Objective:

Rick works as Head of Southern TD Canada Bank. Huge number of customers had to leave the bank due to no proper services from Bank. Rick had a hard time solving internal conflicts, streamlining the process and bringing back customers to the bank. He now wants to make sure everything runs smoothly and work on how to retain customers. For this, he wants us to build an application that would predict which of the customers are more likely to leave the bank soon, so that he can work on how to retain the customer. We will be using machine learning algorithms and help Rick in predicting which of the customers are more likely to leave the bank soon.

Network Architecture and Data Overview:

Data Overview:

Feature Name	DataType
CreditScore	Integer
Geography	String
Gender	String
Age	Integer
Tenure	Integer
Balance	Integer
NumOfProducts	Integer
HasCrCard	Integer
IsActiveMember	Integer
EstimatedSalary	Integer

We are using the above columns to predict whether the customer is more likely to leave the bank or not. The output is represented by “Exited” and it is a boolean output.

Number of input/features: 10

Number of outputs: 1

Total number of records: 10000

Network Architecture:

An artificial neural network (ANN) is a series of algorithms that aim at recognizing underlying relationships in a set of data through a process that mimics the way the human brain operates.

The field of artificial neural networks is often just called neural networks or multi-layer perceptron (MLP) or Feed Forward Neural Network (FFNN). The perceptron is the first and simplest neural network model. This network is said to be simple because it only has two layers: an input layer and an output layer. Neural Network architecture is designed to solve any equation (Linear or Non-Linear) but they are recommended only when our dataset is non-linear. We tried to figure out whether the dataset chosen here is linear or not by running a simple linear regression model on data and checked the R2 score. If the R2 score is high, it means your dataset can be linearly separable or else it can be considered non-linear. We got very less R2 score. Since, the dataset we have cannot be separated linearly, we used Multi-layer

Perceptron to predict outcomes. The Multi-layer Perceptron (MLP) consists of an input layer, an output layer and one or more hidden layers.

Input layer: the input layer neurons receive the information supposed to explain the problem to be analyzed;

Hidden layer: the hidden layer is an intermediate layer allowing neural networks to model nonlinear phenomena. This said to be "hidden" because there is no direct contact with the outside world. The outputs of each hidden layer are the inputs of the units of the following layer;

Output layer: the output layer is the last layer of the network; it produces the result, the prediction.

Theoretically, adding enough neurons to 1-2 hidden layer may be enough to approximate any non-linear function. MLP can be used both for classification and Regression. The MLP learns and improves accuracy of prediction by changing the weights and bias of the hidden layer network by a technique called backpropagation. Several parameters are used by backpropagation to improve overall accuracy. Below are few that are used in this report to improve accuracy.

Parameters:

Parameter Name	Values	Description
Activation Function	Sigmoid: (0 to 1) Tanh: (-1 to 1) ReLu : [0 to infinity)	It is used to limit/determine the output of neuron like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).
Hidden Layers	Minimum 1	Number of layers help in converging quickly
Number of neurons in each layer	Minimum equal to inputs and in output max 1 as per our prediction	
Learning Rate	often in the range between 0.0 and 1.0	Learning rate controls how quickly or slowly a neural network model learns a problem
Batch Size	typically chosen between 1 and a few hundreds. Usually 2^n .Recommended =32 to start with	Batch size controls the accuracy of the estimate of the error gradient when training neural networks
Epochs	Can be any integer	epoch refers to one cycle through the full training dataset
Optimizer	SGD ,RMSProp, Adam	Optimization algorithms are used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).
Early Stopping		Early stopping is a method that allows you to specify an arbitrary large number of

		training epochs and stop training once the model performance stops improving on a hold out validation dataset

Data Pre-Processing:

The dataset has 14 columns in total and below are the name and its datatype

Column Name	Datatype
RowNumber	Integer
CustomerId	Integer
SurName	String
CreditScore	Integer
Geography	String
Gender	String
Age	Integer
Tenure	Integer
Balance	Integer
NumOfProducts	Integer
HasCrCard	Integer
IsActiveMember	Integer
EstimatedSalary	Integer
Excited	Integer

We drop “RowNumber”,“SurName” and “CustomerID” columns as “RowNumber” is just a series identifier and “SurName” logically doesn’t have any impact on leaving the bank.

Below figure shows us how our dataset looks before pre-processing

CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
619	France	Female	42	2	0	1	1	1	101348.88	1
608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
502	France	Female	42	8	159660.8	3	1	0	113931.57	1
699	France	Female	39	1	0	2	0	0	93826.63	0
850	Spain	Female	43	2	125510.8	1	1	1	79084.1	0
645	Spain	Male	44	8	113755.7	2	1	0	149756.71	1
822	France	Male	50	7	0	2	1	1	10062.8	0
376	Germany	Female	29	4	115046.7	4	1	0	119346.88	1
501	France	Male	44	4	142051.0	2	0	1	74940.5	0
684	France	Male	27	2	134603.8	1	1	1	71725.73	0
528	France	Male	31	6	102016.7	2	0	0	80181.12	0

Fig: Original Dataset after dropping unnecessary columns

Data pre-processing has a sequential flow and it starts as follows

a. **Check out the missing values**
   I imported the dataset using python and found that we don’t have any missing values in this dataset.

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

Fig: NULL Records in each column

b. Look for categorial values

Here we have two categorial columns, “Geography” and “Gender”. Machine Learning models deal only with numbers, so let’s convert this string into integer values. We can use any of the below techniques here

1. Label Encoding
2. One-Hot Encoding

The limitation on label encoding is, after encoding, the values in the dataset might confuse the model as if they are somewhat sequential. In our case, both the columns are of some category type, so we would go for “One-Hot Encoding”.

c. Feature Dropping

Features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable. So, when two features have high correlation, we can drop one of the two features. I tried finding correlation between all features and found they don’t have any high correlation.

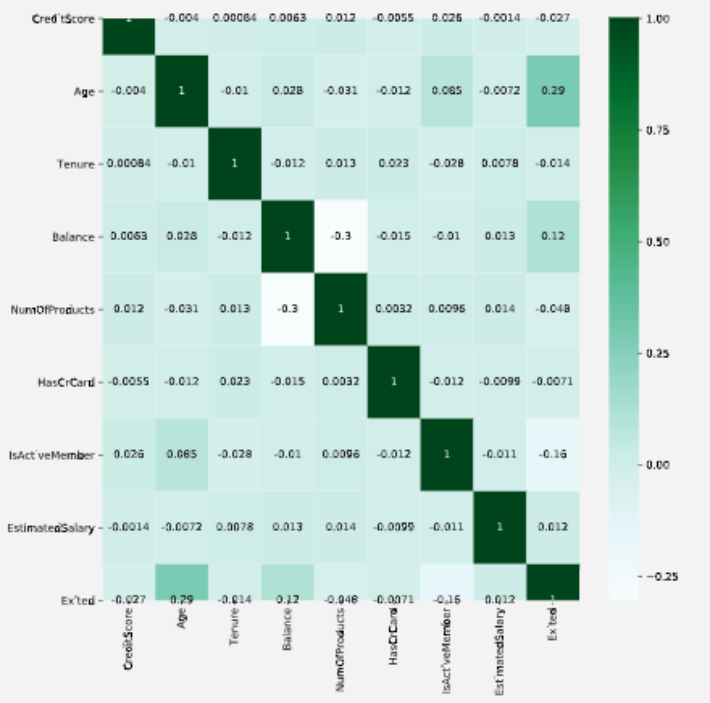


Fig: Correlation Heatmap

d. Split the data into train, DEV/Validation, test

The dataset has 10000 records. We are dividing the dataset into 3 pieces. The purpose of dividing is, we validate the accuracy against "DEV" set after each epoch to understand whether the model overfits or underfits. Based on that it is tuned and ran against test data.

Training	70%
DEV / Validation	15%
Testing	15%

e. Feature Scaling:

We have few columns in our dataset that are at a different range when compared to others. Below are few among them.

Column Name	Range
Credit Score	300-900
Age	10-100
Balance	0.0 – 200000.00
Estimated Salary	0.0 – 200000.00

Since they are in different scales, we need to make every column under a common unit. We have two techniques that can help in scaling

1. Normalization: Data normalization is the process of rescaling one or more attributes to the range of 0 to 1. This means that the largest value for each attribute is 1 and the smallest value is 0.

2. Standardization: Data standardization is the process of rescaling one or more attributes so that they have a mean value of 0 and a standard deviation of 1

Generally, standardization is preferred, and we are trying to standardize our data here. However, we will not be standardizing each column. At this point of data pre-processing, we have categorial data, binary and numerical. We standardize only numeric data and ignore binary columns (one-hot encoding produces binary columns).

NOTE: feature scaling is done on training, testing and DEV data separately to avoid data leaks.

So, we first calculate mean and standard deviation of each column of Test data and use the standardization formula on every column of DEV and TEST data on respective columns.

$$z = \frac{x - \mu}{\sigma}$$

μ = Mean
σ = Standard Deviation

Fig: Standardization

Z = new value, x= existing column value, μ = mean of the respective column , σ = deviation of the column

Mean		Standard Deviation	
CreditScore	651.354821	CreditScore	97.424914
Age	38.949107	Age	10.575480
Tenure	5.004821	Tenure	2.888129
Balance	76899.058895	Balance	62448.500790
NumOfProducts	1.526607	NumOfProducts	0.584377
EstimatedSalary	100557.861963	EstimatedSalary	57567.408450
dtype: float64		dtype: float64	

Below figure shows us the final dataset after pre-processing

CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Geography_France	Geogra…	Geogra…	Gender…	Gender…	HasCrC…	IsActi…
-0.742738993	1.04	-0.0016	0.6537483	-0.9012240636	1.5911250683	0	1	0	1	0	0	1
-0.362925427	1.70	0.69088	0.6411895	-0.9012240636	0.0663889411	0	1	0	1	0	1	1
0.1913970748	0.28	0.34460	0.5674713	-0.9012240636	-0.6054487255	1	0	0	1	0	1	1

Fig: Final Dataset

Model:

As mentioned earlier, we have divided our data into train, DEV and test. We will first train the model and test it against “DEV” data to figure whether our model is underfitting or overfitting. Based on the results, we will tweak the hyperparameters or neural network in the next training set. However, we will test the model against “test” data to find the accuracy

Number of inputs = Number of input neurons = 13

Output layer neurons =1 and activation = sigmoid (this will be the same in all training set)

Training 1:

Hidden Layer(s)	1
Neurons per Hidden Layer(s)	27
Activation function for hidden layer 1	tanh
Optimizer	Adam
Learning Rate	0.01
Epochs	100
Batch size	10
Bias Initializer for each layer	Zeros
Weights initializer for each layer	**He_uniform** (It draws samples from a uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units in the weight tensor)
Shuffle	True (whether to shuffle the training data before each epoch)

We randomly chose these values. Sometimes, one hidden layer with many neurons is good enough for getting good accuracy. So, we will be using one hidden layer only to understand the behavior of our model. Shuffling data serves the purpose of reducing variance and making sure that models remain general and overfit less. Here we shuffle to make sure that your training/test/validation sets are representative of the overall distribution of the data.

Below is the graph for train vs validation dataset.

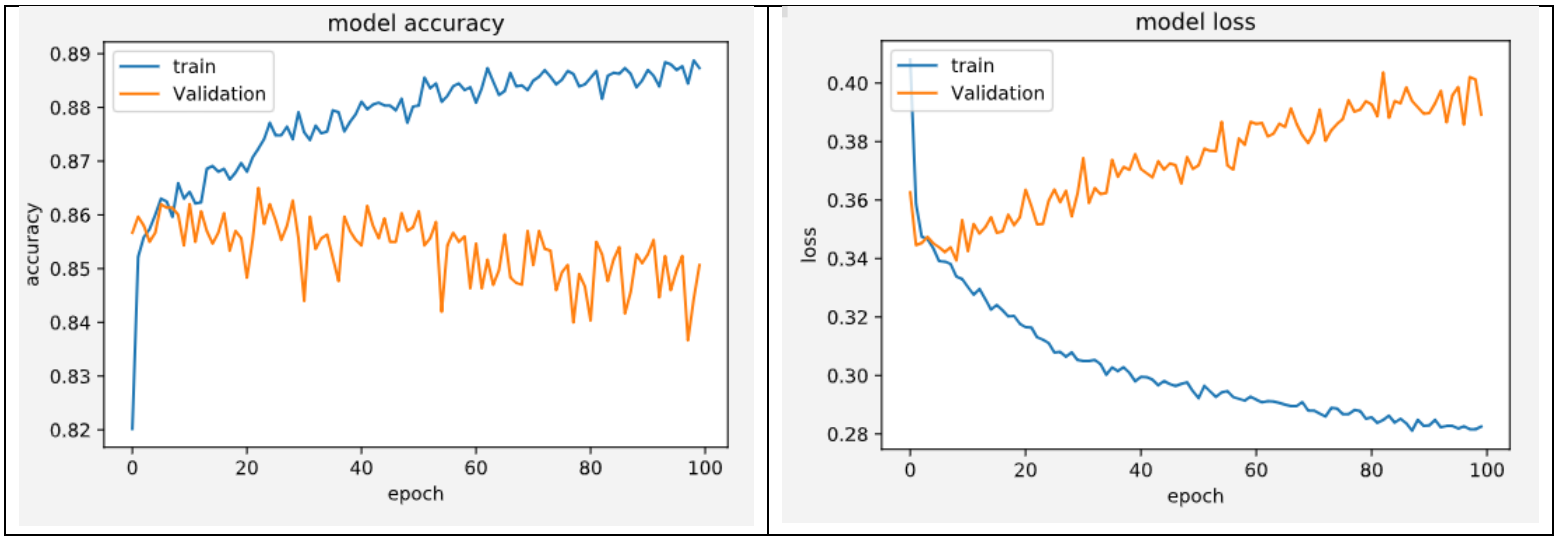


Fig: Model accuracy (accuracy vs epoch) and model loss (loss vs epoch)

As we can see from the left graph, the validation accuracy is less than training accuracy, that implies, our model learned our training data well throughout the epochs but failed to work well on DEV (new data). We can see that there is some scope for improvising our model to work better on new dataset. We can also tell the same by looking at graph on the right (loss vs epoch). The training loss went down as it learned and corrected itself, but the validation loss went high.

Error calculation for Training 1 on TEST Data:

Confusion Matrix: Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.

Since we are dealing with a binary classification problem. We have some samples belonging to two classes: YES or NO. Also, we have our own classifier which predicts a class for a given input sample. On testing our model on 1400 samples, we get the following result.

N=1400	Predicted: NO	Predicted: YES
Actual: NO	TP = 1046	FN = 80
Actual: YES	FP = 134	TN = 140

There are 4 important terms:

True Positives: The cases in which we predicted YES and the actual output was also YES.

True Negatives: The cases in which we predicted NO and the actual output was NO.

False Positives: The cases in which we predicted YES and the actual output was NO.

False Negatives: The cases in which we predicted NO and the actual output was YES.

Accuracy = (True Positive + True Negatives)/(Total Number of samples)

= (1063+130)/1400 = **84.71428571428572%**

F-Measure: F1 Score tries to find the balance between precision and recall. The range for F1 Score is [0, 1]. The greater the F1 Score, the better is the performance of our model

Recall can be defined as the ratio of the total number of correctly classified positive examples divide to the total number of positive examples. High Recall indicates the class is correctly recognized

Recall = (True Positive) / (True Positive + False Negative)

To get the value of precision we divide the total number of correctly classified positive examples by the total number of predicted positive examples. High Precision indicates an example labeled as positive is indeed positive

Precision = (True Positive) / (True Positive + False Positive)

F-Measure = (2*Recall*Precision)/(Recall+precision)

=0.9112730390055722

Mean Average Error: Mean Absolute Error is the average of the difference between the Original Values and the Predicted Values. It gives us the measure of how far the predictions were from the actual output. However, they don't give us any idea of the direction of the error i.e. whether we are under predicting the data or over predicting the data. The lower the MAE the better

$$MeanAbsoluteError = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y}_j|$$

MAE = **0.14785714285714285**

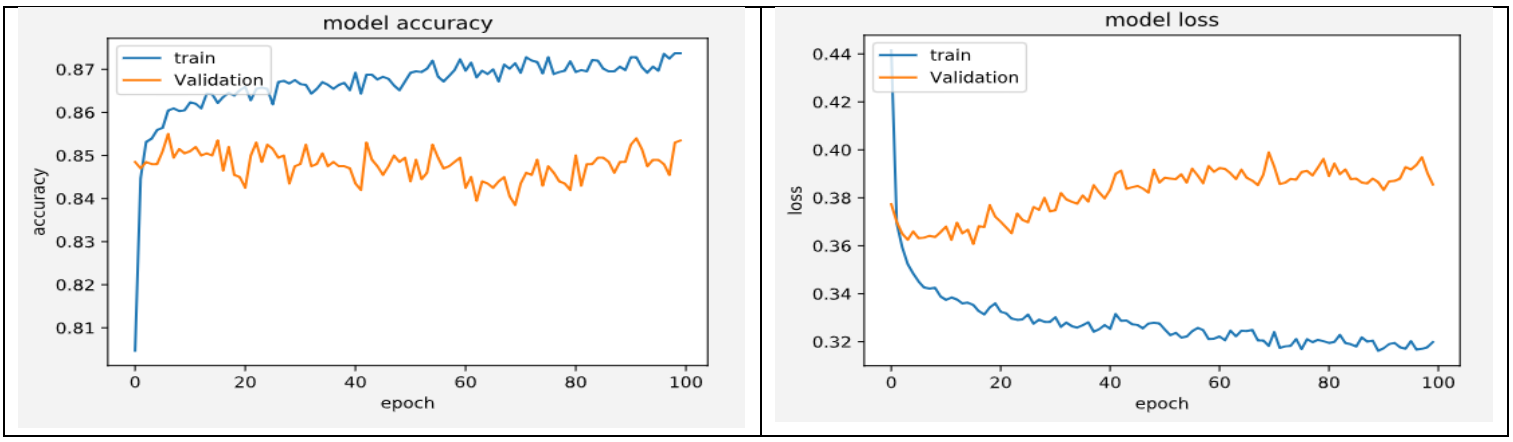
Training 2:

Let us take the same old values but now increase the hidden layer by one and assign equal neurons in each hidden layer

Hidden Layer(s)	2
Neurons per Hidden Layer(s)	5
Activation function for hidden layer(s)	Relu ,Tanh
Optimizer	Adam
Learning Rate	0.01
Epochs	100
Batch size	10
Bias Initializer for each layer	Zeros
Weights initializer for each layer	**He_uniform** (It draws samples from a uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units in the weight tensor)
Shuffle	True (whether to shuffle the training data before each epoch)

In this training set, the only thing we are trying to understand is effect of hidden layers with a combination on neurons per layer. In previous set, we used only one hidden layer with many neurons on it. Here we are increasing the layer and decreasing the neurons per layer to see the model performance

Below is the graph for train vs validation dataset.



From graph on the left, we can see that the training accuracy is little higher than validation accuracy. Also, if we compare the “model accuracy” graphs of Training-1 and Training-2. We can see that our

model improved. We can say this by looking at how much the validation loss decreased in Training-2 when compared to Training-1 and how much validation accuracy improved from Training-1 to Training-2.

Now let us try to understand the accuracy on TEST data.

Error Calculation of Training-2 on TEST data

Confusion Matrix:

N=1400	Predicted: NO	Predicted: YES
Actual: NO	TP = 1092	FN = 34
Actual: YES	FP = 164	TN = 110

Accuracy = (True Positive + True Negatives)/ (Total Number of samples)

= **85.85714285714286%**

F-Measure:

Recall = (True Positive) / (True Positive + False Negative)

Precision = (True Positive) / (True Positive + False Positive)

F-Measure = (2*Recall*Precision)/(Recall+precision)

=0.9168765743073048

Mean Average Error:

MAE = **0.14142857142857143**

If we compare accuracy of previous and the current training set, the accuracy increased by 1%. So 2 hidden layers worked in our case.

Training 3:

Hidden Layer(s)	2
Neurons per Hidden Layer(s)	5
Activation function for hidden layer(s)	Tanh
Optimizer	Adam
Learning Rate	0.01
Epochs	150
Batch size	10
Early Stopping	True
Patience	2
Shuffle	False
Bias and weights	Zeros , he_uniform

A major challenge in training neural networks is how long to train them.

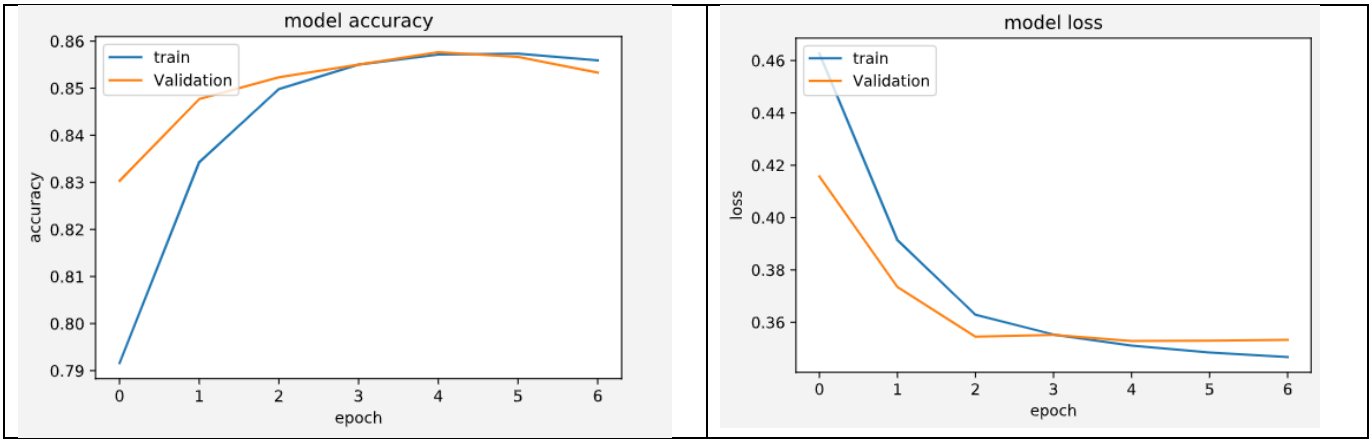
Too little training will mean that the model will underfit the train and the test sets. Too much training will mean that the model will overfit the training dataset and have poor performance on the test set.

A compromise is to train on the training dataset but to stop training at the point when performance on a validation dataset starts to degrade. This simple, effective, and widely used approach to training neural networks is called early stopping.

In this training set, let's try to understand how epoch will help us in improving accuracy by training the model on number of epochs and later select that results in the best performance on the train or a holdout test dataset. The downside of this approach is that it requires multiple models to be trained and discarded. This can be computationally inefficient and time-consuming. So here we will use using a combination of **epoch and early stopping**. The model at the time that training is stopped is then used and is known to have good generalization performance.

Below graph shows accuracy and loss comparison between train and validation.

Below is the graph for train vs validation dataset.



Surprisingly, the model stopped after **6** epochs, that infers, our model's performance started to degrade on validation set and started to overfit the training data. Let us try to understand the performance of this model on test data.

Error Calculation of Training-3 on TEST data

Confusion Matrix: **86%**

F-Measure: 0.917

Mean Average Error: 0.14

We can see that using a combination of early stopping and epoch did work. They stopped the model at the time when it started to overlearn training data. Our test accuracy also went high by 1% when compared to previous. With this we can tell how important it is to stop the model from training more at the right time.

Training 4:

Hidden Layer(s)	2
Neurons per Hidden Layer(s)	5
Activation function for hidden layer(s)	Tanh
Optimizer	Adam
Learning Rate	0.001
Epochs	100
Batch size	16
Early Stopping	True
Patience / Tolerance	2

In this set, let us try to understand the model behavior by changing the batch size and learning rate.

Neural networks are trained using gradient descent where the estimate of the error used to update the weights is calculated based on a subset of the training dataset.

The number of examples from the training dataset used in the estimate of the error gradient is called the batch size and is an important hyperparameter that influences the dynamics of the learning algorithm. Batch size controls the accuracy of the estimate of the error gradient when training neural networks. There is a tension between batch size and the speed and stability of the learning process.

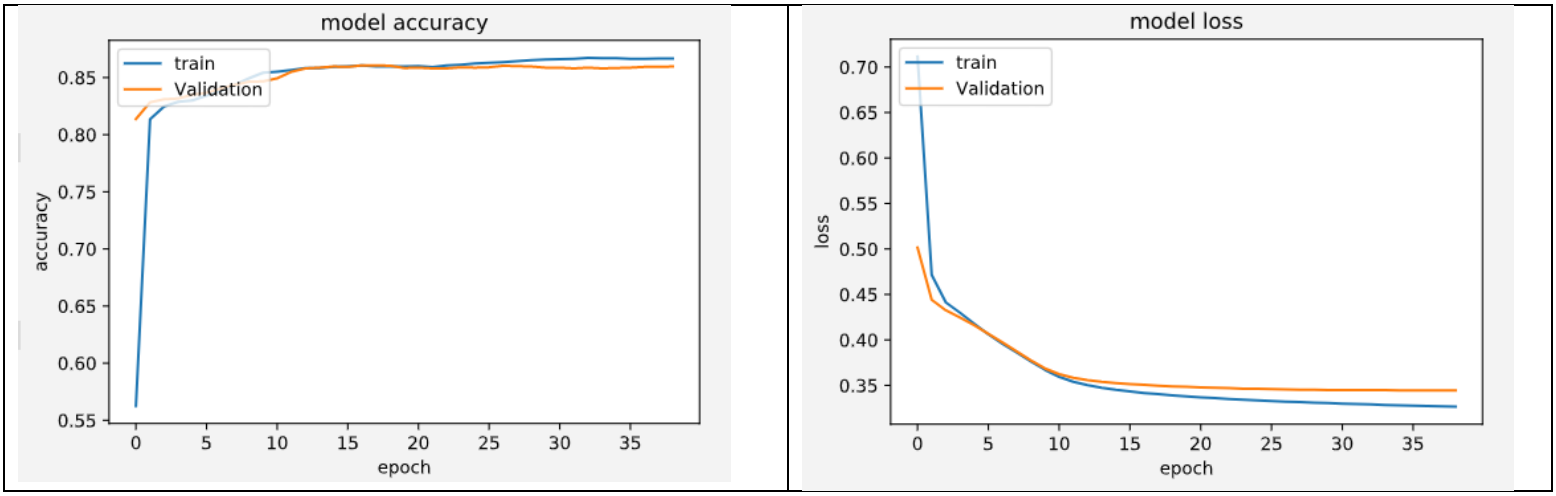
Apart from Batch Size, there is another hyperparameter which effects the learning process. It's called Learning Rate.

The optimization problem addressed by stochastic gradient descent for neural networks is challenging. The amount of change to the model during each step of this search process, or the step size, is called the "learning rate". Learning rate controls how quickly or slowly a neural network model learns a problem.

General saying is batch size and learning rate are directly proportional to learning. This is just logical because bigger batch size means more confidence in the direction of your "descent" of the error surface while the smaller a batch size is the closer you are to "stochastic" descent (batch size 1). Small steps can also work but the direction of each individual "steps" is more stochastic.

It is recommended to start with learning rate 0.001 or 0.01 and smaller batch size in the range of [2,16] and slowly change both. So, in this training set, we have taken batch size as 2^4=16 and learning rate=0.001 to understand the relation between these two.

Below is the graph for train vs validation dataset.



We can observe that validation and train accuracy are almost similar and same goes with loss as well. Our model iterated for like **39** times to reach a state where it can generalize well.

Error Calculation of Training-4 on TEST data

Confusion Matrix: **86.21%**

F-Measure: 0.918

Mean Average Error= 0.137

In this set, the accuracy went high by only 0.21% which can be a good number based on the dataset and prediction criteria. Based on the results we can say that learning rate and batch size together can affect the performance of the model. We need to use right combination of learning rate and batch size to get a good model.

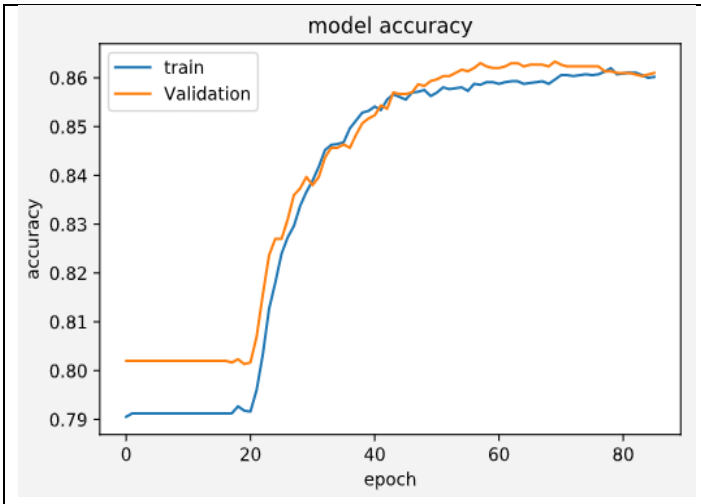
Training 5:

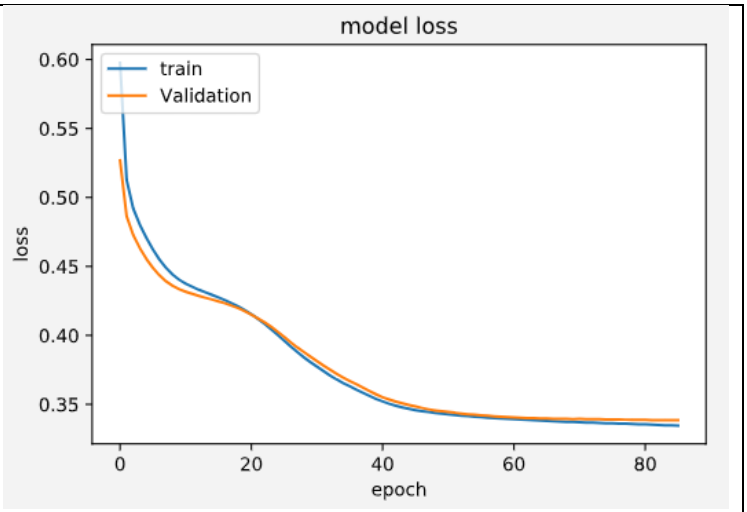
In this set, we will try to check how optimizer and activation functions have an impact on our model. We will also change few parameters like Learning Rate and Batch Size to default values of optimizer as below

Hidden Layer(s)	2
Neurons per Hidden Layer(s)	5
Activation function for hidden layer(s)	ReLu
Optimizer	SGD
Learning Rate	0.01
Epochs	100
Batch size	32
Early Stopping	True
Patience / Tolerance	2
Momentum	0.5

Till previous training set, we were using adam as optimizer and it hardly crossed 50 epochs to stop (by using early stopping). When we used Stochastic Gradient Descent as our optimizer, the model stopped after **89** epochs.

Below is the graph for train vs validation dataset.





From the graphs we can say that our model is being trained well. The accuracy and loss are reasonably good

Error Calculation of Training-5 on TEST data

Confusion Matrix: = **85.28%**

F-Measure: =0.9124

Mean Average Error: MAE = **0.147**

So far, we were using Adam model and saw the progress in accuracy but as we can see in this set, the accuracy went down when we used stochastic gradient descent. SGD keeping its default values didn't help us much here. We will explore SGD with other combinations later in the training sets.

Training 6:

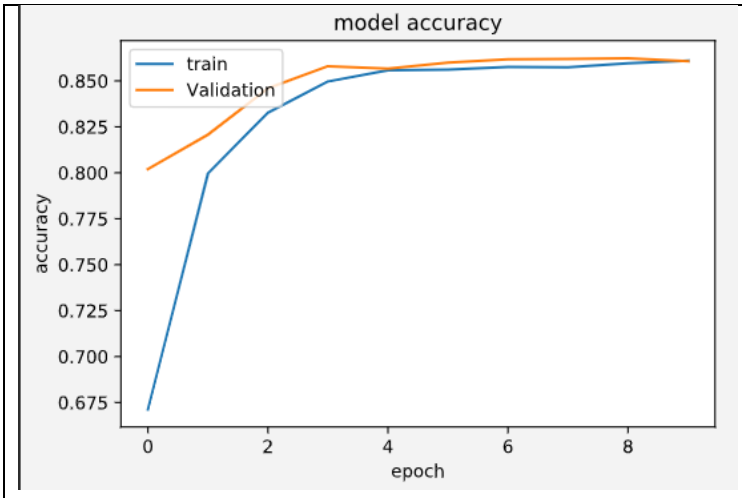
In this training set, let us try to understand the behavior of neural nets by increasing the hidden layer by one more and adding uniform bias along with other parameters mentioned below

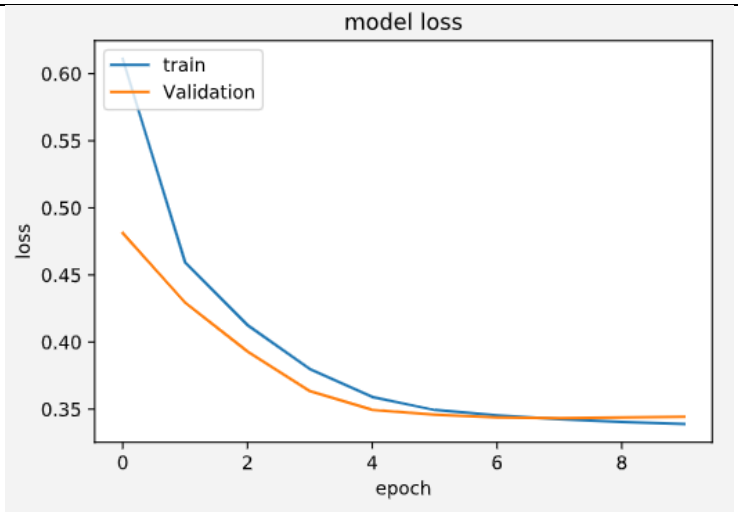
Hidden Layer(s)	3
Neurons per Hidden Layer(s)	3, 3 , 2
Activation function for hidden layer(s)	Tanh
Optimizer	Adam
Learning Rate	0.01
Epochs	100
Batch size	32
Early Stopping	True
Patience / Tolerance	2

Initial Weights	uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units
Initial Bias	Ones

Surprisingly, our training stopped after 10 epochs. Our network was quick enough to understand and develop with 3 hidden layers. Notice that we knowingly decreased the neurons per layer so that we don't overfit. The graphs below show characteristics of a good model.

Below is the graph for train vs validation dataset.





Error Calculation of Training-6 on TEST data

Confusion Matrix: = **86.571428%**

F-Measure: =0.92114

Mean Average Error: MAE = **0.1347**

Adding one more hidden layer helped us bringing up the accuracy by 1.28%. we again went with Adam optimizer thinking it goes well with our dataset and by looking at the accuracy (86.57%), we can say this is the best model we got so far.

Training 7:

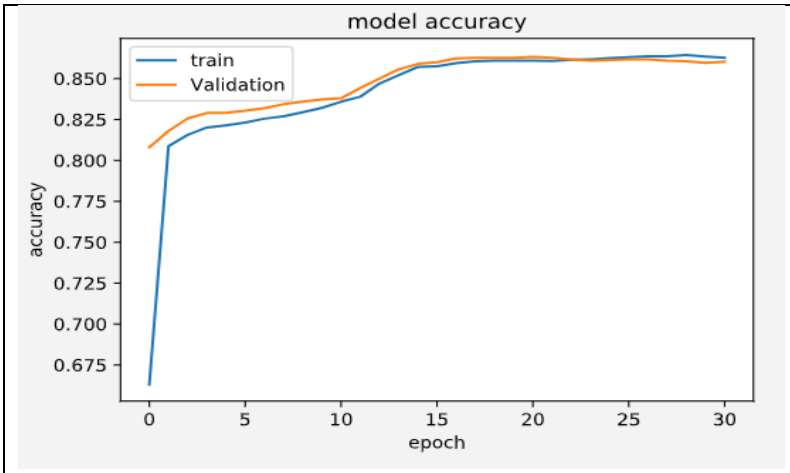
Hidden Layer(s)	3
Neurons per Hidden Layer(s)	3, 3 , 2
Activation function for hidden layer(s)	ReLu , tanh , ReLu
Optimizer	Adam
Learning Rate	0.001
Epochs	100
Batch size	8

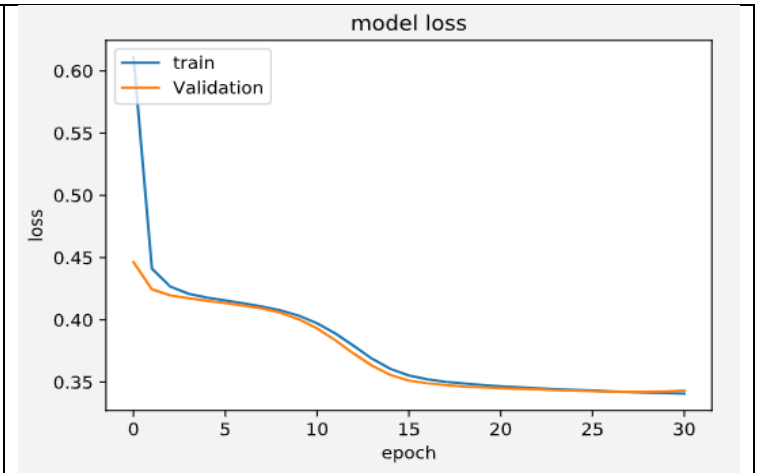
Early Stopping	True
Patience / Tolerance	1
Initial Weights	uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units
Initial Bias	Ones

In this training set, let us understand the behavior by decreasing the learning rate and batch size for 3 hidden layers.

Our model took **31** epochs to train and stopped as it couldn't improve after that.

Below is the graph for train vs validation dataset.





Error Calculation of Training-7 on TEST data

Confusion Matrix: = **85.7142%**

F-Measure: = 0.915325

Mean Average Error: MAE = **0.1428**

The accuracy just decreased a little when we decreased learning rate. However, there comes a point at which reducing the learning rate any more simply wastes time, resulting in taking many more steps than necessary to take the same path to the same minimum. When the step size (here learning rate = eta) gets smaller the function may not converge since there are not enough steps with this small learning rate (step size).

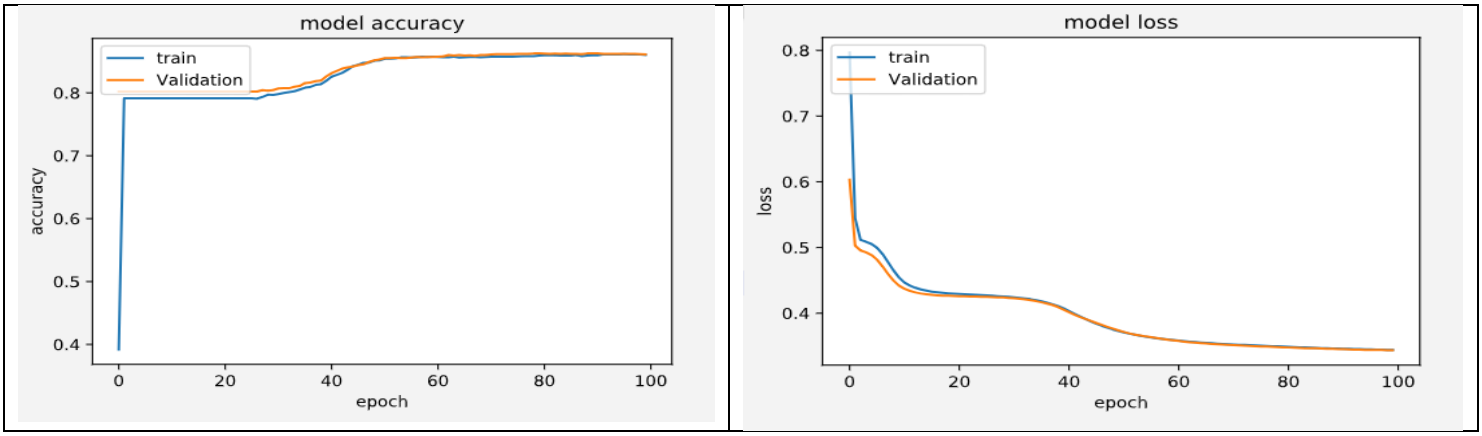
Training 8:

Hidden Layer(s)	3
Neurons per Hidden Layer(s)	3, 3 , 2
Activation function for hidden layer(s)	ReLu , Than , ReLu
Optimizer	SGD

Learning Rate	0.001
Epochs	100
Batch size	8
Early Stopping	True
Patience / Tolerance	1
Initial Weights	uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units
Initial Bias	Ones
Momentum	0.8

In this training set, let us understand the effect of different optimizer on 3 hidden layer net. We will be using Stochastic Gradient Descent instead of Adam and reduce the learning rate. However, we will slightly increase the momentum which indirectly helps learning rate.

Below is the graph for train vs validation dataset.



Error Calculation of Training-8 on TEST data

Confusion Matrix: = **86.35714285714286%**

F-Measure: = 0.9195111672987779

Mean Average Error: MAE = **0.13642857142857143**

SGD worked well with 3 hidden layers for this dataset by increasing the accuracy by 1%.SGD with momentum is method which helps accelerate gradients vectors in the right directions, thus leading to faster converging. Though the model ran on all epochs, by looking at both the graphs, loss and accuracy, we can see that accuracy increased and loss decreased.

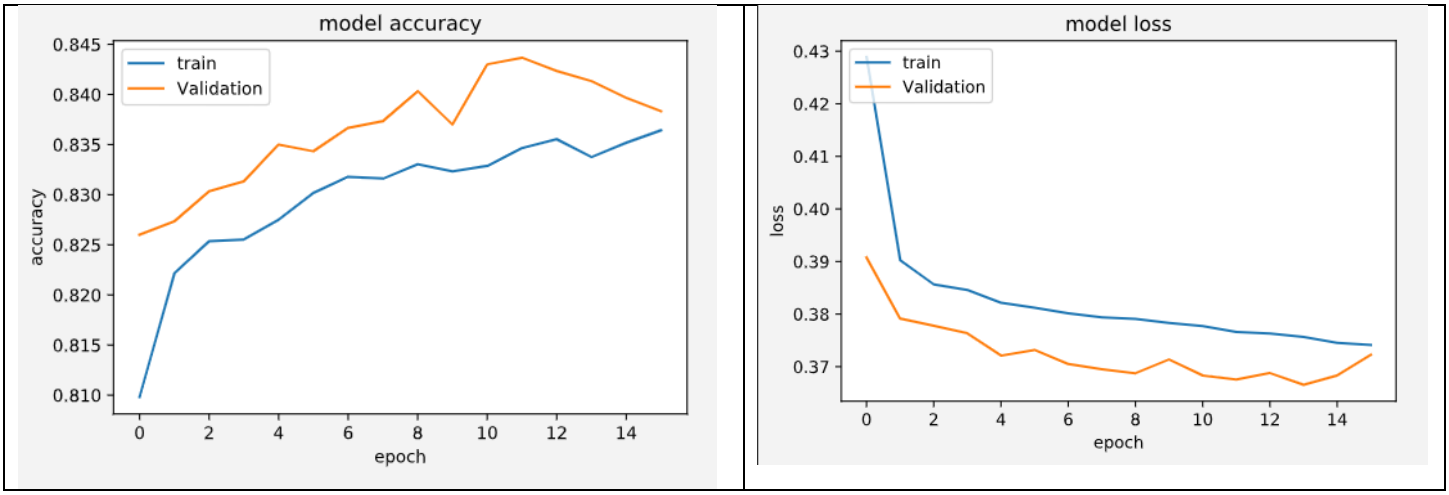
Training 9:

Hidden Layer(s)	4
Neurons per Hidden Layer(s)	2,3 , 2,3
Activation function for hidden layer(s)	ReLu , Tanh , ReLu ,tanh

Optimizer	adam
Learning Rate	0.01
Epochs	100
Batch size	16
Early Stopping	True
Patience / Tolerance	1
Initial Weights	uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units
Initial Bias	Ones

So far, we have calculated by tweaking different parameters on number of hidden layers. Let us try one final test on 4 hidden layers to check whether number of hidden layers really increase the accuracy or decrease. Here we also reduced the number of neurons per layer as hidden layers are more. In an ideal environment, 1-2 hidden layers are more than enough to approximate any dataset. Few datasets have very complex features that require more hidden layers to approximate.

Below is the graph for train vs validation dataset.



Surprisingly, our model stopped at 18 epoch and we can see from the graph that the accuracy went high till 0.84 only and dropped down to ~0.83. similarly, for the loss, it went down for a while and started growing and that's when our model stopped training. Let us look at error metrics against test data.

Error Calculation of Training-8 on TEST data

Confusion Matrix: = **83.2142857142857%** (this is less than what we got for 3-layer network)

F-Measure: = 0.89866321690

Mean Average Error: MAE = **0.1678571427143**

Surprisingly, adding one more hidden layer decreased by 3%. This can be an indication that we are trying to overfit the model by adding more number of layers or neurons per layer than required. As mentioned

earlier, any non-linear model can be approximated by 1-2 hidden layers, increasing too many layers might bring down performance.

Training on Un-Tuned Network:

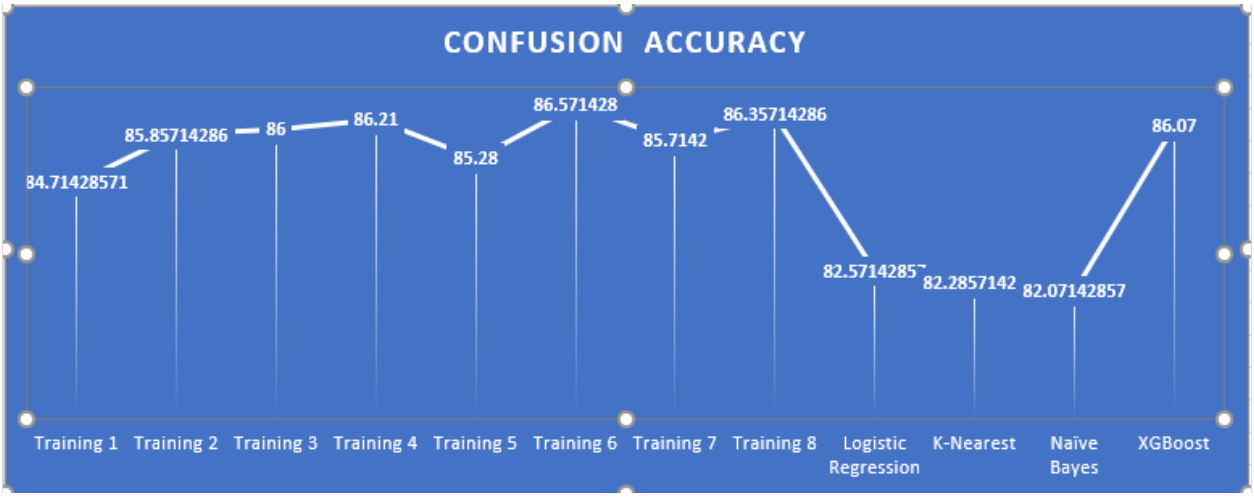
The best training set of ours gave 86.5% However, we need to check whether our neural network works good or not. We can do this by checking the accuracy by running our dataset on un-tuned basic classification models

1. Logistic Regression
2. K-Nearest Neighbor
3. Naïve Bayes
4. XGBoost

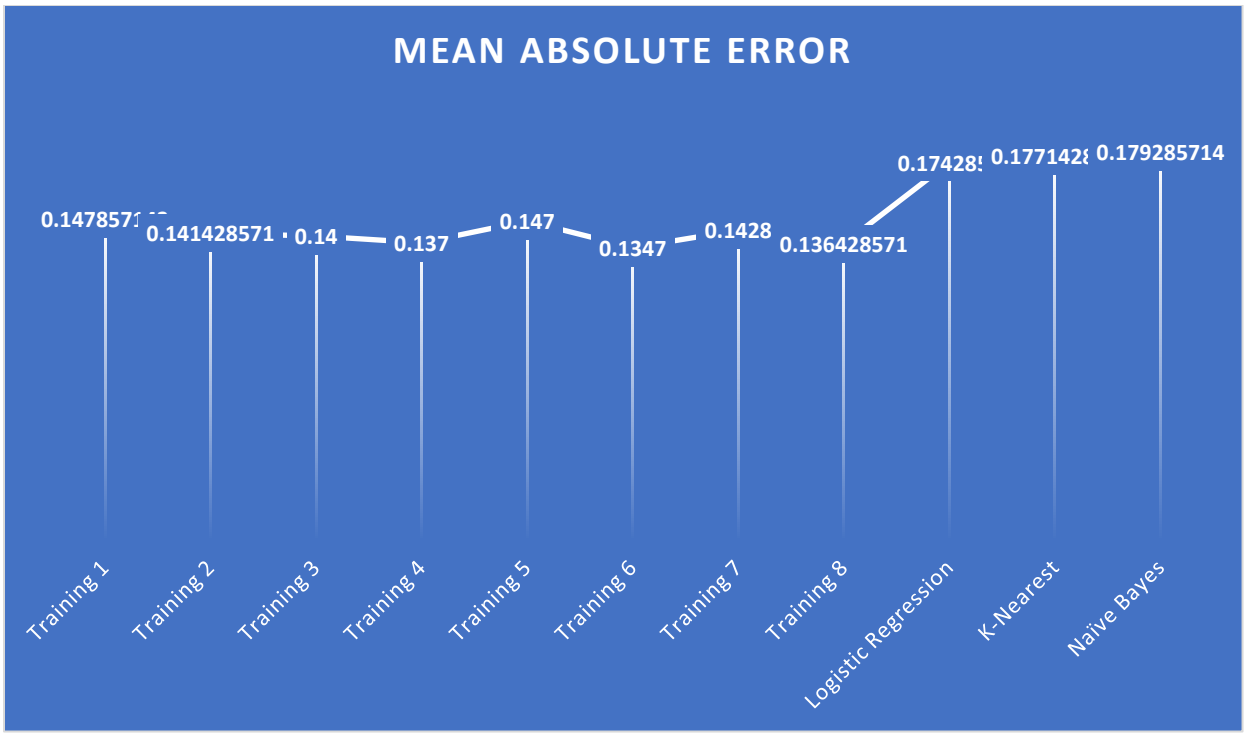
After running our dataset on the above models, we calculated accuracy, F1 score and MAE on each model. Below we have plotted graphs for Accuracy, F1 and MAE for all the models in comparison to our training sets.

Results and Conclution:

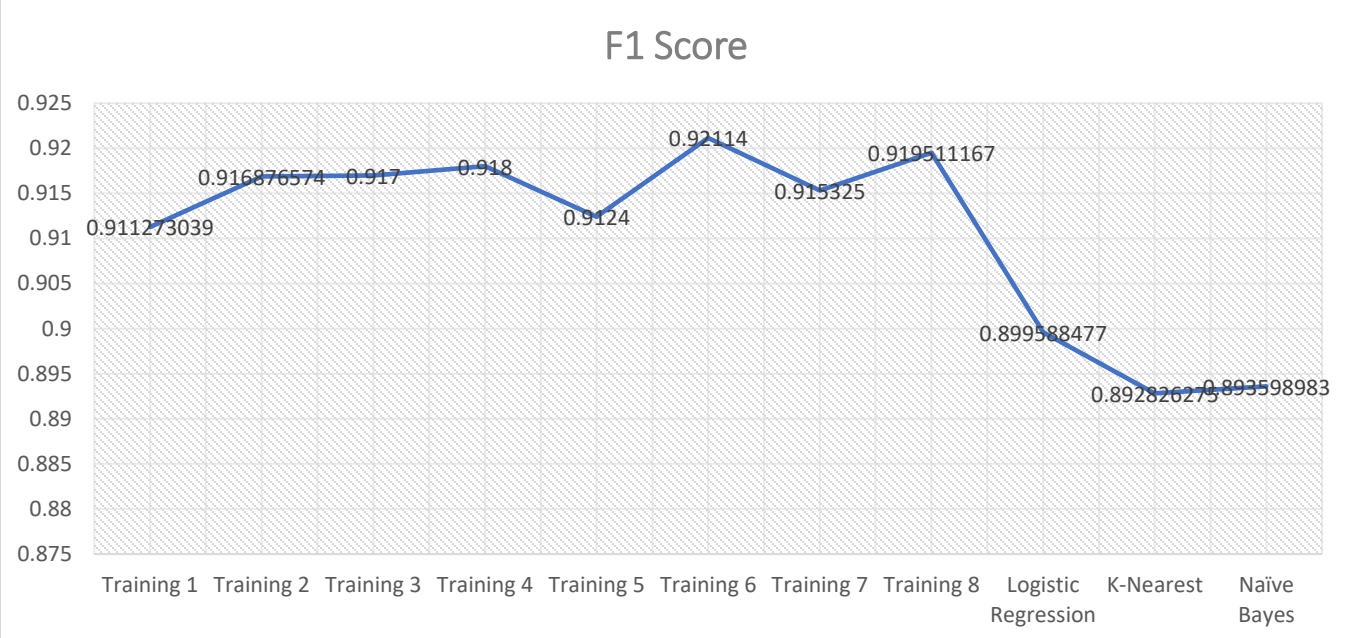
Below graph tells us the confusion matrix accuracy of various training set and un-tuned classification models. We can see that our **Training -6** has highest accuracy when compared to others.



Below graph shows the Mean Absolute Error of all the models. The lower the MAE the better it is. **Training-6** has least MAE.



Below graph shows the F1 score of all the models used. The greater the F1 score the better the model performance. **Training 6** has better score than others.



We can conclude that the below model is the best among all the other models we used for our dataset.

Hidden Layer(s)	3
Neurons per Hidden Layer(s)	3, 3 , 2
Activation function for hidden layer(s)	Tanh

Optimizer	Adam
Learning Rate	0.01
Epochs	100
Batch size	32
Early Stopping	True
Patience / Tolerance	2
Initial Weights	uniform distribution within [-limit, limit] where limit is sqrt(6 / fan_in) where fan_in is the number of input units
Initial Bias	Ones

References:

Choosing layers and neurons: https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw
Dataset: https://www.kaggle.com/filippoo/deep-learning-az-ann
Overfitting theory: https://stats.stackexchange.com/questions/306574/which-elements-of-a-neural-network-can-lead-to-overfitting
Learning Rate: https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/
Columns to Standardize: https://statmodeling.stat.columbia.edu/2009/07/11/when_to_standar/