1. Introduction to models

The usual setup for a basic Machine Learning model looks like this:

The feature is a property of the data that can be used to classify the same data.

The model is essentialy a function that can be tuned to use a feature and classify the data.

2. Basic models

A usual function used in binary (2-object) classification is a sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-wx+b}}$$

`w` controls the steepness or slope of the function, while `b` moves the function left and right so that the function can fit the data better, and map the right values to the right data.

The sigmoid function outputs values that range between 0 and 1, and its input values range from -∞ to + ∞. This means we can use any feature as a real number and the model can classify our data based on this. Low input values output numbers close to zero, while high input values output values close to 1. We can control how high or how low these values need to be so that the data is classified correctly.

This process of letting the computer figure out the best numbers to fit our data is called Training.

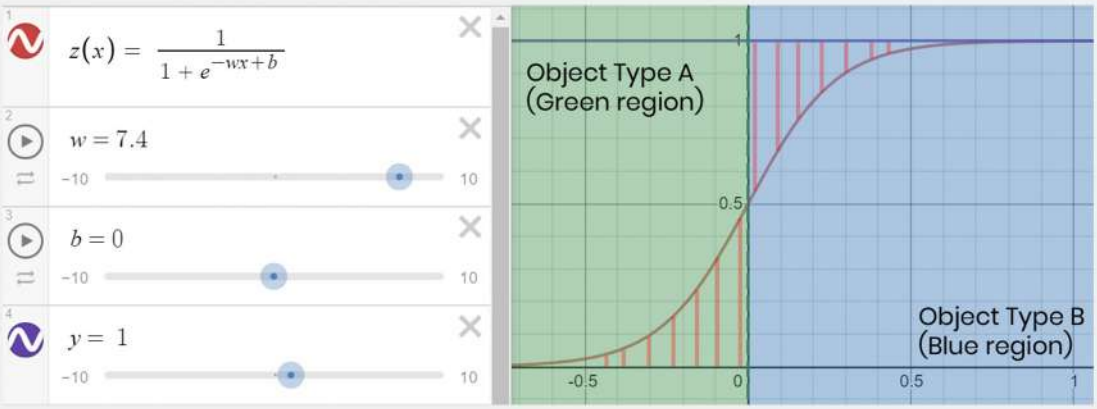
3. Training

To define how bad the model currently is, we define a cost function that measures the deviation from the expected value. One such function is the Mean Squared Error (MSE). It is the mean of the squares of the deviation of the model-returned value from the expected value:

$$MSE = \frac{\sum_{i=1}^{n} (y_i - z_i)^2}{n}$$

where: n = number of predictions made by model
y = expected value per prediction
z = model-output value for prediction

Graphically, the deviation can be represented by the length of the lines as shown:



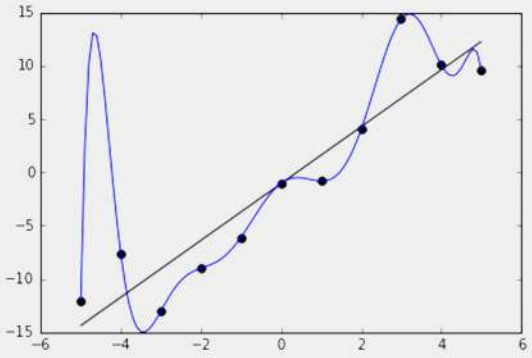
The length of the red lines is the deviation: Objects of type B are encoded as 1 while Objects of type A are encoded as 0.

So, the deviation of the model is the absolute diference between the expected value (0 or 1) and the function's output. The deviations are the lengths of the lines.

The cost of this model is the mean of the squres of these deviations.

4. Overfitting & Model Complexity

Sometimes, training the model too much can be counterproductive: if the model fits the training data too well, then the model will not generalise to new data in some cases and will perform much worse than the moderately trained model.



In the above image[1], we have some noisy data (the points in black), and black line (the linear line of best fit) and a blue line (polynomial line of fit). Clearly, the black line models the data better than the blue line since it has the same general trend as the data. Even though the blue line passes through the data points (it will yield 100% accuracy when tested on the same points), and the black line deviates from the data points, the less numerically "accurate" model does a better job in representing our data.

In other words, the black line generalises better on new data than the more complex model. This can especially be seen around -5 on the x-axis on the same image; the black line models the data accurately while the blue line is way off the general trend of our data.

5. Gradient Descent

Since we need to minimize the cost to our model, we must adjust `b` and `w` to minimize loss. This is what happens when we train our model. In short, we must nudge our `w` and `b` in the direction where the cost decreases.

$$\frac{\partial MSE}{\partial b} \qquad \frac{\partial MSE}{\partial w}$$

We must nudge our w and b in the negative of the directions given by the above partial derivatives as we want to decrease the MSE. This 'nudging' our parameters into their correct place is called Gradient Descent.

6. Handling multiple inputs to model

So far, we have only discussed ways to use the sigmoid function on one input variable: `x`. Now, let us have a look at a sigmoid function that takes in account multiple parameters and classifies data accordingly.

So, for our multiple values of `w` for each `x` value we get, we define two vectors: The `X` vector, and the `W` vector. That way, we can use their dot product to get the output value of our sigmoid function. Now, our sigmoid function is:

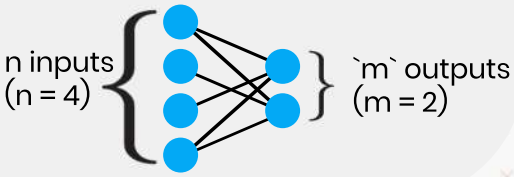
$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \sigma(X) = \frac{1}{1 + e^{-W \cdot X + b}}$$

7. Handling multiple outputs from model

If we want our model to output multiple values for each data point, we define an entire matrix for x. Also, our sigmoid function will now look like this:

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} \quad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \sigma(X) = \frac{1}{1 + e^{-(W \times X) + b}}$$

Our model now has `n` inputs and `m` outputs.
It is common to show the model in a short pictorial way as shown on the right.



Footnotes:

[1] Image by Ghiles on Wikimedia Commons. Image is licensed under the CC BY-SA 4.0 License.
https://commons.wikimedia.org/wiki/File:Overfitted_Data.png
[2] Background by Raul Gaitan on Toptal. Image is licensed under CC BY-SA 3.0 License.
https://www.toptal.com/designers/subtlepatterns/memphis-colorful/ (License is in footer)
[3] Fonts used: Roboto Mono, Poppins on Google Fonts, and Computer Modern (as Latex)licensed under the Apache license 2.0, Open Font license and Open Font License respectively.
https://fonts.google.com/specimen/Poppins
https://fonts.google.com/specimen/Roboto+Mono
https://www.fontsquirrel.com/license/computer-modern

Thanks to:
Julia Academy (for the course), Chris Rackauckas (for the course) and Kyle Woodward (for his Latex to png converter)
Latex to png converter: https://kylewoodward.com/latex.php