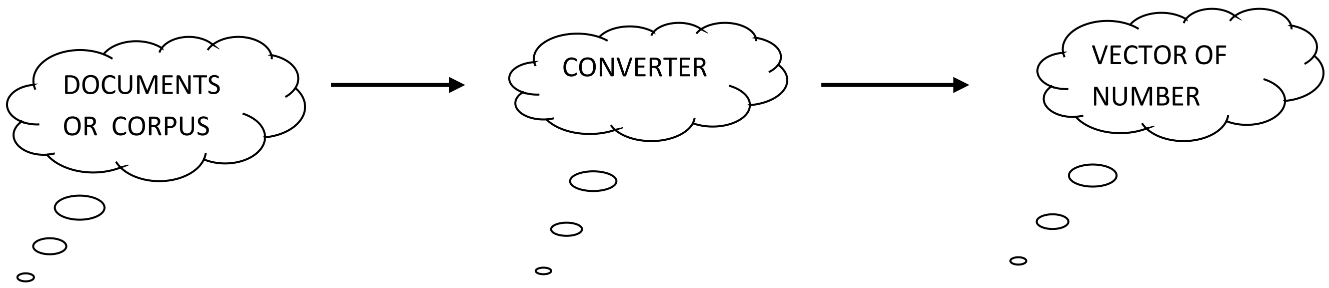
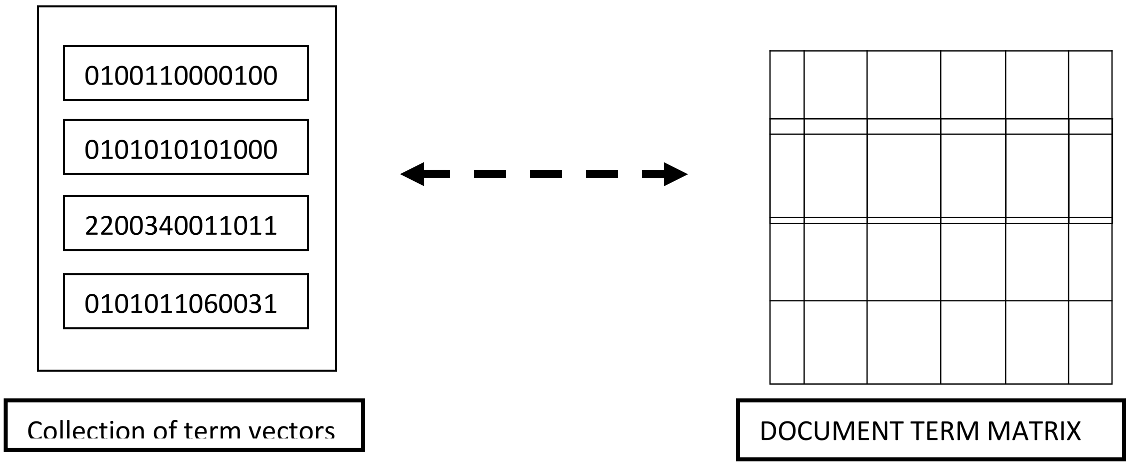
An Neural Network Based Approach

DATA SET ANALYSIS

- We have taken IMDB movie review dataset which consists of 50000 highly polarized reviews from the internet movie database where they are thus split into 25000 reviews termed as training data and other half as testing data , each consisting of 50% positive reviews and 50% negative reviews.
- This splitting into training and testing data is done in order to make the model suitable to predict those target values of data which it has not seen before. Simply speaking the training data is provided labels so that the model is trained well for the data it hasn't yet seen. So it ends up memorizing the mapping between the training and target values of the same dataset.
- IMDB movie review dataset (is similar to MNIST dataset) is a pre-processed dataset package with Keras which contains a sequence of words converted into sequence of integers where each integer is indexed to specific library in Keras.
- Here we will discard the rare words and keep the frequently occurring 10000 top most words making a manageable size vector data. Training data and testing data contains the word indices (encoding a sequence of words). Each training data and test data have train labels and test labels consisting of 1's and 0's which stand for 0 →negative word and 1→positive words.
- Our main aim is to analyse the vector of numbers which represent the word sequence encoding. Thus first the words are converted into numbers using their id's or we say the word frequency. Hence more the frequency of appearance greater is the priority given.
- Example : suppose a term vector is of the form [2, 5, 8] then the unique word encoding is set as [0, 1, 0, 0, 1, 0, 0, 1] where 2 , 5 , 8 are positional indices and hence are replaced by 0 & 1 at there respective position with total number of unique word size is 8.
- But suppose the positional indices appear as [2,5,8,2] then the term vector is shown as [0 , 1+1 , 0 , 0 1 ,0 , 0 , 1] = [0 , 2 , 0 , 0 , 1 , 0 , 0 , 1] as 2 appeared twice whereas 5 ,8 appeared once. Now the positional meaning is finished and we have created a cluster of term vectors as matrix called as Document Term Matrix.





PREPARATION OF DATA :

- For feeding into the neural networks we have to turn our list into tensors which can be carried out by various techniques but the most suitable one is by using one hot encoding method which manually vectorizes the data. In this for example [3 , 5] sequence is turned into 10000 dimensional vector with all zeros except the indices 3 and 5. These floating point vector is handled as the first layer in our network.
- While building the network we have 'input data → vector' and the ' labels → scalars of 1's and 0's' thus creating a fully connected dense layer using the activation function as ReLu. Understanding the above concept let us say

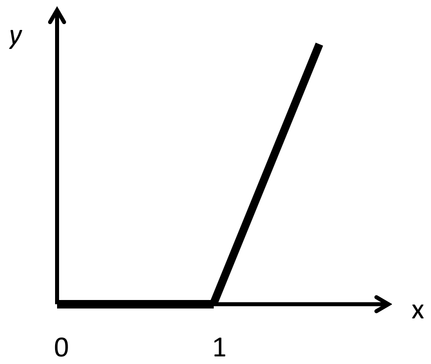
  w_i - value of i number of weights

  x_i - value of the i number of the inputs

  b – value of bias

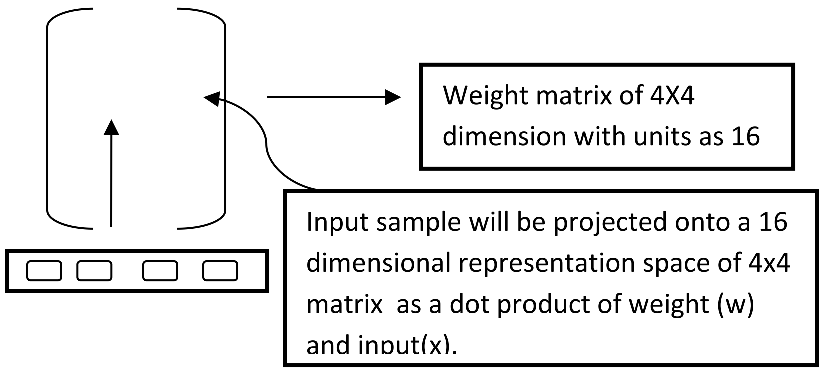
  $z = f(\sum w_i x_i + b)$

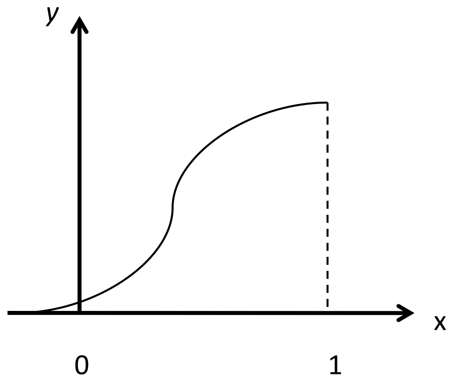
Here we have the activation function as ReLu or we rectified linear unit whose graph is shown as



Output = relu(dot(W , input) + b) *known as the tensor operation in NN*

So 10000 word samples as inputs passed through 16 hidden units (dimension in the representational space in the layer) of dense layer. Here the dimension means the amount of freedom we have given to the network when learning the internal representation but complex representation means higher dimension which results in NN to be computationally expensive and even over-fits the model by training the unwanted pattern in the train data. Thus to avoid such we consider using CNN - convolutional neural networks.

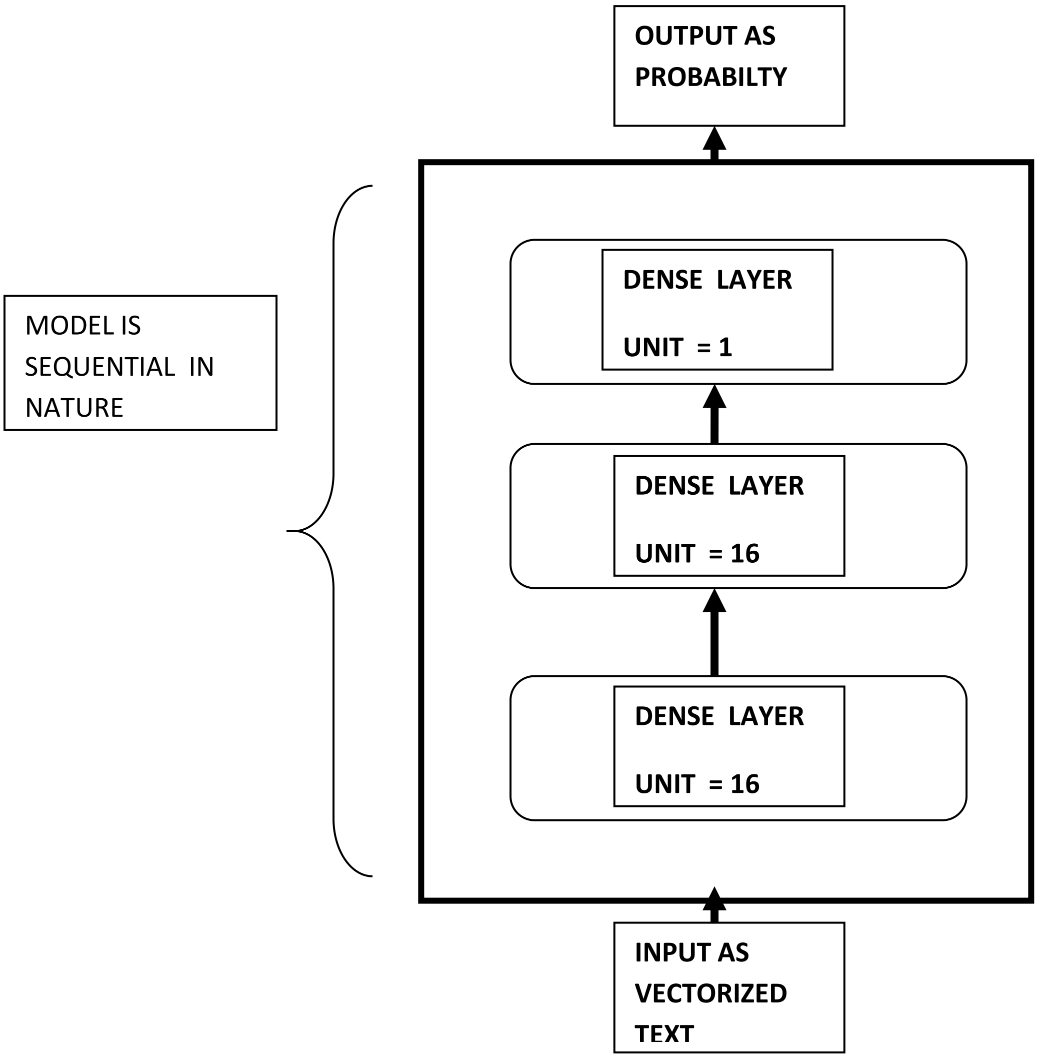

Weight matrix of 4X4 dimension with units as 16
Input sample will be projected onto a 16 dimensional representation space of 4x4 matrix as a dot product of weight (w) and input(x).



NOTE : ReLu activation function is used to zero out negative values whereas sigmoid squashes arbitrary values into [0,1] interval outputting something that can be seen or interpreted as a probability.

Sigmoid function is used at output so that the output is a probability with a score between 0 and 1 . We can also prefer using softmax activation function output for better classification.

HERE'S HOW THE NETWORK LOOKS LIKE :



- The input undergoes a linear transformation or more generalised version we say affine transformation where input data is piped into a hypothesis space (the space containing set of all possible linear transformation) into 16 dimensional space. This hypothesis space is too restricted and a deep stack of layer will still implement a linear operation and does not affect or extend the hypothesis space.
- Finally we choose the loss function and an optimizer. As we know the output is a single unit layer with a sigmoid activation thus a output of the network is probabilistic in nature or we say binary class problem. Henceforth we choose a binary_crossentropy loss. Other option are mean squared error loss (mse loss). But in ANN crossentropies are best suitable. So what is a cross entropy??
- Crossentropy is the quantity from the field of information theory that measures the distance between probability distributions or say the distance between the distribution of actual and predicted data hence the loss!!
- There are various optimizers available, here we have used rms_prop optimizers for fine tuning the network. Some of the optimizers are explained below:
  I. GRADIENT DESCENT : *It is a simple and effective method to find the optimum values for the neural network. The objective of all optimizers is to reach the global minima where the cost function attains the least possible value.*
  II. RMS PROP : *The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster***.**
  III. ADAM : *Adam is an optimization algorithm that can used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. he method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients*

WHAT IS METRICS? HOW DO WE DEFINE ACCURACY METRICS?

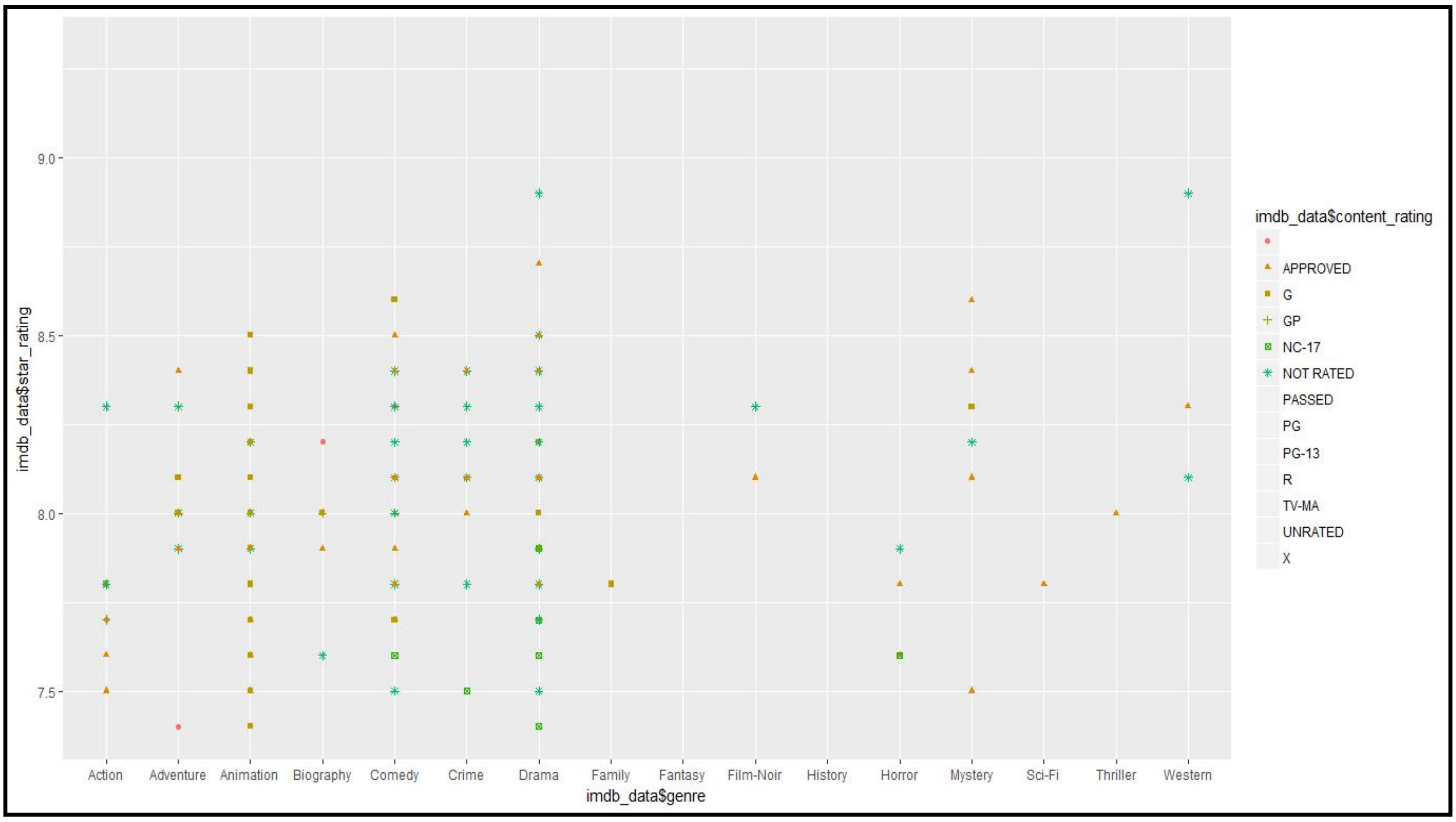
It is the performance test measurement value typically shown by the confusion matrix which gives us an idea how near we are to the predicted value and how much deviation we have regarding the actual data.

IMPLEMENTATION :

a) VISUALZATION AND PLOTTING OF THE IMDB DATASET :

USAGE OF PACKAGE : ggplot2

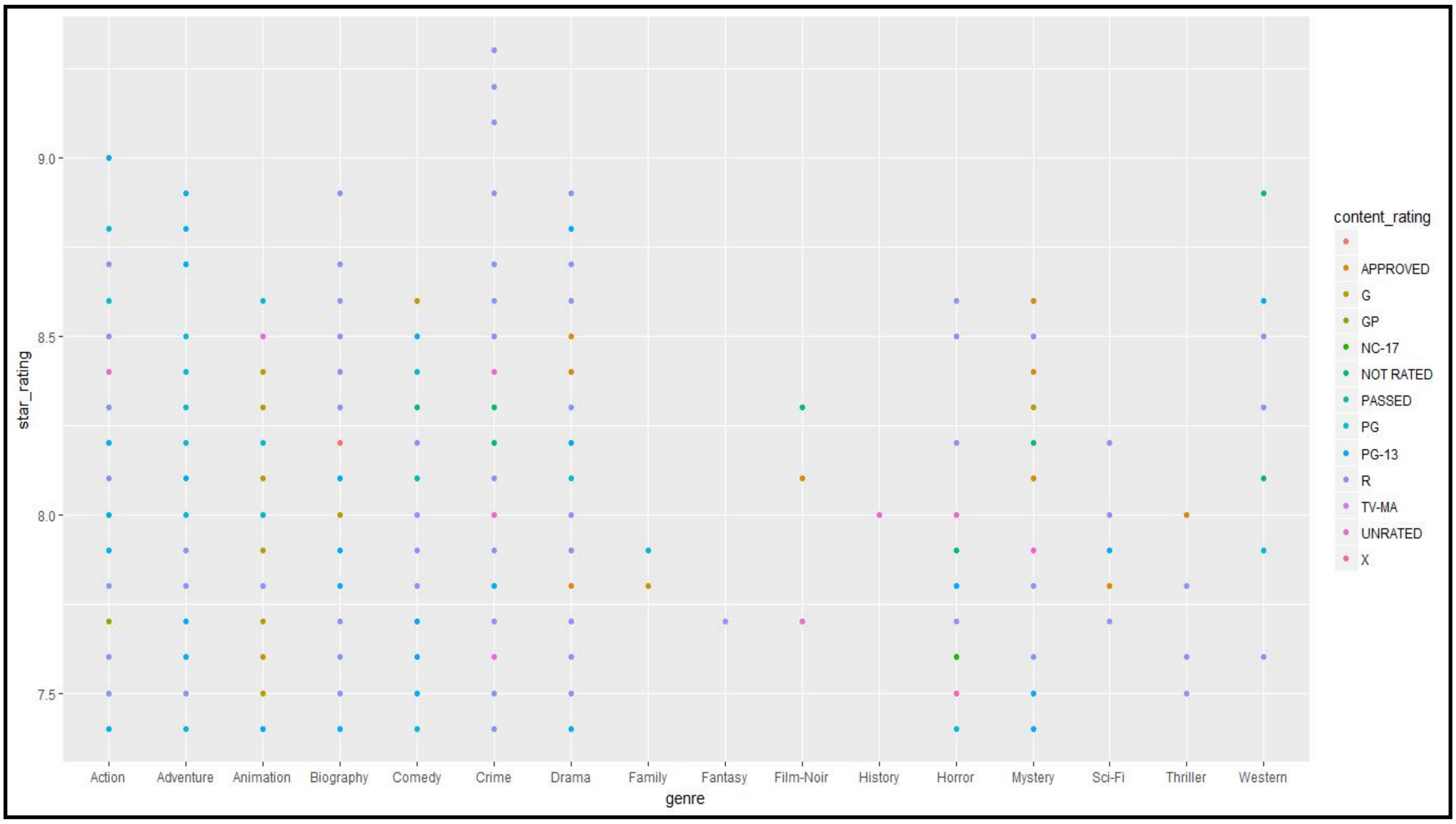
Developed by Hadley Wickham this package enabled us in R to have good visualization and exploration of data. Defined as Grammar of Graphics its approach in handling the elements of a graph is versatile and provides a number of feature for better control. Some of the important components of ggplot( ) are namely 1) Aesthetics 2) qplot( ) function 3) ggplot( ) object 4) geoms , facets and scales.



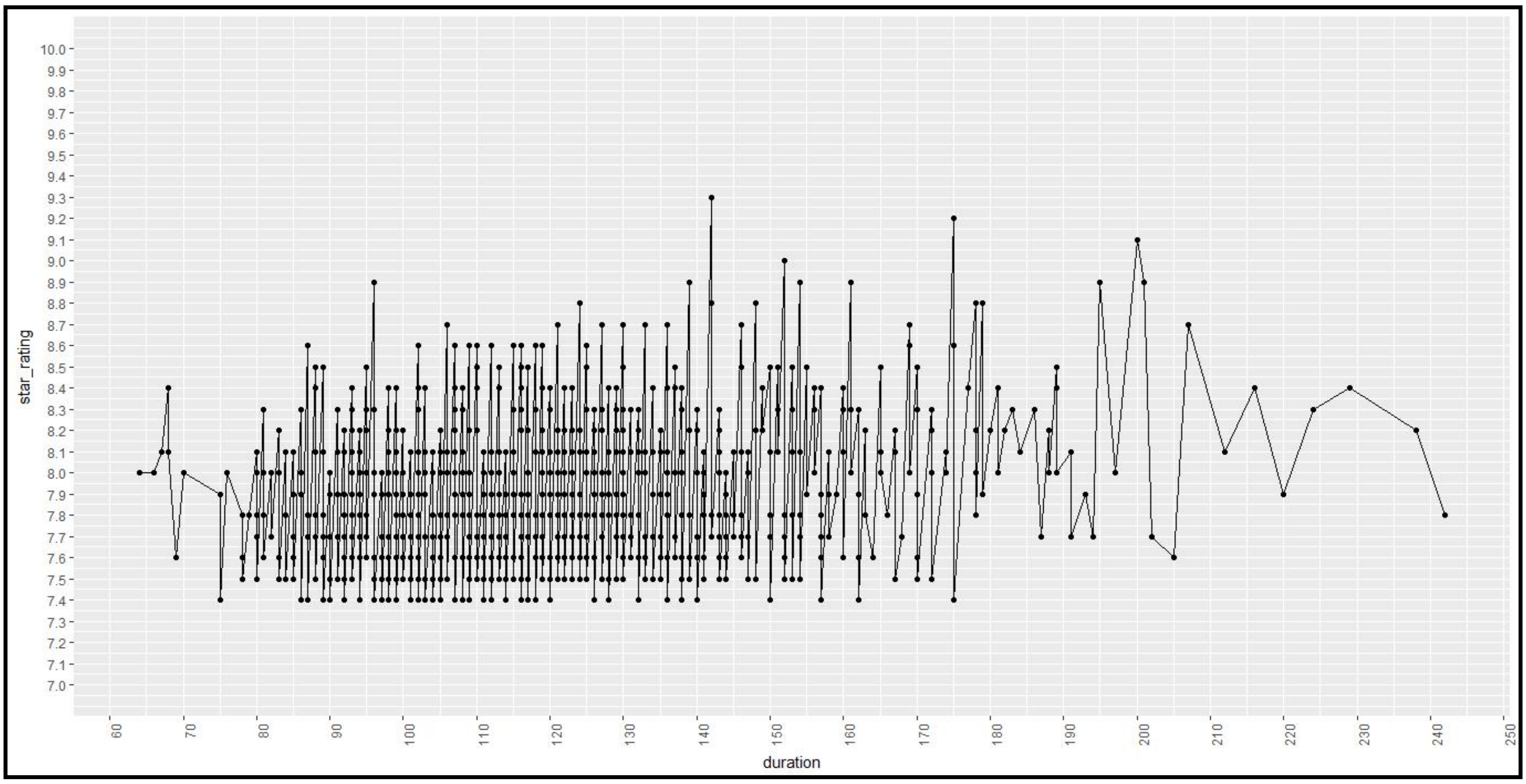
Here we have used qplot( ) for the plotting the star_rating vs genre with content_rating as the shape for distinguishing but the major problem is that the plot is not well defined and cumbersome at some areas. So we switch to more efficient method using ggplot2( )

A small overview of ggplot( ) we can describe – grammar of graphics by Hadley Wickham breaks the graphs into elements and then building up layer for each element for better controlling the plot visualization. This methodology has two aspects – number one the aesthetics of the graph which are x and y attribute and colour , shape , size and secondly the geometric which can be bar , point or line with some statistical element.

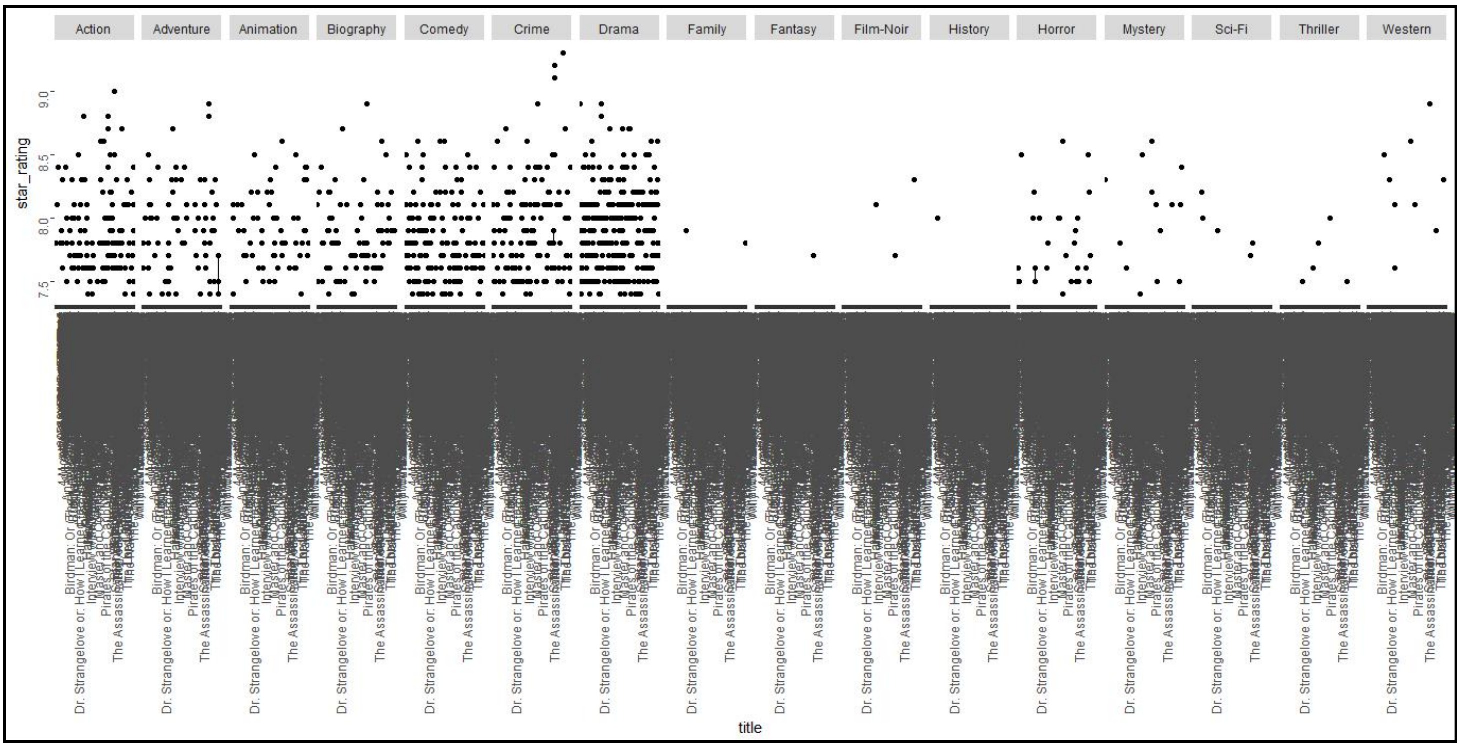
```
ggplot(data_frame, aes(x_label, y_label , colour , size , shape)) +
geom_point/bar/line()
```



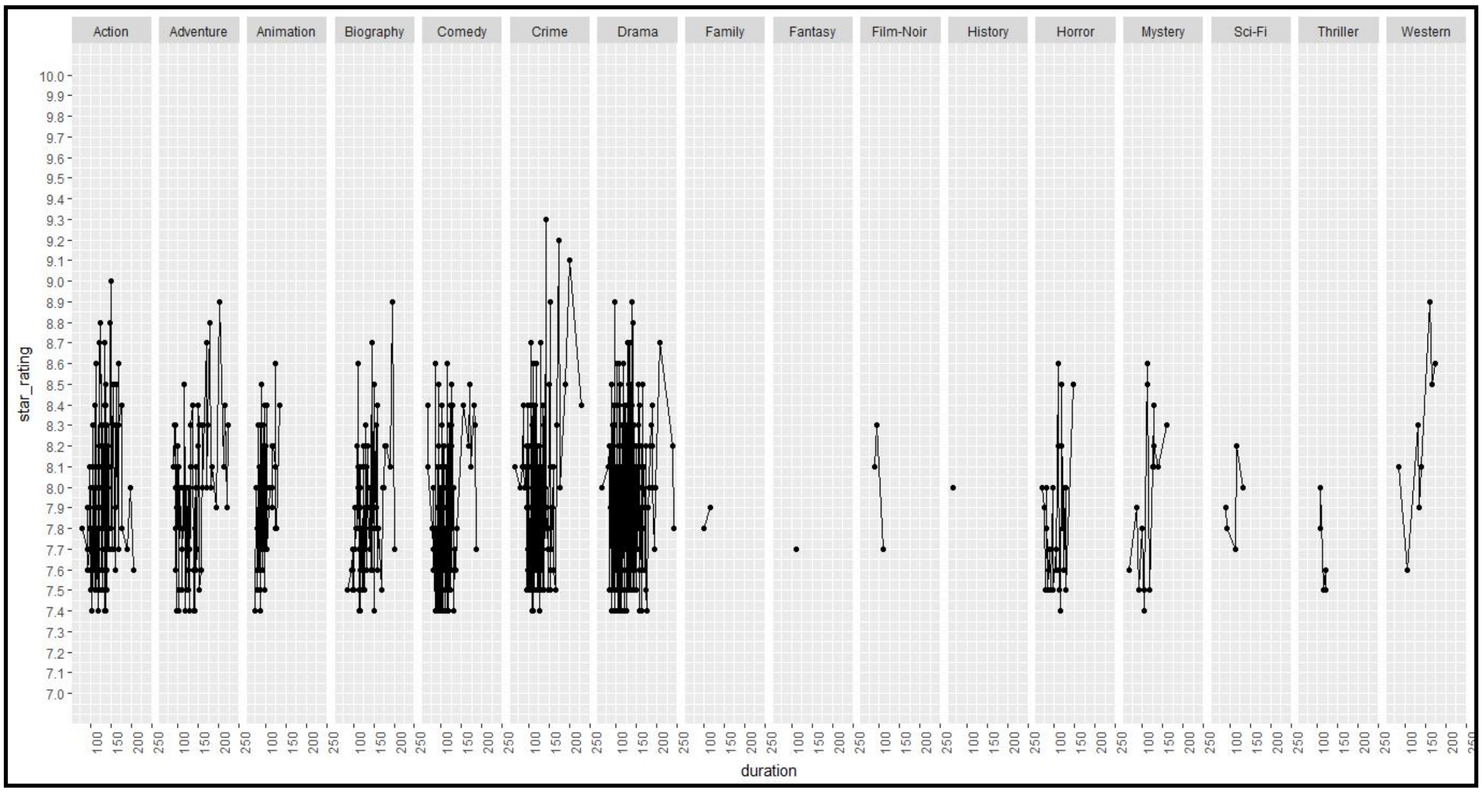
Here we have plotted the same columns data what we did for qplot( ) but using the ggplot( )



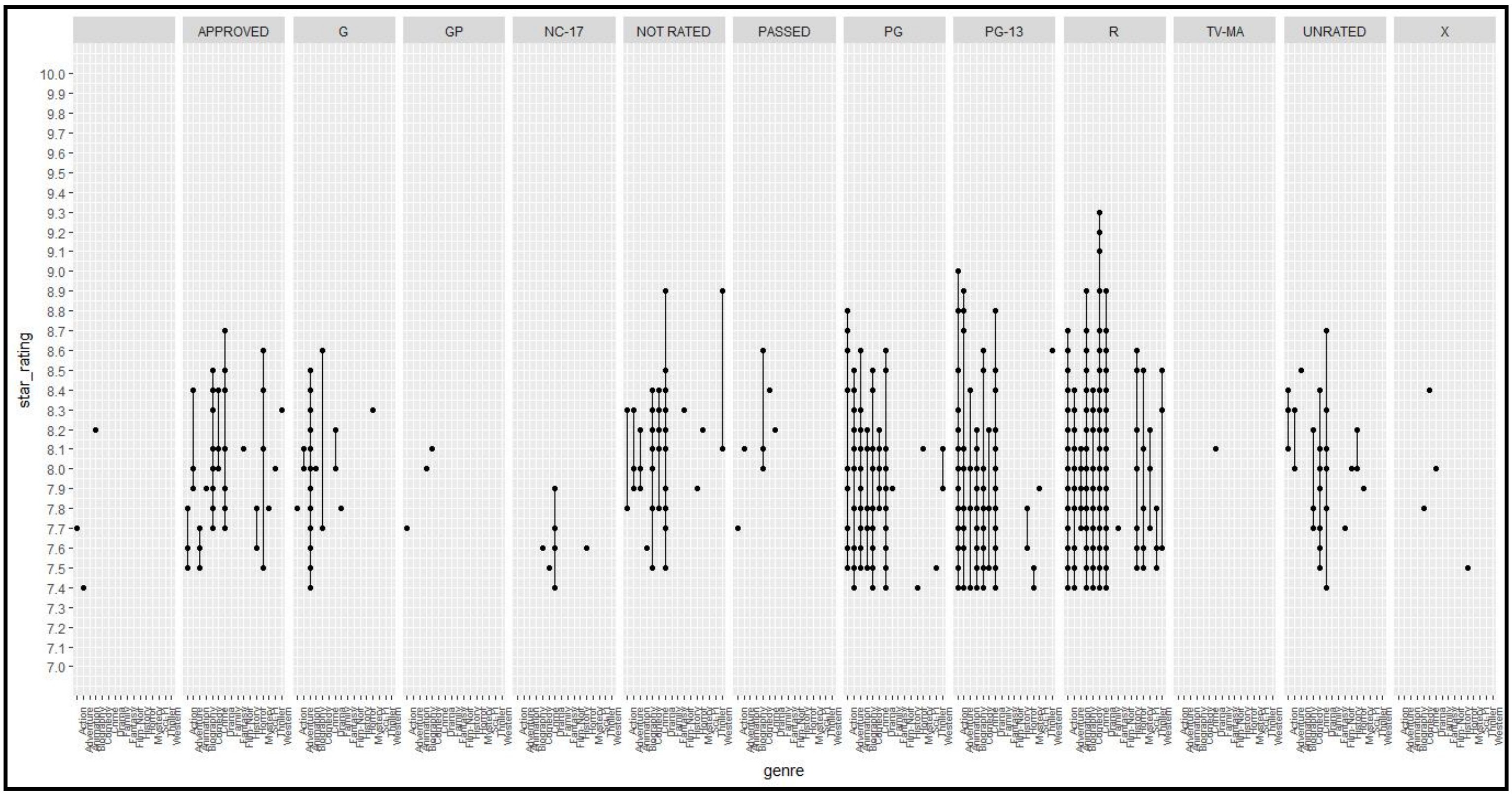
Here our plot shows duration vs star rating and how it impacts the graph.



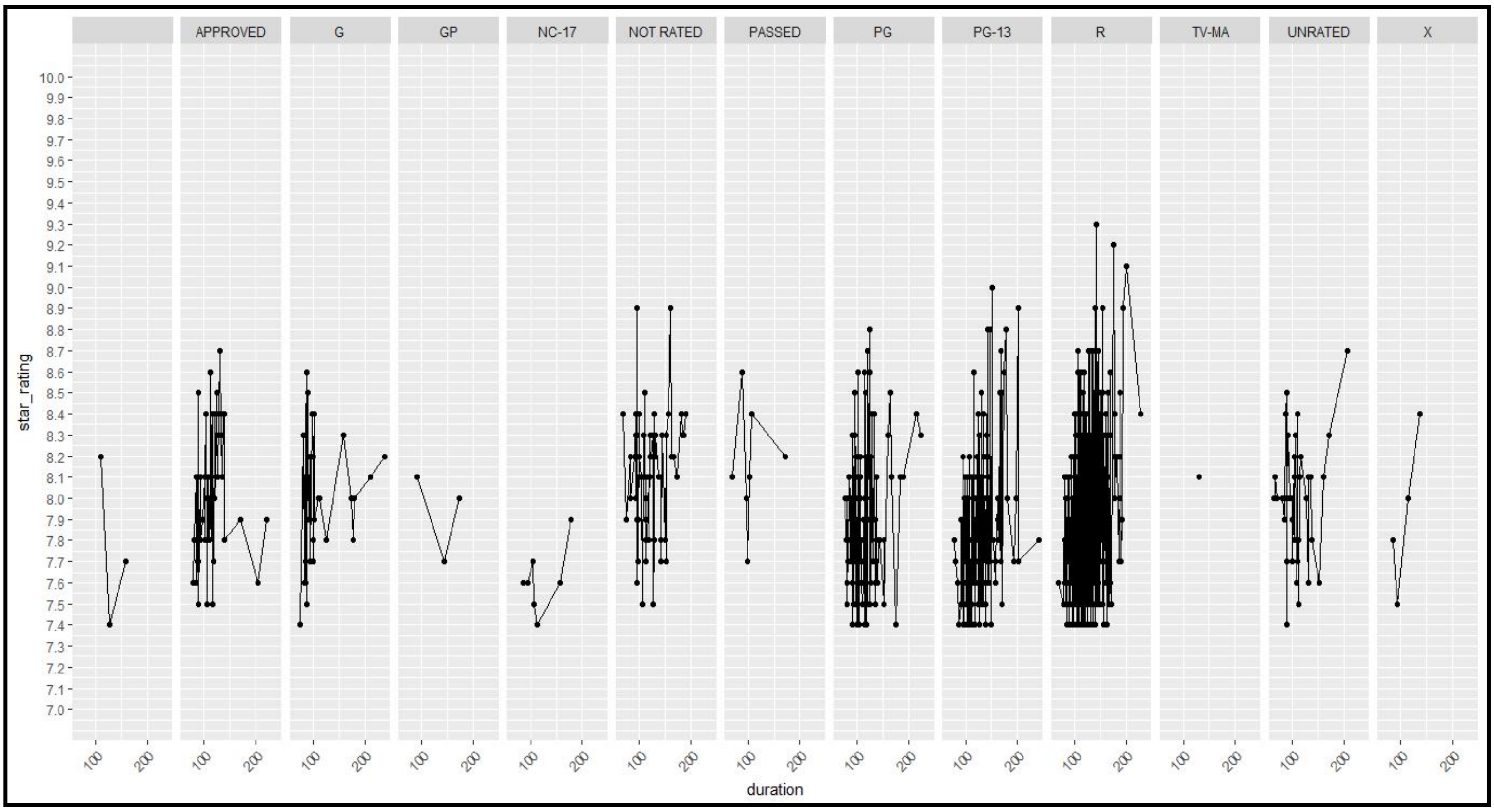
Here facets have been created with classifying title and its respective genre vs the star rating.



Here we specific classification based on genre with duration vs star_rating



Here we have plotted genre vs star_rating with the facets as content_rating



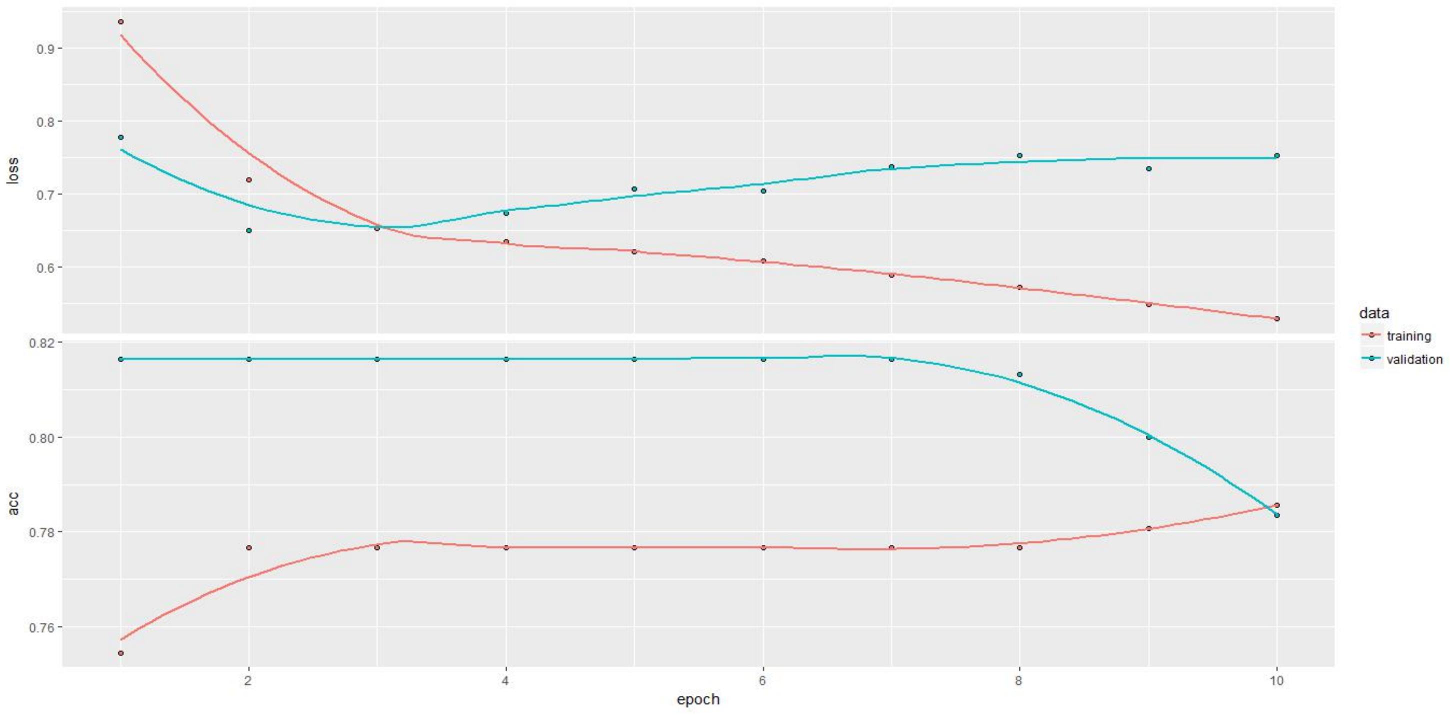
Here we have plotted duration vs star_rating with facets as content rating

Code Review:

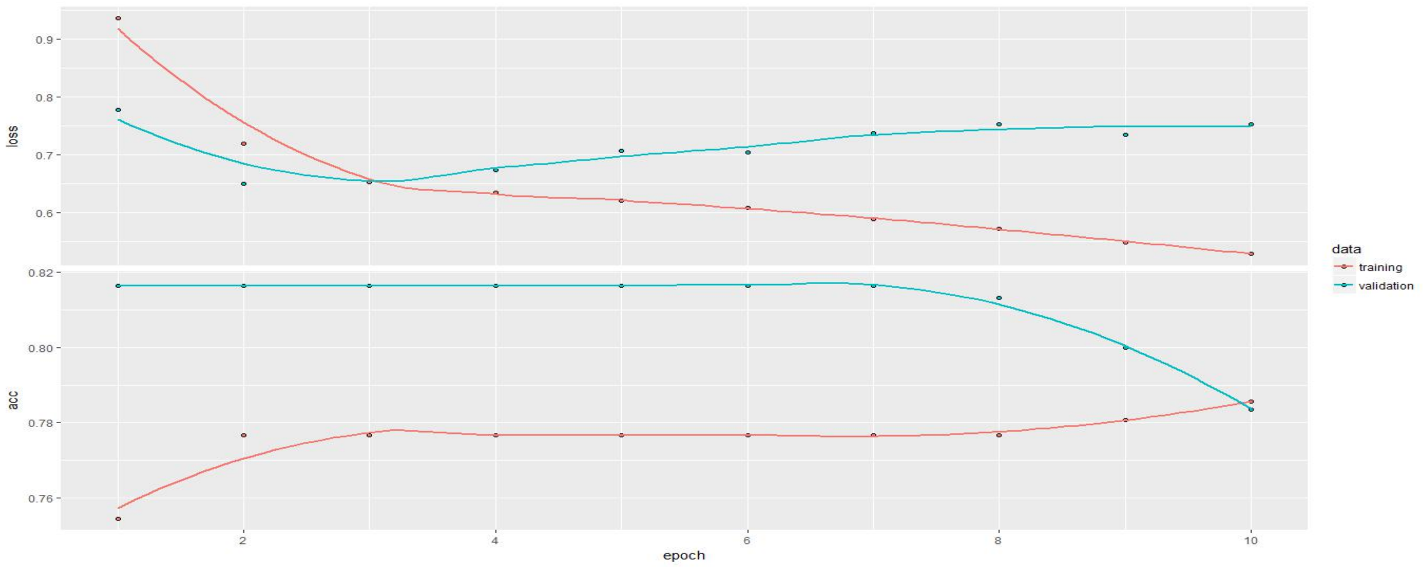
```
library(ggplot2)

library(plyr)

imdb_data <- read.csv(file.choose() , header = T)

names(imdb_data)

## here we are plotting title against content rating

qplot(data = imdb_data , imdb_data$genre , imdb_data$star_rating , shape =imdb_data$content_rating , colour = imdb_data$content_rating)

## using gglot2 for plotting the various elements ##

ggplot(imdb_data , aes(x = genre , y = star_rating , colour = content_rating , shape = content_rating)) + geom_point()

ggplot(imdb_data , aes(x = duration , y = star_rating)) + geom_point() + geom_line() + theme(axis.text.x = element_text(angle = 90 , hjust = 1)) + scale_x_continuous(breaks = seq(from = 0 , to = 250 , by = 10)) + scale_y_continuous(limit = c(7 , 10) , breaks= seq(from = 7 , to = 10 , by = 0.1))

## using ggplot2 for facet construction ##

ggplot(imdb_data , aes(x = title , y = star_rating)) + geom_point() + geom_line() + theme(axis.text = element_text(angle = 90 , hjust = 1)) + facet_grid(~genre)

ggplot(imdb_data , aes(x = duration , y = star_rating)) + geom_point() + geom_line() + theme(axis.text.x = element_text(angle = 90 , hjust = 1)) + scale_y_continuous(limit = c(7 , 10) , breaks = seq(from = 7 , to = 10 , by = 0.1)) + facet_grid(~genre)

ggplot(imdb_data , aes(x = genre , y = star_rating)) + geom_point() + geom_line() + theme(axis.text.x = element_text(angle = 90 , size = rel(0.8) , hjust = 1)) + scale_x_discrete(labels = c("Action " , "Adventure " , "Animation " , "Biography " , "Comedy " , "Crime " , "Drama " , "Family " , "Fantasy " , "Film-Noir " , "History " , "Horror " , "Mystery " , "Sci-Fi " , "Thriller " , "Western ")) + scale_y_continuous(limit = c(7,10) , breaks = seq(from = 7 , to = 10 , by = 0.1)) + facet_grid(~content_rating)

ggplot(imdb_data , aes(x = duration , y = star_rating)) + geom_point() + geom_line() + theme(axis.text.x = element_text(angle = 45 ,hjust = 1)) + scale_x_continuous(breaks = seq(from = 0 , to = 250 , by =100)) + scale_y_continuous(limit = c(7 , 10) , breaks = seq(from = 7 , to = 10 , by = 0.1)) + facet_grid(~content_rating)
```

DEEP LEARNING IMPLEMENTATION WITH VARIOUS RESULTS:

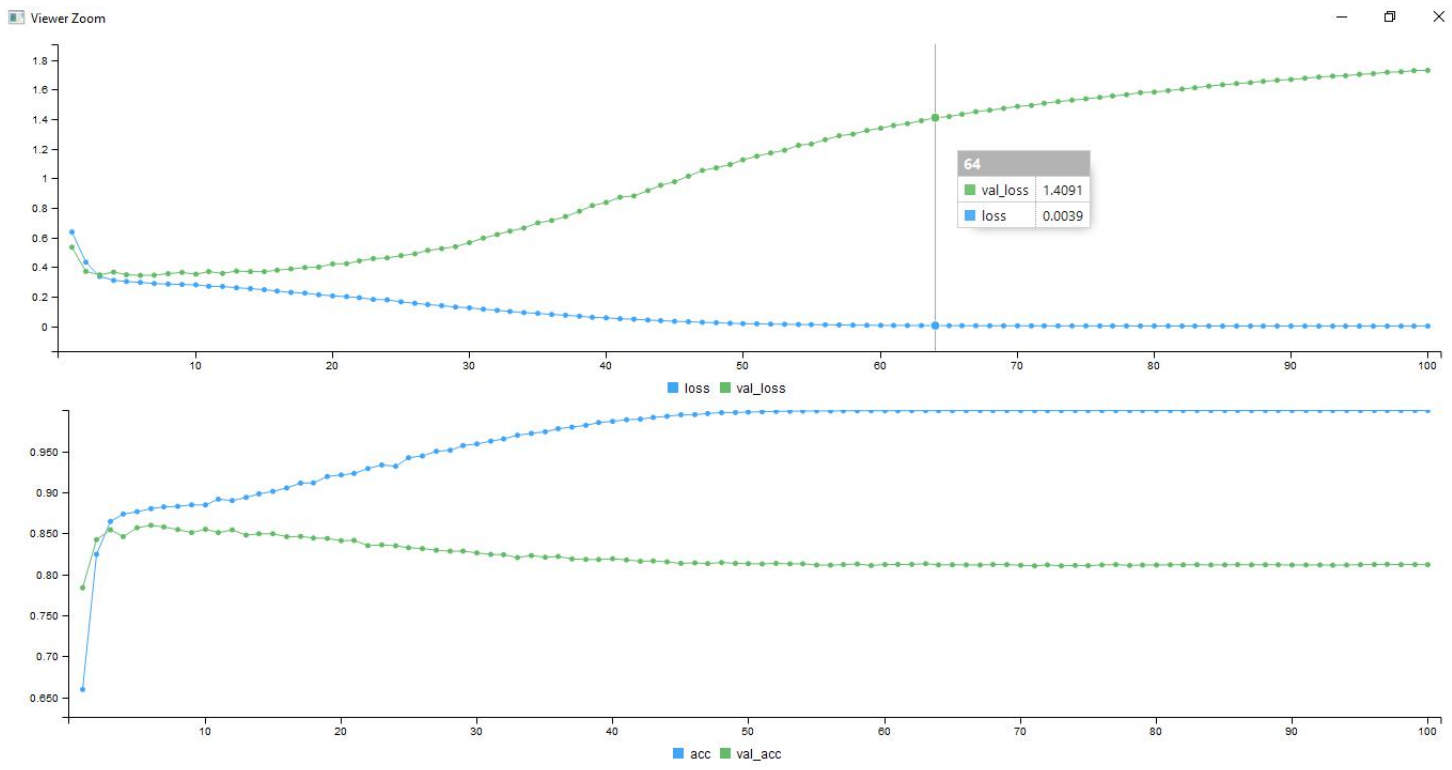
- (1) SINGLE LAYER HIDDEN UNITS : 16 OPTIMIZER : **RMSPROP**
  ACTIVATION AT INPUT : **RELU** ACTIVATION AT OUTPUT : **SIGMOID**
  LOSS FUNCTION : **BINARY CROSSENTROPY**

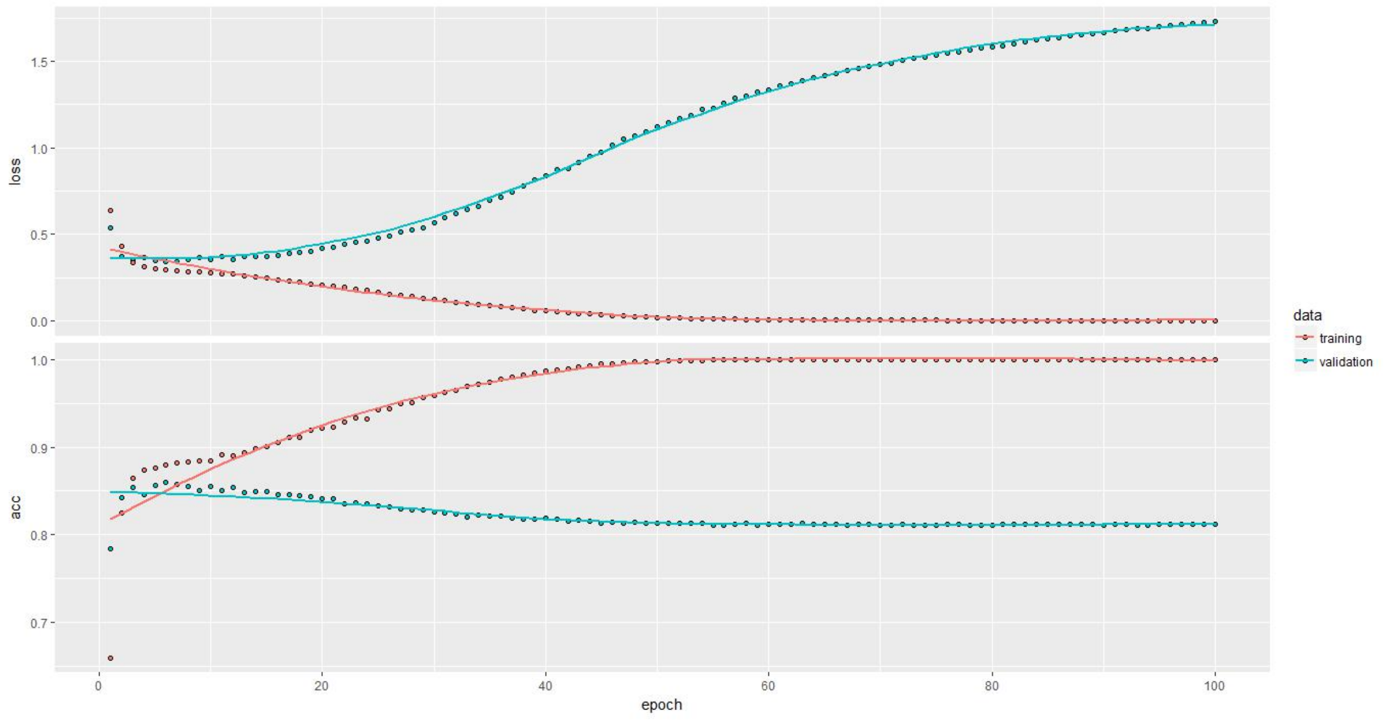


- (2) DOUBLE LAYER HIDDEN UNITS : 16 OPTIMIZER : **RMSPROP**
  ACTIVATION AT INPUT : **RELU** ACTIVATION AT OUTPUT : **SIGMOID**
  LOSS FUNCTION : **BINARY CROSSENTROPY**



- (3) DOUBLE LAYER HIDDEN UNITS : 16 OPTIMIZER : **ADAM**
  ACTIVATION AT INPUT : **RELU** ACTIVATION AT OUTPUT : **SIGMOID**
  LOSS FUNCTION : **BINARY CROSSENTROPY**
  TRAINING DATA





TESTING DATA

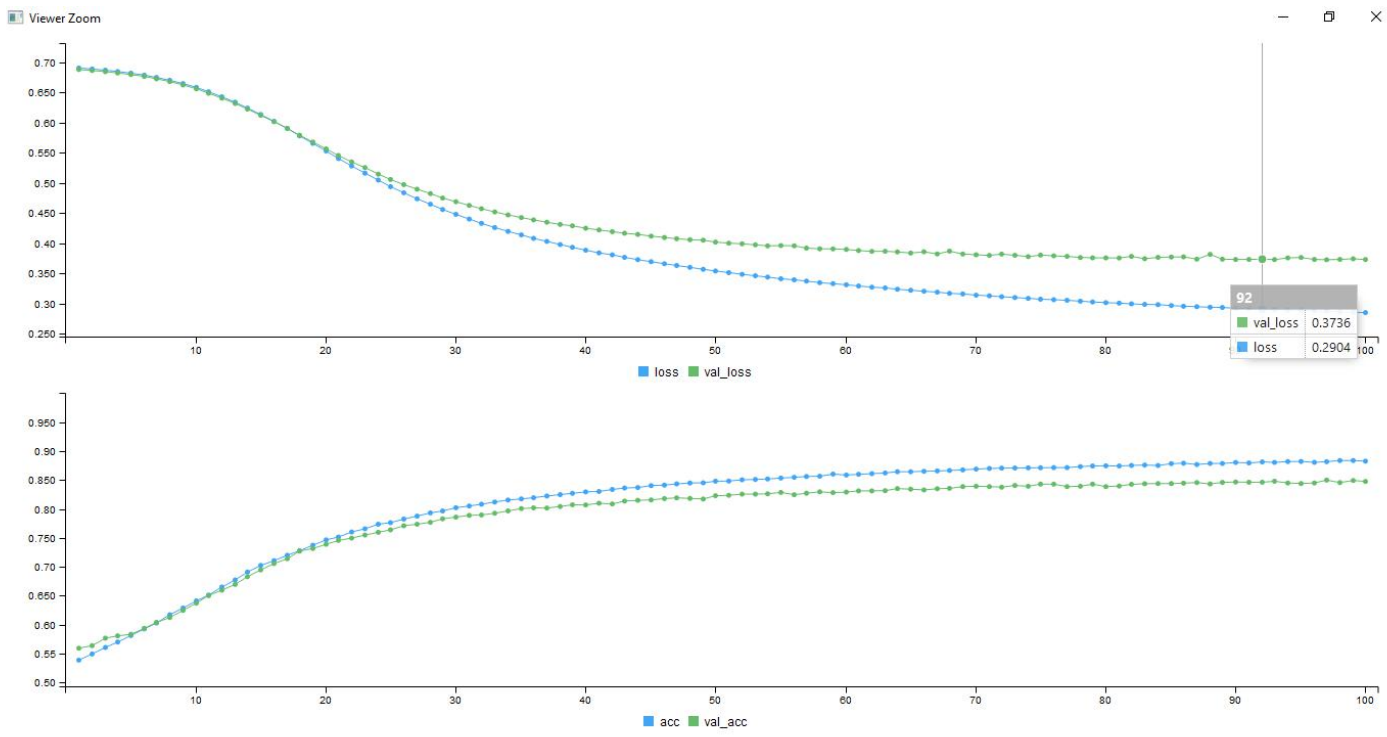
```
evaluate(x_test , y_test)
25000/25000 [==============================] - 1s 39us/step
> results
$loss
[1] 1.789788

$acc
[1] 0.80312

model%>%
+   predict(x_test[1:10,])
                [,1]
 [1,] 4.264363e-04
 [2,] 1.000000e+00
 [3,] 9.918563e-01
 [4,] 1.000000e+00
 [5,] 3.984695e-09
 [6,] 9.999999e-01
 [7,] 1.000000e+00
 [8,] 2.367968e-33
 [9,] 9.999998e-01
[10,] 9.999722e-01
```

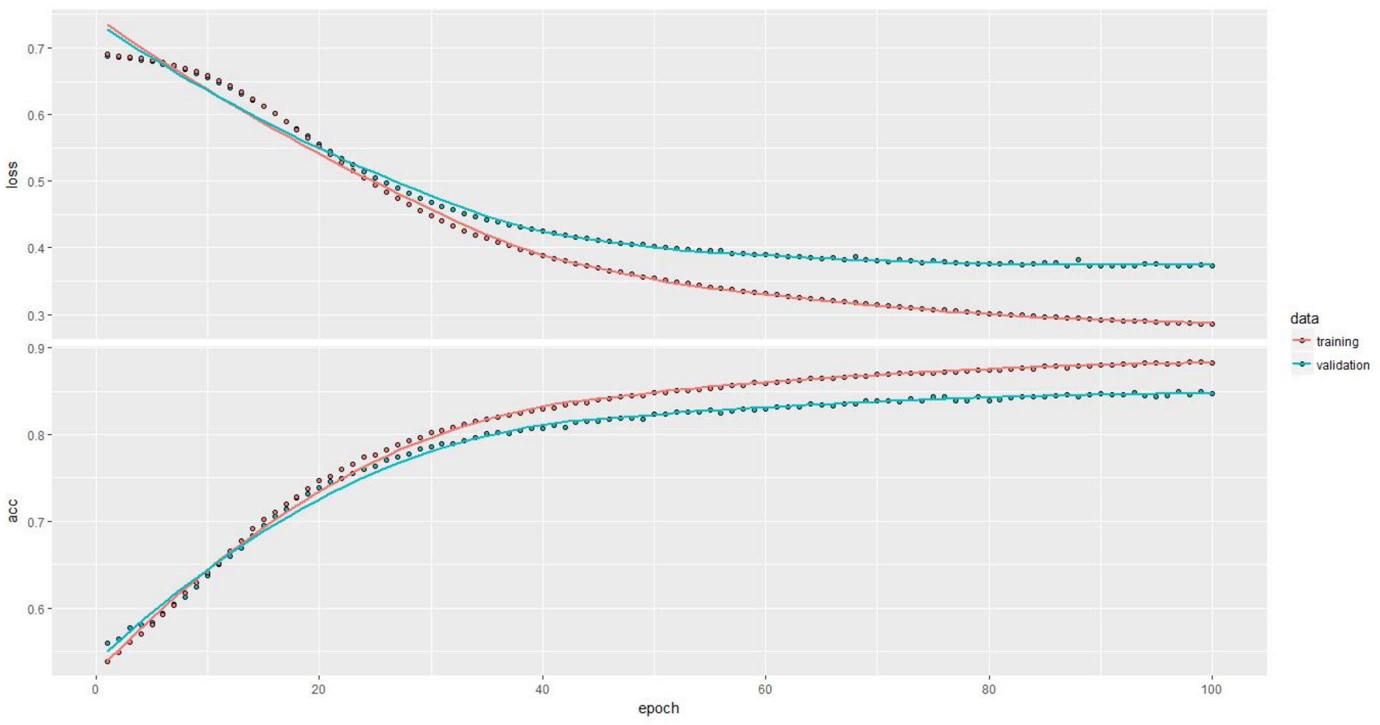
- (4) DOUBLE LAYER HIDDEN UNITS : 16 OPTIMIZER : **SGD**

  ACTIVATION AT INPUT : **RELU** ACTIVATION AT OUTPUT : **SIGMOID**

  LOSS FUNCTION : **BINARY CROSS ENTROPY**

TRAINING DATA





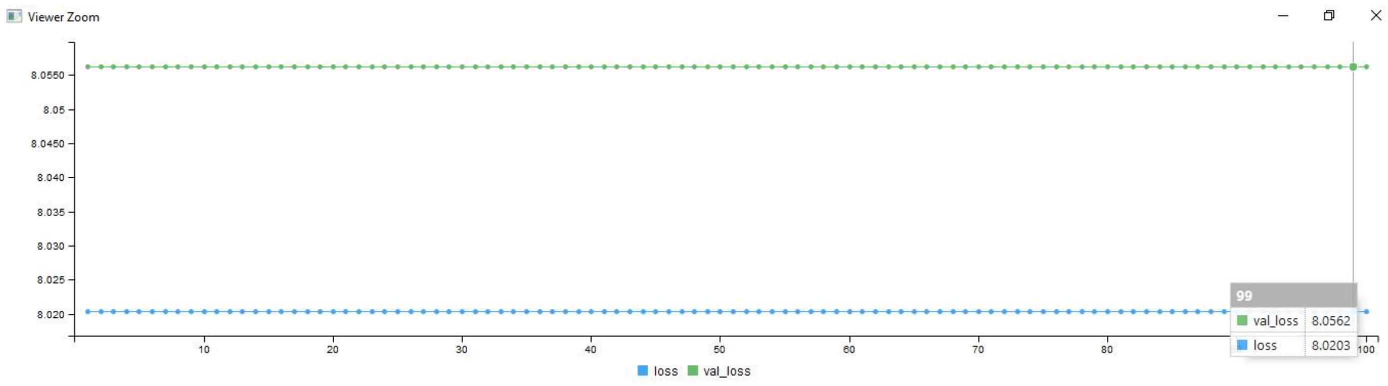
TESTING DATA

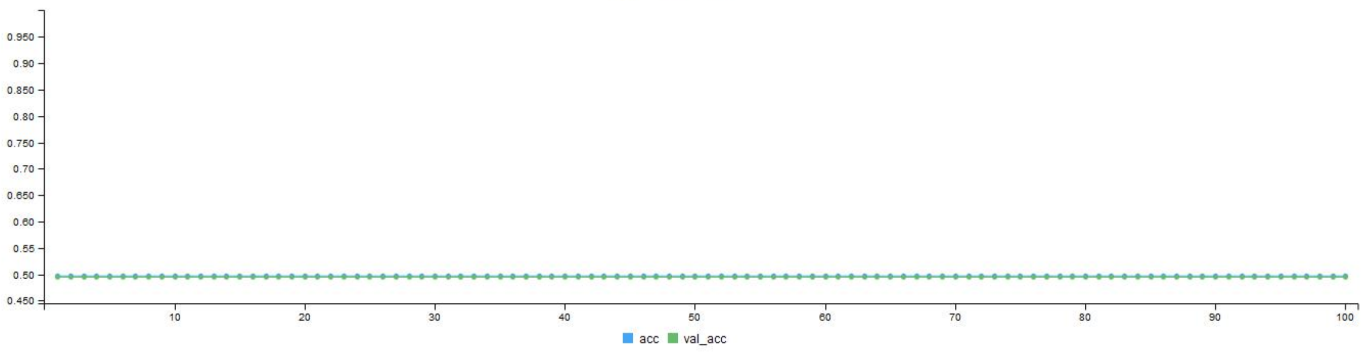
```
results <- model%>%

+ evaluate(x_test , y_test)
```

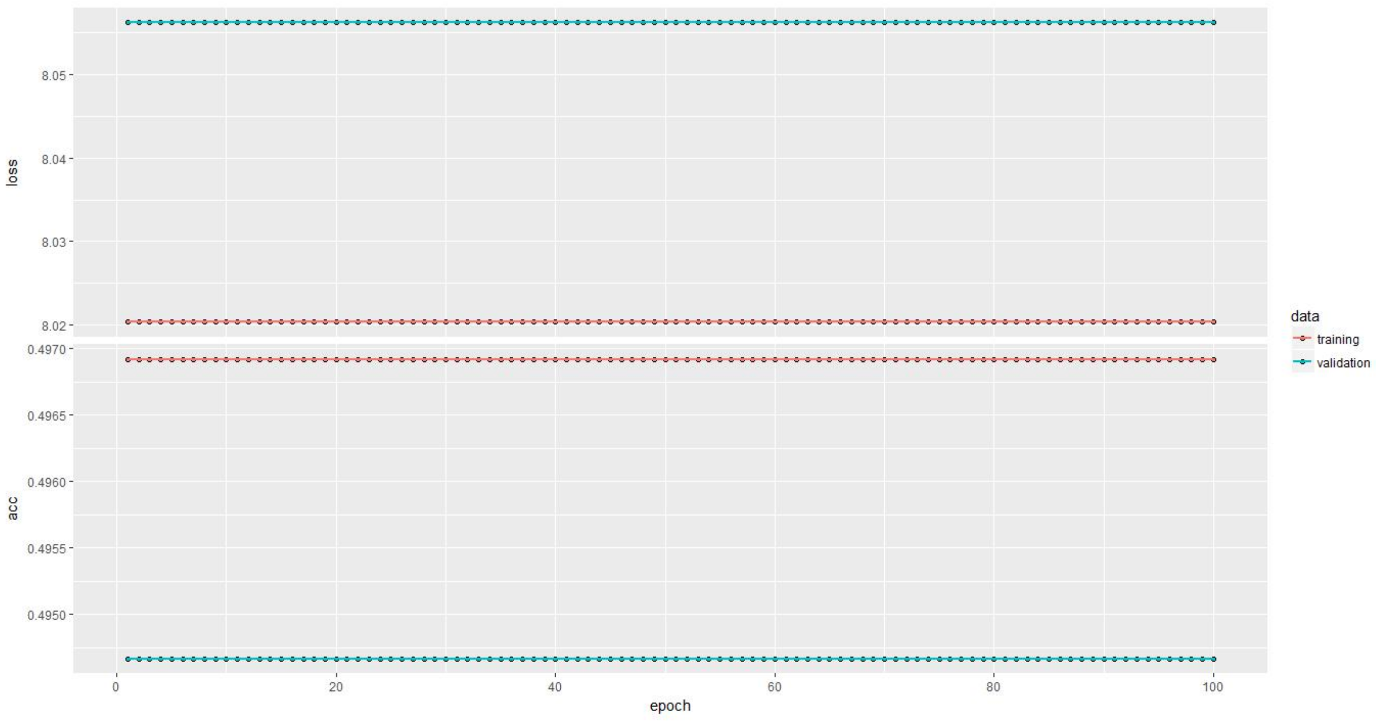
```
25000/25000 [==============================] - 1s 53us/step
> results
$loss
[1] 0.3582359

$acc
[1] 0.84836


model%>%
+   predict(x_test[1:10,])
             [,1]
 [1,] 0.42266974
 [2,] 0.98292542
 [3,] 0.77043903
 [4,] 0.65409279
 [5,] 0.83771586
 [6,] 0.58505362
 [7,] 0.99567032
 [8,] 0.04521511
 [9,] 0.96939003
[10,] 0.85922599
```

- (5) DOUBLE LAYER HIDDEN UNITS : 16 OPTIMIZER : **ADA DELTA** ACTIVATION AT INPUT : **RELU** ACTIVATION AT OUTPUT : **SOFTMAX** LOSS FUNCTION : **BINARY CROSS ENTROPY**

TRAINING DATA







TESTING DATA

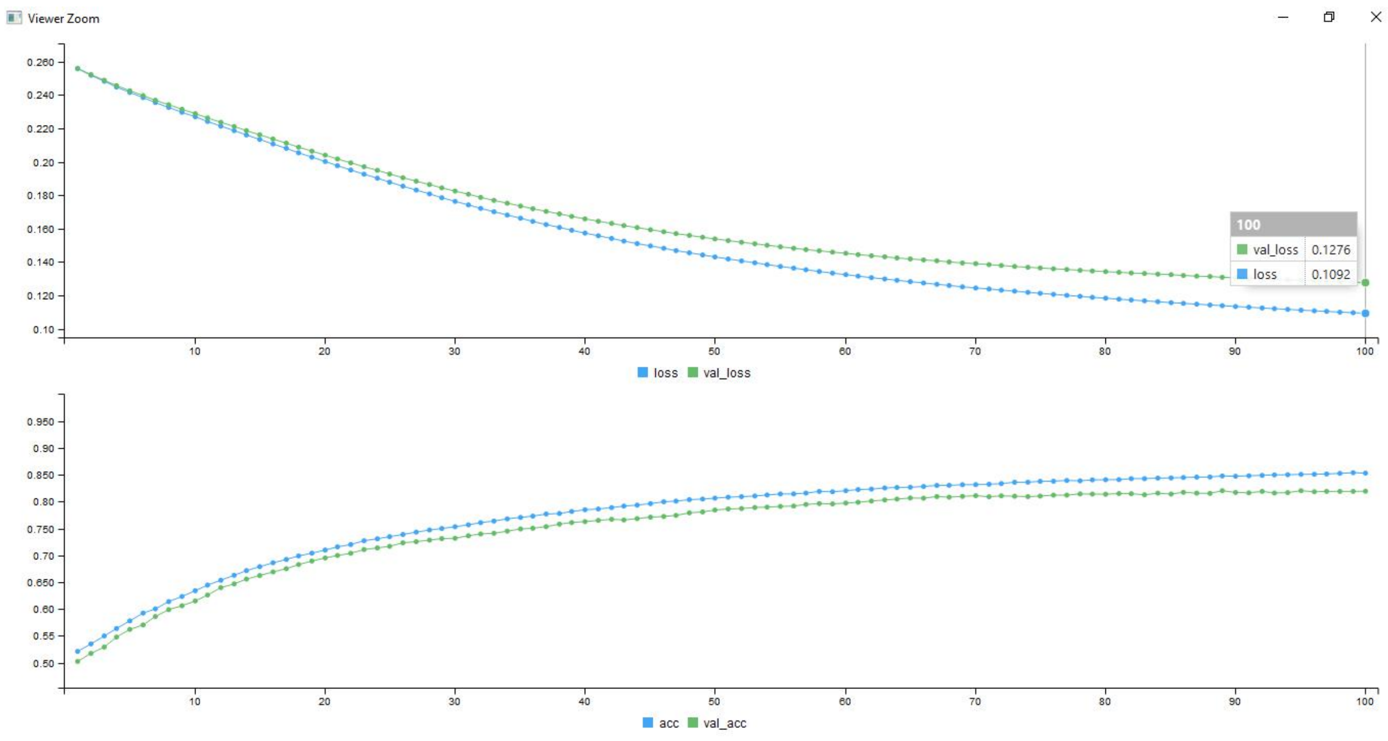
```
results <- model%>%
+   evaluate(x_test , y_test)
25000/25000 [==============================] - 1s 42us/step
> results
$loss
[1] 7.971192

$acc
[1] 0.5

model%>%
+   predict(x_test[1:10,])
      [,1]
 [1,]    1
 [2,]    1
 [3,]    1
 [4,]    1
 [5,]    1
 [6,]    1
 [7,]    1
 [8,]    1
 [9,]    1
[10,]    1
```

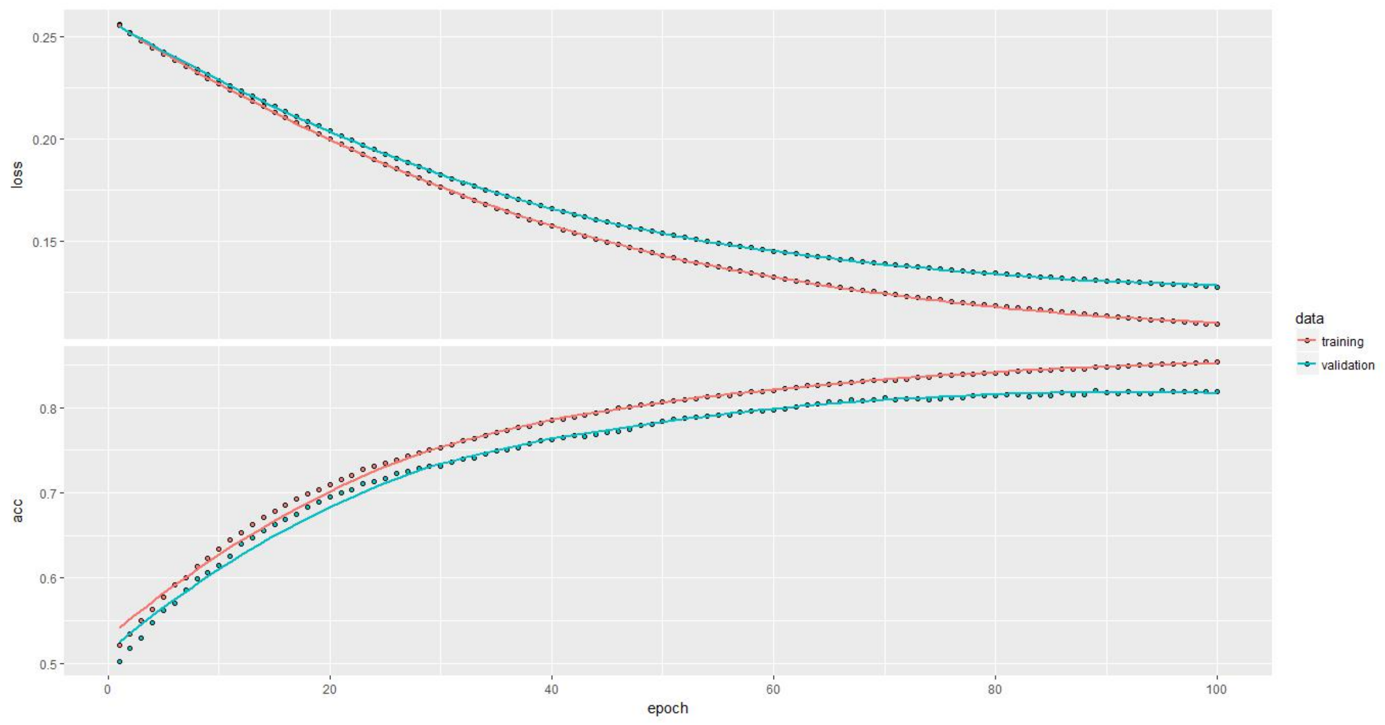
- (6) DOUBLE LAYER HIDDEN UNITS : 32 OPTIMIZER : **SGD**

  ACTIVATION AT INPUT : **TANH** ACTIVATION AT OUTPUT : **SIGMOID**

  LOSS FUNCTION : **MEAN SQUARE ERROR (MSE)**

  TRAINING DATA





TESTING DATA

```
results <- model%>%
+   evaluate(x_test , y_test)
25000/25000 [==============================] - 1s 51us/step
> results
$loss
[1] 0.1219174

$acc
[1] 0.82936


> model%>%
+   predict(x_test[1:10,])
           [,1]
 [1,] 0.5976627
 [2,] 0.9040789
 [3,] 0.7784562
 [4,] 0.8444042
 [5,] 0.8492339
 [6,] 0.4906103
 [7,] 0.9205060
 [8,] 0.1456023
 [9,] 0.8339341
[10,] 0.8140877
```

CONCLUSION

Here the conclusion is very simple but trivial at certain areas. Simple because we can easily see which model architecture is best suited for adoption by calculating accuracy and loss. Furthermore it is equally trivial in a sense that loss and accuracy do not comprehend each other , where we are getting better accuracy the same model describes a loss at a greater rate. Such is the vice-versa scenario where the loss value is minimum, the accuracy is not so good in comparison to other obtained results. Hence we might say that there exists a trade-off between loss and accuracy.

Observing the results we can round off to some hypothesis of concern. Taking into account model (1) and (2) , they were used as sample model in order to see how our network functions. As the title of the project suggests deep learning we can thus conclude that the architecture for the neural network model must be build on 2 or more than 2 layer with the exception for model (6) where double layer is used but the number of hidden units are increased by factor 2 .i.e. 32 hidden units.
Taking into consideration model (5) – this is the worst one among all however it had an profound effect on loss value giving the highest of all – 7.9(approx) yet it is the rejected one due to its least value in terms of accuracy.
Though model (3) has the second highest value of the loss but its accuracy is lesser than model (4) and model (6).
Henceforth the efficient model , not to be mistaken as the best as there can be many other combination of activation function , optimizers etc with varying and good , is model (4).
Without comprising the accuracy , the loss is somewhat acceptable and good.

After the model is well trained and tested we can now predict the positive and negative reviews by scoring them between ‘0’ and ‘1’ – ‘1’ being likely to be positive and ‘0’ being likely to be negative. Thus the prediction value for first 10 instances of test data shows a value ranging between 0 and 1 which concludes that no review is completely positive or completely negative but contains both the elements simultaneously. Therefore 0.9 shows the review to be positive and 0. 1 shows the review to be negative.
A score of 0.5 to 0.7 might intuitively say that the movie is good but some elements aren’t that great which might have led to some negative criticism.
** Super Hero genre atleast one can say got good rating whereas Horror genre got rave reviews. Old classics like Citizen Kane and To Kill A Mockingbird got the highest score because of excellent plotline and great acting.

Here at the end I have described the code for study that has been used in getting the output...

CODE FOR REVIEW :

```
library(keras)
## choosing the first 10000 words for analysis ##
## 10000 means we will have top most frequently occurring words the training data ##
## train data and test data classification ##
imdb <- dataset_imdb(num_words = 10000)
train_data <- imdb$train$x
train_labels <- imdb$train$y
test_data <- imdb$test$x
test_labels <- imdb$test$y
## indices and labels ##
str(train_data[[1]])
train_labels[[1]]
## restricting to only top 10000 words ##
max(sapply(train_data , max))
## sequence vector and one hot encoding manually##
vectorize_sequences <- function(sequences , dimension = 10000){
 ## matrix formation of all zeros with number of rows equal to sequences and columns equal to
dimension ##
   results <- matrix(data = 0 , nrow = length(sequences) , ncol = dimension)

  for (i in 1:length(sequences))
    results[i , sequences[[i]]] <- 1
  results
}
x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)

str(x_train[1,])

y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)

str(y_train)

## validation data ##
val_indices <- 1:10000

x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]

y_val <- y_train[val_indices]
partial_y_train <- y_train[-val_indices]
```

```
      ## model building ##

      model <- keras_model_sequential() %>%
        layer_dense(units = 16 , activation = 'relu' , input_shapes = c(10000)) %>%
        layer_dense(units = 16 , activation = 'relu'))%>%
        layer_dense(units = 16 , activation = 'relu') %>%
        layer_dense(units = 1 , activation = 'sigmoid')

      model %>% compile (
        optimizer = 'rmsprop',
        loss = 'binary_crossentropy' ,
        metrics = c("accuracy")
      )

      history <- model %>%
        fit(
          partial_x_train,
          partial_y_train,
          epochs = 20,
          batch_size = 512,
          validation_data = list(x_val , y_val)
          )
      plot(history)
results <- model%>%

  evaluate(x_test , y_test)

results

model%>%

  predict(x_test[1:10,])
```

note :

In the above the values of activation function and optimizers are taken as for model (1) and during execution these values were subsequently changed for getting the different outputs which is however not shown.

The steps that were followed are as follows :

1) Reading the data and printing the elements in it.
2) Changing the data into matrix or more specifically document term matrix.
3) Normalization of data which is already present in the dataset itself.
4) Data partitioning where labels for train data and test data are separated.
5) One hot encoding to classify the data into classes for better understanding.
6) Sequential modelling where the layers are added as well as the activation function is decided for execution.
7) Compilation of the data and the model together for plotting the dynamic and static graph.
8) Now the final take will be fitting the model in training data.
9) Then after training the model, test data is evaluated and accuracy & loss is calculated.
10) At last the model is used for prediction data at various instances showing the probability of positive and negative reviews.