MODEL 2

THEDION V. DIAM JR.

2022-12-13

Load Data Sets

The data contains 197 rows and 431 columns with *Failure.binary* binary output.

```
rawd <- read.csv("C:/Users/redee/OneDrive/Desktop/STAT 325 Final Project/FP DATA.csv")
```

#== Reprocessing the Raw Data ==#

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
```

```
## Warning: package 'ggplot2' was built under R version 4.2.2
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(bestNormalize)
```

```
## Warning: package 'bestNormalize' was built under R version 4.2.2
```

Check for null and missing values

Using *anyNA()* function, We can determine if any missing values in our data.

```
anyNA(rawd)
```

```
## [1] FALSE
```

```
#The result shows either *True* or *False*. If True, omit the missing values using *na.omit()*

#[1] FALSE

#Thus, our data has no missing values.
```

Check for Normality of the Data

We used *Shapiro-Wilk's Test* to check the normality of the data.

```
rd <- rawd%>%select_if(is.numeric)
rd <- rd[,-1]
test <- apply(rd,2,function(x){shapiro.test(x)})
```

To have the list of p-value of all variables, the *unlist()* function is used and convert a list to vector.

```
pvalue_list <- unlist(lapply(test, function(x) x$p.value))
```

```
sum(pvalue_list<0.05)  # not normally distributed
```

```
## [1] 428
```

```
sum(pvalue_list>0.05)  # normally distributed
```

```
## [1] 1
```

```
test$Entropy_cooc.W.ADC
```

```
##
##  Shapiro-Wilk normality test
##
## data:  x
## W = 0.98903, p-value = 0.135
```

```
# [1] 428
# [1] 1

#  Thus, we have 428 variables that are not normally distributed and Entropy_cooc.W.ADC is normally dis
```

We use *orderNorm()* function, the *x.t* is the elements of orderNorm() function transformed original data.Using the *Shapiro-Wilk's Test*

```
TRDrawd=rawd[,c(3,5:length(names(rawd)))]

TRDrawd=apply(TRDrawd,2,orderNorm)
TRDrawd=lapply(TRDrawd, function(x) x$x.t)
TRDrawd=TRDrawd%>%as.data.frame()
test=apply(TRDrawd,2,shapiro.test)
test=unlist(lapply(test, function(x) x$p.value))
```

#Testing Data

```
sum(test <0.05)  # not normally distributed
```

```
## [1] 0
```

```
sum(test >0.05)  # normally distributed
```

```
## [1] 428
```

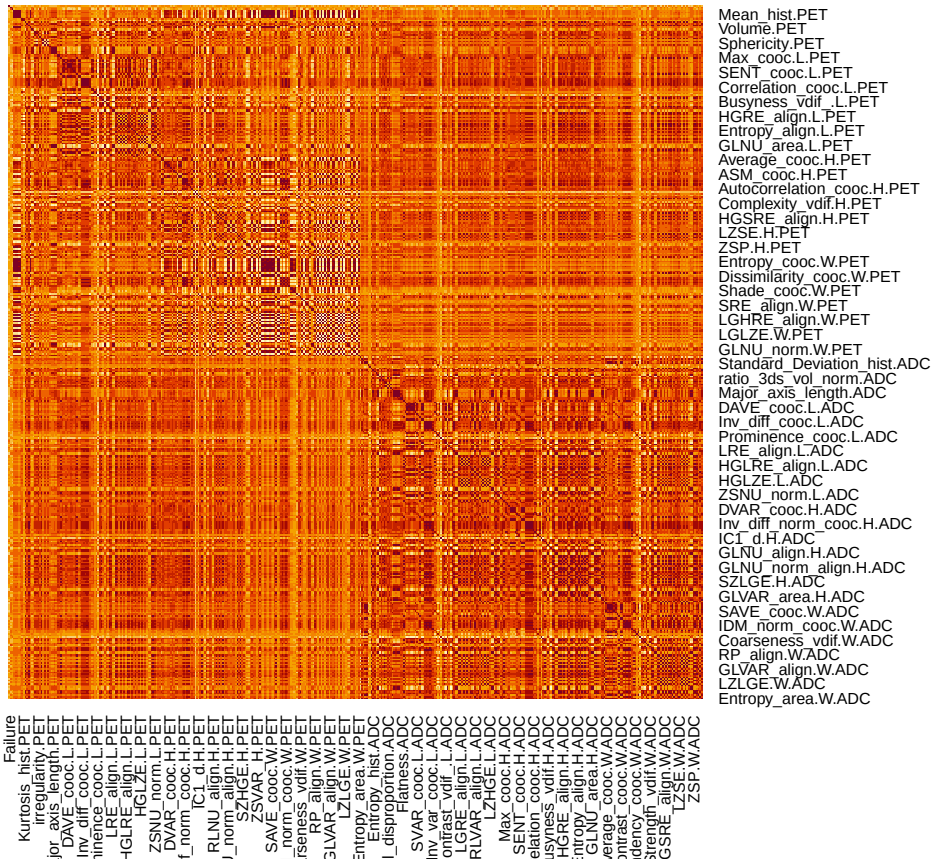
```
#[1] 0
#[1] 428

# Thus, our data is normally distributed.
```

```
rawd[,c(3,5:length(names(rawd)))]=TRDrawd
```

Get the correlation of the whole data expect the categorical variables

```
CorMatrix=cor(rawd[,-c(1,2)])
heatmap(CorMatrix,Rowv=NA,Colv=NA,scale="none",revC = T)
```



#Splitting the Data Split the data into training (80%) and testing (20%).

```
rawd$Institution=as.factor(rawd$Institution)
rawd$Failure.binary=as.factor(rawd$Failure.binary)
```

```
splitter <- sample(1:nrow(rawd), round(nrow(rawd) * 0.8))
trainND <- rawd[splitter, ]
testND  <- rawd[-splitter, ]
```

The data frame output of data reprocessing will be converted into to "csv", which will be used for entire project.

Load new Data

```
Final<- read.csv("C:/Users/redee/OneDrive/Desktop/STAT 325 Final Project/newdat.csv")
View(Final)
```

#========== NETWORK-BASED CLASSIFICATION MODEL ==========#

Helper Packages and Model Packages

```
library(dplyr)
library(keras)
```

```
## Warning: package 'keras' was built under R version 4.2.2
```

```
library(tfruns)
```

```
## Warning: package 'tfruns' was built under R version 4.2.2
```

```
library(rsample)
library(tfestimators)
```

```
## Warning: package 'tfestimators' was built under R version 4.2.2
```

```
## tfestimators is not recomended for new code. It is only compatible with Tensorflow version 1, and is
```

```
library(readr)
library(tensorflow)
```

```
## Warning: package 'tensorflow' was built under R version 4.2.2
```

Data Splitting

Split the data into training (80%) and testing (20%).

```
Final<-Final %>%
  mutate(Failure.binary=ifelse(Failure.binary== "No",0,1))

set.seed(123)
split = initial_split(Final,prop = 0.8 ,strata = "Failure.binary")
split_train <- training(split)
# [1] 157 rows and 431 columns

split_test  <- testing(split)
# [1] 40 rows and 431 columns

xtrain <- split_train[,-c(1,2)]%>%as.matrix.data.frame()
xtest <- split_test[,-c(1,2)]%>%as.matrix.data.frame()
ytrain <- split_train$Failure.binary
ytest <- split_test$Failure.binary
```

Reshaping the dataset

```
xtrain <- array_reshape(xtrain, c(nrow(xtrain), ncol(xtrain)))
xtrain <- xtrain

xtest <- array_reshape(xtest, c(nrow(xtest), ncol(xtest)))
xtest <- xtest

ytrain <- to_categorical(ytrain, num_classes = 2)
```

```
## Loaded Tensorflow version 2.9.3
```

```
ytest <- to_categorical(ytest, num_classes = 2)
```

Run the model

keras_mod_sequential() of keras package, allows us to create our network with a layering approach. We will create five hidden layers with *256, 128, 128, 64, 64* neurons respectively, With activation functions *Sigmoid*. Also *2* neurons for output layer with activation functions of *Softmax*.

```
mod <- keras_model_sequential() %>%

# Network architecture
  layer_dense(units = 256, activation = "sigmoid", input_shape = c(ncol(xtrain))) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 128, activation = "sigmoid") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 128, activation = "sigmoid") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 64, activation = "sigmoid") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 64, activation = "sigmoid") %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = 2, activation = "softmax")

    # Every layer is followed by a dropout to avoid overfitting.
```

Backpropagation Compiler Approach

```
mod %>%

compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(),
    metrics = c("accuracy")
  )
```

Trained the model

To achieve this, we input our training data and mod into a *fit()* function. The *epoch* indicates how many times the algorithm views the entire dataset.

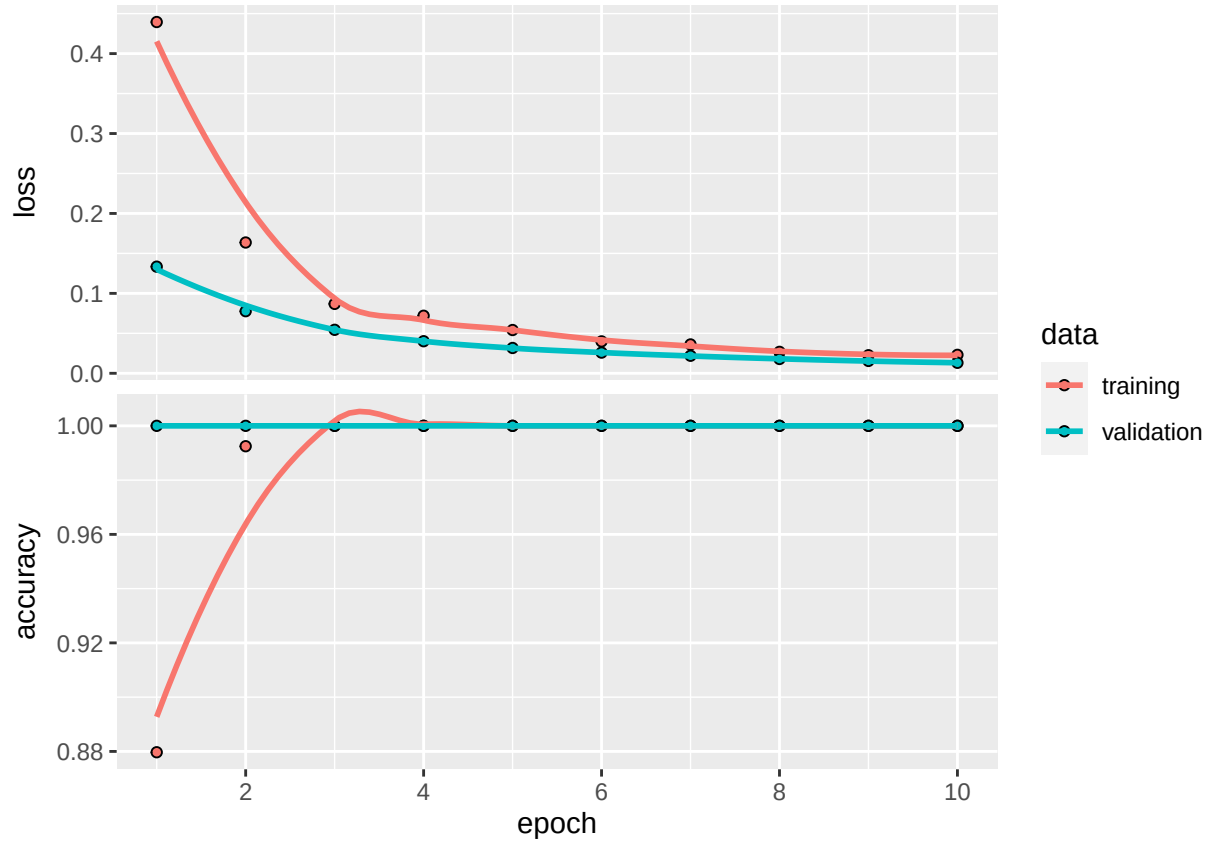
```
fitted <- mod %>%
  fit(xtrain, ytrain,
      epochs = 10,
      batch_size = 128,
      validation_split = 0.15)


# Displaying the output
fitted
```

```
##
## Final epoch (plot to see history):
```

```
##         loss: 0.02304
##     accuracy: 1
##     val_loss: 0.01311
## val_accuracy: 1
```

```
# Plotting the training and validation performance over 10 epochs
plot(fitted)
```



Evaluate the Trained Model using Testing Dataset

```
mod %>%
  evaluate(xtest, ytest)
```

```
##       loss   accuracy
## 0.01278048 1.00000000
```

```
dim(xtest)
```

```
## [1]  40 429
```

```
dim(ytest)
```

```
## [1] 40  2
```

Model Prediction using Testing Dataset

```
mod   %>% predict(xtest) %>% `>`(0.5) %>% k_cast("int32")
```

```
## tf.Tensor(
## [[0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]
##  [0 1]], shape=(40, 2), dtype=int32)
```