Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2022

The problem

Now, our goal is to create a **good model** for our problem, **binary classification**.

To do this, we can **try** using our 0-1 loss $\mathcal{L}$:

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(\text{sign}(\theta^\mathsf{T} x^{(i)} + \theta_0), y^{(i)}) \tag{1}$$

The **first** thing to note is that there isn't an easy **analytical** solution, no simple **equation**: $\text{sign}(u)$ isn't a function that we can explicitly **solve**, like we could for **linear regression**.

> To be fair, this is true for most possible problems: most of them can't be solved analytically.

So, we refer to our other approach, **gradient descent**.

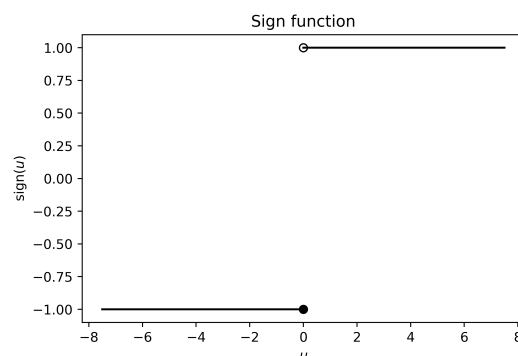But in order to do that, we'll just need to get the **gradient**.

$$\nabla_\theta J = 0 \tag{2}$$

...Well that's not good.

> Why not? Because we use our **gradient** to decide **how** to change θ, if the gradient is 0, we'll never **improve** θ at all!

The real problem: $\text{sign}(u)$ is flat

What's going on here? Let's look at the sign function:

Sign is a flat function! The slope is 0 everywhere, except $u = 0$, where it's **undefined**.

Well, that explains why we can't use the gradient: the function is **flat**.

Another way to say this is that our function doesn't **tell** us when we're **closer** to being right.

There's **no difference** between being **wrong** by 1 unit or being wrong by 10 units: you can't tell if you're getting **closer** to a correct answer.

And the **gradient** doesn't tell you which way to move in **parameter space** to further improve.

> Remember, parameter space is what we move through as we change our parameter vector θ.

In fact, the best way we know how to approach this kind of problem takes **exponential** time: it takes exponentially **longer** to solve based on our **number** of data points.

That's way too **slow**. So, we'll have to come up with a **better** function: something to **replace** sign(u), that still serves the same role.

Concept 1

The **sign function** is difficult to optimize, because it isn't **smooth**: not only is the slope undefined at 0, it is 0 everywhere else.

This causes two problems:

- We can't tell whether one **hypothesis** is **closer** to being **correct**, if it has gotten **better**, unless its accuracy has increased.

  - This makes it harder to **improve**.

- We can't indicate how **certain** we are in our answer: sign(u) is **all-or-nothing**: we choose one class, with no information about how **confident** we are in our choice.

  - Knowing how **uncertain** we are can be **helpful**, both for **improving** our machine and also **judging** the choices or machine makes.

So, we need to explore a **new** approach: we'll **replace** sign(u) with something else.

The sigmoid function

So, what do we **replace** sign with? We like the way sign **works** (choosing between two different classes based on a **threshold**), so maybe we want a **smoother** version of it.
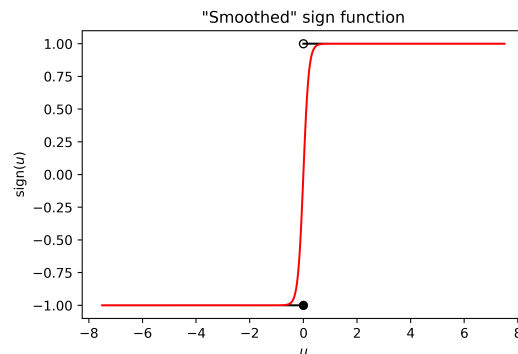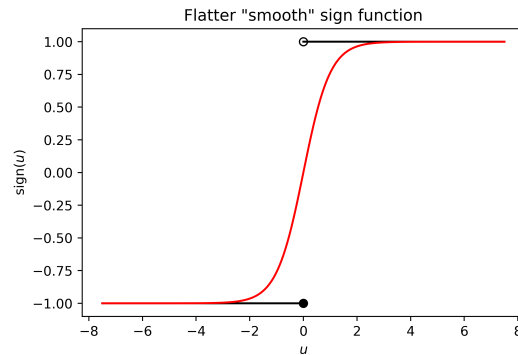

The red line shows a "**smoother**" sign function, that mostly behaves the same, while solving our problem.

This solves **one** of our two problems: the **gradient** is **nonzero**.

We could also make it less steep:

> It's hard to see visually, but the function is **smooth**, and the slope is nonzero **everywhere**!

Flatter "smooth" sign function

So, we need a **function** that accomplishes this. It turns out there are **several** that work: $\tanh u$, for example.

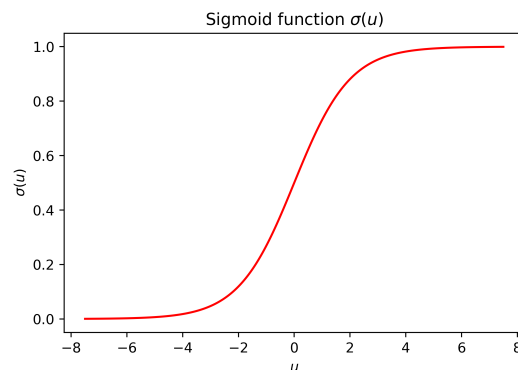For our purposes, we'll use the following function:

Definition 2

The **sigmoid** function

$$\sigma(u) = \frac{1}{1 + e^{-u}} \tag{3}$$

...is a nonlinear function that we use to **compute** the output of our **classification** problem.

It is also called the **logistic** function.

The function looks like this:

Sigmoid function $\sigma(u)$

Sigmoid as a probability

Something you may **notice** is that $\sigma(x)$ is always between 0 and 1. But before, $\text{sign}(x)$ was **always** between -1 and +1. Why would we use *this* function?

Because going between 0 and 1 has a different advantage: we can interpret it as a **probability**.

Your **value** of $\sigma(u)$ can be stated as, "what does the machine think is the **probability** we **classify** this data point as +1".

And, on the **flip** side, $1 - \sigma(u)$ is the **probability** we **classify** as -1.

This solves the second problem we mentioned **earlier**: we can indicate how **confident** the machine is in its answer!

Concept 3

The output of the **sigmoid function** $\sigma(u(x))$ gives the **probability** that the data point x is classified **positively**.

$$\sigma(u) = \mathbf{P}\{x \text{ is classified } +1\}$$

$$1 - \sigma(u) = \mathbf{P}\{x \text{ is classified } -1\}$$

Note that this works because $\sigma(u) \in (0, 1)$.

Logistic Regression

So, we've seen the benefits of switching from $\text{sign}(u)$ to $\sigma(u)$. So we'll do that: _____

> We're using $u(x) = \theta^\mathsf{T} x + \theta_0$

Key Equation 4

Logistic Regression is a **modification** of **linear regression**.

$$h(x; \theta) = \sigma(\theta^\mathsf{T} x + \theta_0)$$

where

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$

It outputs the **probability** of a **positive** classification.

If we **plug** this in, we get this slightly ugly expression:

$$h(x; \theta) = \frac{1}{1 + e^{-(\theta^\mathsf{T} x + \theta_0)}}$$

We have a problem, though: **logistic regression** is a... **regression** function. It takes in a real **vector**, and outputs a real **number**: $\mathbb{R}^d \to \mathbb{R}$.

We can't use this to do **classification**, where want $\mathbb{R}^d \to \{-1, +1\}$!
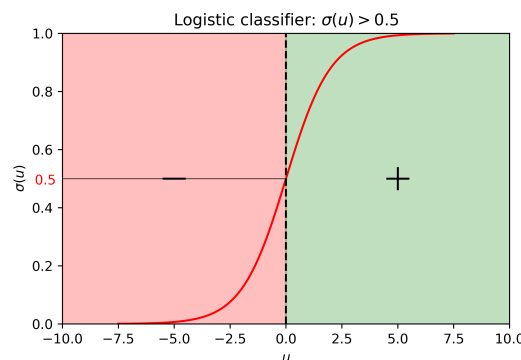
Prediction Threshold

When we were just using $u(x) = \theta^T x + \theta_0$, we classified data points by saying whether $u(x) > 0$. Our boundary was $u(x) = 0$.

We can't quite do that here, because $\sigma(u) = 0$ is **impossible**: $\sigma(u)$ is **always** greater than 0.
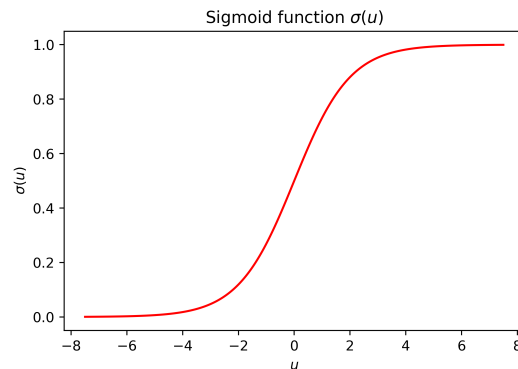
$\sigma(u)$ approaches 0 as u approaches $-\infty$, but it never reaches it.

Well, what happens when $u(x) = 0$? We get $\sigma(0) = .5$. So, we could use that as our classification: $\sigma(u) > .5$

But, we don't necessarily always want to use .5:

Example: Imagine if you wanted to **classify** whether someone needs **life-saving** treatment. Classify -1 if sick (they need it), $+1$ if healthy (they don't).

Let's say you got $\sigma(u) = .6$, so you're only 60% sure they **don't** need it. You'd classify that as $\sigma(u) > .5$: they're '**healthy**'.

Even so, you probably shouldn't **refuse** someone treatment that's 40% likely to **save** their life. We might not want to use $\sigma(u) > .5$ after all.

We call the **boundary** between positive and negative the **prediction threshold**.
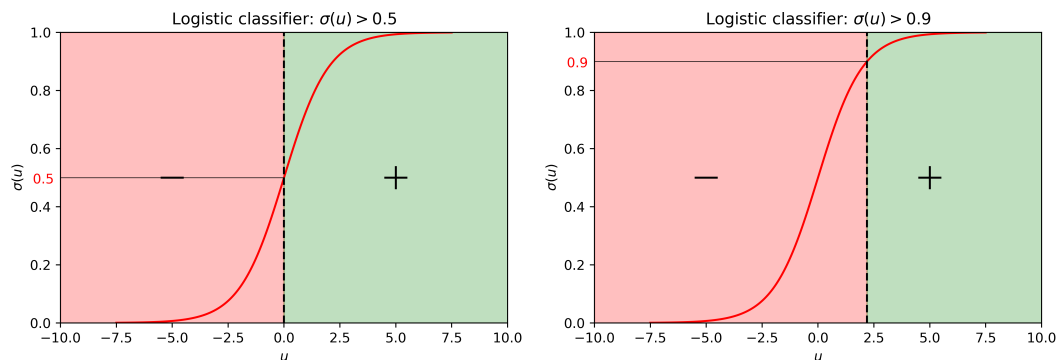

Definition 5

The **prediction threshold** $\sigma_{\texttt{thresh}}$ is the value where you go from **negative** classification to **positive**.

In general, we say

Our **default** value is a threshold of .5, but our threshold can be **anywhere** in the range

$$0 < \sigma_{\texttt{thresh}} < 1$$

Example: If $\sigma_{\texttt{thresh}} = .9$, we would see:

We switch from a .5 threshold to a .9 threshold.

Linear Logistic Classifier

This finally gives us our **linear logistic classifier** (LLC)

Key Equation 6

The **linear logistic classifier** is a **binary** classifier of the form

$$h(x;\theta) = \begin{cases} +1 & \text{if } \sigma(u(x)) > \sigma_{\texttt{thresh}} \\ -1 & \text{otherwise} \end{cases}$$

where

$$u = \theta^\mathsf{T} x + \theta_0 \qquad \sigma(u) = \frac{1}{1 + e^{-x}} \tag{4}$$

We call it linear because of the linear inner function $u(x)$, and logistic because of the outer function $\sigma(u)$.