Model 2: Neural Network Based Model

Junaira I. Ibrahim

2022-12-15

Importing Packages

```
library(dplyr)         # for data manipulation
```

```
## Warning: package 'dplyr' was built under R version 4.1.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(keras)         # for fitting DNNs
```

```
## Warning: package 'keras' was built under R version 4.1.3
```

```
library(tfruns)        # for additional grid search & model training functions
```

```
## Warning: package 'tfruns' was built under R version 4.1.3
```

```
library(tensorflow)
```

```
## Warning: package 'tensorflow' was built under R version 4.1.3
```

```
library(tfestimators)  # provides grid search & model training interface
```

```
## Warning: package 'tfestimators' was built under R version 4.1.3
```

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

```
library(rsample)
```

```
## Warning: package 'rsample' was built under R version 4.1.3
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.1.3
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.8     v stringr 1.4.1
## v tidyr   1.2.1     v forcats 0.5.2
## v readr   2.1.3
```

```
## Warning: package 'ggplot2' was built under R version 4.1.3
## Warning: package 'tibble' was built under R version 4.1.3
## Warning: package 'tidyr' was built under R version 4.1.3
## Warning: package 'readr' was built under R version 4.1.3
## Warning: package 'purrr' was built under R version 4.1.3
## Warning: package 'stringr' was built under R version 4.1.3
## Warning: package 'forcats' was built under R version 4.1.3
## -- 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 4.1.3
```

Importing the dataset

The dataset used in this model is imported from `radiomics data`. It has 197 observations and 431 variables.

```
datard <- read_csv("radiomics_completedata.csv")
```

```
## Rows: 197 Columns: 431
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (1): Institution
## dbl (430): Failure.binary, Failure, Entropy_cooc.W.ADC, GLNU_align.H.PET, Mi...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
dim(datard)
```

```
## [1] 197 431
```

Data Pre-Processing

Preprocessing the data

####Checking for null and missing values

```
is.na(datard)
colSums(is.na(datard))
```

```
anyNA(datard)
```

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

Based on the results, there is no missing values.

####Checking for normality

```
dp1 = datard%>%select_if(is.numeric)
datadl1 = lapply(dp1[,-1], shapiro.test)
r = lapply(datadl1, function(x)x$p.value) #Extracting p-value only
s=unlist(r)    #to convert a list to vector
sum(s[s>0.05])
```

```
## [1] 0.1350113
```

```
r$Entropy_cooc.W.ADC
```

```
## [1] 0.1350113
```

Based on the results, there is only one variable who is normally distributed (i.e. Entropy_cooc.W.ADC). All the rest are not normally distributed. Hence, we will try to normalize the data using `orderNorm()` function.

####Normalizing the data

```
datard_norm = datard[,c(3,5:length(names(datard)))]
datard_norm = apply(datard_norm,2,orderNorm)
datard_norm = lapply(datard_norm, function(x) x$x.t)   #to transformed original data
datard_norm = datard_norm%>%as.data.frame()
```

Test again using shapiro-wilk's test.

```
datadl2 = lapply(datard_norm, shapiro.test)
r2 = lapply(datadl2, function(x) x$p.value)
s2 = unlist(r2)
sum(s2>0.05)
```

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

Based on the results, the rest of the variables is now normally distributed.

Substituing the normalized values into the original data, we have

```
r3 = select(datard, c("Failure.binary",  "Entropy_cooc.W.ADC"))
datard_n = cbind(r3,datard_norm)
```

Splitting

Split the data into training (80%) and testing (30%).

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

set.seed(123)
rdsplit = initial_split(datard_n, prop = 0.8, strata = "Failure.binary")
rdtrain <- training(rdsplit)
rdtest  <- testing(rdsplit)

train1 <- rdtrain[,-c(1,2)]%>%as.matrix.data.frame()
train2 <- rdtrain$Failure.binary
test1 <- rdtest[,-c(1,2)]%>%as.matrix.data.frame()
test2 <- rdtest$Failure.binary
```

Reshaping the dataset

```
train1 <- array_reshape(train1, c(nrow(train1), ncol(train1)))
train1 <- train1

test1 <- array_reshape(test1, c(nrow(test1), ncol(test1)))
test1 <- test1

train2 <- to_categorical(train2, num_classes = 2)
```

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

```
test2 <- to_categorical(test2, num_classes = 2)
```

Run the model

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

   # Network architecture
  layer_dense(units = 256, activation = "sigmoid", input_shape = c(ncol(train1))) %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 128, activation = "sigmoid") %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 128, activation = "sigmoid") %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 64, activation = "sigmoid") %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 64, activation = "sigmoid") %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 2, activation = "softmax") %>%

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

```
## Model: "sequential"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_5 (Dense)                    (None, 256)                     109824
##  dropout_4 (Dropout)                (None, 256)                     0
##  dense_4 (Dense)                    (None, 128)                     32896
##  dropout_3 (Dropout)                (None, 128)                     0
##  dense_3 (Dense)                    (None, 128)                     16512
##  dropout_2 (Dropout)                (None, 128)                     0
##  dense_2 (Dense)                    (None, 64)                      8256
##  dropout_1 (Dropout)                (None, 64)                      0
##  dense_1 (Dense)                    (None, 64)                      4160
##  dropout (Dropout)                  (None, 64)                      0
##  dense (Dense)                      (None, 2)                       130
## =========================================================================
## Total params: 171,778
## Trainable params: 171,778
## Non-trainable params: 0
## _________________________________________________________________________
```

Trained the model

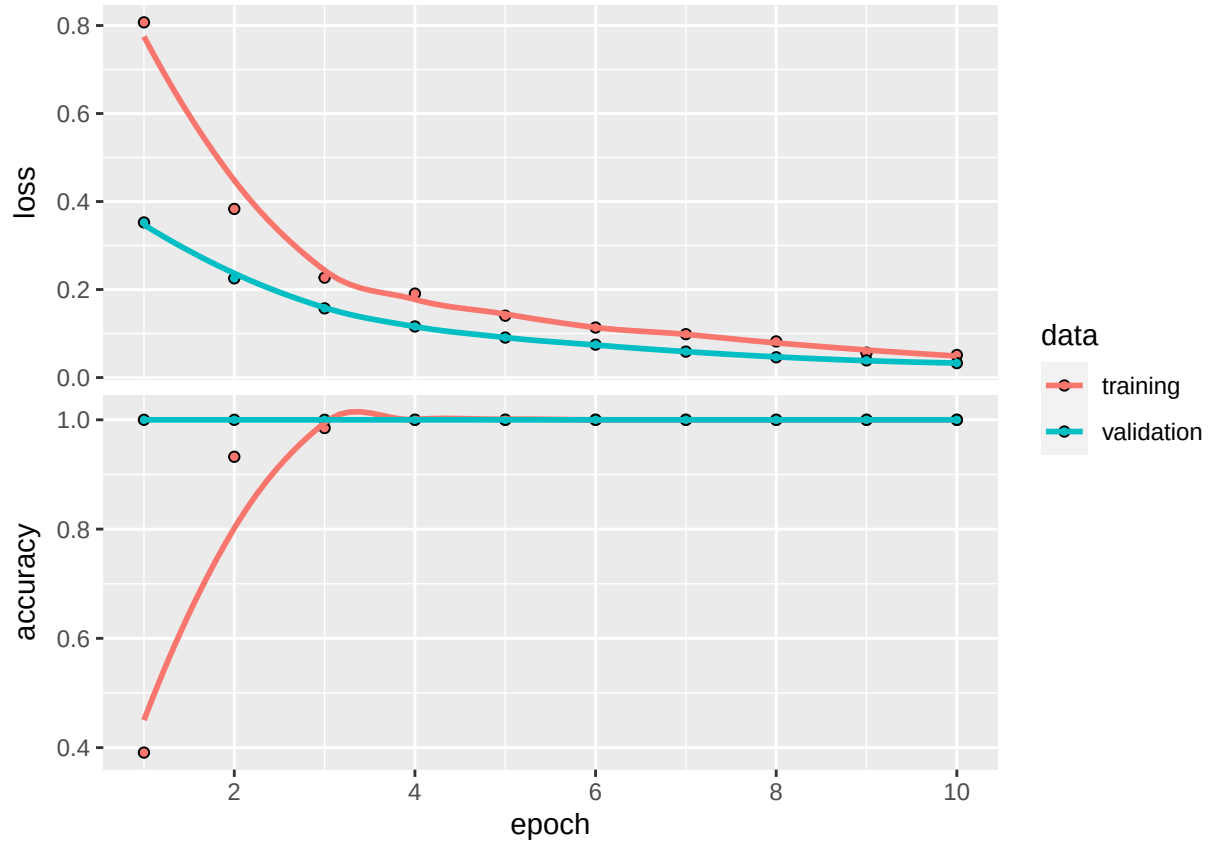
```
#trained model history
fitdl <- modeldl %>%
```

```
  fit(train1, train2,
      epochs = 10,
      batch_size = 128,
      validation_split = 0.15)

# Display output
fitdl
```

```
##
## Final epoch (plot to see history):
##         loss: 0.05131
##     accuracy: 1
##     val_loss: 0.03279
## val_accuracy: 1
```

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



Evaluate the trained model using testing dataset

```
modeldl %>%
  evaluate(test1, test2)
```

```
##       loss   accuracy
## 0.03197459 1.00000000
```

```r
dim(test1)
```

```
## [1]  40 428
```

```r
dim(test2)
```

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

Model prediction using testing dataset

```r
modeldl %>%
  predict(test1) %>% `>`(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)
```