---

# PROJECT - 1

Sai Teja Cherukuri (50418484)

10th October 2021

---

## 1. Project Overview

The goal of the project is to perform classification using machine learning. In the first part of the project, I have performed Data Pre-processing. Then, I trained the model using gradient descent for logistic regression and calculated the accuracy by tested the model on the testing set. Later I implemented Neural networks using different regularization methods and then calculated accuracy for each of them.

## 2. Dataset

To implement machine learning models with Pima Indian Diabetes dataset with 768 samples, I have split the data samples as Training, Validation and Testing, each constituting of 60%, 20% and 20% respectively of the overall data. Training dataset has 460 samples, Validation dataset has 154 samples and Testing dataset has 154 samples.

## 3. Python Editor

I have used Jupiter Notebook on Google Collab for implementation and shared.

## 3.1 Data processing

1. **Extract feature values from the data**
   I have processed the given CSV dataset (diabetes.csv) and extracted the features (**Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age)**

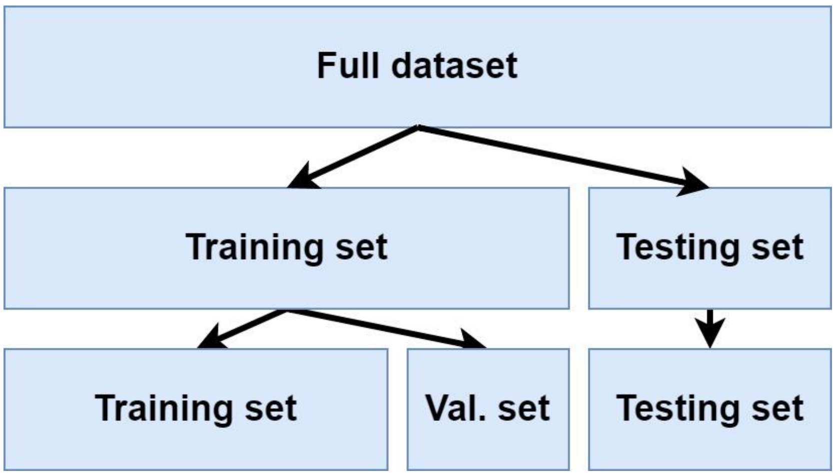
2. **Correlation matrix**
   Correlation matrix is constructed showing correlation coefficients between features.

3. **Data Normalization**

- Normalization of data is done to scale the feature values to a particular range, else they will over-estimate or under-estimate the importance of each feature.
- Next, I removed the Outcome column from the dataset to extract the features alone.
- Now process the feature dataset to contain values between 0 and 1 inclusive, by using the Min-Max Scaler method from sklearn.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

4. **Data Partitioning**
I have partitioned the data into Training, Validation and Testing sets using *train_test_split* function from sklearn in 60%, 20% and 20% manner.



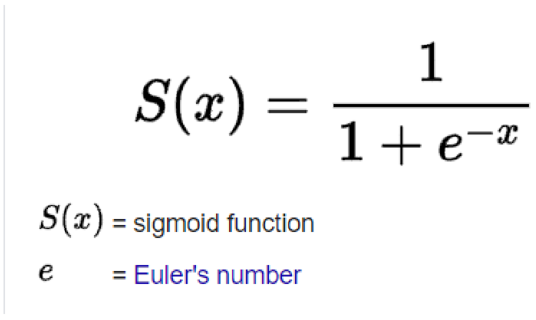
5. **Model Building**

**Part – 1: Implementing Logistic Regression**
First, I have applied Transpose for Training, Validation and Testing matrices so we can perform multiplications.

Then I have initialized the weights as numpy array having the dimension as one and filled with 0.01 and bias is initially assigned to zero.

Then taking the pre-processed dataset, I have defined the sigmoid activation function and gradient descent. Logistic regression is mainly used for binary classification, hence the outputs are always between 0 and 1. The role of sigmoid function is to convert linear model to predicted output.

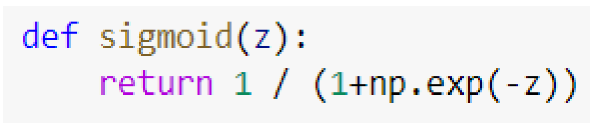
Here we take a linear model (z) of form **Z = W*X + B ,** where
W is the weights matrix and B is the bias



$$S(x) = \frac{1}{1 + e^{-x}}$$

$S(x)$ = sigmoid function

$e$ = Euler's number



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

Now, I have used the Forward-Backward propagation to find gradient descent and cost function.

Forward propagation:

- For the Forward propagation, we need to find to value of z. It is expressed in the form of linear expression of line.

$$z = \sigma(W^T * X + b)$$

```
z = np.dot(w.T,train_x) + b
```

- Then apply sigmoid function of z to calculate cost of the function.

**A = sigmoid(z)**

- Then we find the loss for each feature by the below equation.

**Loss = Y * log (A) + (1 – A) * log (1 – A)**

- Next we find the Cost function which is Summation of losses of all features divided by total number of data entries.

**Cost function :**

$$cost = -\frac{1}{m}\sum_{i=1}^{m}[Y * log(A) + (1 - A) * log(1 - A)]$$

Backward propagation:

- For the Backward propagation, we need to traverse the graph downward, hence we find derivative. (formulae as below)

$$dW = \frac{\partial COST}{\partial W} = (A - Y) * X^T$$

$$dB = \frac{\partial COST}{\partial B} = (A - Y)$$

- Hence using Forward and Backward propagation, gradient descent is applied and we get the derivative weight and derivative bias.

Now I declared Logistic regression function using training data, learning rate and Epochs (number of iterations):

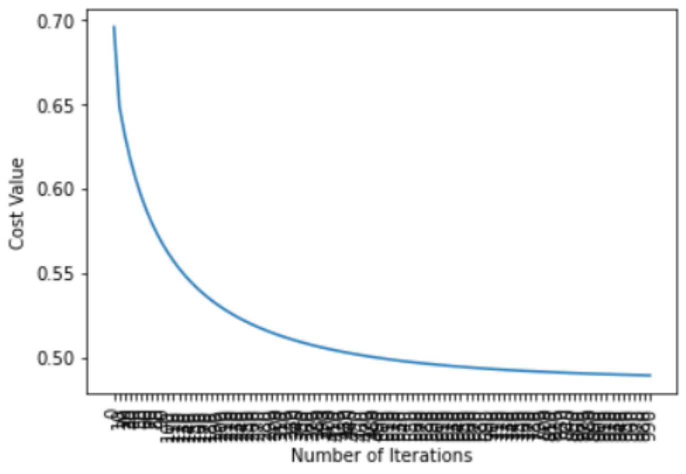
1. Inside the function, we call the **Update** function with training data, learning rate and Epochs to update the Weights and Bias.
2. Using the Weight and Bias parameters, we call the **Prediction** function passing these as parameters
3. In the Predict function, I have taken the default threshold as 0.5 and made the prediction (If data value is less than 0.5, then we predict as 0 (meaning the patient has no diabetes), else if the data value is greater than 0.5, then we predict as 1 (meaning the patient has diabetes)

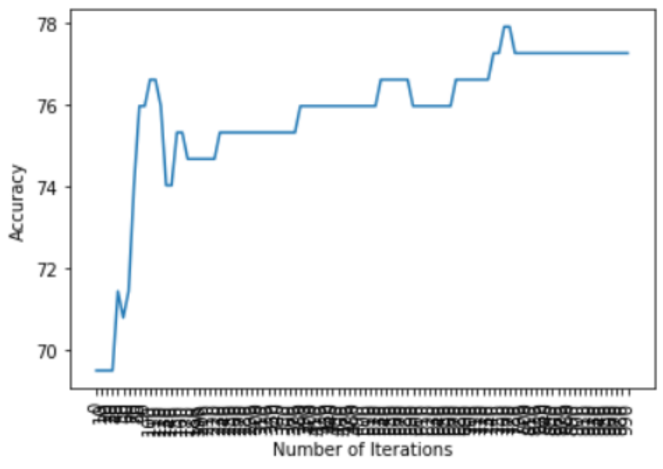
**Visualization for Validation dataset**

I have plotted two graphs as below :

1. Cost vs Number of Iterations
2. Accuracy vs Number of Iterations

Accuracy achieved = **77.27 %** .



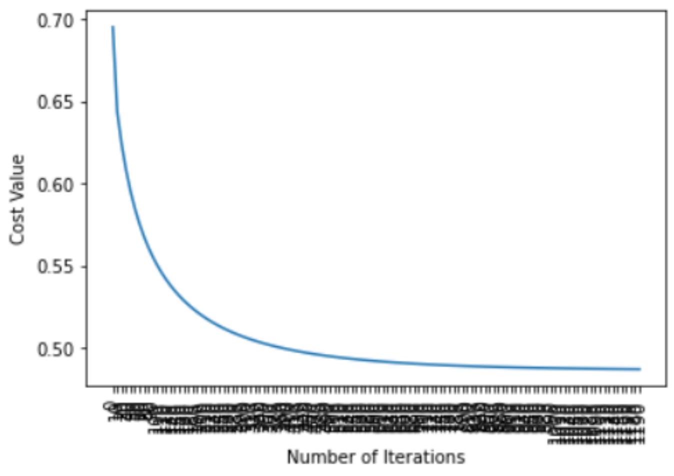


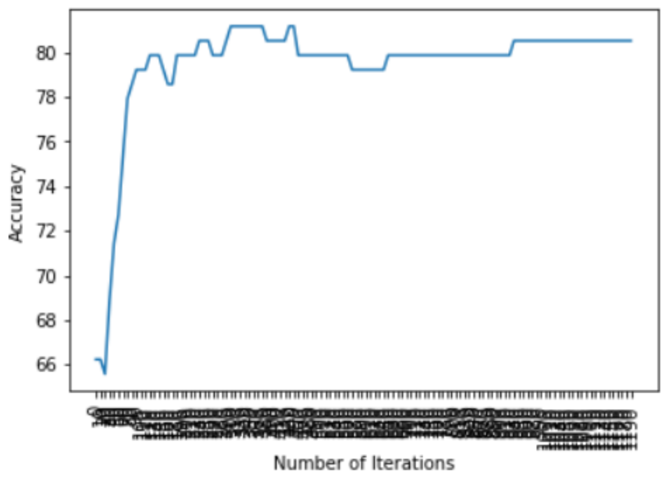
**Visualization for Testing dataset**

I have plotted two graphs as below :

1. Cost vs Number of Iterations
2. Accuracy vs Number of Iterations

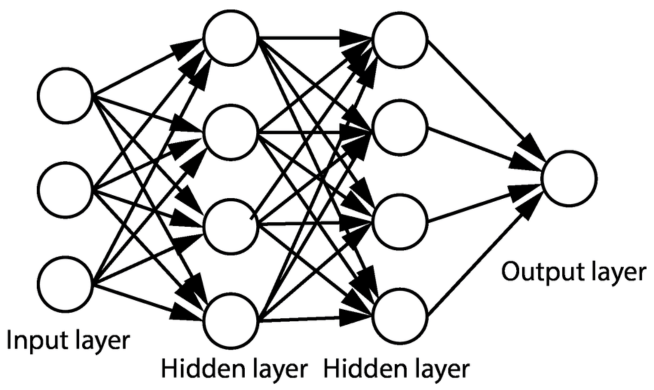
Accuracy achieved = **80.51 %**





**Part – 2: Implementing Neural Networks**

- First we load the libraries Sequential and Dense from Keras.
- Now, we start building the artificial neural network which is a sequential model of 3 layers.
  1. The first layer will have 12 neurons and uses the ReLu activation function.
  2. The Second layer will have 15 neurons and used ReLu activation function.
  3. Last layer which is the Output layer has only 1 neuron which uses Sigmoid function.



- Next, I have used a Regularization technique in the model to reduce model overfitting.
  - I have used L1 Regularization for first layer and L2 Regularization for second layer.

  Regularization is a technique which makes modifications to the algorithm such that the machine learning model becomes better.

  Here, I have used L1 and L2 regularization methods which are the most common types of regularization. These update the cost function of the model by adding a regularization term.

$$\textit{Cost function} = \textit{Loss (say, binary cross entropy)} + \textit{Regularization term}$$

Due to the addition of this regularization term, the values of weight matrices decrease because it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting of the model.

L2 Regularization:

- This regularization is also known as Weight decay. This method forces the weights to decay towards zero.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|^2$$

L1 Regularization:

- This method applies a penalty to the absolute value of the weights. Unlike L2 regularization, the weights may be reduced to zero.

$$Cost\,function = Loss + \frac{\lambda}{2m} * \sum \|w\|$$

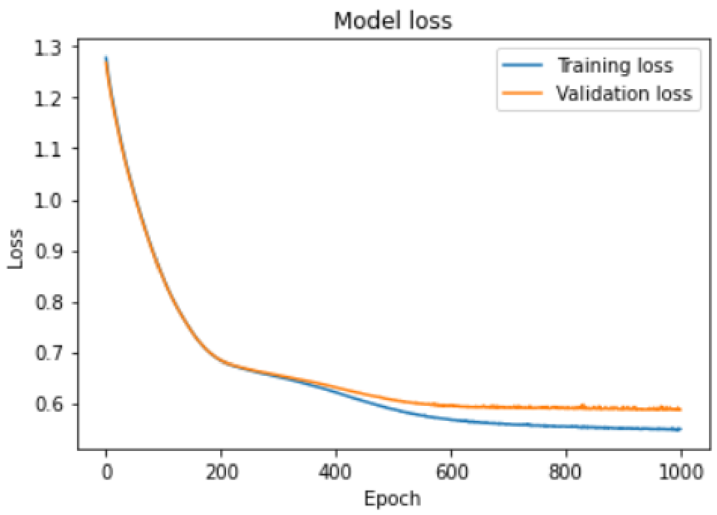
Here lambda (λ) is the regularization parameter, it is the hyperparameter whose value is optimized for better results.

- I have used Regularization parameter (λ) as 0.01
- Now, we compile the model with loss function as 'Binary_crossentropy' (as we need the output to be 0/1) and I have used Stochastic Gradient Descent 'sgd' optimizer to improve upon the loss. I have added 'Accuracy' to the metrics to measure the accuracy of the model.
  - I have used a tunable hyper parameter for SGD optimiser with a learning_rate = 0.01

- Next, we train the model using fit method with a batch size of 32 and 1000 Epochs (Number of iterations).
- Also, gave the model validation data by splitting the training data into 20%.

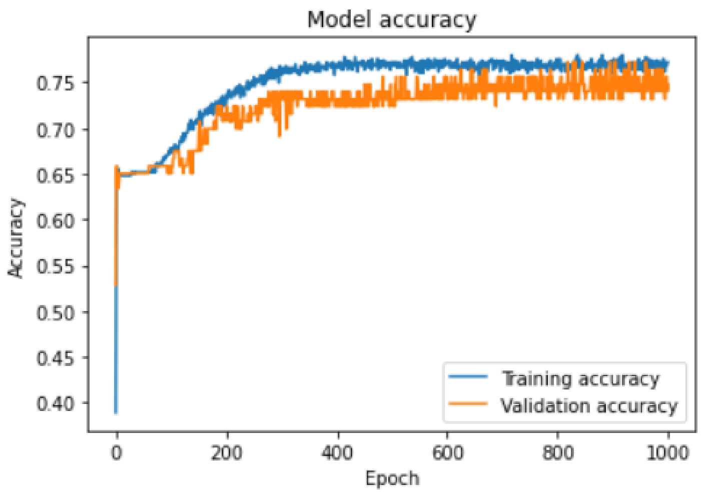
**Visualization for Model's Training Loss and Validation Loss:**

I have plotted the Training and Validation Loss with respect to Epoch as below :



*The model loss is decreasing significantly after 200 Epochs*

**Visualization for Model's Training Accuracy and Validation Accuracy:**
I have plotted the Training and Validation Accuracy with respect to Epoch as below :



- Now, we evaluate the model on the testing data set by predicting that all the values which are less than 0.5 as 0 and values which are greater than 0.5 as 1.

The Testing Accuracy achieved using the above model = **81.16 %**

## Part – 3: Implement different regularization methods for the Neural Networks

Regularization method : **Dropout**

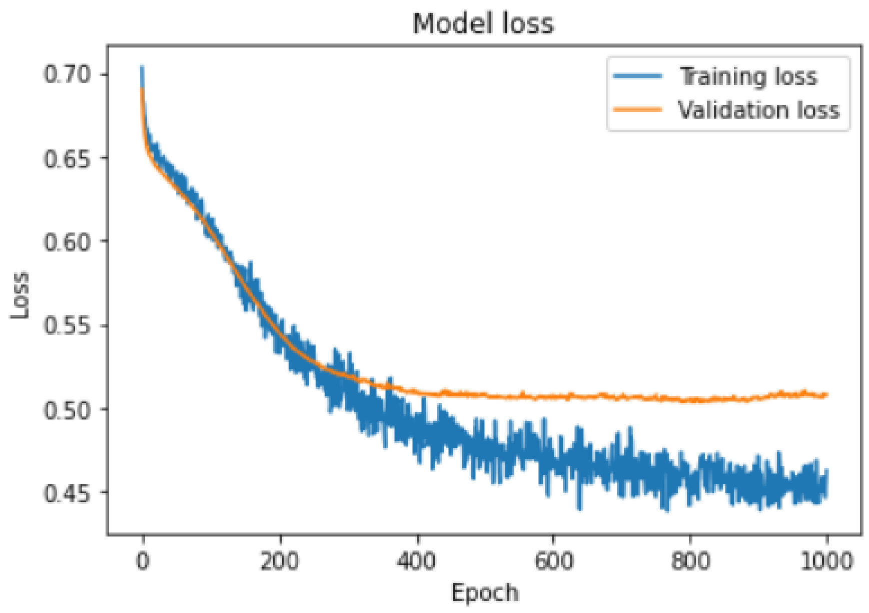
- Dropout is one of the regularization techniques, which is the most used one.
- In this method, it randomly selects some nodes and removes them, so each iteration has different set of nodes and hence this results in a different set of outputs.
- Dropout method produces best results when applies on larger datasets.

Here, I have used probability of dropping = 0.15

- I have used the same hyper-parameters as previous model (Epochs = 1000 and Batch size = 32)
- I have used Stochastic Gradient Descent 'sgd' optimizer to improve upon the loss. I have added 'Accuracy' to the metrics to measure the accuracy of the model.
    - I have used a tuneable hyper parameter for SGD optimiser with a learning_rate = 0.01
- Also, gave the model validation data by splitting the training data into 20%.

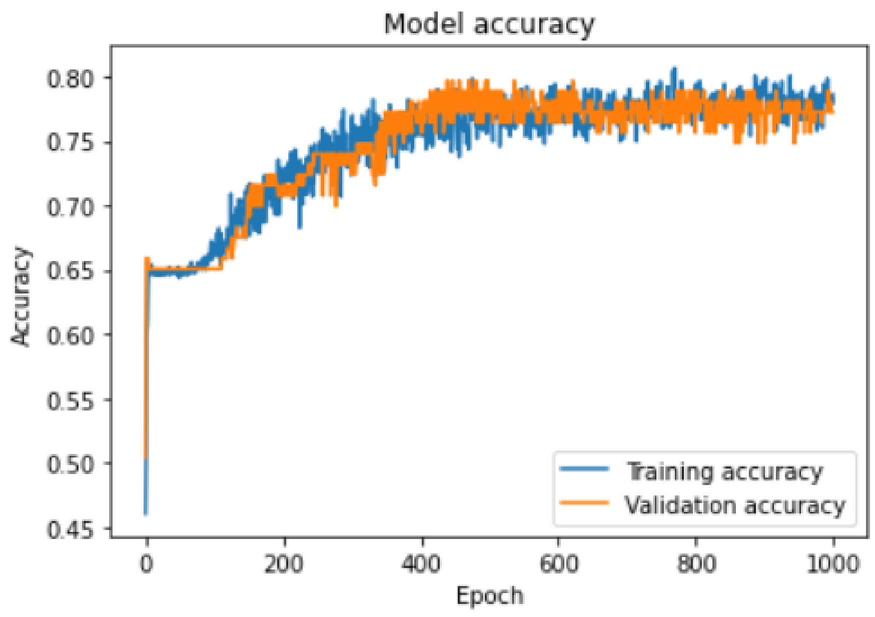
**Visualization for Model's Training Loss and Validation Loss:**

I have plotted the Training and Validation Loss with respect to Epoch as below:



**Visualization for Model's Training Accuracy and Validation Accuracy:**

I have plotted the Training and Validation Accuracy with respect to Epoch as below:



- Now, we evaluate the model on the testing data set by predicting that all the values which are less than 0.5 as 0 and values which are greater than 0.5 as 1.

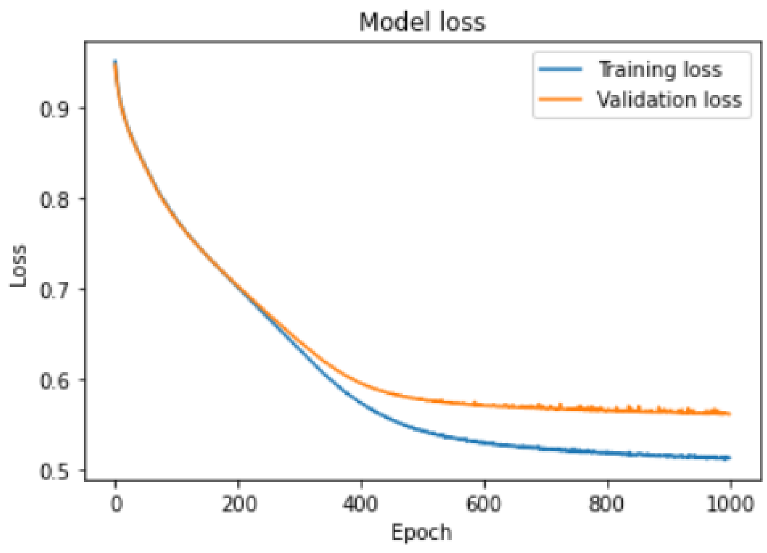
The Testing Accuracy achieved using the above model = **77.92 %**

Regularization method : **L2**

- Similar to above model explained in part - 2, I have implemented the model with L2 regularization.
- Batch size = 1000 and Epochs = 32
- Regularization parameter = 0.01 and Learning rate for SGD Optimizer = 0.01

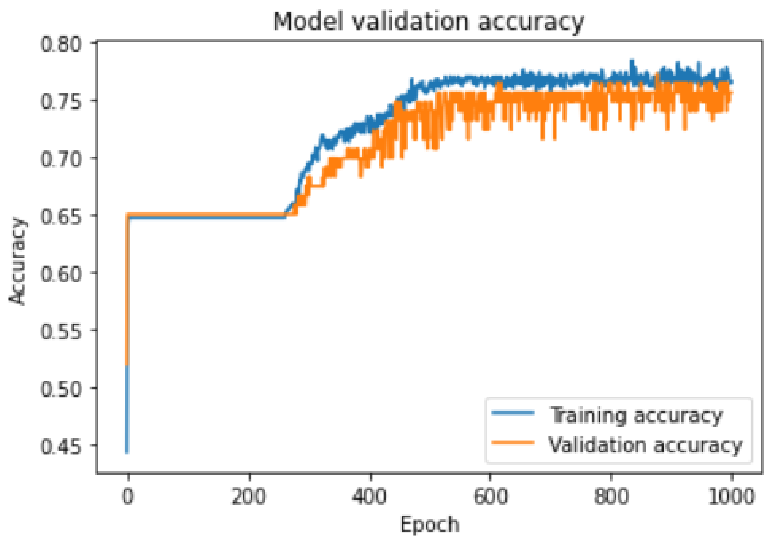
**Visualization for Model's Training Loss and Validation Loss:**

I have plotted the Training and Validation Loss with respect to Epoch as below:



**Visualization for Model's Training Accuracy and Validation Accuracy:**

I have plotted the Training and Validation Accuracy with respect to Epoch as below:



- Now, we evaluate the model on the testing data set by predicting that all the values which are less than 0.5 as 0 and values which are greater than 0.5 as 1.

The Testing Accuracy achieved using the above model = **79.87 %**

## Comparison between Dropout and L2:

- Though both methods (Dropout and L2) are used to reduce over-fitting, there is difference in execution and usage.
- The main idea of dropout is to randomly drop the neurons during the training so that the model doesn't overly depend upon the neurons for the output.
- Dropout method provides higher predictive accuracy than L2 for larger networks, but for smaller networks like our case, L2 regularization yields better results.
- Comparing the above model results, we can see L2 regularization yielded slightly better accuracy than dropout.

## Conclusion:

- We have implemented a logistic regression model developed with gradient descent yielded an accuracy of 80.51%.
- Splitting dataset into 60% training and 20% Validation dataset, helped in tuning the hyper parameters which improved the accuracy of the model.
- Implemented Neural networks with L1, L2 and Dropout regularization methods
- Neural network with L2 regularization method yielded better results than the Dropout as we have a smaller network.

## References:

1. Build Your Own Artificial Neural Network Using Python (https://randerson112358.medium.com/build-your-own-artificial-neural-network-using-python-f37d16be06bf)
2. Overview of Regularization Techniques in Deep Learning (https://www.analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques/)
3. Analysis of Regularization between Dropout and L2 in Neural Networks (https://uksim.info/isms2016/CD/data/0665a174.pdf)