# Logistic Regression Implementation on Diabetes Dataset

**Nandini Chinta**
SUNY Buffalo, NY
nandinic@buffalo.edu

## Abstract

This documentation refers to a Logistic Regression model which predicts and differentiates the diabetic patients from the given Pima Indians Diabetes Database dataset. The dataset is first partitioned into 3:1:1(training, validation, and test) ratio and data pre-processing is implemented on the data (data normalization). The dataset is fed to the model which learns on training data. Sigmoid activation function and gradient descent learning strategy are used for model improvement using hyper parameters. The model here is tuned using the validation set divided and is tested on the test data for accuracy. The decision is made on 0.5 decision boundary probability. This model achieved 78% accuracy.

## 1 Loading and splitting the Dataset

The given data set Pima Indians Diabetes Database dataset consists of 768 rows of females with age above 21diagnostic measurements. This dataset is first loaded using Pandas and split into training (60%), validation (20%) and test (20%) datasets.

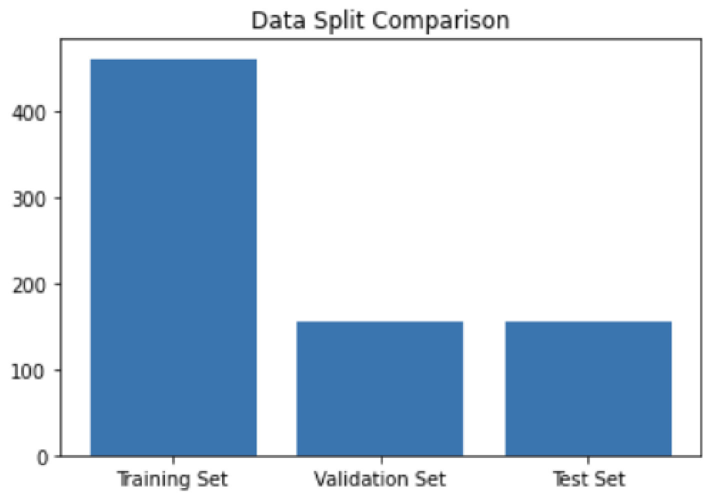


Figure 1.1: Instances of Pima Indian Diabetes Database dataset after splitting

## 2 Data Pre-processing

Data Pre-processing is done to avoid any anomalies or outliers and fetch the result as accurate as possible. This is the most critical step in any machine learning algorithm. Here, we clean the data by removing the missing values (zeroes and NAN values) and replace them with the mean of that column. Data prep-processing gives you cleansed data by which you can train the model with utmost efficacy.

### 2.1 Data Cleansing

As discussed, avoiding outliers and anomalies is a crucial step in data pre-processing. Outliers and anomalies include missed values and zeroes. We eliminate them by replacing them with the mean of that feature set. In the given Dataset, after having a glimpse, we get to know that few features are having the values as 0, which practically can not have zeroes as their values. So, to eliminate such anomalies, we cleanse the data using mean of that feature set.

### 2.2 Data Normalization

Data Normalization is basically used to clean the data, i.e. to be appeared in the same range in all features. This process includes getting rid of unstructured data and redundancy to make sure of logical data storage. Depending on the dataset, data normalization looks differently. The goal of data normalization is to make every data (all columns) use a common scale without any different ranges and data loss. Here we have used Min Max Normalization technique to process the data which converts all the data into range between 0 and 1.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Where, x – individual values from each column
Min(x) – minimum value from the column
Max(x) – maximum value from the column

## 3 Model Building

Now that the data is cleansed, the data can be used to train the model. As said earlier, logistic regression is going to be used to build the model function, this model was built by using sigmoid function (converts any value to 0-1 range) and gradient descent. But any model is not 100% accurate on results. There will be minimal error or loss. This loss can be calculated using log loss function every time gradient descent is iterated.

Here we are using a linear function $\boldsymbol{\hat{y} = W * x + b},$ where $\boldsymbol{\hat{y}}$ is the output result predicted, W is the weights calculated, x is the input vector (excluding outcome column) of the feature set and b is the bias. For calculating weights, we initially assign values between 0 and 1 to the weight vector (#featureSet x 1). We usually take weight vector as zero vector and update it using gradient descent, as needed.

### 3.1 Gradient Descent

Gradient Descent is one of the most vital optimization techniques in machine learning algorithms which helps us to build best machine learning model. An ML model requires to find the hyper parameters to fine tune the model for best results. Here in our function $(\boldsymbol{h_\theta(x)}), \boldsymbol{\hat{y} = W * x + b},$ W and b are hyper parameters. These weights and bias are updated on iterations. For every iteration cost function is calculated and is stopped when previous cost function is lesser than the current cost. The resulted y_predicts are then sent to sigmoid function to convert the outcomes into 0-1 range to make the decision whether the given patient (input) is a diabetic.

Initially, in the given database, input vector is considered as 460x8 (#instances x #featuresExcludingOutcomecolumn). Since we have 8 features, weight vector is considered as 8*1 and bias is a scalar (any minimal random value). The outcome is predicted using $\boldsymbol{h_\theta(x)}$ function. To begin with, zeroes are assigned to the weight vector and $\boldsymbol{h_\theta(x)}$ function is calculated to update weights and bias using a hyper parameter which we call it a learning factor $(\boldsymbol{\alpha})$. to make viable solutions. Learning factor is taken randomly around 0-1.

$$\boxed{\boldsymbol{L = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log\big(h_\theta(x)\big) + (1 - y_i) \cdot log\big(1 - h_\theta(x)\big)}}$$

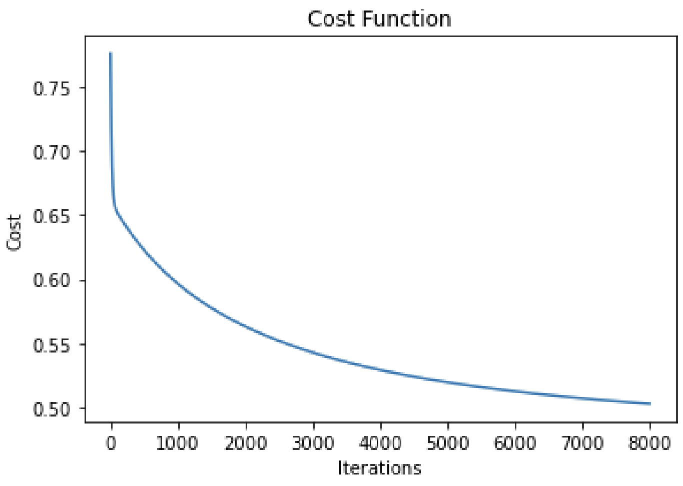


Figure 3.1.1: Cost Function Plot

$$\omega_{new} = \omega - \alpha \frac{\partial L}{\partial w}$$

$$b_{new} = b - \alpha \frac{\partial L}{\partial w}$$

Where ,

$\omega_{new}$ - updated weights,
w - old weights,
b - old bias
$b_{new}$ – updated bias
$\alpha$ is the learning factor
$\frac{\partial L}{\partial w}$ is the derivative of cost function.

### 3.2 Sigmoid Function

Logistic Regression is used to predict values for binary classification. So, the outcome should be in between 0 and 1 to predict the classification. To convert the outcome into categorical values, we use a mathematical activation function called Sigmoid Activation function which maps any value in between 0 and 1. This outcome will be used to predict which category this instance belongs to.

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

Where,

x – input instances

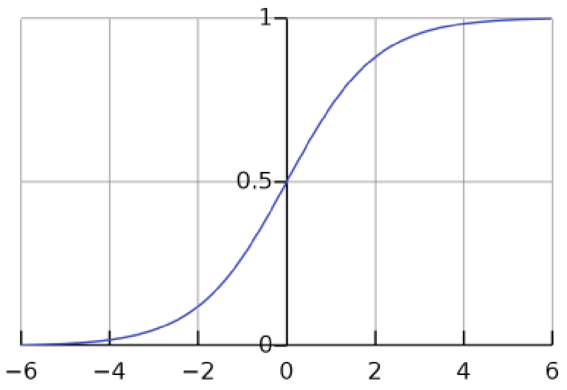


Figure 3.2.1: Sigmoid Activation Function Plot

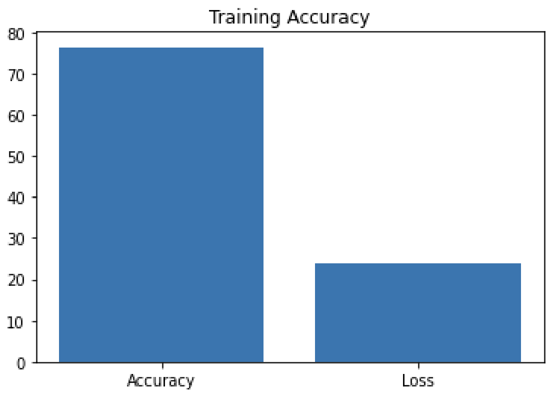


Figure 3.2.2: Model performance on Training dataset

## 3.3 Validation

Validation for any ML model is a process of deciding whether the model is giving a good fit of results and validating if the predictive performance is increasing, and error rate is decreasing. It also fine tunes the hyper parameters to make sure to maximize the accuracy. We use a threshold value to validate the predicted outcomes and the actual outcomes. This process also fine tunes the threshold so that the model doesn't get used to test dataset values.

$$\boxed{\hat{y} = \begin{cases} 1, & \text{if } h_\theta(x) \geq \text{threshold} \\ 0, & \text{if } h_\theta(x) < \text{threshold} \end{cases}}$$

Here, threshold considered for this project is 0.5

For this project we obtained 79.87% accuracy in validation.

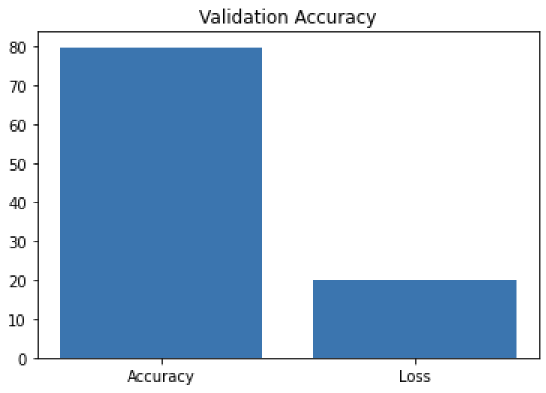


Figure 3.3.1: Model Performance on Validation dataset

## 3.4 Testing

Now Validating the model over validation data set is done, we need to test the model over test dataset that was split. With all the optimal hyper parameters assigned to the model, the accuracy of the model over test dataset has resulted as 78.57%. So, it meant that, for any unknown input values, this model gives you 78.57% probability that the predicted outcome to be exact as the actual outcome.

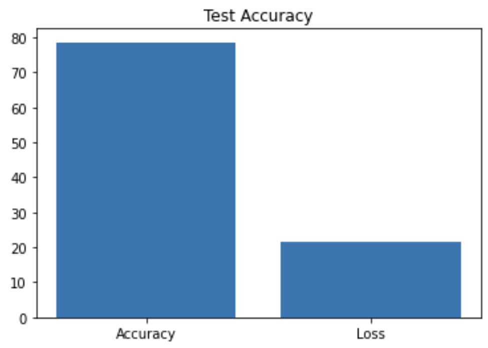


Figure 3.4.1: Model Performance on Test dataset

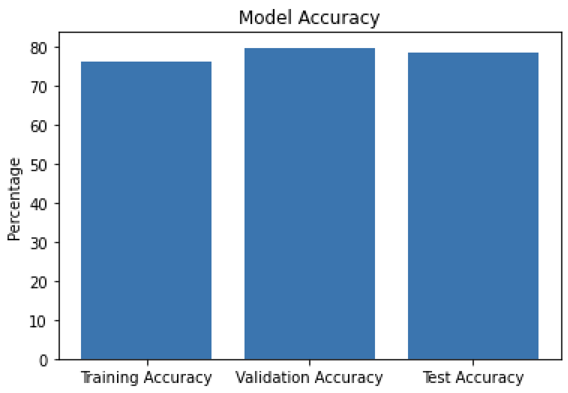


Figure 3.4.2: Accuracy Plot on Training, Validation and Test dataset

## 4 Neural Networks

Artificial Neural Networks are inspired by 'Neurons' in our brain. However, ANNs are bit different from biological neurons. The similarity is both have branches which are used to transfer the data. In biological neurons, they become active when it receives a certain amount of neurotransmitter while ANNs activates its output when certain number of input units become active.

Currently, ANNs are at the core od deep learning starting the field from using perceptron. Here in this project, I have used Sequential() model from Keras in tensor flow library. This library gives us the liberty to implement the model by just adding few parameters to the function. The model I used here takes feature set as input and two hidden layers are added to represent the non-linear functionality to the model. The loss function used here is 'cross-entropy loss' and optimization function used is gradient descent algorithm to best fit the data.

### 4.1 Optimizer

Even in ANNs, parameters are to be tuned to get the best possible result. An optimizer in ANN model implementation is an optimization function that tunes the hyper parameters like weights and learning factor by reducing the loss that can be occurred. Various optimizer functions available in Keras are SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam and Ftrl. The optimization function used here is Adam (Adaptive Movement Estimation Algorithm), a stochastic gradient descent algorithm which estimates based on first and second order functions.

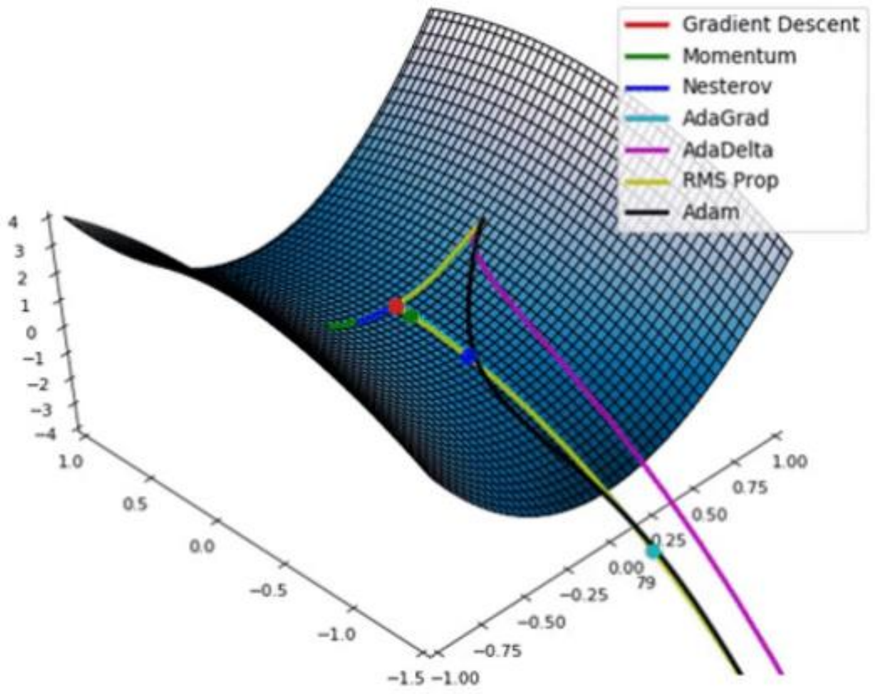


Figure 4.1.1: Various Optimizers Performance Visualization

The loss function used here is cross-entropy loss. This calculates the loss occurred between the outcomes predicted and the golden outcomes (actual outcomes).

$$L_{CE}(\hat{y},y) = -\log p(y|x) \quad = \quad -[y\log\hat{y}+(1-y)\log(1-\hat{y})]$$

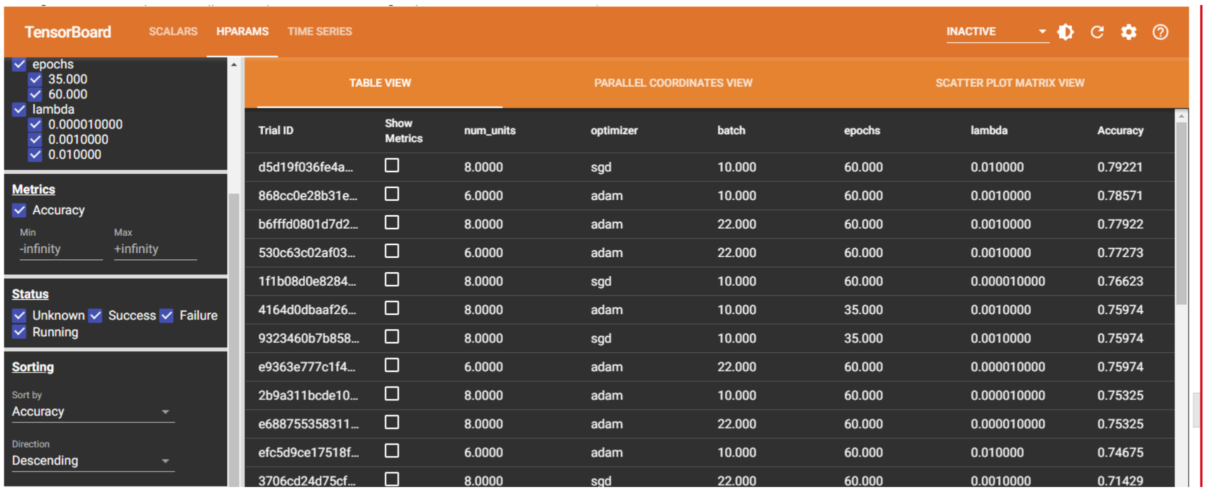
## 4.2 Activation Function

Mathematically an activation function maps any given value between [0,1] which can be used later to decide the classification or class it belongs to. The unit calculates the linear function using weights and biases and the resultant is given to activation function as input. This resultant is again given to the next layers as input for further calculations. In simple terms, activation function is a mathematical equation which decide whether a neuron to be activated or de-activated. Few activation functions available in Keras are Relu, Sigmoid, Softmax, Tanh etc.

## 4.3 Implementation

Neural Networks deals with layers of networks containing neurons which takes inputs and computes outputs using weights and biases. Any neural network or ML model’s final goal is to get the curve and decision boundary that best fits the data. This can be achieved by tuning hyper parameters and using the best of all values for hyper parameters. Here in this project, I have implemented Sequential model using Keras. For obtaining best fit to the data, number of neurons, epochs, number of batches, list of optimizers, list of lambda values (learning factors) serve as variables. I’ve implemented sequential model on all these parameters with various values. Parameters used when accuracy is the highest are taken as actual parameters and used for training the model. The highest values are visualized using TensorBoard’s HParams.

After training the model and validated against validation dataset. Then it is tested on test dataset that was split earlier. Plot of accuracy for training, validation and test data sets are given below.



☑ epochs
☑ 35.000
☑ 60.000
☑ lambda
☑ 0.000010000
☑ 0.0010000
☑ 0.010000

**Metrics**
☑ Accuracy
Min -infinity
Max +infinity

**Status**
☑ Unknown ☑ Success ☑ Failure
☑ Running

**Sorting**
Sort by: Accuracy
Direction: Descending

| Trial ID | Show Metrics | num_units | optimizer | batch | epochs | lambda | Accuracy |
|---|---|---|---|---|---|---|---|
| d5d19f036fe4a... | ☐ | 8.0000 | sgd | 10.000 | 60.000 | 0.010000 | 0.79221 |
| 868cc0e28b31e... | ☐ | 6.0000 | adam | 10.000 | 60.000 | 0.0010000 | 0.78571 |
| b6fffd0801d7d2... | ☐ | 8.0000 | adam | 22.000 | 60.000 | 0.0010000 | 0.77922 |
| 530c63c02af03... | ☐ | 6.0000 | adam | 22.000 | 60.000 | 0.0010000 | 0.77273 |
| 1f1b08d0e8284... | ☐ | 8.0000 | sgd | 10.000 | 60.000 | 0.000010000 | 0.76623 |
| 4164d0dbaaf26... | ☐ | 8.0000 | adam | 10.000 | 35.000 | 0.0010000 | 0.75974 |
| 9323460b7b858... | ☐ | 8.0000 | sgd | 10.000 | 35.000 | 0.0010000 | 0.75974 |
| e9363e777c1f4... | ☐ | 6.0000 | adam | 22.000 | 60.000 | 0.000010000 | 0.75974 |
| 2b9a311bcde10... | ☐ | 8.0000 | adam | 10.000 | 60.000 | 0.000010000 | 0.75325 |
| e688755358311... | ☐ | 8.0000 | adam | 22.000 | 60.000 | 0.000010000 | 0.75325 |
| efc5d9ce17518f... | ☐ | 6.0000 | adam | 10.000 | 60.000 | 0.010000 | 0.74675 |
| 3706cd24d75cf... | ☐ | 8.0000 | sgd | 22.000 | 60.000 | 0.0010000 | 0.71429 |

Figure 3.3.1: TensorBoard having parameters that gives highest accuracy

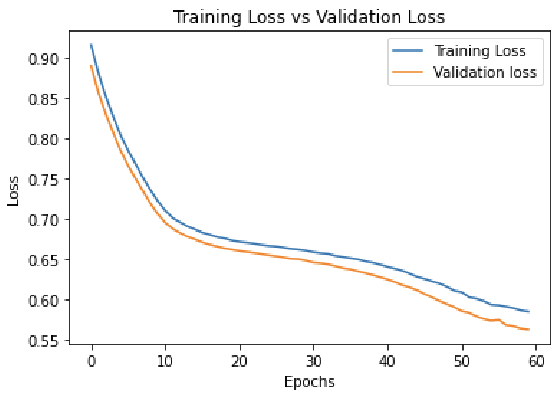


Figure 4.3.2: Training vs Validation Loss

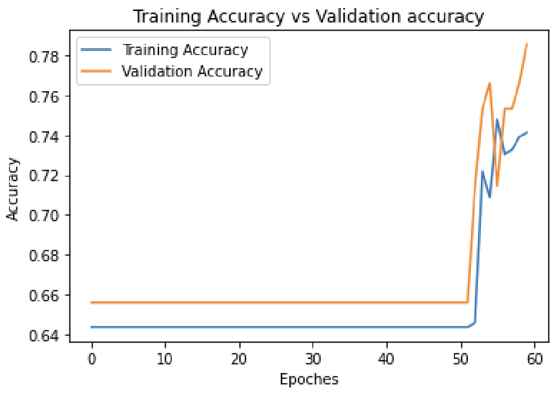


Figure 4.3.3: Training vs Validation Accuracy

## 4.4 Testing

After implementing neural networks onto the given dataset, the model fit the curve with 79.87% accuracy. On any unknown data, this model gives you the correct outcome with a probability of around 0.79. The probability of success obtained for validation dataset is around 79.87%.

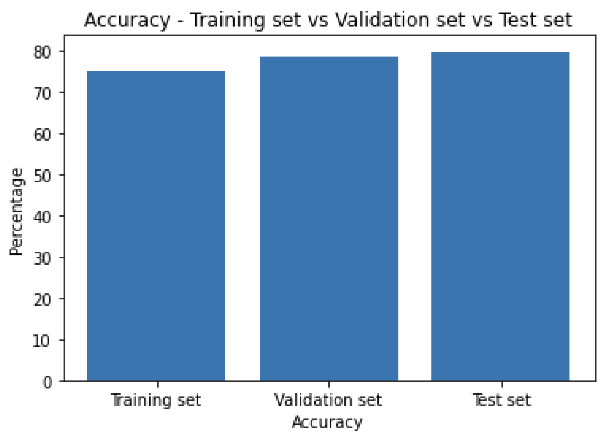


Figure 4.4.1: Training vs Validation vs Test Accuracy

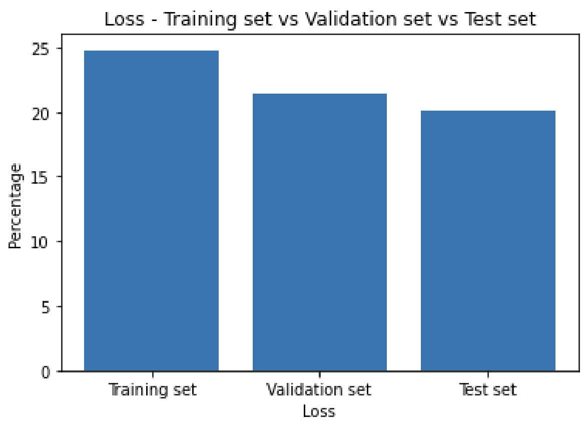


Figure 4.4.2: Training vs Validation vs Test Loss

## 5 Conclusion

To conclude, this project implements a logistic regression ML model using sigmoid activation function and gradient descent strategy. Hyper parameters and threshold values are fine-tuned using validation process and are set to best fit values. After testing, the model prediction performance turned out to be 78.57% accurate. When used neural networks, this model gave 79.87% on test data with very low loss. Here in neural networks, hyper parameters are fine tuned using 3 layers of networks with various values for all parameters where best fit values are selected for every parameter using Tensor Board.

## References

[1] https://en.wikipedia.org/wiki/Logistic_regression
[2] Christopher Bishop (2006) *Pattern Recognition and Machine Learning*
[3] https://www.youtube.com/watch?v=yIYKR4sgzI8&ab_channel=StatQuestwithJoshStarmer
[4] https://towardsdatascience.com/5-reasons-why-you-should-use-cross-validation-in-your-data-science-project-8163311a1e79

[5] https://www.ibm.com/cloud/learn/gradient-descent
[6] https://towardsdatascience.com/getting-to-know-activation-functions-in-neural-networks-125405b67428
[7] https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams