Advanced Modelling

We did the basic modelling and found out that tuned Light Gradient Boosting machine performs really well with a F1-score of 0.93. Now we move on to some advanced modelling. We will use a neural network for doing so.

Artificial Neural Network

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity. Tasks in speech recognition or image recognition can take minutes versus hours when compared to the manual identification by human experts. One of the most well-known neural networks is Google's search algorithm.

How do neural networks work?

Think of each individual node as its own linear regression model, composed of input data, weights, a bias (or threshold), and an output. The formula would look something like this:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

$$\text{output} = f(x) = \begin{cases} 1 \text{ if } \sum w_1 x_1 + b \geq 0 \\ 0 \text{ if } \sum w_1 x_1 + b < 0 \end{cases}$$

Once an input layer is determined, weights are assigned. These weights help determine the importance of any given variable, with larger ones contributing more significantly to the output compared to other inputs. All inputs are then multiplied by their respective weights and then summed. Afterward, the output is passed through an activation function, which determines the output. If that output exceeds a given threshold, it "fires" (or activates) the node, passing data to the next layer in the network. This results in the output of one node becoming in the input of the next node. This process of passing data from one layer to the next layer defines this neural network as a feedforward network.

Let's break down what one single node might look like using binary values. We can apply this concept to a more tangible example, like whether you should go surfing (Yes: 1, No: 0). The decision to go or not to

go is our predicted outcome, or y-hat. Let's assume that there are three factors influencing your decision-making:

1. Are the waves good? (Yes: 1, No: 0)
2. Is the line-up empty? (Yes: 1, No: 0)
3. Has there been a recent shark attack? (Yes: 0, No: 1)

Then, let's assume the following, giving us the following inputs:

- X1 = 1, since the waves are pumping
- X2 = 0, since the crowds are out
- X3 = 1, since there hasn't been a recent shark attack

Now, we need to assign some weights to determine importance. Larger weights signify that particular variables are of greater importance to the decision or outcome.

- W1 = 5, since large swells don't come around often
- W2 = 2, since you're used to the crowds
- W3 = 4, since you have a fear of sharks

Finally, we'll also assume a threshold value of 3, which would translate to a bias value of –3. With all the various inputs, we can start to plug in values into the formula to get the desired output.

Y-hat = (1*5) + (0*2) + (1*4) – 3 = 6

If we use the activation function from the beginning of this section, we can determine that the output of this node would be 1, since 6 is greater than 0. In this instance, you would go surfing; but if we adjust the weights or the threshold, we can achieve different outcomes from the model. When we observe one decision, like in the above example, we can see how a neural network could make increasingly complex decisions depending on the output of previous decisions or layers.

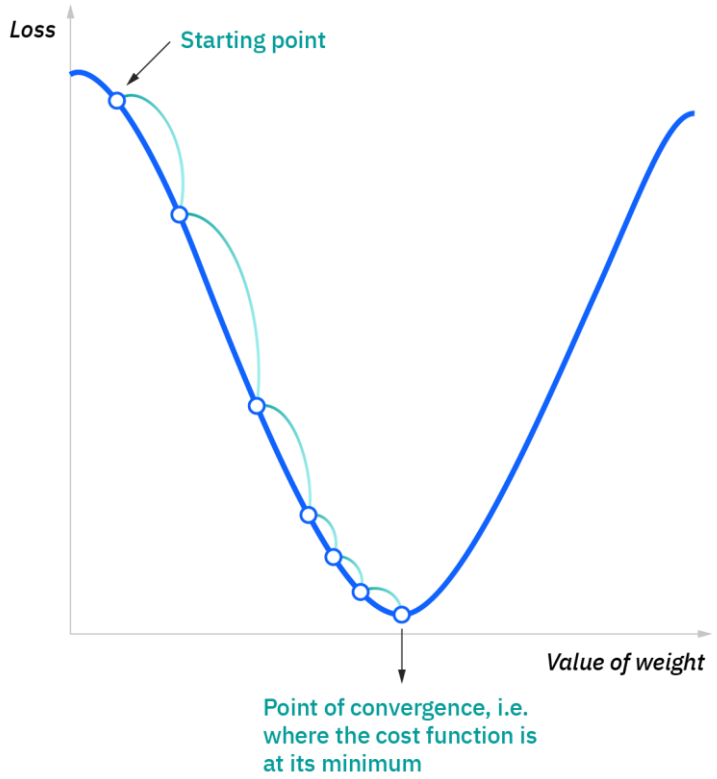
In the example above, we used perceptrons to illustrate some of the mathematics at play here, but neural networks leverage sigmoid neurons, which are distinguished by having values between 0 and 1. Since neural networks behave similarly to decision trees, cascading data from one node to another, having x values between 0 and 1 will reduce the impact of any given change of a single variable on the output of any given node, and subsequently, the output of the neural network.

As we start to think about more practical use cases for neural networks, like image recognition or classification, we'll leverage supervised learning, or labeled datasets, to train the algorithm. As we train the model, we'll want to evaluate its accuracy using a cost (or loss) function. This is also commonly referred to as the mean squared error (MSE). In the equation below,

- *i* represents the index of the sample,
- y-hat is the predicted outcome,
- y is the actual value, and
- *m* is the number of samples.

$$\text{Cost Function} = \text{MSE} = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2$$

Ultimately, the goal is to minimize our cost function to ensure correctness of fit for any given observation. As the model adjusts its weights and bias, it uses the cost function and reinforcement learning to reach the point of convergence, or the local minimum. The process in which the algorithm adjusts its weights is through gradient descent, allowing the model to determine the direction to take to reduce errors (or minimize the cost function). With each training example, the parameters of the model adjust to gradually converge at the minimum.



Most deep neural networks are feedforward, meaning they flow in one direction only, from input to output. However, you can also train your model through backpropagation; that is, move in the opposite direction from output to input. Backpropagation allows us to calculate and attribute the error associated with each neuron, allowing us to adjust and fit the parameters of the model(s) appropriately.

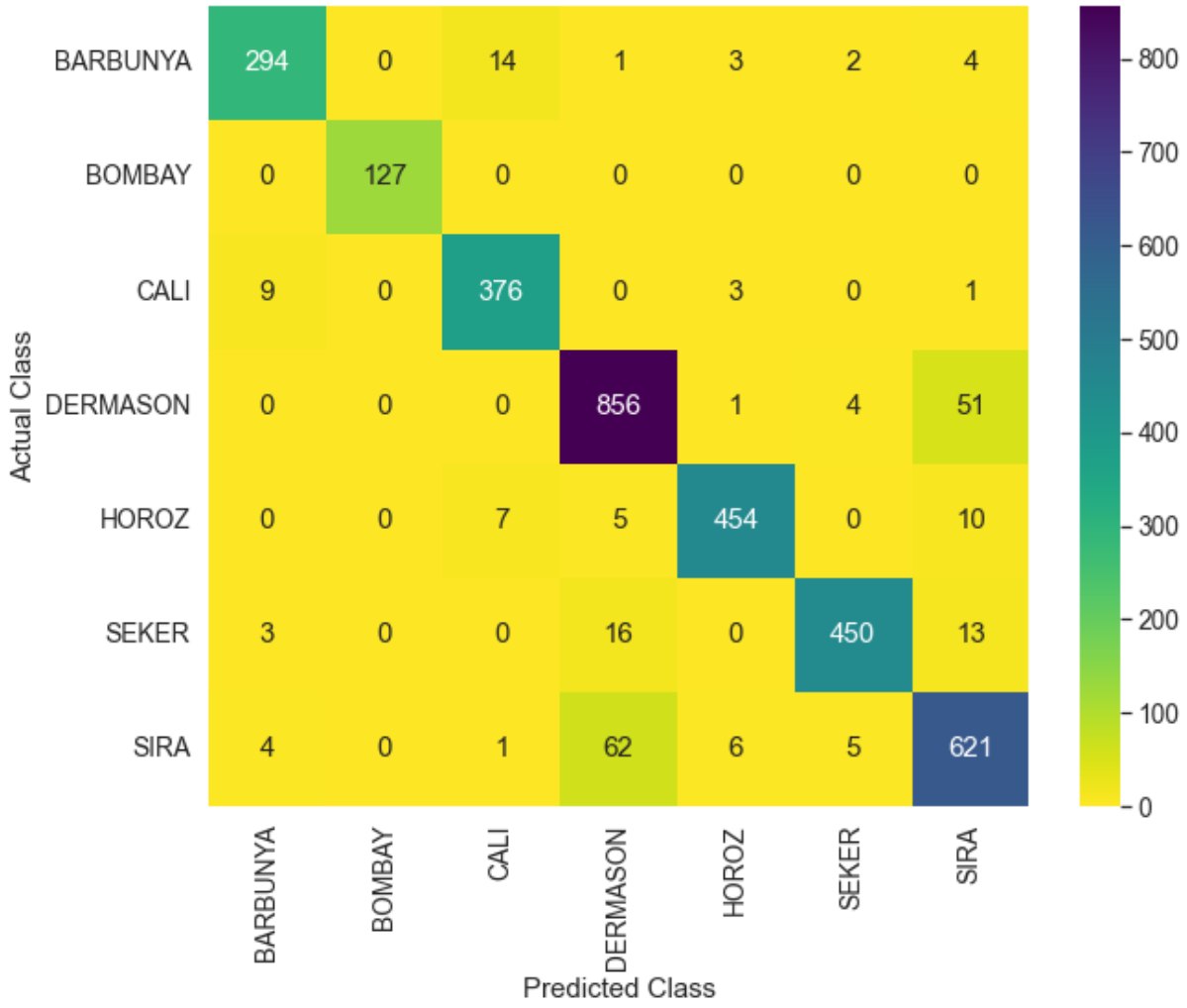
Vanilla Net

So, for our modelling I call our model "Vanilla Net"

1) It's a 2 layer NN with relu activation.
2) The first hidden layer has 512 nodes
3) The second one has 256 nodes
4) Both the layers use `relu` activation
5) Optimizer: `Adam` with a `lr=3e-4`
6) loss: `SparseCategoricalCrossEntropy(logits=True)`
7) Epochs: 20

Results

With resampling (Oversampling with SMOTE) We even beat the best accuracy and f1-score in the paper with an accuracy of **93.39%** & f1-score of **0.9340**


BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Actual Class
294 0 14 1 3 2 4
0 127 0 0 0 0 0
9 0 376 0 3 0 1
0 0 0 856 1 4 51
0 0 7 5 454 0 10
3 0 0 16 0 450 13
4 0 1 62 6 5 621
BARBUNYA
BOMBAY
CALI
DERMASON
HOROZ
SEKER
SIRA
Predicted Class
800
700
600
500
400
300
200
100
0