Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2022

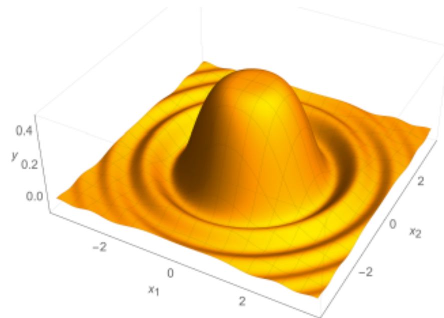
What's still missing?

Last chapter, we used our linear regression model to do classification: we created a "hyperplane" to **separate** the the data that we placed in each class.

We also mentioned that regularization can increase **structural error**, by limiting what possible θ models we're allowed to use.

Our goal was to decrease estimation error, but that's beside the point right now.

But, what if our linear model is already **too limited**? What if we need a more complicated model? This is true in a lot of real-world problems, like vibration:



This wave doesn't seem particular friendly to a planar approximation.

These kinds of situations are called, appropriately, **non-linear**.

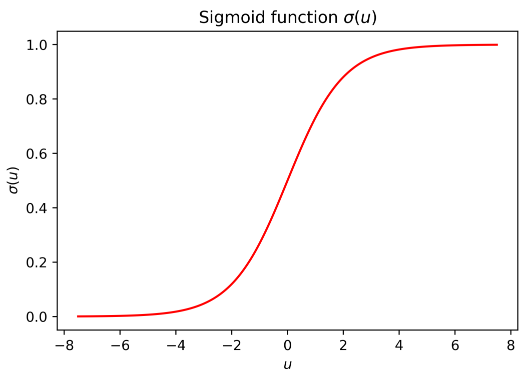
> **Concept 1**
>
> **Non-linear** behavior cannot be accurately represented by any **linear** model.
>
> In order to create an accurate model, we have to use some **nonlinear** operation.

If we could create effective, non-linear models, we might even be able to deal with data that was previously "**linearly inseparable**".

Possible Solutions: Polynomials

Let's try to think of ways to approach this problem. We'll start with a 1-D input, for simplicity.

Upon hearing "non-linear", we might remember the function we introduced last chapter: the **sigmoid**.



Your friendly neighborhood sigmoid.

Can we use this to create a new model class? For now, unfortunately not: remember that we used this in the last chapter, and we still got a **linear** separator. The reasons were discussed there.

We'll show ways we can use this kind of approach, when we discuss Neural Networks.

Instead, we can get inspiration from our example of "structural error". For now, let's focus on **regression** (though classification isn't too different):

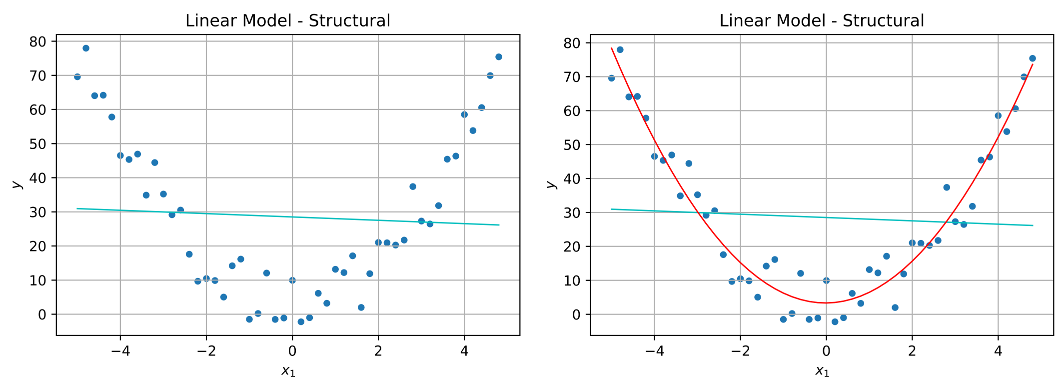


Figure 1: A linear function can't represent this dataset. However, a parabola can!

We're still using our input variable x, but this time, we've "**transformed**" it: we have squared x, giving us a model of the form

Remember that x is 1-D right now!

$$h(x) = Ax^2 + Bx + C \tag{1}$$

It should be clear that his model is more **expressive** than the one before: it can create every model that our linear approach could (just by setting $A = 0$), and it can create new models in a parabola shape.

Reminder: "expressiveness" or "richness" of a hypothesis class is how many models it can represent: a more expressive model can handle more different situations.

Concept 2

We can make our **linear** model more **expressive** by add a squared term, and turning it into a **parabolic** function.

This concept can be extended even further, to any **polynomial**.

Transformation

How do we *generalize* this concept? Well, we have a set of constant parameters A, B, C. These are similar to our constants θ_i. Let's change our notation:

$$h(x) = \theta_2 x^2 + \theta_1 x + \theta_0 \tag{2}$$

Now, we've got something more familiar. We could imagine extending this to any number of terms $\theta_i x^i$: if we needed a cubic function, for example, we could include $\theta_3 x^3$.

This is starting to look pretty similar to our previous model: in fact, we could even separate out θ as a variable:

Notice that θ_0 corresponds to $x^0 = 1$.

$$h(x) = \overbrace{\sum_{i=1}^{k} \theta_i x^i}^{\text{Polynomial sum}} = \overbrace{\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_k \end{bmatrix} \cdot \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix}}^{\text{Store as vectors}} = \overbrace{\theta^T}^{\text{Simplify}} \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \tag{3}$$

This really *is* starting to look like our linear transformation! That's helpful: we might be able to use the techniques we developed before.

In fact, we can argue that they're **equivalent**: we've just changed what our input vector is. Consider our new input $\phi(x)$:

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^k \end{bmatrix} \qquad h(x) = \theta^T \overbrace{\phi(x)}^{\text{New input}} \tag{4}$$

This is called **transforming** our input.

Definition 3

A **transformation** $\phi(x)$ takes our input vector x and converts it into a **new** vector.

This transformation can be used to:

- Allow our model to handle new, more **complex** situations (more **expressiveness**)
- **Pre-process** our data to make it **easier** for our model to find **patterns**.
- Convert our data into a **usable** format (if, say, the original format doesn't fit into our equations)

Example: Taking our input x and converting it into a polynomial is a **transformation** of our input.

This chapter will focus on these kinds of transformations.

Features

One benefit of only changing out input is that we can continue to use our linear representation: we will be able to optimize a "linear" model θ, over data that has been made **nonlinear**.

These transformations can be complex, especially for multi-dimensional inputs. In this first case, we only combined one input with **itself**. But, often, we can combine multiple together!

Thus, we should be careful to distinguish each input variable from each other. We often call these "**features**". However, we need to be careful:

Clarification 4

We often use the word **feature** in related (but not identical) contexts:

- A **feature** can be one **aspect** of our **original data**: for example, whether or not something is a cat or a dog, or the height of a patient.
- A **feature** can also be one mathematical **variable** in our **transformed input**. x_i is a feature of the data, while each variable of $\phi(x)$ is a feature of the transformed data.

Just like how we have an input space, we call the collection of possible values for our features the **feature space**.

Example: x in our previous example was a feature of the data, while x^i is a feature of our transformed vector.

Combined, this is why we called this technique the **feature transformation**: we apply some *transform* to the *features* of a data, to create a new set of *features*.

Since these transforms only apply to our features, we can keep the structure of a linear function:

Definition 5

Feature transformation allows us to do **linear** regression or classification on a set of **features** we have **non-linearly transformed**:

$$h(x) = \theta^T \phi(x)$$

$\phi(x)$ is our (often nonlinear) transformation of our features x.