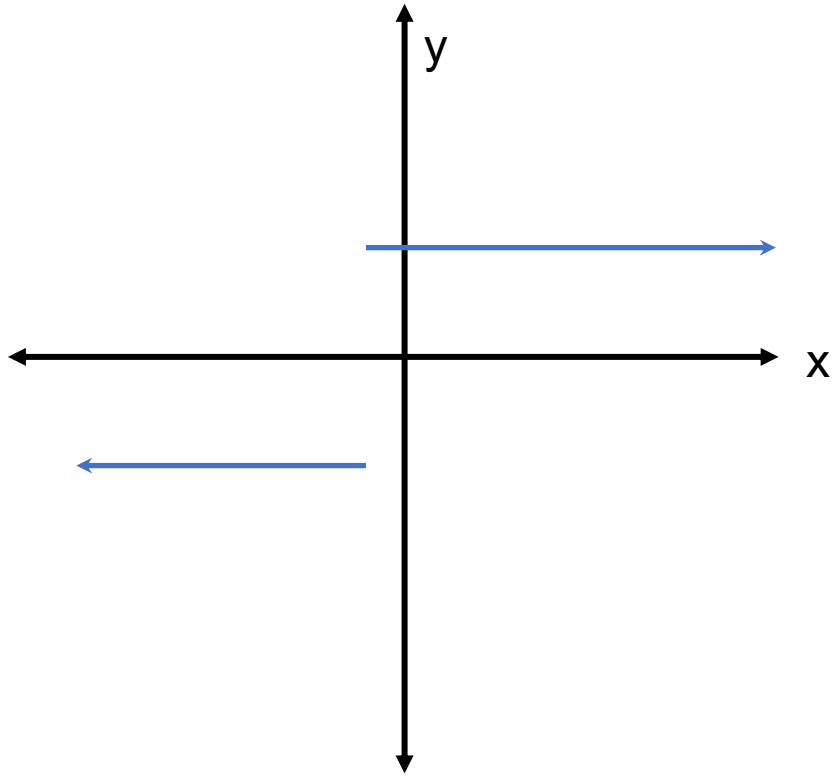
- In the previous session we said that input features are ready to go once they are one-hot encoded.
- What about if the target is a category? We can one-hot encode the target, but linear regression predicts a continuous response, not a discrete 1 or 0.
- How can we utilize linear regression to produce a 1 or 0?
  - If we can use linear regression we gain all the methods used to solve linear regression.
  - The issue to tackle is how to make the output discrete.

Continuous vs. Discrete

Continuous Output
(y can be any value)



Discrete Output
(y can only be specific values)



Introducing Logistic Regression

- Take our original linear regression equation:

$$m_1x_1 + m_2x_2 + ... = y$$

$$\Sigma m_i x_i = y$$

- Use a new equation that we will call logistic regression:

$$\frac{1}{1+e^{-\Sigma m_i x_i}} = y$$

- Or:

$$\sigma(m_i x_i) = y$$

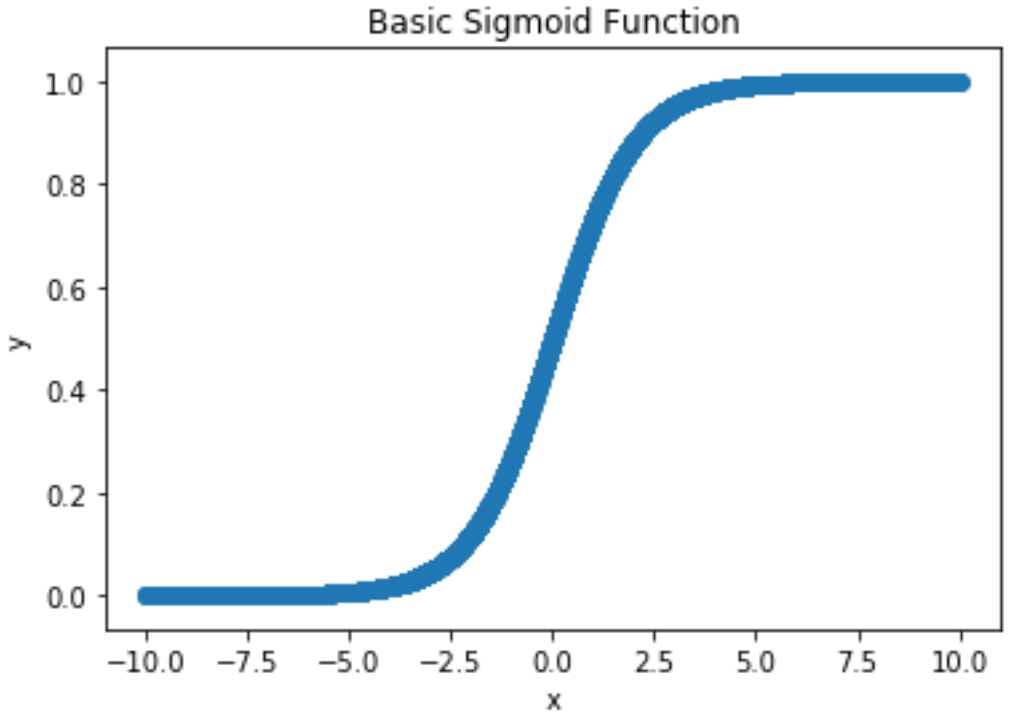
Note that I dropped the summation sign for clarity.

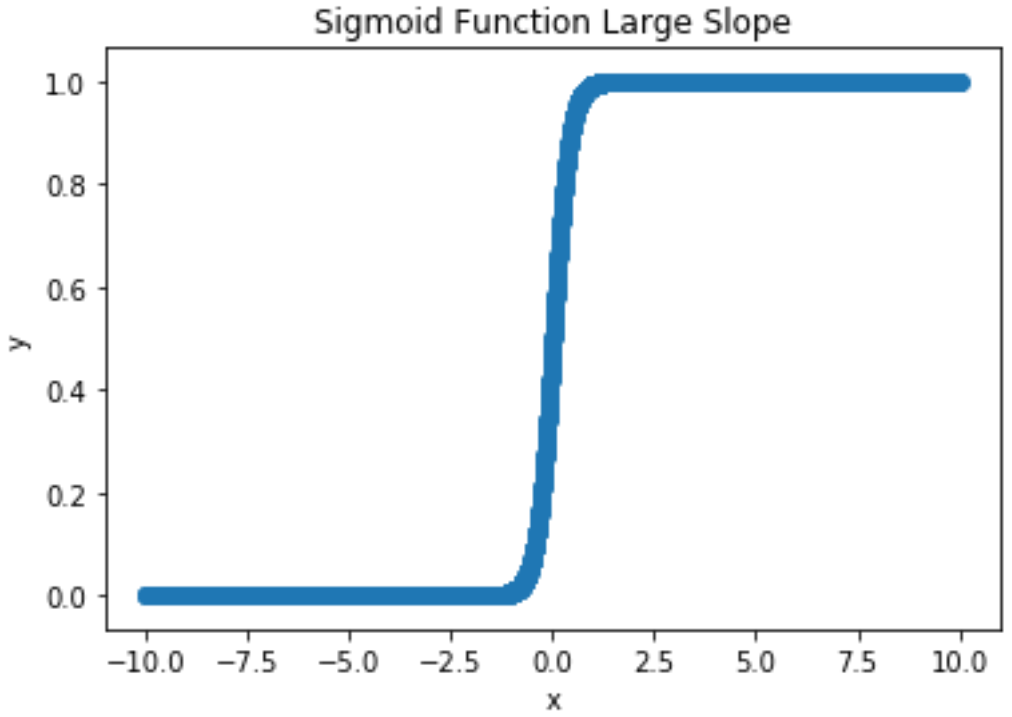
DataScience@SMU

The Sigmoid

Introducing the Sigmoid Function

- The sigmoid function takes our inputs and mimics a step function.
- It allows the output to be continuous, but in essence “squashes” the output to between 1 and 0.





Sigmoid Function Properties

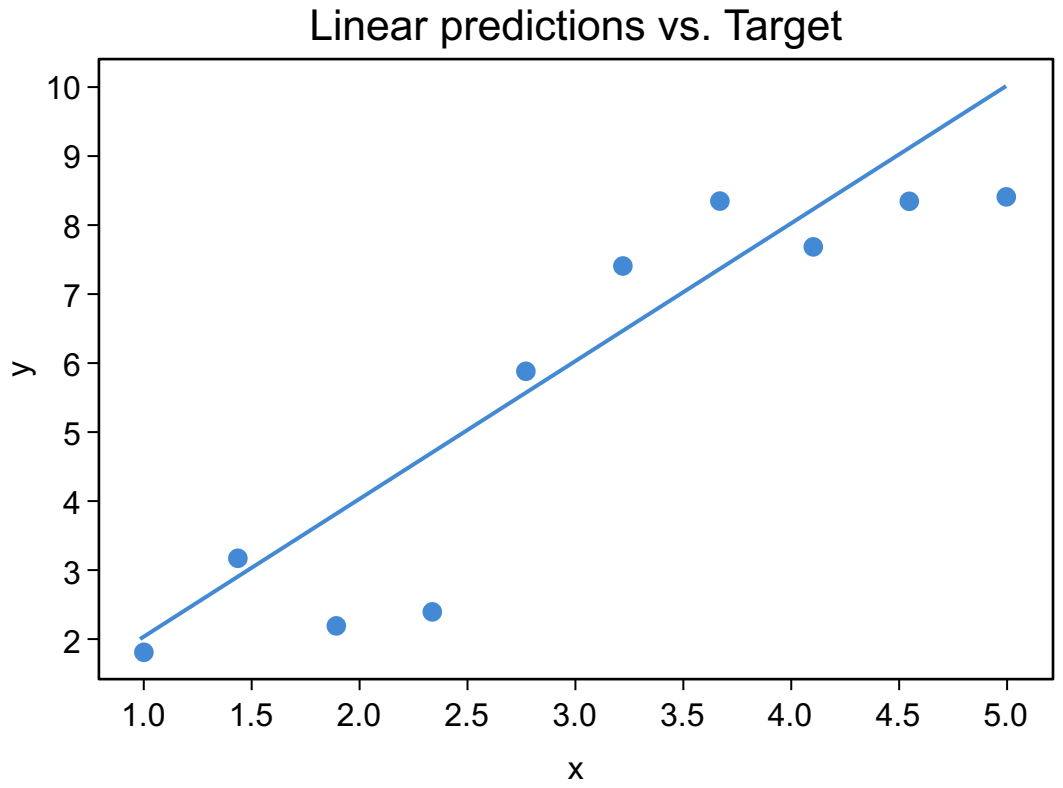
- Equation: $y = \frac{1}{1+e^{-mx}}$
- If “mx” is positive and large, then the exponential part is nearly 0, so y will be very close to 1.
- If “mx” is negative (−5 or more) the exponential becomes very large and the denominator of the fraction becomes very large, thus y is very close to zero.
- If “mx” is very small, than y will vary from 0 to 1.
- This simulates the step function.
- Best of all, if we need to take a derivative for slope via gradient descent, exponentials are super easy!

DataScience@SMU

Log Loss

Linear Loss

- In linear regression models, we use mean squared error or mean absolute error to measure the distance from the target to the prediction.
- The loss is the sum of the vertical distance from the prediction (dot) to the prediction (line).
- The object is to get the points as close to the line as possible.



Logarithmic (Log) Loss

Since y takes on only two values, let us modify the loss function from a distance function to a two-part loss function.

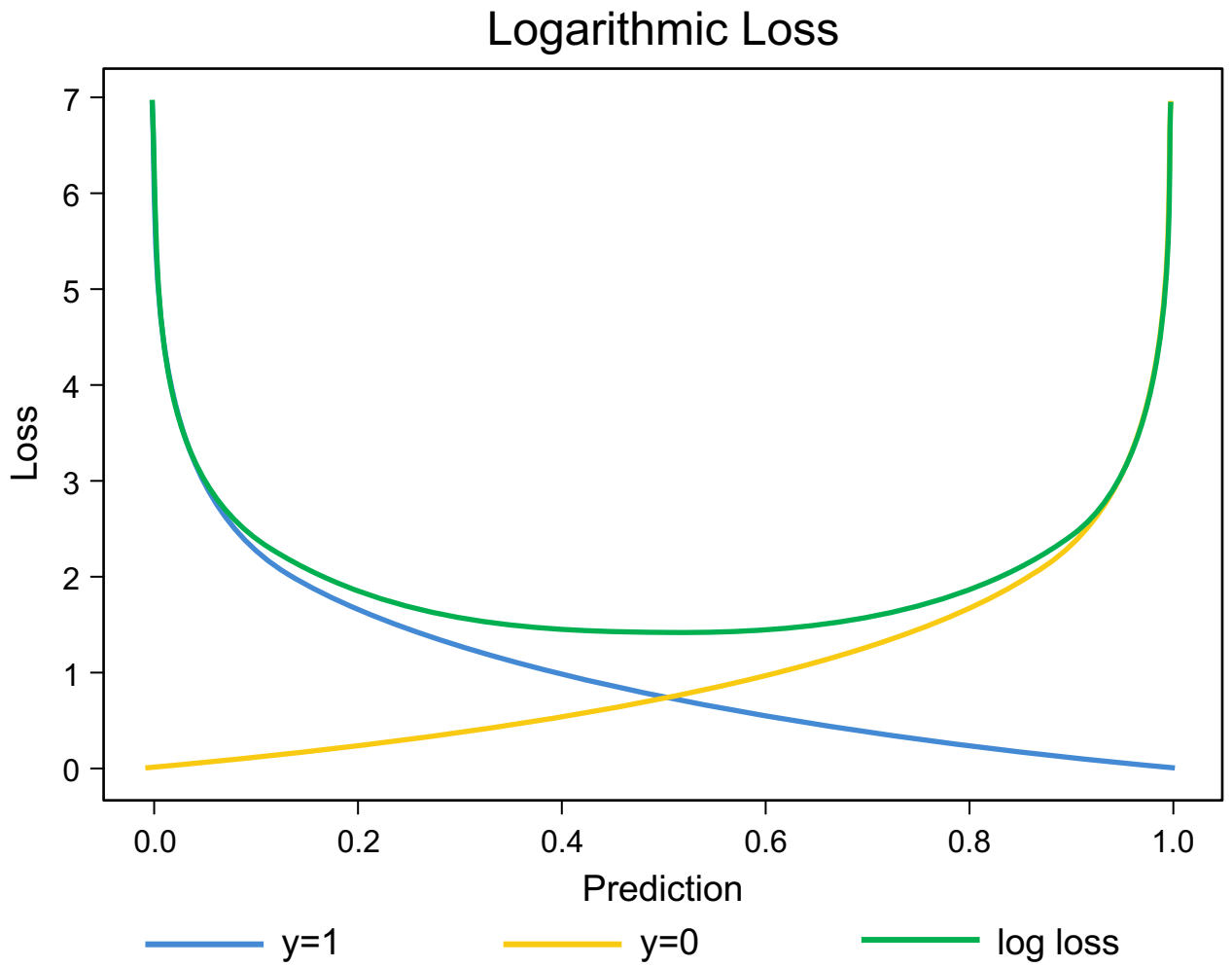
- We have two cases, when the target (y) is 1: $-y \log p$
- When the target is 0: $(1-y) \log (1-p)$
- p is the prediction or output of the sigmoid
- Together they form the log-loss function

$$\frac{-1}{N} \Sigma y_i \log p_i + (1-y_i) \log (1-p_i)$$

- Since y is 1 or 0 only, one of the terms is zero
- This penalizes scores or predictions far from the target
  - If y is 1, then only predictions close to 0 contribute significantly to the loss.
  - If y is 0, only predictions close to 1 contribute significantly to the loss.

Graph of the Log Loss

- When the target is 1, we see predictions that are close to zero, are heavily penalized, or contribute a larger amount to the loss.
- Likewise when the target is 0, predictions close to 1 are also heavily penalized.



Note: Log loss is also called “binary cross entropy” since it deals with binary (1 and 0) targets.

DataScience@SMU

Minimizing Log Loss

How Do We Minimize Log Loss?

- To find a good model, we want a small loss.
- We have a loss function.
- We have a prediction function.
- Let’s combine them similar to linear regression.
  - Minimize loss by taking the derivative.
  - Remember, the prediction p depends on the input x.

Equations for Log Loss

$$J = \frac{-1}{N}\Sigma y_i \log p_i + (1 - y_i) \log (1 - p_i)$$

$$p_i = \frac{1}{1+e^{-\Sigma m_i x_i}}$$

$$\frac{\partial J}{\partial m_i} = \frac{-1}{N}\frac{\partial}{\partial m_i}\Sigma y_i \log p_i + (1 - y_i) \log (1 - p_i)$$

$$\frac{\partial J}{\partial m_i} = (p_j - y_j)x_j$$

- This is a pretty big leap, but there are tons of derivations. Check out: Logistic Regression
- It turns out the derivative of the loss gives the **same** equation as squared error loss for linear regression. Thus, all the tools to solve linear regression can be easily adapted to logistic regression. In fact, the only big difference between linear and logistic regression is the target and the loss function utilized!

DataScience@SMU

Multiclass

How Do We Go from Two Classes to Many?

- Sigmoid only gives us a binary output.
  - Thus, the secondary loss name “binary cross entropy loss”
- Many problems have more than two classes.
- How can we use our binary logistic regression?

One vs. All

- The first method for multiclass is known as “one versus all” or “one versus rest.”
- For each class, train a model on a true/false basis.
  - Example: three classes—red, blue, green
  - Three models
    1. Red/Not Red
    2. Green/Not Green
    3. Blue/Not Blue
  - Form a vector with a length of the number of classes (in this case three) with the value of the positive class (Red, Green, Blue target = 1) as the value.
  - The position with the highest value is the class.

OvA Example

Red (1)	Not Red (0)	Blue (1)	Not Blue (0)	Green (1)	Not Green (0)
0.87	0.13	0.22	0.18	0.45	0.55

Red = 1	Blue = 1	Green = 1
0.87	0.22	0.45

- Max value is the first position, so the predicted color is **red**.
- Problem: Classifiers see a large number of negative examples. Each classifier can have different confidence.

One vs. One

- Set up multiple binary classifiers to distinguish among classes in a head-to-head “battle.”
- For n classes requires $n(n-1)/2$ classifiers.
  - For n=3, this is a total of three classifiers, same as OvA.
  - For n=4, this is a total of six classifiers.
  - For n=10, this is 45 classifiers!
- The class that wins the most votes is the assigned class.

OvO Example

- Four states (classes): Texas, Iowa, California, Florida
- Six classifiers

Texas	Florida
0.77	0.23

Texas	Iowa
0.74	0.26

Texas	California
0.99	0.01

California	Iowa
0.12	0.82

California	Florida
0.45	0.55

Florida	Iowa
0.37	0.63

- Texas wins three votes, Iowa wins two votes, Florida wins one, California wins one.
- However, what would happen if, in the Texas vs. California, California got the higher score? Then three states would have two votes! This is a big downside to OvO.

Summary

- Both methods are imperfect.
- OvO can have ties.
- OvR can suffer from unbalanced negative examples.
- Both are usually implemented “under the hood” of the code—meaning you just tell the model: “I want to use OvR” or “I want to use OvO.”
- It allows us to use binary classifiers on multiclass problems.

DataScience@SMU