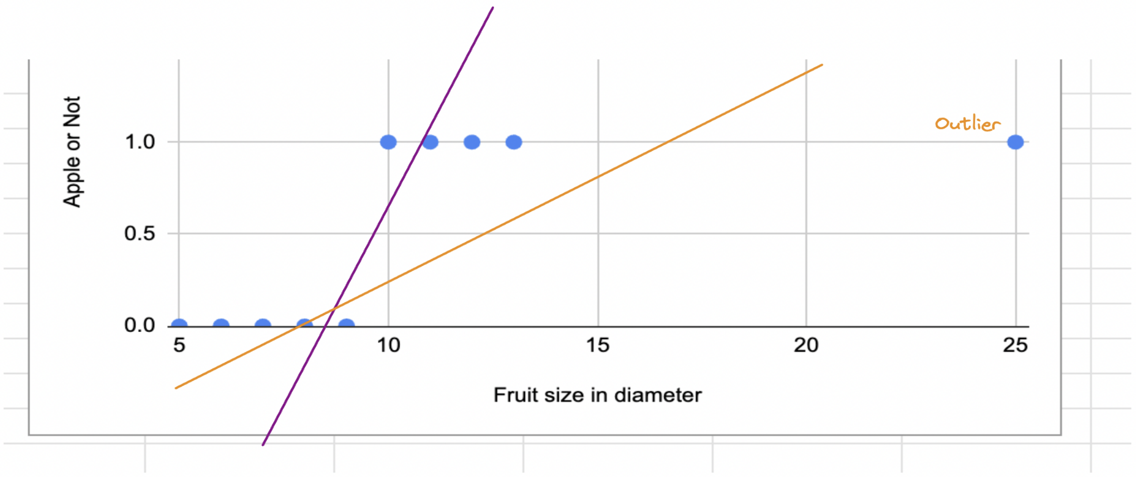
Logistic Regression

Before reading this blog post, please read the post on Linear Regression.

Why we need logistic regression

In supervised learning, the label can be numerical and categorical, where category can be a predefined set of things. In linear regression where we had numerical labels, the output is continuous. When we have categorical labels, the output is discrete. Linear regression works well if the data points are roughly aligned. For example, based on a set of features, if we are classifying whether a fruit is apple (1) or not (0), we get a graph with data points which are discrete and not aligned. Additionally, for discrete outputs, when we have an outlier the linear regression becomes unstable.



In the above example, we can see that one outlier has skewed the line in a significant fashion.

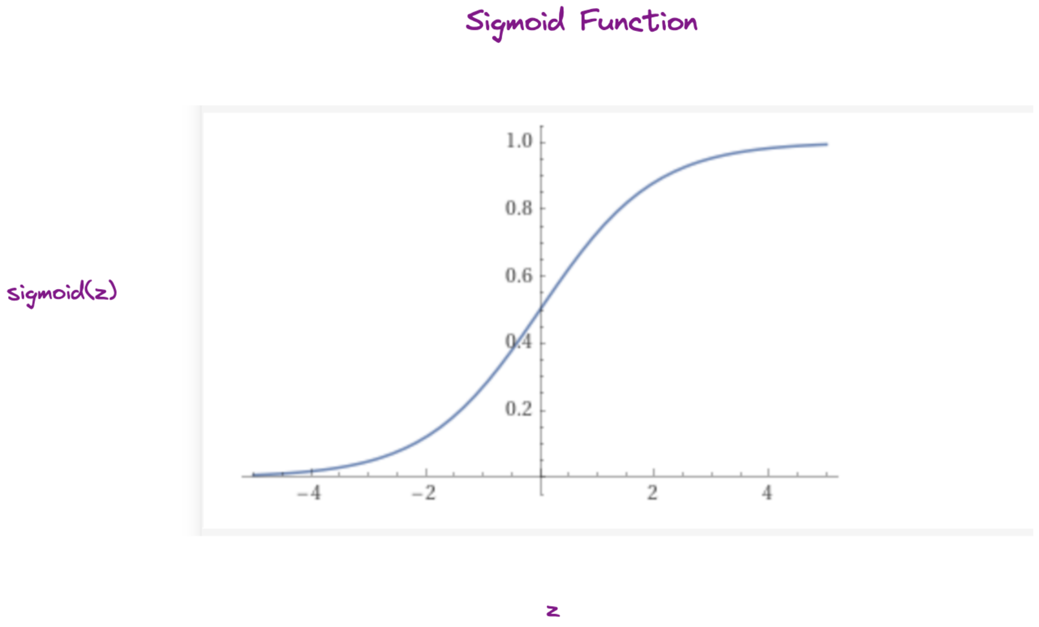
Sigmoid Activation Function

Logistic regression helps in binary classification. To overcome the shortcomings mentioned above, we need a function which can give the output within a certain range, in this case between 0 and 1. So, for logistic regression, we take the weighted sum of inputs and then pass it to a sigmoid function which will compress the output value between 0 and 1.

Weighted sum of inputs = $w_0 x_0 + w_1 x_i + w_2 x_2 + w_3 x_3 \ldots + w_m x_m$ where **w_0** is for bias, **$x_0 = 1$**

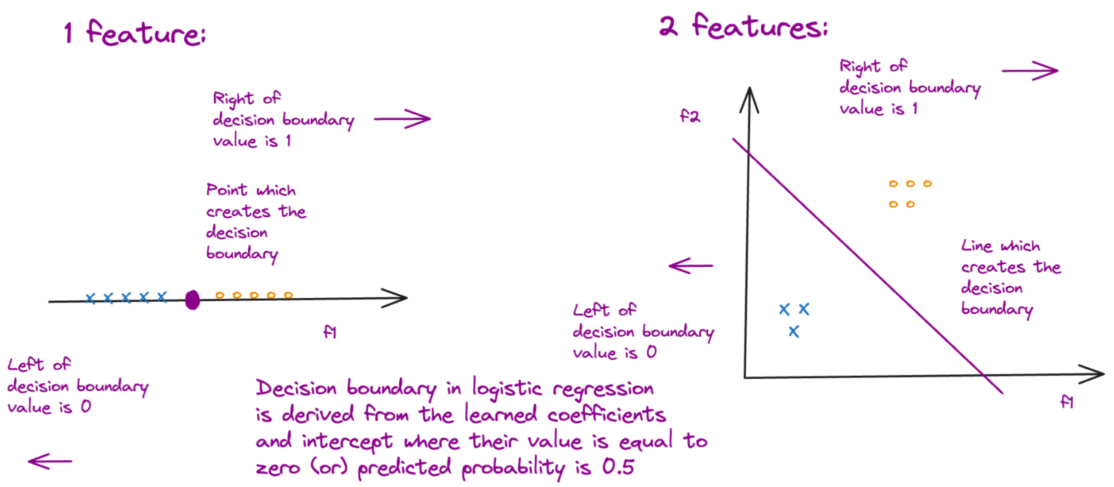
Sigmoid function, σ (z) = $\dfrac{1}{1+e^{-z}}$

Sigmoid function, σ (weighted sum of inputs) = $\dfrac{1}{1 + e^{-(w0\ x0\ +\ w1\ xi +\ w2\ x2 +\ w3x3\ .....\ +\ wm\ xm)}}$



The sigmoid function, also known as the logistic function, is used to transform the linear combination of input features into probabilities as shown above. It smoothly transitions between 0 and 1 but it never reaches the value 0 as well as 1.

Through logistic regression, we end up creating a decision boundary from the learned coefficients and intercept that best fit the data during the training process. The decision boundary happens when the weighted sum of the inputs is equal to 0, which is when the sigmoid function is 0.5. When the weight sum of inputs >= 0, we classify the output to 1 as the probability is greater than 0.5, and when the weighted sum of inputs < 0 we can classify the output to 0 as the probability is less than 0.5. The decision boundary can be shown as below:


1 feature:
Right of decision boundary value is 1
Point which creates the decision boundary
f1
Left of decision boundary value is 0
Decision boundary in logistic regression is derived from the learned coefficients and intercept where their value is equal to zero (or) predicted probability is 0.5
2 features:
Right of decision boundary value is 1
f2
Line which creates the decision boundary
Left of decision boundary value is 0
f1

Log Loss Function

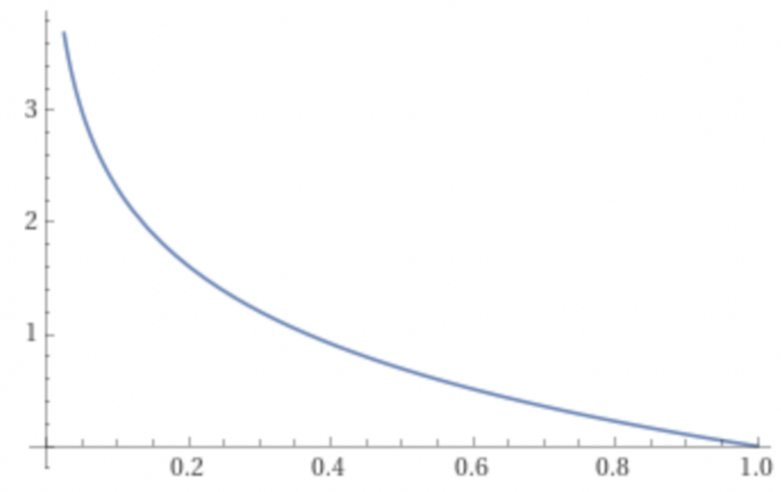
While discussing gradient descent we said the function needs to have 3 properties: **convex**, **continuous** and **differentiable**. Unfortunately, the mean square loss function does not work well with sigmoid functions as the loss function is not convex anymore, which means that we could have many local minima mingled with global minima. We need to use a different loss function called **log loss** function.

Cost function = $- \frac{1}{n} \sum_{i=1}^{n} (y_i * log(y_i^{prime}) + ((1 - y_i) * log(1 - y_i^{prime})$

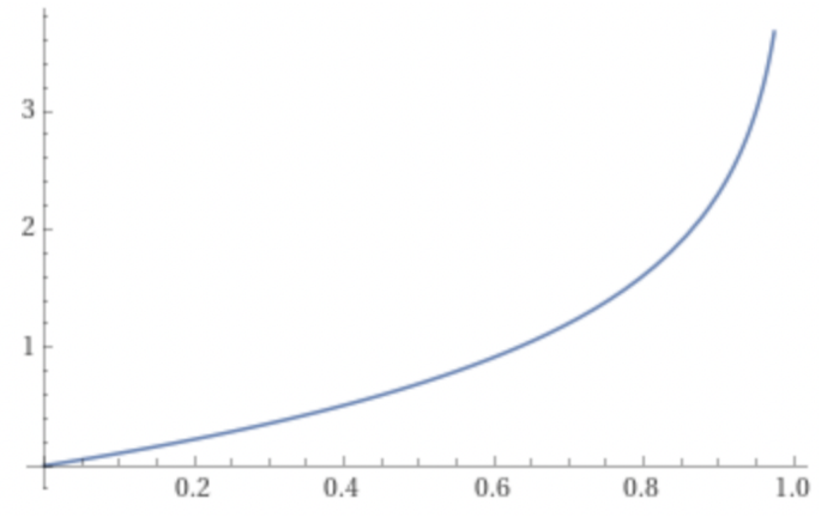
where, y_i- Output label, y_i^{prime} - Predicted value

This function says, if the output label is 1 and the predicted value is closer to 1, the log loss is closer to 0, else it will be a huge value. Similarly, if the output label is 0 and the predicted value is closer to 0, the log loss is closer to 0, else it will be a huge value. We also subtract the overall result, as the log values between 0 and 1 are negative and we want the loss function to be positive so that it will be easy to understand that our optimization goal is to reduce the loss value. It can be shown in the below graphs:

- log(x) between 0 and 1
When prediction is 1, we can see loss is close to zero
When prediction deviates more from 1, larger is the log loss



- log(1 - x) between 0 and 1
When prediction is 0, we can see loss is close to zero
When prediction deviates more from 0, larger is the log loss



Gradient Descent of Log Loss Function

Partial derivative of the log loss function is very similar to the mean squared loss function. The partial derivative of log loss function can be shown as below:

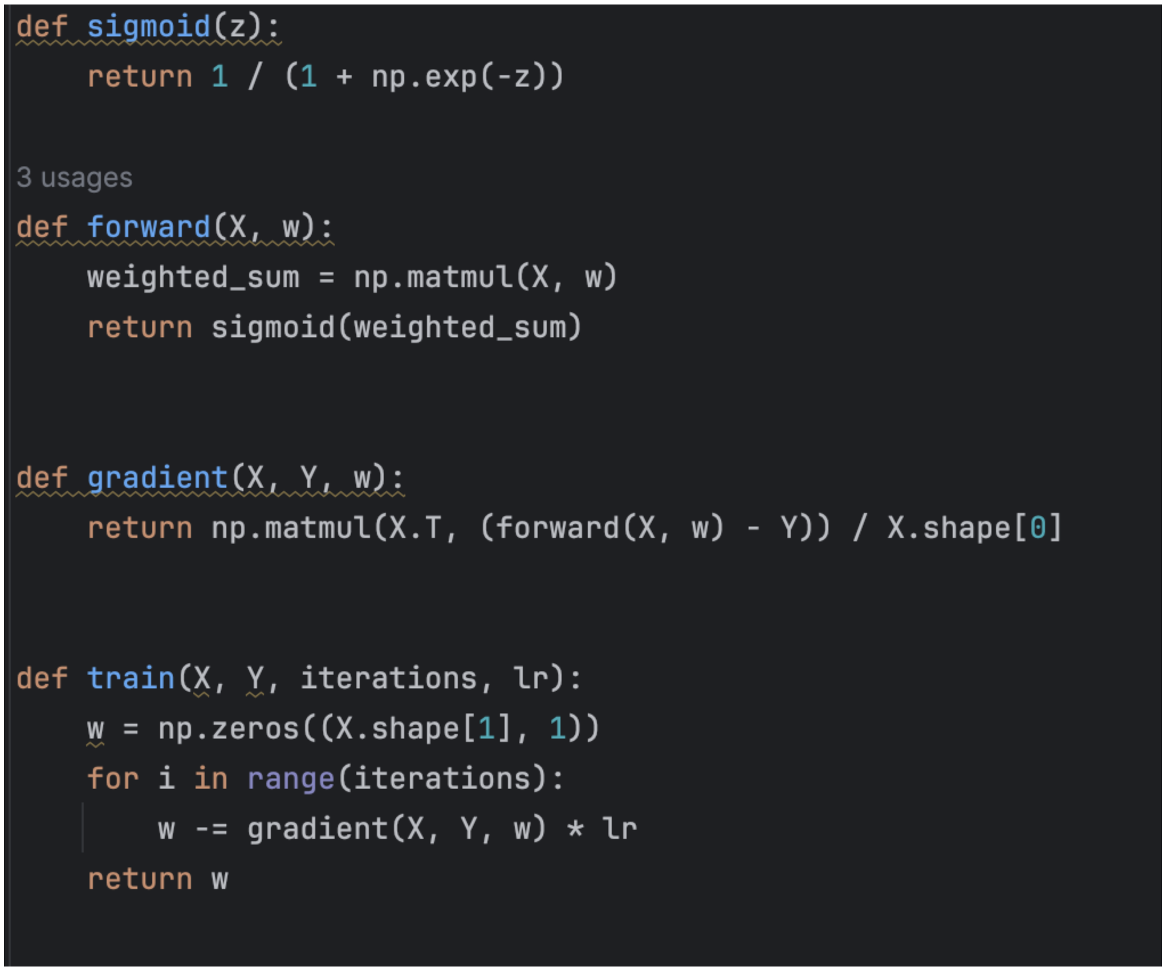
Partial Derivative = ($\sum_{i=1}^{n}$ (($w_0 x_{0(i)} + w_1 x_{1(i)} + \ldots + w_m x_{m(i)}$) - Ground truth) * $x_{m(i)}$) / n

($J(w_m)$ w.r.t w_m)

The derivative can be worked out by knowing the derivatives of a few key functions and applying the **"Chain rule"** of Calculus.

Solution

Similar to linear regression, to solve for logistic regression, all we have to do is compute the partial derivatives for our cost function w.r.t each of the features and using the **magnitude** and **direction** of their derivative value, we will adjust all the feature's weights accordingly so that we find the appropriate weights where the cost function is minimum. The adjustment is made using **learning rate**, which will be a small value like 0.1, 0.01, 0.001 etc.,



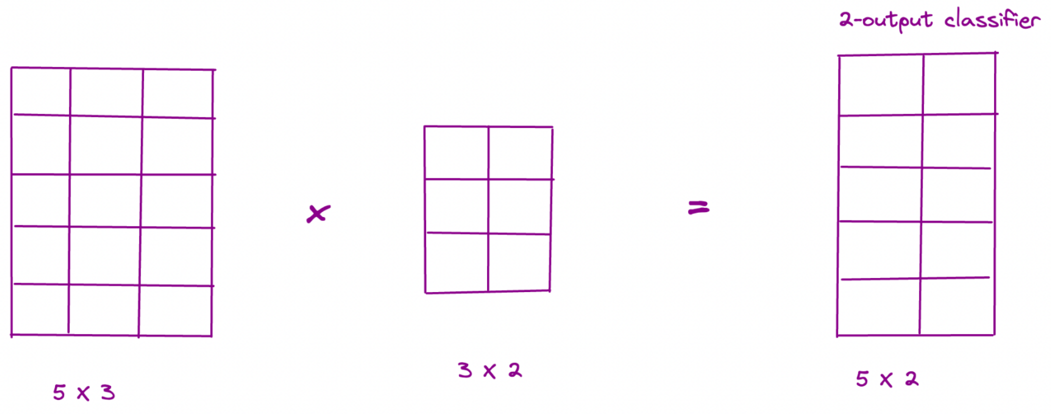
```python
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

3 usages
def forward(X, w):
    weighted_sum = np.matmul(X, w)
    return sigmoid(weighted_sum)


def gradient(X, Y, w):
    return np.matmul(X.T, (forward(X, w) - Y)) / X.shape[0]


def train(X, Y, iterations, lr):
    w = np.zeros((X.shape[1], 1))
    for i in range(iterations):
        w -= gradient(X, Y, w) * lr
    return w
```

Multinomial logistic regression

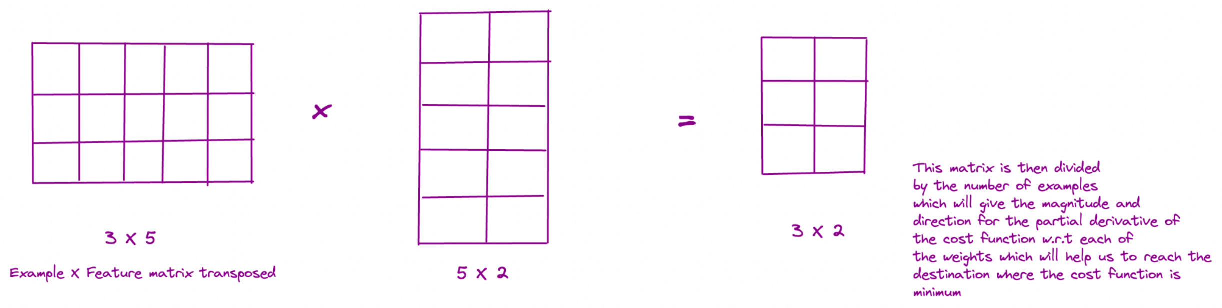
In our example, we used logistic regression to do binary classification, whether it is an apple or not. What if we need to classify a fruit into 4 different types, say whether it is an apple, guava, orange, or lemon. When we need to handle more than 2 classes, we call it multinomial logistic regression.

Solving for multinomial is relatively straightforward using a **one vs all** approach. We simply run the n-examples through all the n-classifiers. During the classification phase, we run the input across all the n-classifiers and pick the one whose probability is highest which is computed using the sigmoid function.

For ex, if we have 5 examples, 3 features(including bias), 2-output classifier, matrix multiplication will be,



For the partial derivatives, now we need to multiply **$x_{m(i)}$** with **y^{prime}** - **Ground Truth**. Again, this is similar to what we did during 1-output classifier and linear regression, it is just that we need to factor in for the other output classifier using matrix columns.


3 x 5
Example x Feature matrix transposed
x
5 x 2
=
3 x 2
This matrix is then divided by the number of examples which will give the magnitude and direction for the partial derivative of the cost function w.r.t each of the weights which will help us to reach the destination where the cost function is minimum

Why logistic regression is considered linear classification algorithm

Logistic regression is considered a linear classification algorithm, even though we use the sigmoid function as the activation function whose output is non-linear after applying multiple linear regression. The reason why logistic regression is considered a linear classification algorithm is because it models the relationship between the independent variables (features) and the log-odds of a binary outcome in a linear way. It can be shown as below:

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}$$

sigmoid transforms real valued numbers into probabilities

$$P = \frac{1}{1 + e^{-z}}$$

Rearranging on both sides,

$$1 + e^{-z} = 1 / p$$

$$e^{-z} = (1 - p) / p$$

Taking natural logrithm,

$$-z = \ln (1 - p) / p$$

$$\boxed{z = \ln p / (1 - p)} \longrightarrow$$

ln p / 1 - p is called log odds or logits

where z is the multiple linear regression equation

Conclusion

In this post, we covered Logistic regression which is used when we need to categorize a set of things. We saw how we combined multiple linear regression model with a sigmoid activation function which made the output non-linear. This will pave the way for us to explore and understand one of the most improved breakthroughs in machine learning, which is neural networks.

References

[Programming Machine Learning : From Coding to Deep Learning]
[Logistic Regression]