# Logistic Regression

The past three chapters have introduced you to a number of machine learning algorithm that are used to solve *regression problems*.

To date we’ve been solving regression problems, where the values being predicted (like the price of a house) are entirely unconstrained. Classification problems are those where we want a machine learning algorithm to predict a specific, discrete result from a predefined set of data.

## Binary Classification

Binary classification is the simplest form of classification problem to solve for, but can help us answer some incredibly valuable questions. Examples of binary classification problems include spam filtering (is this email spam? yes/no) fraud detection (is this transaction legitimate? yes/no) and medicine (is this tumor malignant? yes/no).

For each of these example, the variable that we’re trying to predict will take on one of two distinct values, 0 or 1. More formally, we want to predict a value y which will take on the value 0 or 1, defined as:

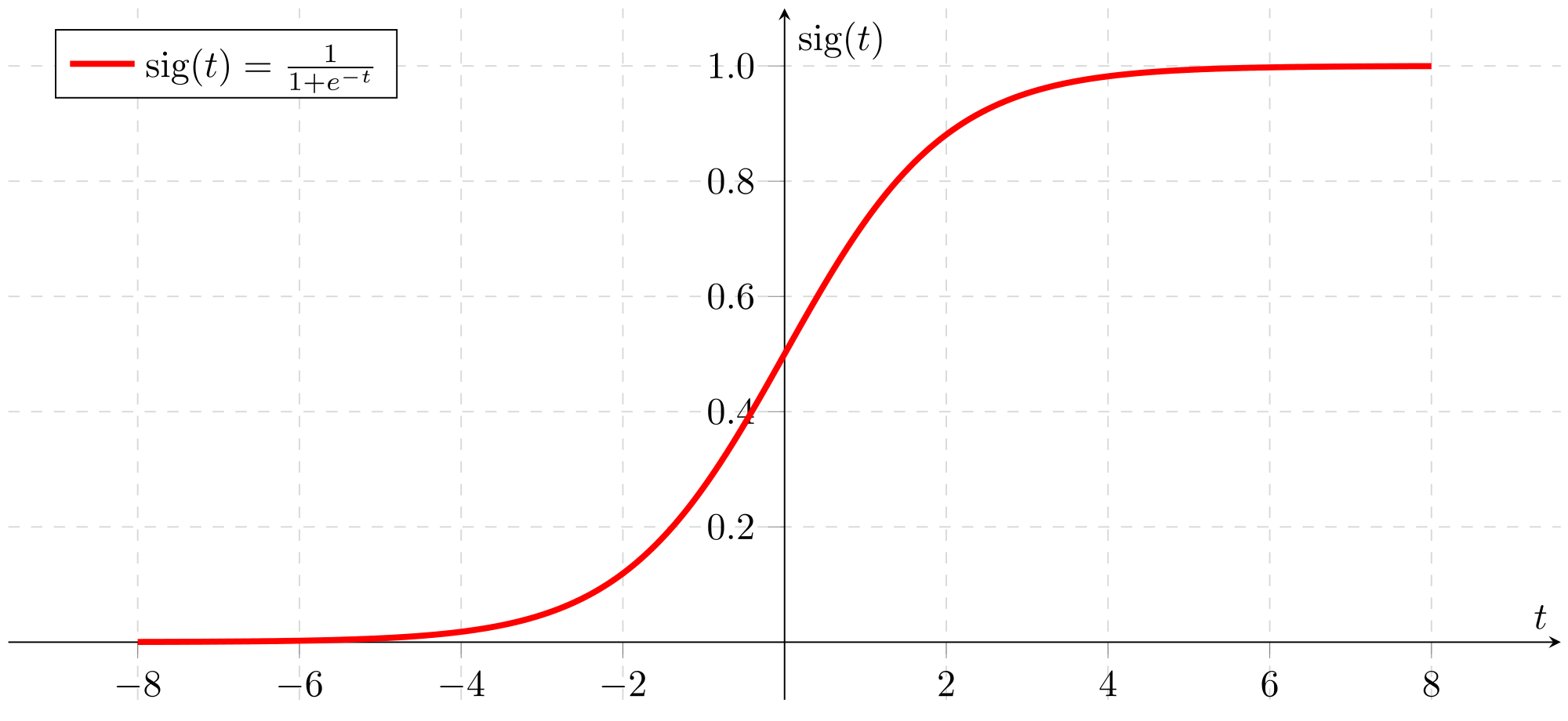
y∈{0,1}

By convention, the value 0 is referred to as the “negative class”, and the value 1 is referred to as the “positive class”.

### The Logistic Function

The Sigmoid function (also referred to as the Logistic Function) is a function that neatly satisfies our requirements for classification. Don’t get confused about the terminology here - although this function is called the Logistic Regression function, it is in fact used for classification problems.

Figure 3: The Sigmoid Function



Firstly, the Sigmoid function maps any real number along the x axis to the [0, 1] interval we require for classification. For any input value x, the Sigmoid function guarantees that we will always be in our “yes”/”no” boundary. Secondly, it has a sharp transition between the [0, 1] boundary, which will help us produce a well-defined decision boundary (i.e. the point where we decide what qualifies as a yes, and what qualifies as a no).

### The Hypothesis Function

We need to “hook” our hypothesis function into the Logistic Regression function, here’s how to do it.

Our existing hypothesis function is defined as:  
h(x) = Θ0x0 + Θ1x1 + Θ2x2 + … + Θnxn

First, we temporarily assign our existing hypothesis function to a variable z, so we have:  
z = Θ0x0 + Θ1x1 + Θ2x2 + … + Θnxn

The Logistic Regression function is defined as:

(z)=11+e−z

We can formally define our new classification hypothesis function as:

h(x) = g(z)

When we substitute z with our definition above, we get:

h(x) = g(Θ0x0 + Θ1x1 + Θ2x2 + … + Θnxn)

When we expand the function g, our fully implementable hypothesis function is:

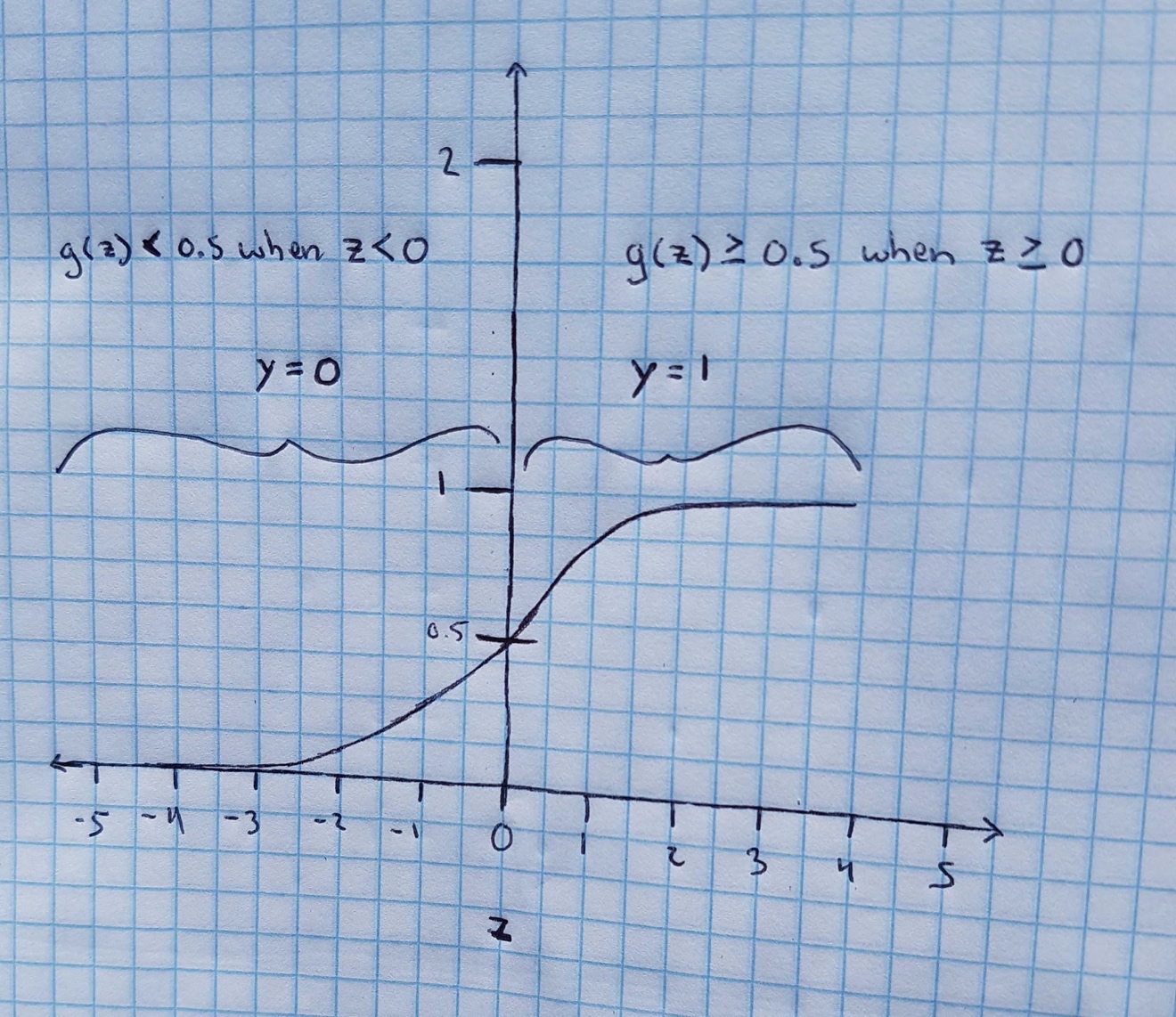
h(x)=11+e−(θ0x0+θ1X1+...+θnXn)

It’s also important to note the behavioral difference between the Sigmoid function and our linear regression hypothesis function. For any input value x, the Sigmoid function will return a probability of that y = 1 for the input value.

For example, if we found h(x) = 0.55, the Sigmoid function is telling us that h(x) as a 55% chance of being 1, and therefore a 45% chance of being 0.

### The Decision Boundary

The goal of our hypothesis function is to predict a discrete value, a 1 or a 0. In this section, we’ll understand how the hypothesis function will make these predictions with the use of a decision boundary. We’ll use the following graph to get a better sense of what the logistic regression hypothesis function is computing.



The image above shows our hypothesis function (Logistic Regression or Sigmoid function), with the function asymptoting at 0 and 1. Since our predictions can only result in one of two possible values, the first question we need to ask ourselves is when should a value of h(x) fall into the “0” camp, and when should a value of h(x) fall into the “1” camp.

We now know that if h(x) = 0.55, the hypothesis function is telling us that h(x) as a 55% chance of being 1, and therefore a 45% chance of being 0. Let’s define our first decision boundary as:

when h(x) >= 0.5, the predicted value (y) should be equal to 1.   
when h(x) < 0.5, the predicted value (y) should be equal to 0.

This is a pretty sensible definition - the arbitrary component here is that when h(x) is exactly equal to 0.5, the predicted value could really go in either camp. As a convention, we’ll make the predicted value 1. If you really wanted to, you could make the predicted value 0 - there’s no right or wrong answer.

We can start reasoning about how the algorithm will work. If you look at the value z = 0 on the graph, you’ll notice that the function intersects at y = 0.5. The value of y increases (up to 1) as the values of z get bigger.

We can generalize and say that g(z) >= 0.5 when z >= 0. Likewise, we can say that g(z) < 0.5 when z < 0.

Looking at our graph, we know that g(z) >= 0.5 when z >= 0. That’s the right hand side of the graph. Given our hypothesis function h(x) = g(z), we can also say that

h(x) >= 0.5 when z >= 0, and h(x) < 0.5 when z < 0

Of course, 0.5 is our decision boundary for deciding when a predicted value should go into the “1” camp or “0” camp, so we can also say that

h(x) should produce a predicted value of 1 when z >= 0,  
h(x) should produce a predicted value of 0 when z < 0

We also know what z is, so we can more formally say

h(x) should produce a predicted value of 1 when Θ0x0 + Θ1x1 + Θ2x2 + … + Θnxn >= 0 h(x) should produce a predicted value of 0 when Θ0x0 + Θ1x1 + Θ2x2 + … + Θnxn < 0

### The Decision Boundary in Practice

Sdfsd

sdfsdf