Stroke (Heart Attack) Prediction

Machine Learning Final Project

Kiran Ram Ganugula

1 Abstract:

Every year, heart disease, stroke, and other cardiovascular disorders claim more than 877,500 lives in the United States. The major causes of death in the US are heart disease and stroke, respectively, which rank first and fifth. This dataset is used to assess a patient's risk of suffering a stroke. To predict I used the following K-Nearest Neighbors Algorithm, Logistic Regression, Decision Tree, and Artificial Neural Network.

2 Problem Definition and Goals:

Based on input characteristics like gender, age, numerous diseases, and smoking status, this dataset is used to determine whether a patient is likely to get a stroke. The data rows each provide pertinent information about the patient. Our goal is to predict whether the patient is going to get a stroke or not by using the above-mentioned models.

This dataset contains 12 variables and 5000 observations, and the target variable is the **Stroke**, one of the biggest problems in the dataset is data is **unbalanced**, we need to balance the data before performing the models. In Dataset "id" are the patent id number, "Gender", and "age" of the patent, and "Hypertension" is whether the patient had it or not. The previous history of heart disease "Ever Married" marital status of the patient, work type, Glucose level of the patent, BMI Rate, Smoking status of the patent, and **Stroke** status.

The Variable is as follows:

we can download the dataset from the below link.

Stroke Prediction Dataset | Kaggle

1. Id
2. Gender
3. Age
4. Hypertension
5. Heart disease
6. Ever married
7. Work type
8. Residence type
9. Avg glucose level
10. Bmi
11. Smoking status
12. Stroke

3 Related Work:

We are doing the Stroke prediction on patients and it's the largest data in the Kaggle(Stroke Prediction Dataset | Kaggle), I found other data sets on heart failure and related to heart disease but all the other datasets with fewer observations and fewer variables compared to our dataset. We find those sets from this link Search | Kaggle

Also, I get to know more information about USA heart failure information https://www.ahajournals.org/doi/10.1161/CIR.0000000000001052 from this I understand more information about heart failure and other heart-related statistics, they mention the information clearly by using graphs and understandable way. So, it helps me understand the best way to represent the graphs and plots.

4 Data Exploration and Pre-Processing:

- I did the normalization and one hot encoding for ANN model to deal with my data. Because my data is imbalanced.

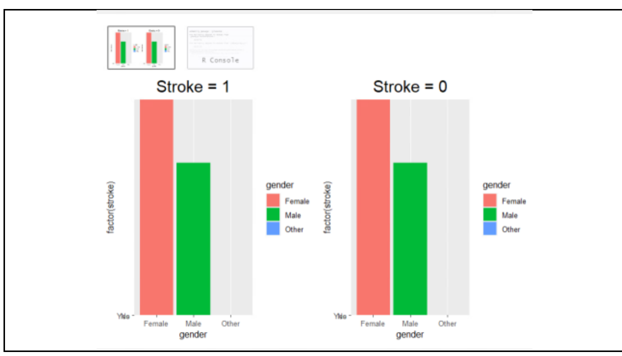
- My data was imbalanced data so when doing the models (KNN, Logistic Regression, ANN, etc.) I used the reference as Module 12 to deal with them like (Class weights etc...)

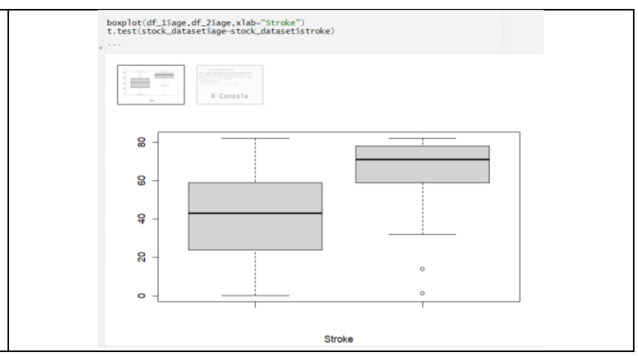
- There are 201 missing values in the variable “bmi” to deal with them I used the Mean and mode to deal with them, it will calculate the column mean and fill l the missing values

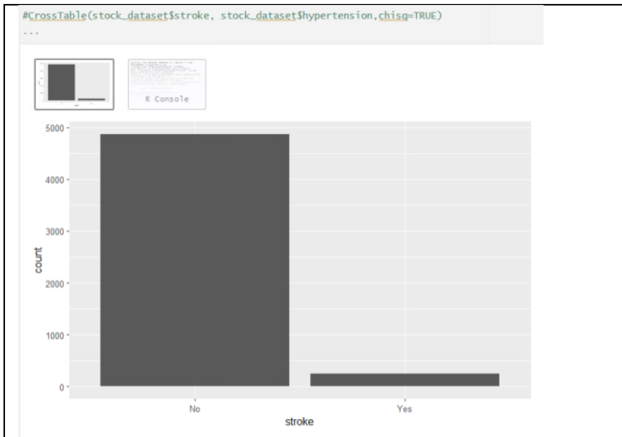
- Our data has 6 numeric and 6 categorical variables we converted the categorical data into factorial and the target variable “Stroke” integer into factor after that we replaced the 0,1 with “no” and “yes”

- We removed the first column called “ID” which is not important, which is patent ID.

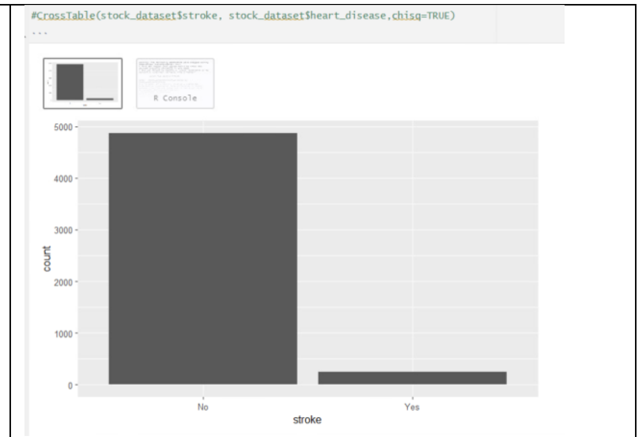
Data Exploration:



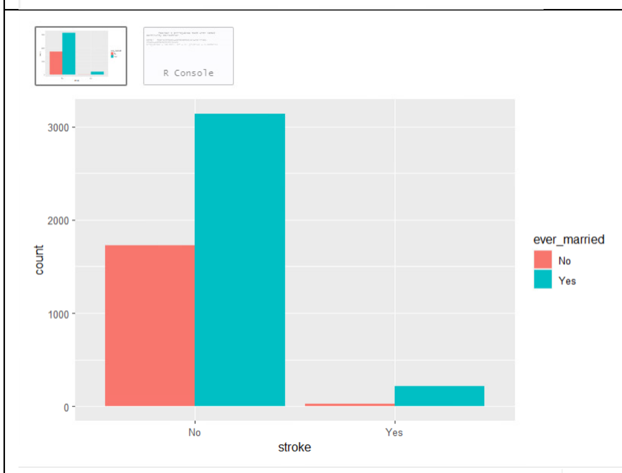




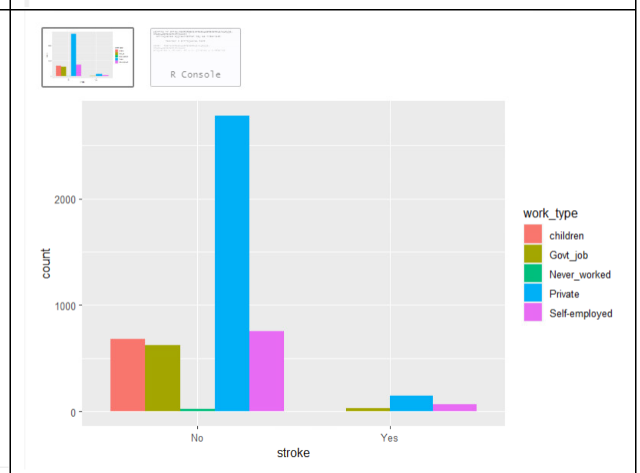
#CrossTable(stock_dataset$stroke, stock_dataset$hypertension,chisq=TRUE)
R Console
count
0
1000
2000
3000
4000
5000
No
Yes
stroke



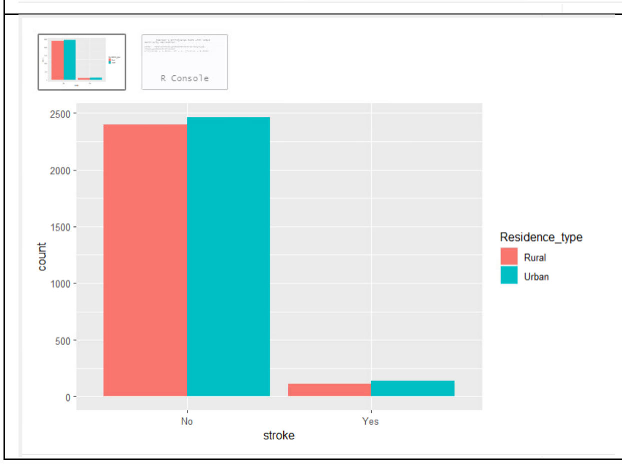
#CrossTable(stock_dataset$stroke, stock_dataset$heart_disease,chisq=TRUE)
R Console
count
0
1000
2000
3000
4000
5000
No
Yes
stroke



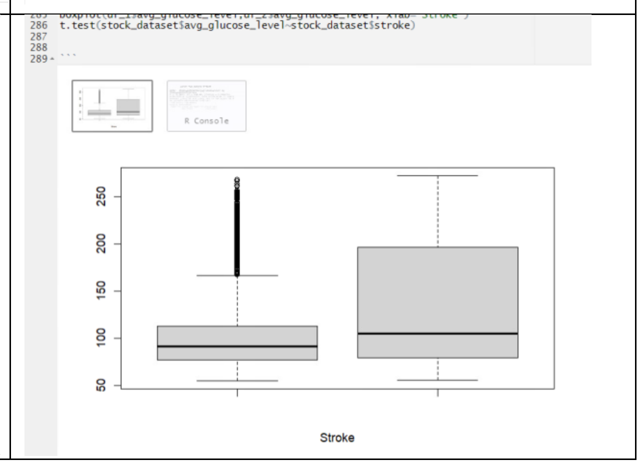
R Console
count
0
1000
2000
3000
No
Yes
stroke
ever_married
No
Yes



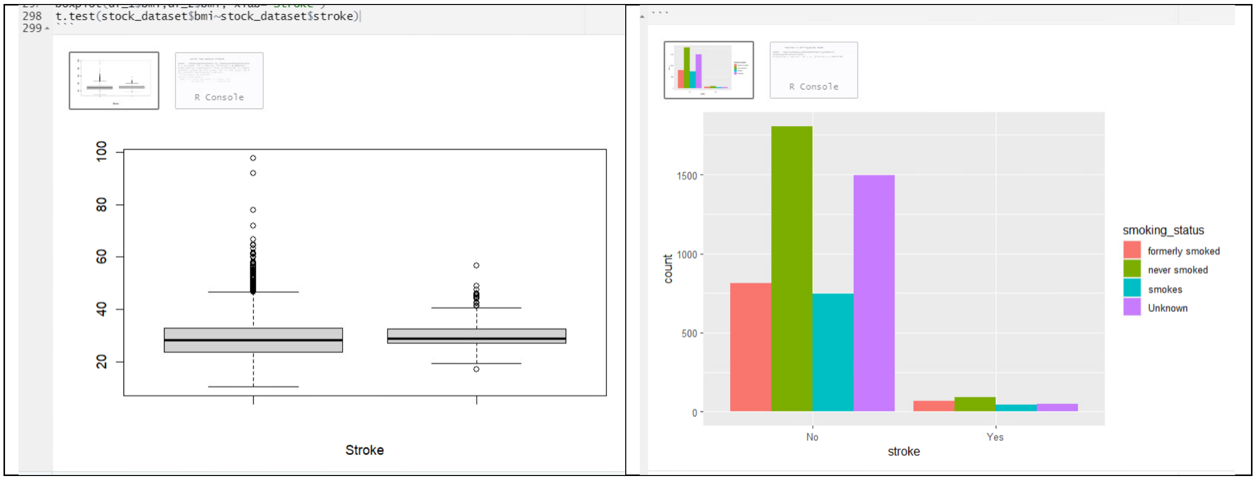
R Console
count
0
1000
2000
No
Yes
stroke
work_type
children
Govt_job
Never_worked
Private
Self-employed



R Console
count
0
500
1000
1500
2000
2500
No
Yes
stroke
Residence_type
Rural
Urban



286 t.test(stock_dataset$avg_glucose_level~stock_dataset$stroke)
287
288
289
R Console
50
100
150
200
250
Stroke



- **For Variable Gender**
- We have a chi-squared value of 0.47259. Since the p-Value 0.7895 is not less than the significance level of 0.05, we cannot reject the null hypothesis and conclude that the two variables are in fact dependent.

- **For Variable Age**
- Here the output of t-test shows that the p-value is less than the significance level and we can reject null hypothesis and accept alternate hpothesis which says that difference in age means for two groups with (no stroke)0 and (stroke)1 is not equal to 0.

- **For Variable Hypertension**
- T_test we obtained a p-value of 1.978e-09, lower than the threshold of 0.05. You can reject the null (H0) hypothesis.true difference in means between group 0 and group 1 is not equal to 0

- **For Variable Heart Disease**
- t-test we obtained a p-value of 4.095e-08, lower than the threshold of 0.05. You can reject the null (H0) hypothesis. true difference in means between group 0 and group 1 is not equal to 0

- **For Variable Ever Married**

- We have a chi-squared value of 58.924. Since we get the p-Value is 1.639e-14 less than the significance level of 0.05, we can reject the null hypothesis and conclude that the two variables are in fact dependent.

- **For Variable Work Type**
- We have a chi-squared value of 49.164. Since we get the p-Value 5.398e-10 is less than the significance level of 0.05, we can reject the null hypothesis and conclude that the two variables are in fact dependent.

- **For Variable Residence Type**
- We have a chi-squared value of 1.0816. Since the p-Value 0.2983 is not less than the significance level of 0.05, we cannot reject the null hypothesis and conclude that the two variables are in fact dependent.

- **For Variable Avg Glucose Level**
- t-test we obtained a p-value of 2.401e-11, lower than the threshold of 0.05. You can reject the null (H0) hypothesis. true difference in means between group 0 and group 1 is not equal to 0

- **For Variable BMI**
- t-test we obtained a p-value of 0.0003591, lower than the threshold of 0.05. You can reject the null (H0) hypothesis. true difference in means between group 0 and group 1 is not equal to 0

- **For Variable Smoking Status**
- We have a chi-squared value of 49.164. Since we get the p-Value 2.085e-06 is less than the significance level of 0.05, we can reject the null hypothesis and conclude that the two variables are in fact dependent.

5 Data analysis and experimental Results:

For this Dataset, I'm using the 4 models, for doing these models with my Imbalanced data I referred the module 12&13, in those modules we clearly got an overview of how to deal with Data Imbalance.

- Gradient Boosting Machine (Decision Tree)
- K-Nearest Neighbors Algorithm (KNN)
- Logistic Regression (LASSO, Ridge, Elastic Net)
- Artificial Neural Network (ANN)

5.1. Gradient Boosting Machine (Decision Tree):

For doing decision trees I use ensemble trees like GBM, I do these models by using the **class weights that** were thought in module 12, This method of controlling data imbalance is cost-sensitive and involves giving instances belonging to the minority vs. the majority class distinct weights. then we retrain the model with class weights and later we compute Area under the ROC Curve.

````
#Decision Tree (Gradient Boosting Machine )

```{R}

t=table(stroke_train$stroke)
#t=table(stroke_train$stroke)
w_yes=sum(t)/(t["Yes"]*2)
w_no=sum(t)/(t["No"]*2)
w_yes
w_no

```

```{R}
class_weights= ifelse(stroke_train$stroke=="Yes", w_yes, w_no)
str(class_weights)
```

```{R}
ctrl=trainControl(method="cv", number=10, classProbs = TRUE, summaryFunction =twoClassSummary)
m_weighted_gbm<-train(stroke ~ ., data = stroke_train, method = "gbm",weights=class_weights,verbose=FALSE, metric ="ROC", trControl= ctrl)

```

```{R}
gbm_predictions_prob=predict(m_weighted_gbm, stroke_test, type="prob")
gbm_predictions= prediction(gbm_predictions_prob$Yes,stroke_test$stroke)
performance(gbm_predictions, measure = "auc")@y.values


```
````

Now we predict the confusion matrix for the Gradient Boosting Machine (Decision Tree), with that can examine the model performance.

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  689   9
       Yes 283  40

               Accuracy : 0.714
                 95% CI : (0.6852, 0.7416)
    No Information Rate : 0.952
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1437

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.81633
            Specificity : 0.70885
         Pos Pred Value : 0.12384
         Neg Pred Value : 0.98711
              Precision : 0.12384
                 Recall : 0.81633
                     F1 : 0.21505
             Prevalence : 0.04799
         Detection Rate : 0.03918
   Detection Prevalence : 0.31636
      Balanced Accuracy : 0.76259

       'Positive' Class : Yes
```

5.2. K-Nearest Neighbors Machine (KNN):

The amount of data we are producing has increased the requirement for sophisticated machine learning algorithms. The K Nearest Neighbor algorithm is one such technique. Again, for this model also I'm using I used module 12 to predict the imbalanced data.

````
#knn


```{R}

ctrl=trainControl(method="cv", number=10, classProbs = TRUE, summaryFunction = twoClassSummary,
sampling="smote")
knn_balanced_smote<-train(stroke~., data = stroke_train, method = "knn",  metric ="ROC", trControl= ctrl)


```

```{R}
 library(ROCR)
knn_predictions_prob=predict(knn_balanced_smote, stroke_test, type="prob")
knn_predictions= prediction(knn_predictions_prob$Yes,stroke_test$stroke)
performance(knn_predictions, measure = "auc")@y.values

```

 [[1]]
 [1] 0.7878979
````

Now we predict the confusion matrix for the K-Nearest Neighbors Algorithm (KNN) that can examine the model performance.

```{R}
knn_predicted_labels = predict(knn_balanced_smote, stroke_test)
confusionMatrix(knn_predicted_labels, stroke_test$stroke, positive="Yes", mode="everything")
```

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  758  15
       Yes 214  34

               Accuracy : 0.7757
                 95% CI : (0.7489, 0.801)
    No Information Rate : 0.952
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1618

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.69388
            Specificity : 0.77984
         Pos Pred Value : 0.13710
         Neg Pred Value : 0.98060
              Precision : 0.13710
                 Recall : 0.69388
                     F1 : 0.22896
             Prevalence : 0.04799
         Detection Rate : 0.03330
   Detection Prevalence : 0.24290
      Balanced Accuracy : 0.73686

       'Positive' Class : Yes
```

5.3. Logistic Regression (LASSO, Ridge, Elastic Net):

Lasso, Ridge, and Elastic Net can be used to regularize a logistic regression model by penalizing big coefficients, much like linear regression. Caret's train function allows us to perform regularized logistic regression by setting method="glmnet"

In this logistic regression, we have 3 types of models to predict.

- LASSO
- Ridge
- Elastic Net

5.3.1. LASSO Regression:

In this, we put the tune lambda (we use a grid with 100 values between -3,3) and set alpha to 1 to perform lasso.

```
#Logestic Regression

#LASSO

```{R}

set.seed(1)

library(caret)

lasso_regression_f = train(stroke ~., data = stroke_train, method = "glmnet", preProc=c("knnImpute", "nzv"),metric ="ROC", na.action = na.pass,trControl = trainControl("cv", number = 10,classProbs = TRUE,sampling="smote"),tuneGrid = expand.grid(alpha = 1, lambda = 10^seq(-3, 3, length =100)))


```


```{R}
 library(ROCR)
lasso_predictions_prob=predict(lasso_regression_f, stroke_test, type="prob")
lasso_predictions= prediction(lasso_predictions_prob$Yes,stroke_test$stroke)
performance(lasso_predictions, measure = "auc")@y.values

```

 [[1]]
 [1] 0.5
```

After implementing the LASSO now, we do the prediction by using the prediction function, with that we can do Confusion Matrix.

With that data we can find the accuracy of model.

```
```{R}

#head(knn_predicted_labels)
#head(stroke_test$stroke)
lasso_predicted_labels = predict(lasso_regression_f, stroke_test)
confusionMatrix(lasso_predicted_labels, stroke_test$stroke, positive="Yes", mode="everything")

```

Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No 972 49
       Yes   0   0

               Accuracy : 0.952
                 95% CI : (0.937, 0.9643)
    No Information Rate : 0.952
    P-Value [Acc > NIR] : 0.5379

                  Kappa : 0

 Mcnemar's Test P-Value : 7.025e-12

            Sensitivity : 0.00000
            Specificity : 1.00000
         Pos Pred Value :   NaN
         Neg Pred Value : 0.95201
              Precision :   NA
                 Recall : 0.00000
                     F1 :   NA
             Prevalence : 0.04799
         Detection Rate : 0.00000
   Detection Prevalence : 0.00000
      Balanced Accuracy : 0.50000

       'Positive' Class : Yes
```

5.3.2. Ridge Regression:

As in the Ridge linear regression model, to do Ridge regularization, we set alpha=0 and we put the tune lambda (we use a grid with 100 values between -3,3)

```
#Ridge

```{R}

set.seed(1)

ridge_regression_f = train(stroke ~., data = stroke_train, method = "glmnet", preProc= c("knnImpute", "nzv"),metric ="ROC",
na.action=na.pass,trControl = trainControl("cv", number = 10,classProbs = TRUE,sampling="smote"),tuneGrid = expand.grid(alpha = 0, lambda = 10^seq(-3,
3, length =100)))

```

```{R}

 library(ROCR)
ridge_predictions_prob=predict(ridge_regression_f, stroke_test, type="prob")
ridge_predictions= prediction(ridge_predictions_prob$Yes,stroke_test$stroke)
performance(ridge_predictions, measure = "auc")@y.values

```

[[1]]
[1] 0.5
```

Same as LASSO, after implementing the ridge now, we do the prediction by using the prediction function, with that we can do Confusion Matrix.

With that data we can find the accuracy of the model.

```
```{R}
ridge_predicted_labels = predict(ridge_regression_f, stroke_test)
confusionMatrix(ridge_predicted_labels, stroke_test$stroke, positive="Yes", mode="everything")

```

Confusion Matrix and Statistics

          Reference
Prediction No Yes
       No 972 49
       Yes   0   0

               Accuracy : 0.952
                 95% CI : (0.937, 0.9643)
    No Information Rate : 0.952
    P-Value [Acc > NIR] : 0.5379

                  Kappa : 0

 Mcnemar's Test P-Value : 7.025e-12

            Sensitivity : 0.00000
            Specificity : 1.00000
         Pos Pred Value :   NaN
         Neg Pred Value : 0.95201
              Precision :   NA
                 Recall : 0.00000
                     F1 :   NA
             Prevalence : 0.04799
         Detection Rate : 0.00000
   Detection Prevalence : 0.00000
      Balanced Accuracy : 0.50000

       'Positive' Class : Yes
```

5.3.3. Elastic Net:

As in the Ridge linear regression model, to do Ridge regularization, we set alpha=0 .

````
#Elasic Net


```{R}
set.seed(1)

enet_f <- train(stroke ~., data = stroke_train, method = "glmnet", preProc= c("knnImpute", "nzv"),metric ="ROC", na.action=na.pass,trControl =
trainControl("cv", number = 10,classProbs = TRUE,sampling="smote"),tuneGrid = expand.grid(alpha =seq(0,1, length=10), lambda = 10^seq(-3, 3, length =
100)))


```

```{R}


library(ROCR)
enet_predictions_prob=predict(enet_f, stroke_test, type="prob")
enet_predictions= prediction(enet_predictions_prob$Yes,stroke_test$stroke)
performance(enet_predictions, measure = "auc")@y.values


```
````

```
[[1]]
[1] 0.5
```

Same as LASSO, after implementing the Elastic Net now, we do the prediction by using the prediction function, with that we can do Confusion Matrix.

With that data we can find the accuracy of the model.

````
```{R}

enet_predicted_labels = predict(enet_f, stroke_test)
confusionMatrix(enet_predicted_labels, stroke_test$stroke, positive="Yes", mode="everything")


```
````

```
Confusion Matrix and Statistics

          Reference
Prediction  No Yes
       No  972  49
       Yes   0   0

               Accuracy : 0.952
                 95% CI : (0.937, 0.9643)
    No Information Rate : 0.952
    P-Value [Acc > NIR] : 0.5379

                  Kappa : 0

 Mcnemar's Test P-Value : 7.025e-12

            Sensitivity : 0.00000
            Specificity : 1.00000
         Pos Pred Value :     NaN
         Neg Pred Value : 0.95201
              Precision :      NA
                 Recall : 0.00000
                     F1 :      NA
             Prevalence : 0.04799
         Detection Rate : 0.00000
   Detection Prevalence : 0.00000
      Balanced Accuracy : 0.50000

       'Positive' Class : Yes
```

5.3.4. Now we compare the three logistic models LASSO, Ridge, Elastic Net

```
# Compare Lasso,Ridge,Elastic net

```{R}

compare=resamples(list(L=lasso_regression_f,R=ridge_regression_f,E=enet_f))
compare
summary(compare)

```
```

By using the resamples function we can compare all the three logistic regression models.

```
Call:
resamples.default(x = list(L = lasso_regression_f, R = ridge_regression_f, E = enet_f))

Models: L, R, E
Number of resamples: 10
Performance metrics: Accuracy, Kappa
Time estimates for: everything, final model fit

Call:
summary.resamples(object = compare)

Models: L, R, E
Number of resamples: 10

Accuracy
        Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
L 0.04901961 0.9511002 0.9511002 0.8608922 0.9511002 0.9511002    0
R 0.04901961 0.9511002 0.9511002 0.8608922 0.9511002 0.9511002    0
E 0.04901961 0.9511002 0.9511002 0.8608922 0.9511002 0.9511002    0
```

In the above after we are comparing, we got that Min, Median, Mean, Max of all the Lasso, Ridge, and Elastic Net.

5.4 Artificial Neural Network (ANN):

5.4.1. Artificial Neural Network (ANN) without Class Weights:

To predict strokes, neural network models with and without class weights will be developed. We need to normalize the numerical variables in the stroke dataset before preparing it for neural networks.

Our Data is imbalanced data so for that we are doing it by reference to Module 12 about dealing with imbalanced data to create the ANN Model.

```
```{R}
numeric_cols= c("age","hypertension","heart_disease","avg_glucose_level","bmi")
col_means_train<-attr(scale(stroke_test [,numeric_cols]), "scaled:center")
col_stddevs_train<-attr( scale(stroke_train[,numeric_cols]), "scaled:scale")
stroke_train[numeric_cols]= scale(stroke_train[numeric_cols])
stroke_test[numeric_cols]<-scale(stroke_test[numeric_cols], center = col_means_train, scale =col_stddevs_train)



```
```

After we normalize, the data now do need to convert the target variable “Stroke” to 0-1 indices

```{R}
stroke_train$stroke = as.numeric(stroke_train$stroke)-1
stroke_test$stroke = as.numeric(stroke_test$stroke)-1


```

And now we need we need to one-hot encode the factor variables

```{R}
library(mltools)
library(data.table)
stroke_train=as.matrix(one_hot(as.data.table(stroke_train)))
stroke_test=as.matrix(one_hot(as.data.table(stroke_test)))


```

Now let's separate features(x) and labels (y). The labels are the "Stroke" variable, the 11th column in Stroke Train and Stroke Test.

```{R}
stroke_train_x= stroke_train[,-11]
stroke_test_x= stroke_test[,-11]
stroke_train_y= stroke_train[,11]
stroke_test_y=stroke_test[,11]
```

Create a neural network model with two hidden layers and configure it for binary classification.

```{R}

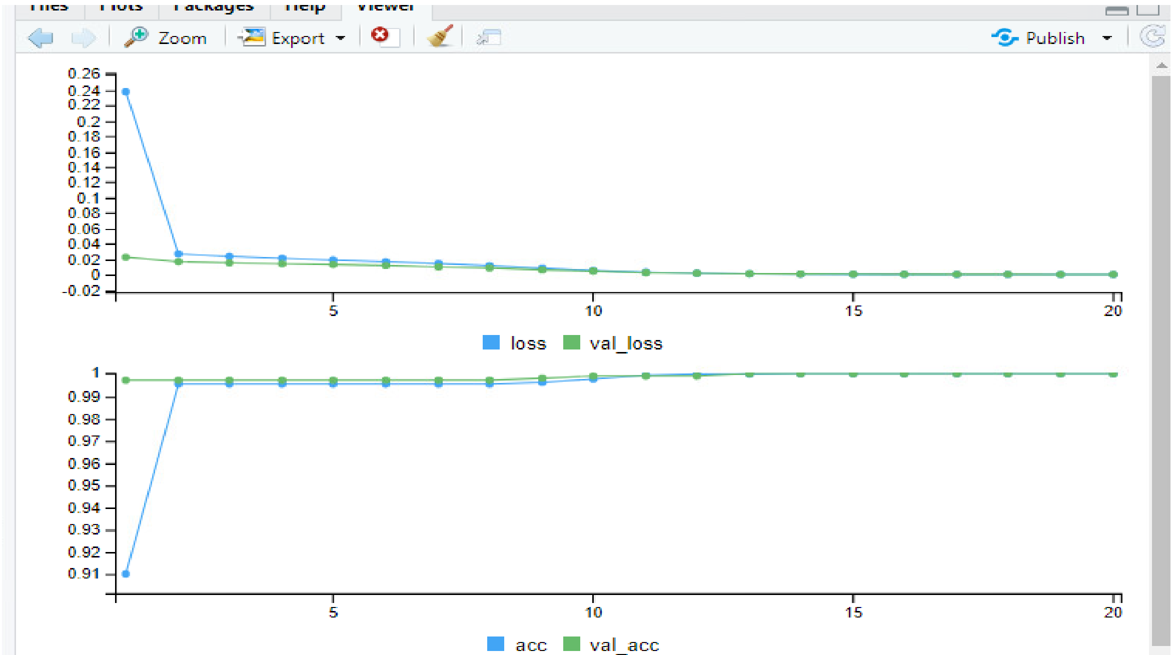
model <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
  input_shape= dim(stroke_train_x)[2]) %>%
  layer_dense(units = 64,activation="relu" )%>%
  layer_dense(units = 1, activation="sigmoid")
  model %>% compile(
  loss = "binary_crossentropy",
  optimizer = "adam" , metrics="acc")

history <-model %>% fit(stroke_train_x, stroke_train_y, batch_size=50, epochs = 20,verbose=2, validation_data=list(stroke_test_x,stroke_test_y))


```

Graphs:

We got the graphs as follow below; we got the most accurate way of neural network graphs because. We have imbalanced data,




```
Epoch 12/20
82/82 - 0s - loss: 0.0018 - acc: 0.9998 - val_loss: 0.0018 - val_acc: 0.9990 - 358ms/epoch - 4ms/step
Epoch 13/20
82/82 - 0s - loss: 9.9056e-04 - acc: 0.9998 - val_loss: 0.0011 - val_acc: 1.0000 - 410ms/epoch - 5ms/step
Epoch 14/20
82/82 - 0s - loss: 6.0775e-04 - acc: 1.0000 - val_loss: 7.4488e-04 - val_acc: 1.0000 - 370ms/epoch - 5ms/step
Epoch 15/20
82/82 - 0s - loss: 4.4182e-04 - acc: 1.0000 - val_loss: 5.4460e-04 - val_acc: 1.0000 - 392ms/epoch - 5ms/step
Epoch 16/20
82/82 - 0s - loss: 3.1000e-04 - acc: 1.0000 - val_loss: 4.3233e-04 - val_acc: 1.0000 - 466ms/epoch - 6ms/step
Epoch 17/20
82/82 - 0s - loss: 2.3653e-04 - acc: 1.0000 - val_loss: 3.3878e-04 - val_acc: 1.0000 - 376ms/epoch - 5ms/step
Epoch 18/20
82/82 - 0s - loss: 1.8691e-04 - acc: 1.0000 - val_loss: 2.8137e-04 - val_acc: 1.0000 - 391ms/epoch - 5ms/step
Epoch 19/20
82/82 - 0s - loss: 1.4992e-04 - acc: 1.0000 - val_loss: 2.3987e-04 - val_acc: 1.0000 - 338ms/epoch - 4ms/step
Epoch 20/20
82/82 - 0s - loss: 1.2326e-04 - acc: 1.0000 - val_loss: 1.9686e-04 - val_acc: 1.0000 - 352ms/epoch - 4ms/step
```

Now we do the ANN Model Predictions:

```{R}
predicted_probs= model%>% predict(stroke_test_x)
ann_predictions= prediction(predicted_probs, stroke_test_y)
performance(ann_predictions, measure = "auc")@y.values

```

```
32/32 [==============================] - 0s 1ms/step
[[1]]
[1] 1
```

Now we do the confusion Matrix to the Predicts:

```
```{R}
predicted_labels = factor(ifelse(predicted_probs>0.5, "1", "0"))
confusionMatrix(predicted_labels, as.factor(stroke_test_y), mode="everything", positive="1")

```
```

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1018    0
         1    0    3

               Accuracy : 1
                 95% CI : (0.9964, 1)
    No Information Rate : 0.9971
    P-Value [Acc > NIR] : 0.04957

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000000
            Specificity : 1.000000
         Pos Pred Value : 1.000000
         Neg Pred Value : 1.000000
              Precision : 1.000000
                 Recall : 1.000000
                     F1 : 1.000000
             Prevalence : 0.002938
         Detection Rate : 0.002938
   Detection Prevalence : 0.002938
      Balanced Accuracy : 1.000000

       'Positive' Class : 1
```

5.4.2. Artificial Neural Network (ANN) without Class Weights:

Here we are doing with the class weights, we did the class weights when we did that in the Decision Tree (Gradient Boosting Machine) model, and we are going to use the same now here.

```
```{R}

t=table(stroke_train$stroke)
#t=table(stroke_train$stroke)
w_yes=sum(t)/(t["Yes"]*2)
w_no=sum(t)/(t["No"]*2)
w_yes
w_no

```
```

```
    Yes
10.2225
       No
0.5257136
```

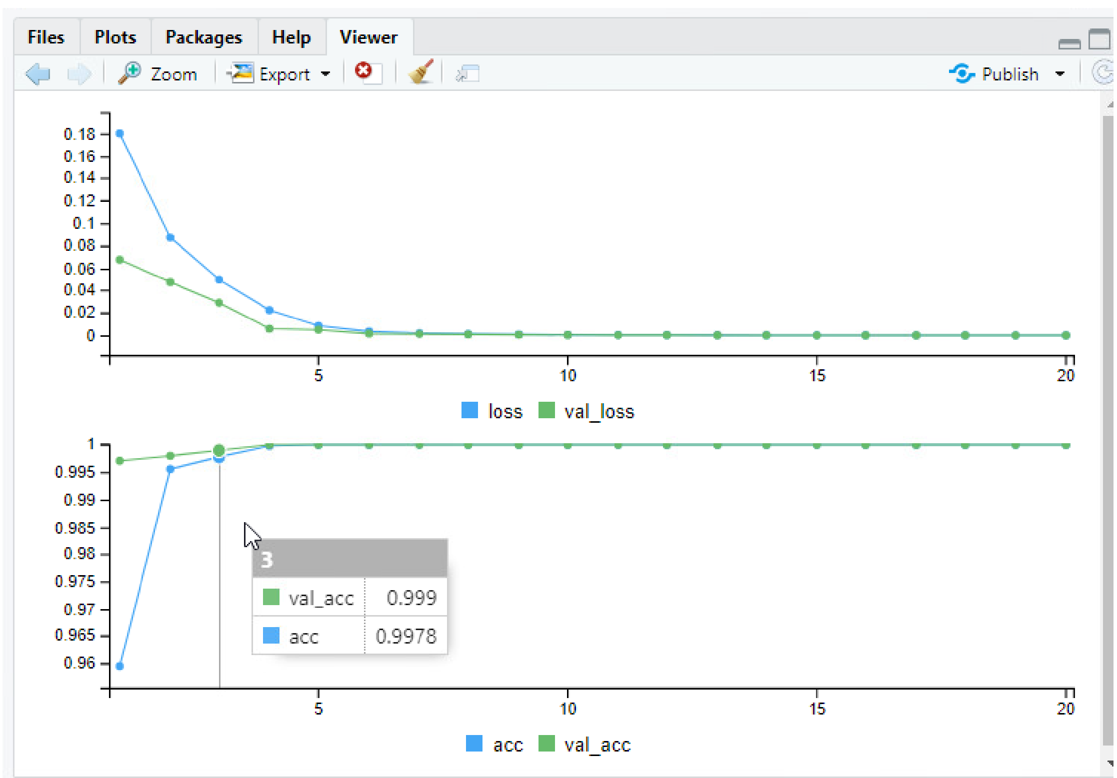
To use class_weights in keras, we can use the parameter “class_weight” in “fit” function and set it to a list of weights for each class.

````
```{r}
model <-keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
  input_shape= dim(stroke_train_x)[2]) %>%
  layer_dense(units = 64,activation="relu" )%>%
  layer_dense(units = 1, activation="sigmoid")
  model %>% compile(
  loss = "binary_crossentropy",
  optimizer = "adam" , metrics="acc")

history <-model %>% fit(stroke_train_x, stroke_train_y,batch_size=50, epochs = 20,verbose=2,
validation_data=list(stroke_test_x,stroke_test_y),class_weight=list("0"=w_no, "1"=w_yes))

```
````

After that, we got the graph below



When comparing with the ANN without class weights graph, the ANN with graph weights got more accurate.

Tuning R File code

```r
FLAGS1 <- flags(
  flag_numeric("nodes1", 128),
  flag_numeric("batch_size", 100),
  flag_string("activation1", "relu"),
  flag_string("activation2", "relu"),
  flag_numeric("learning_rate", 0.001),
  flag_numeric("epochs", 30),
  flag_numeric("nodes2", 250)
)


model =keras_model_sequential()

model %>%
  layer_dense(units = FLAGS1$nodes1, activation =FLAGS1$activation1, input_shape = dim(stroke_train_x)[2]) %>%
  layer_dropout(0.5) %>%
  layer_dense(units = FLAGS1$nodes2, activation = FLAGS1$activation2) %>%
  layer_dropout(0.5) %>%
  layer_dense(units = 1, activation = 'sigmoid')

model %>% compile(
  optimizer = optimizer_adam(lr=FLAGS1$learning_rate),
  loss = 'binary_crossentropy',
  metrics = c('accuracy'))

model %>% fit(as.matrix(stroke_train_x)    , as.numeric(stroke_train_x_lab) , epochs = FLAGS1$epochs, batch_size= FLAGS1$ba
  validation_data=list(as.matrix(stroke_validation_x) , as.numeric(stroke_validation_x_lab)))
```

```r
```{R}


runs <- tuning_run("C:/Users/csc/Desktop/New folder/ML_Final_Tune.R",
                   flags = list(
                     nodes1 = c(64, 128),
                     nodes2 = c(128,392),
                     learning_rate = c(0.01, 0.05, 0.001, 0.0001),
                     batch_size=c(100,200,500,1000),
                     epochs=c(30,50,100),
                     activation1=c("relu","sigmoid","tanh"),
                     activation2 =c("sigmoid", "tanh", "relu")

                   ),
                   sample = 0.02
)


```
```

Obtain the model's predictions on the test data and compute the weighted model's area under the ROC curve.

````
```{R}
predicted_probs= model%>% predict(stroke_test_x)
ann_predictions= prediction(predicted_probs,stroke_test_y)
performance(ann_predictions, measure = "auc")@y.values

```
````

```
32/32 [==============================] - 0s 1ms/step
[[1]]
[1] 1
```

Compute Confusion Matrix:

````
```{R}
predicted_labels = factor(ifelse(predicted_probs>0.5, "1", "0"))
confusionMatrix(predicted_labels, as.factor(stroke_test_y), mode="everything", positive="1")
```
````

```
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1018    0
         1    0    3

               Accuracy : 1
                 95% CI : (0.9964, 1)
    No Information Rate : 0.9971
    P-Value [Acc > NIR] : 0.04957

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.000000
            Specificity : 1.000000
         Pos Pred Value : 1.000000
         Neg Pred Value : 1.000000
              Precision : 1.000000
                 Recall : 1.000000
                     F1 : 1.000000
             Prevalence : 0.002938
         Detection Rate : 0.002938
   Detection Prevalence : 0.002938
      Balanced Accuracy : 1.000000

       'Positive' Class : 1
```

When comparing with the ANN without class weights graph, the ANN with graph weights got more AUC Value

Conclusion:

Predicting the Stroke with this Imbalance data is more difficult than compared to balanced data, we need to do a lot of work on the data to make it balanced. This project needed a thorough review of the exploratory data and numerous decisions regarding the shrinking of high cardinality predictors and missing values.

In this top 2 most AUC is both the “ANN” and “ANN with class weights” the value of Most AUC is ANN with class weights. (0.9991). we got the least AUC in the Decision Tree which is 0.714. because of the imbalance we got most of the accurate values.

Comparing All the models

While comparing all the models i got the most accuracy "ANN With Class Weights" which is 0.9971

the accuracy of GBM Decision tree is 0.714

The Accuracy of KNN is 0.7757

The Accuracy of LASSO is 0.952

The Accuracy of Ridge is 0.932

The Accuracy of Elastic Net is 0.923

The Accracy of ANN is 0.996