Logistic Regression

> ⚠ **Warning**
>
> It is considered a classification model, producing discrete outputs.

> **Its name comes from the logistic function (Sigmoid Function):**
>
> $$\sigma(z) = \frac{1}{1 + e^{-z}}$$
>
> Actually this function is called logistic, and it belong to the a whole family of functions called sigmoid.

The key idea behind logistic regression is to take the features, putting it into some kind of linear function and such as $z = f(x) = w_0 + w_1 x$, which then is passed into the sigmoid and its outputs is the final output of the model. This way we use the sigmoid to define some threshold. Because the sigmoid gives as an output with 0 as a center points, we can use this to define the threshold. If the value is negative then it belong into one class and vise-versa.

Changing the linear function, would change this threshold. The slope of the line tells us how steep or how smooth the sigmoid is. And the bias tells us where the 0.5-crossing of the sigmoid is located. This intermediate function is define as:

$$z = \alpha x + \beta$$

Where α is the slope and β is the bias.

$$f_w(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^T \boldsymbol{x}}}$$

Since we have the max at 1 and the min at 0, we can reason the results of this model as a probability. This particular model would be called **Binary Logistic Regression**. As mentioned at the beginning we consider this a classification model because of its discrete outputs, despite of the model given us a probability.

This can be proven just by seeing that the probabilities of the two possible outputs sum to 1:

$$P(y = 0|\boldsymbol{x}, \boldsymbol{w}) + P(y = 1|\boldsymbol{x}, \boldsymbol{w}) = 1$$
$$P(y = 0|\boldsymbol{x}, \boldsymbol{w}) = 1 - P(y = 1|\boldsymbol{x}, \boldsymbol{w})$$

> **This model would be called a Discriminative Model, which its just classifies into to classes. These types of model just care about the classes output, without taking into account its precision. If we care about the precision we would use another kind of models called Generative Models.**

Derivative

The derivative of the sigmoid is:

$$\frac{d\sigma}{dx} = \sigma(1 - x)$$

Therefore, we can define easily the gradient descent method with which we can optimize this model. The only remaining step would be to apply the chain rule in order the get the derivative that we care about. And of course, depending on the loss the final result would be different, as the loss is what we want to derive.

Decision Boundary

As seen before, there is a point when we make a decision, in binary logistic regression, it would be if a number is above or below 0.5. We would call this a **decision boundary**. These surfaces need to be one dimension less as the feature space because we are essentially splitting the space into two, whatever its dimension is.

It would be defined as $\forall x, f_w(x) = 0.5$, which is equivalent to when $\boldsymbol{w}^T \boldsymbol{x}$

Non-Linear DBs

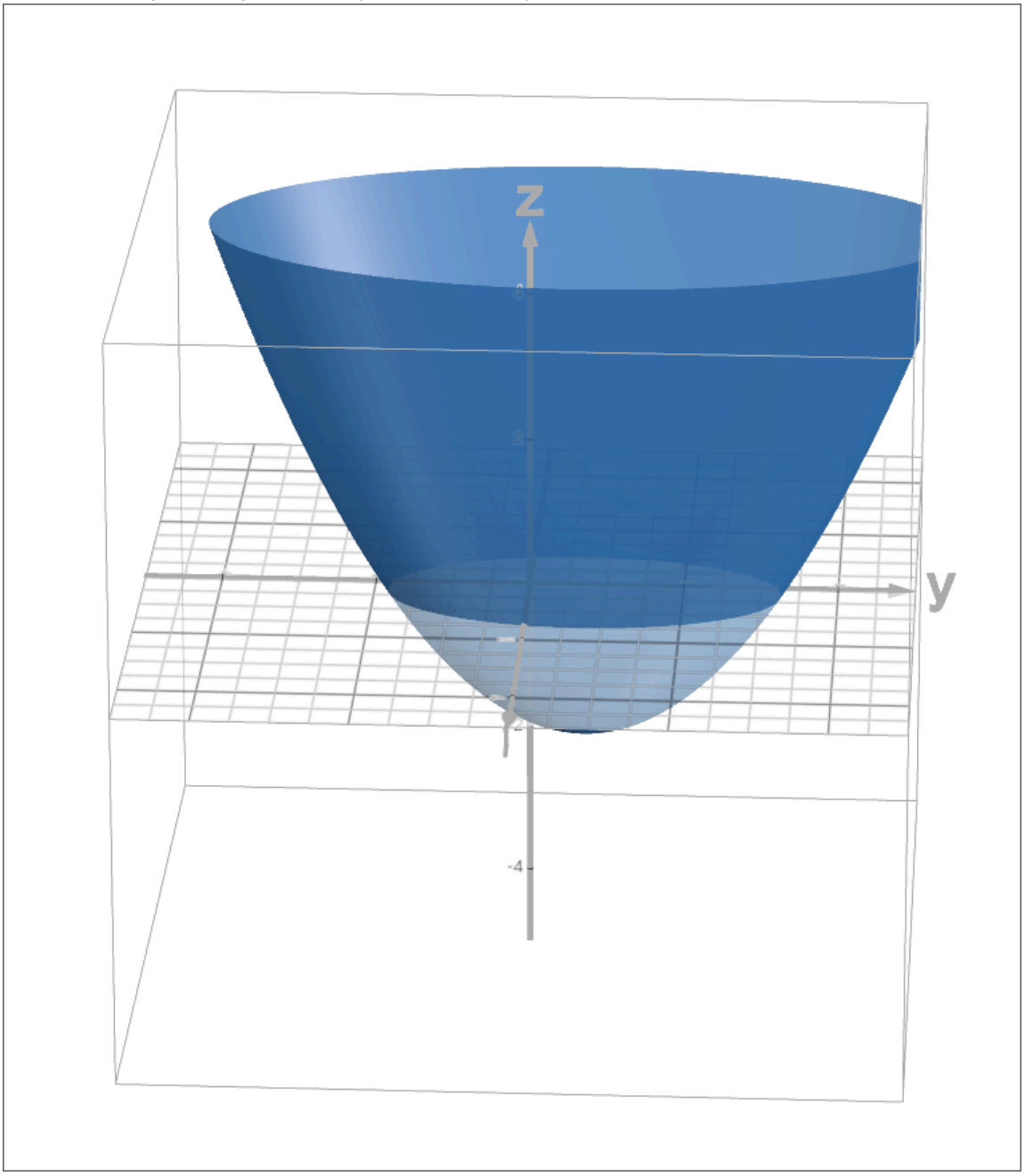
> **Use projections from a surface in the feature spaces with an added dimension to create more complex non-linear decision boundaries.**

We define them as the projection of the sigmoid when its value is equal to 0.5. The input to the sigmoid would this high-dimensional surface (in the feature space + one dimension). The projection can also be defined as the roots of this surface.

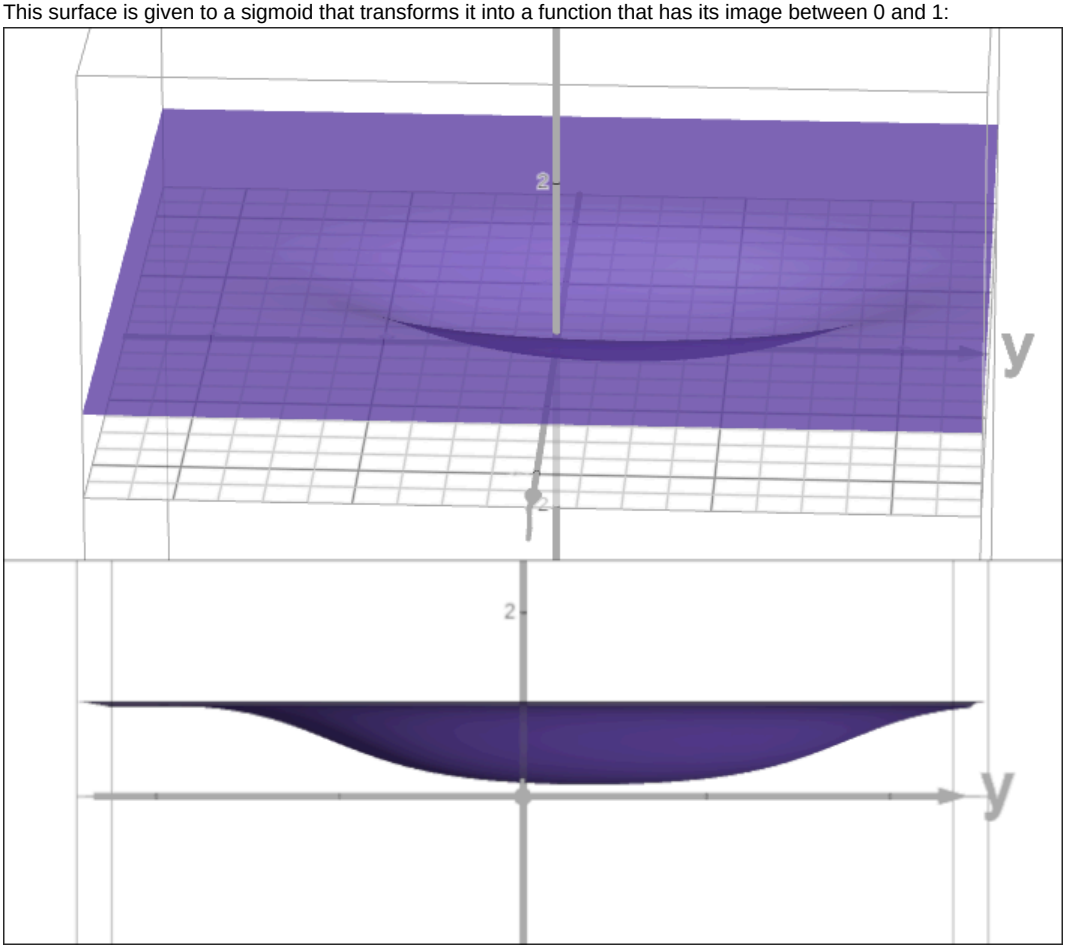
Two ways of defining the decision boundary:

- root of surface
- sigmoid equal to 0.5

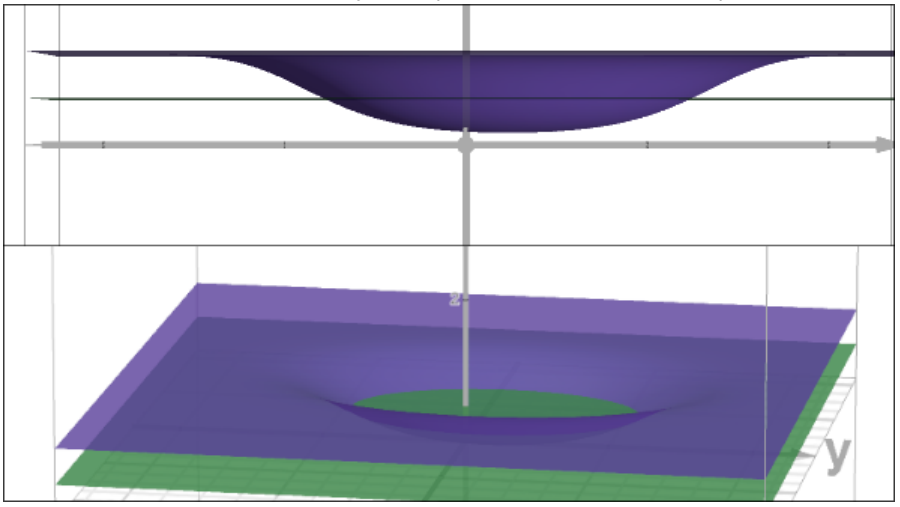
What are we doing is creating a more "complex" function like a parabola:



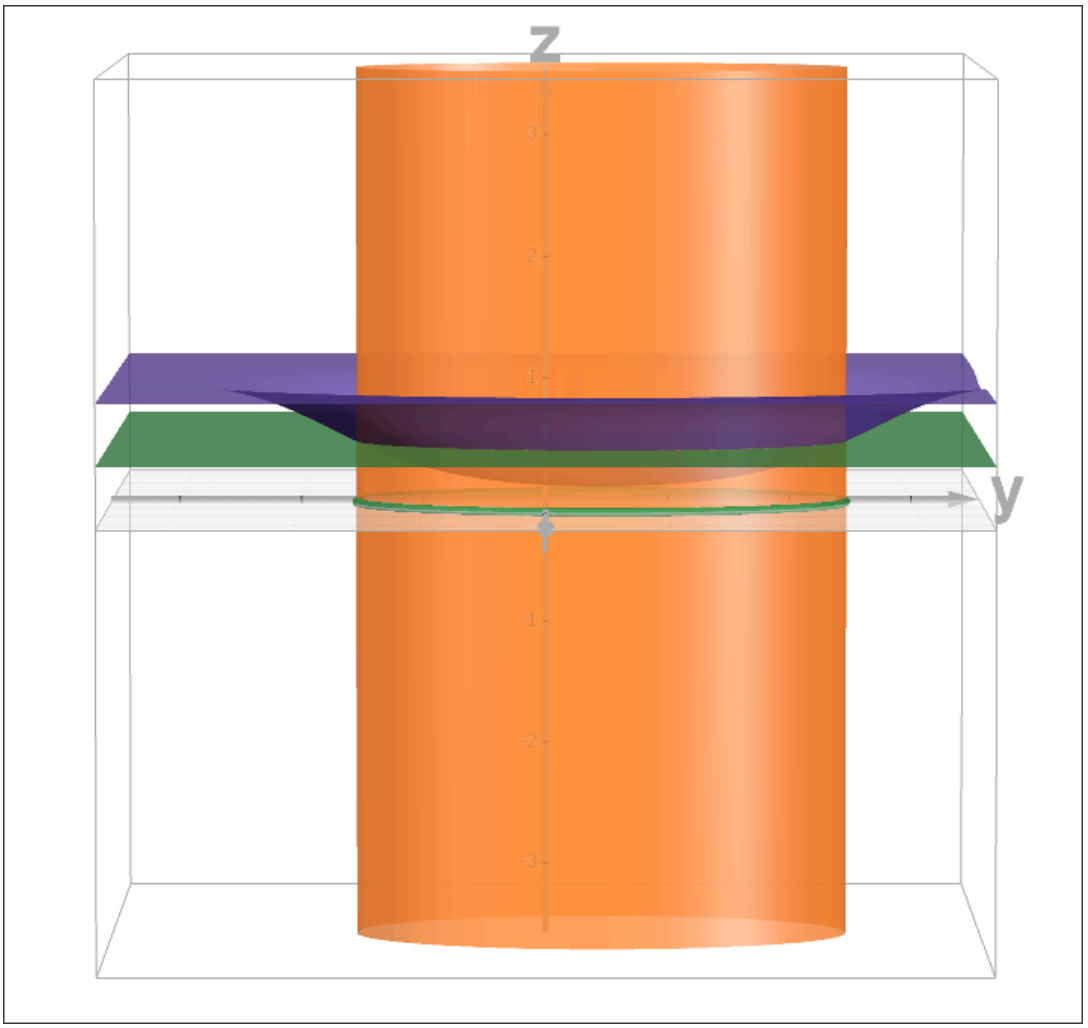
This surface is given to a sigmoid that transforms it into a function that has its image between 0 and 1:

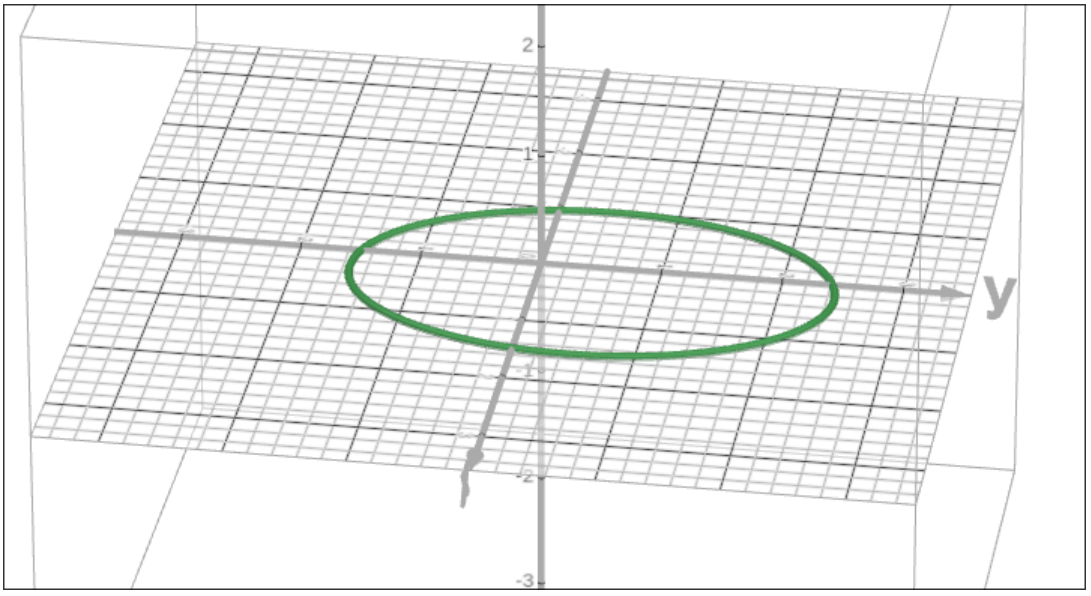


As said before, we define the boundary as the points where this function is equal to 0.5:

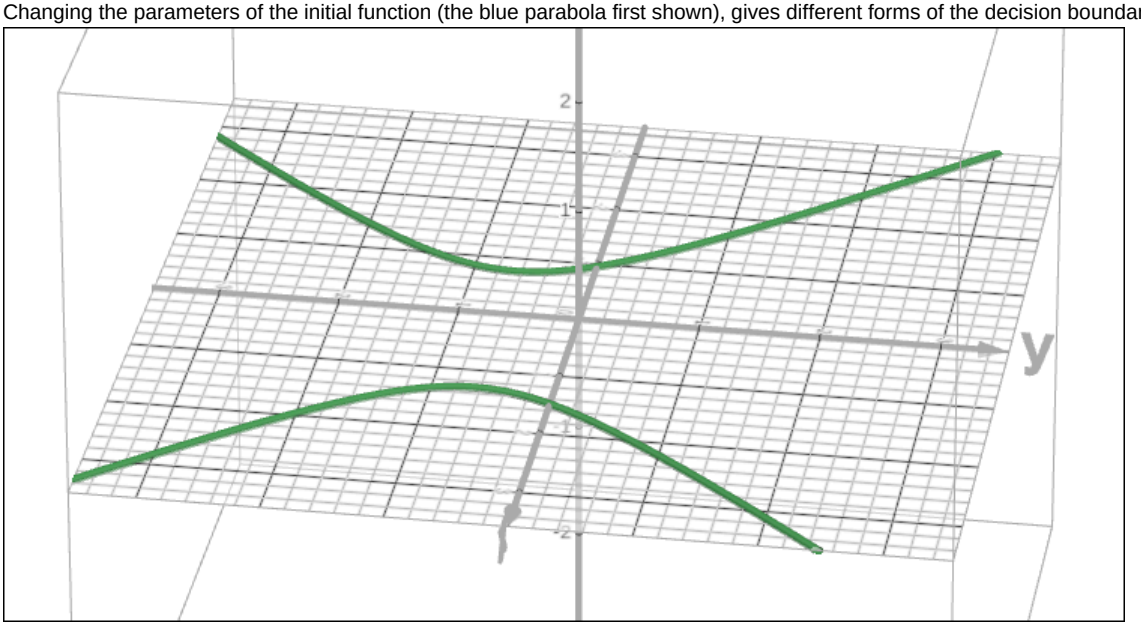


Notices how pictured here its just a plane (in gree), where $z = 0.5$, the decision boundary would just the intersection between these two figures. Which is just equal to the green ring located at $z = 0$:

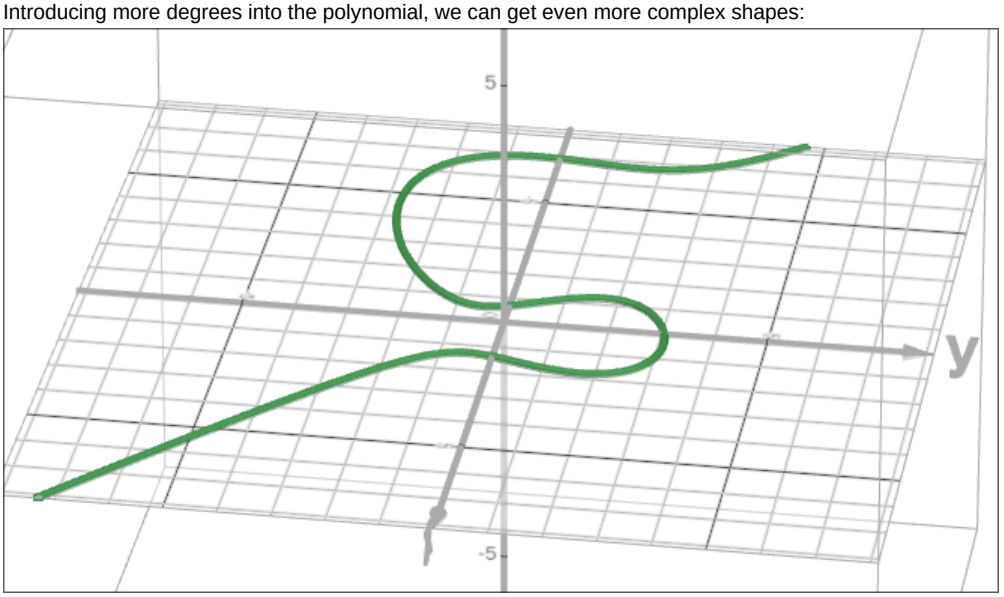

z
3
2
1
y
-1
-2
-3


2
1
y
-1
-2
-3

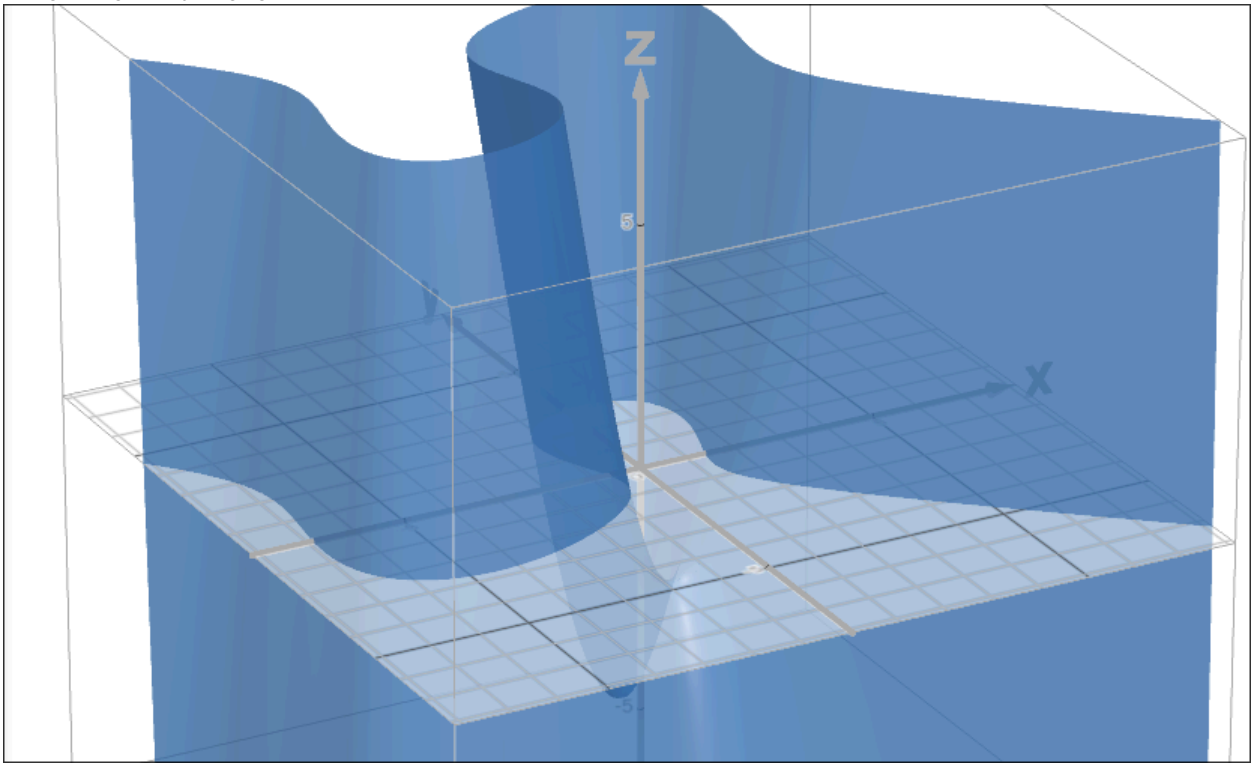
Changing the parameters of the initial function (the blue parabola first shown), gives different forms of the decision boundary:

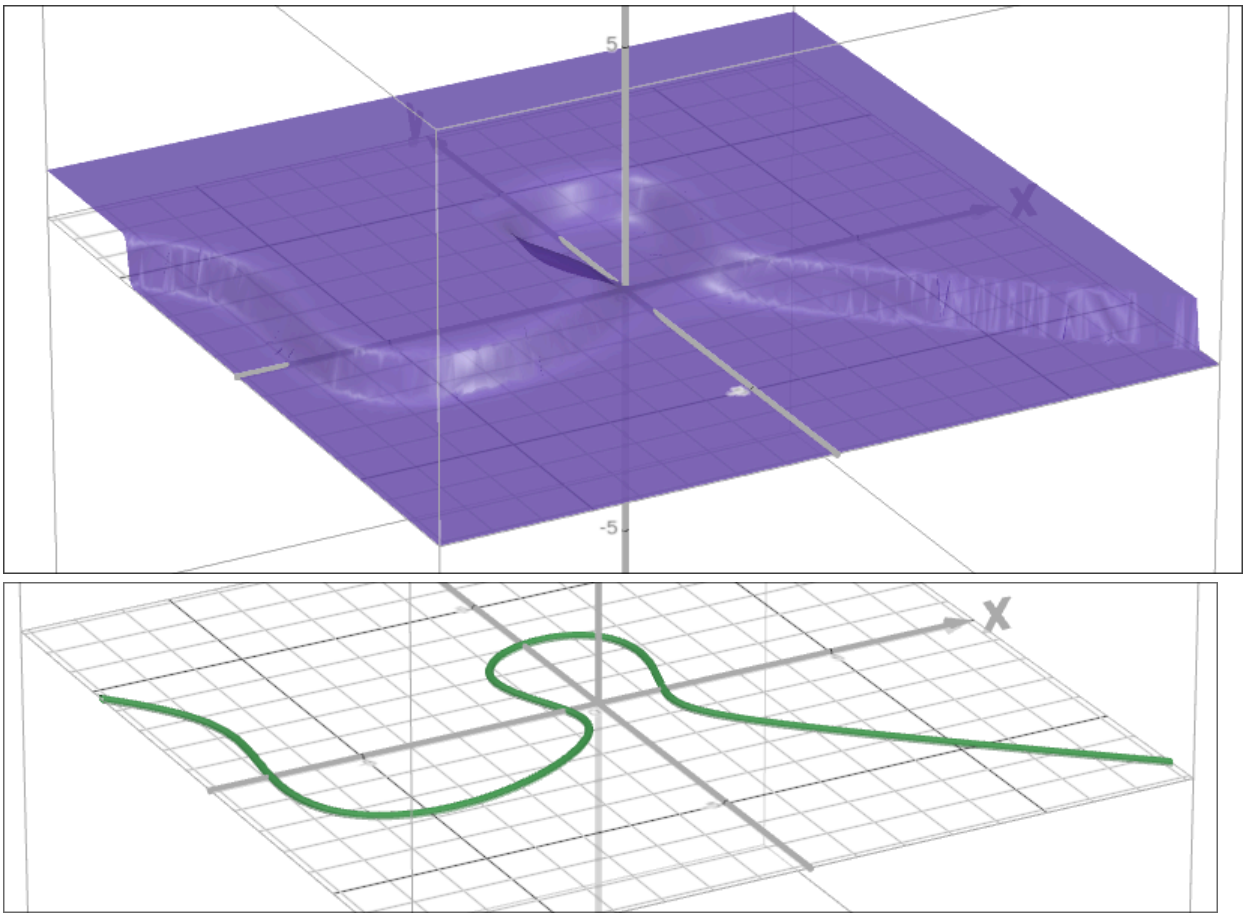


Introducing more degrees into the polynomial, we can get even more complex shapes:



Going through everything again:





PLAY WITH THE GRAPH
I just added more degrees to the function in order to create more complex shapes. Play with the sliders!

Losses For Binary Classification

Mean Squared Error

The most important issue that we have is that the noise that we have in the problem is not Gaussian, therefore using mean squared is not optimal, since Gaussian distributions are designed to work with this loss function historically. We should use another type of loss function for this problem.

Binary Cross Entropy

We can understand how the MSE is not ideal for this type of problem, since the outputs are always binary (i.e. classification problem), using a loss that is designed for a regression problem may not be the way to go.

Thus, we introduce a new type of loss that is designed for binary classification algorithms. Keep in mind that we would like to maximize two things:

- The correct classification (if a datapoint is 0 or 1)
- The probability of each datapoint to be the correct class

In other words, we want to classify the data correctly (choose the class as accurately as possible), and do it with as much certainty as possible (with high probability in the correct class). Therefore, we would like to have a loss function that accomplishes both things.

One good option would be to use a $-\log$ function to input the probabilities into it. When the predicted value is close to 0 it would give a large number, and as it get close to 1, would get smaller:

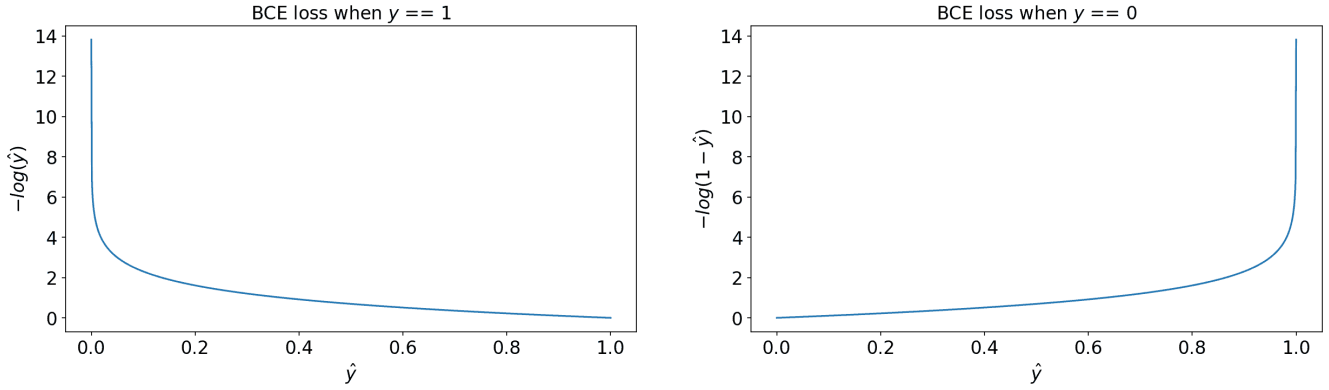
- $-\log(0.01) = 2$
- $-\log(0.5) \approx 0.3$
- $-\log(0.9) \approx 0.045$
  Thus, if we minimize this negative log for the datapoints where the ground truth probability is 1, we are correctly describing what our model should do.

On the other hand, where the ground truth probability should be as close as 1, we can invert the number that goes into the negative log ($-\log(1 - f_w(x))$):

- $-\log(1 - 0.01) = 0.004$
- $-\log(1 - 0.5) = 0.3$
- $-\log(1 - 0.9) = 1$

  How, if we minimize this function, we are teaching our model that the probabilities that need to be 0, are as close as possible to 0.

In other words, with this two function we are punishing our model heavily if the predicted probability is far from the ground truth, and rewarding heavily if its close to it:



Note how it rapidly the loss increases as the output $\hat{y}$ gets very close to the opposite side from the result that we expect. This will result on a model that rapidly learns which results are the complete opposite from the expected ones and corrects them.

The result loss function will be:

$$\begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases}$$

However, having an if-statement in our code can be cumbersome. We can introduce the `if` directly into our loss function:

> **Notice the hidden if statement in the following equation:**
>
> $$-y \log(f_w(x)) - (1 - y) \log(f_w(x))$$
>
> Where y is a binary variables, thus when $y = 1$, the loss is $-\log(f_w(x))$. When $y = 0$, the loss is $-\log(1 - f_w(x))$.

How depending on the ground truth we are optimizing model to get as close to it as possible.

BCE Derivative

The resulting derivative for the complete BCE is the following

$$\begin{aligned} \frac{\partial \text{BCE}}{\partial \hat{y}} &= \frac{\partial}{\partial \hat{y}} \Big[-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \Big] \\ &= -\frac{\partial}{\partial \hat{y}} \Big[\log(\hat{y}) \Big] - \frac{\partial}{\partial \hat{y}} \Big[(1 - y) \log(1 - \hat{y}) \Big] \\ &= -\frac{y}{\hat{y}} - (1 - y) \Big(-\frac{1}{(1 - \hat{y})} \Big) \qquad \Leftarrow \frac{d}{d\hat{y}} \log(1 - \hat{y}) = -\frac{1}{1 - \hat{y}} \\ &= -\frac{y}{\hat{y}} + \frac{(1 - y)}{(1 - \hat{y})} \end{aligned}$$

Uses in Multiclass Classification

> **Instead of a single multiclass classifier, we make several binary classifier, one for each class. They are task to distinguish one class and all of the others. Note that not all classifier are able to do this.**

Having several classes to distinguish between, we can create $f_{\boldsymbol{w}}^{(c)}(\boldsymbol{x})$ for each c class, that just tells us whether $\boldsymbol{x}$ belongs to c or not. That means that we are ignoring the rest of the classes. We would create as many of these classifiers as the numbers of classes that we have.

To predict for some datapoint $\boldsymbol{x}$ we would compare between all the outputs of all the classifiers and pick the one that has the highest probability. In other words, pick the c class that maximizes the probability of $f_{\boldsymbol{w}}^{(c)}(\boldsymbol{x})$:

$$\text{argmax}_c \, f_{\boldsymbol{w}}^{(c)}(\boldsymbol{x}) \iff \text{argmax}_c \, P(y = c | \boldsymbol{x}, \boldsymbol{w})$$

This means that we want to pick the argument c that maximizes $f_{\boldsymbol{w}}^{(c)}(\boldsymbol{x})$ in such a way that also maximizes the probability that the predicted label y is equal to c for some $\boldsymbol{x}$ and $\boldsymbol{y}$.

Train as many classifiers as labels and see which label has the highest probability for some datapoint. Pick that one has the predicted value.