# Latent variable models

## Problems that are not even close to being linearly separable

- Let's make an artificial example of a classification problem
- Load the Iris data, but make a new class structure in which there are two classes: versicolor (leave as is) and "setosica" - the union of setosa and virginica
- The result will look like this:

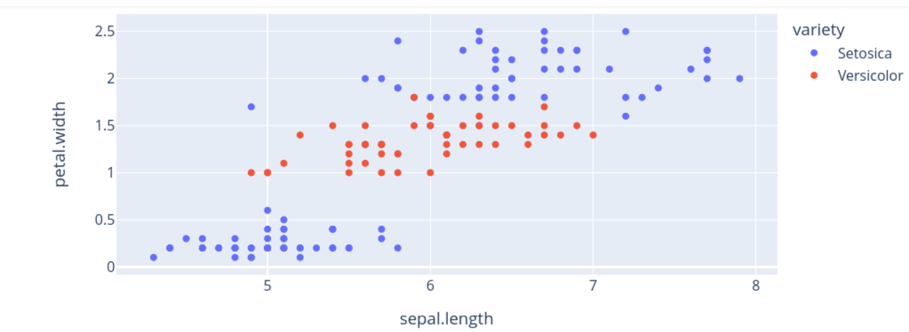



Figure 1: image

- How might we draw a line to separate these two classes?
- Well, we mightn't
- Looking at the other two dimensions won't help

## Learning latent variables

- The intuition behind **neural networks**, as well as behind a popular other approach called **support vector machines,** is that, **if we had the right representations then the problem would be easy**
- In particular, the classes would be **linearly separable**
- As we will see, neural network training asks **what set of features** (in the broad sense: not necessarily discrete like in linguistics) **would make this**

**problem linearly separable**

- It seems like magic, but it isn't if you think about it: a feature extractor is just a function; that function can be optimized, indirectly, so that its output (in conjunction with that of one or more other feature extractors) can solve a particular classification problem linearly
- What are some features that would help here (forget about how you might compute them, and no need to be too precise)? In other words, if I give you a new point, instead of telling you sepal width and pedal width, what information **that could in principle be derived from sepal width and pedal width** might group all the blue points off to one side? Think of two features, perhaps.
- Indeed, if we were somehow able to extract the feature **is off in the blue blob on top** and the feature **is down in the blue blob on the bottom** then I could combine these two using **and** and get my answer; and **and** is a linear function (how)?
- (Of course, I got this idea from the fact that I know they are actually two different species)
- There is however one important qualification to this; the features must be non-linear functions of the input for any of this to have any effect
- If they were not, then we would just some other linear classifier: linear functions are closed under composition!

# Feed-forward neural networks

- Let's see what this idea can do in this case
- I'm going to run this problem through the same optimizer that scikit-learn uses to do logistic regression (even though I would not recommend this for larger networks: we usually use a trick that takes into account which parameters affect the input to which other ones and which are independent - more on that another time)
- I'm going to try and optimize to find two linear features that then go through the inverse logit function, making some non-linear features - I'll name the parameters as follows (it's the parameters, of course, that we have to find):

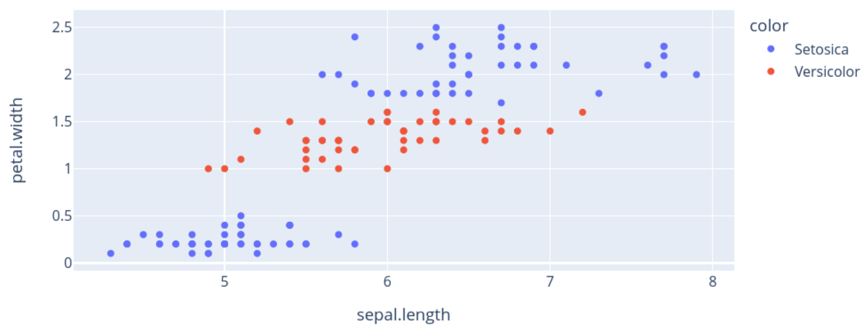
$$z_1 = a + bx_1 + cx_2$$
$$z_2 = d + ex_1 + fx_2$$

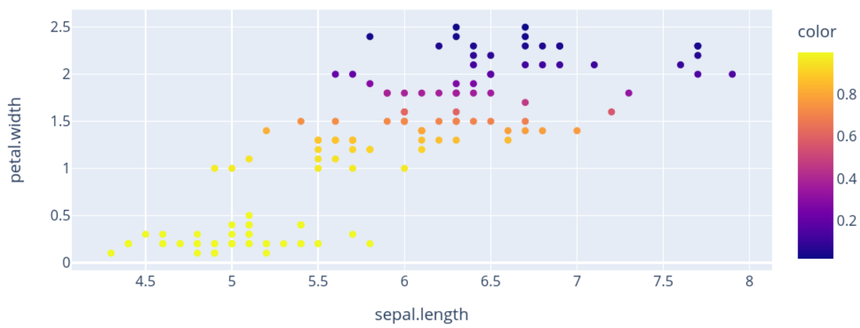
- For pedagogical purposes I prefer letters of the alphabet because otherwise I would have to give at least two indices to each parameter
- And now remember that there is going to be another set of parameters as we attempt to build a classifier with respect to these learned features - let's make it a logistic regression to keep it simple - and so I'll call the inverse logit function $G$

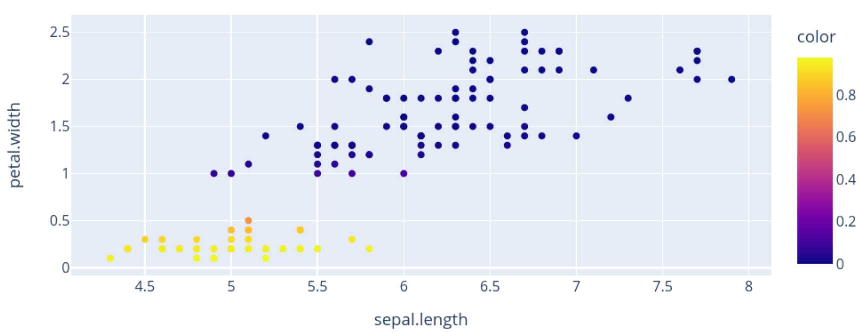
$$y = G(p + qG(z_1) + rG(z_2))$$

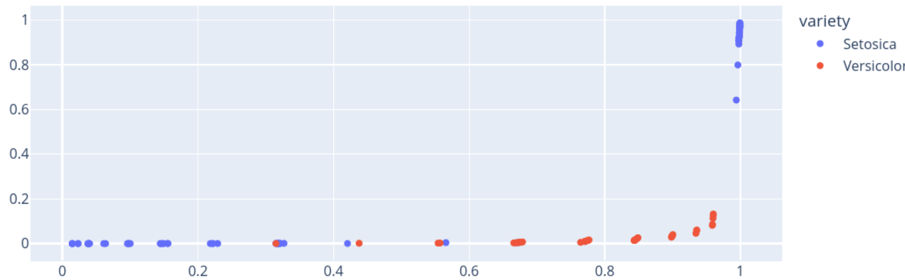
- In total I have nine parameters (right?)
- **Fit this**
- It would be most appropriate to use Pytorch, for the purposes of this workshop, but for a simple model like this, L-BFGS will also do fine

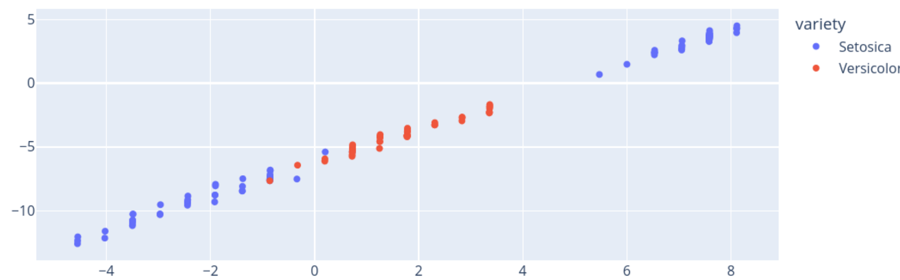
# Magic




1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
variety
Setosica
Versicolor


5
0
−5
−10
−4
−2
0
2
4
6
8
variety
Setosica
Versicolor