MODEL 2 NEURAL NETWORK-BASED CLASSIFICATION MODEL

JUDISMA SALI

2022-12-16

Creating a Neural Network-Based Classification Model.

Note that we used the reprocessed data of radiomics_complete.csv (*RAD. NORMAL DATA.CSV*) in performing neural network-based classification model

LOAD PACKAGES

```
# Helper packages

library(dplyr)         # for data wrangling
```

```
##
## 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(tidyverse)     # for filtering
```

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

## v ggplot2 3.4.0     v purrr   0.3.5
## 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
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(readr)         # load dataset
library(rsample)       # for creating validation splits
library(bestNormalize) # for normalizing the dataset

# Modeling packages

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

```r
library(tensorflow)

# Modeling helper package

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

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

LOAD THE REPROCESSED DATASET

Radiomics Dataset 197 Rows (Observations) of 431 Columns (Variables) Failure.binary: binary property to predict

```r
radiomicsdt <- read_csv("RAD. NORMAL DATA.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.
```

```r
View(radiomicsdt)
head(radiomicsdt)
```

```
## # A tibble: 6 x 431
##   Institution Failure.~1 Failure Entro~2 GLNU_~3 Min_h~4 Max_h~5 Mean_~6 Varia~7
##   <chr>            <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 A                    0   1.15     12.9  -0.433  -0.270 -0.257   -0.192  0.0509
## 2 A                    1  -0.533    12.2  -1.02    0.671  0.405    0.490  0.687
## 3 A                    0   2.24     12.8   0.179  -1.41  -1.57    -1.53  -1.57
## 4 A                    1  -0.140    13.5   2.00   -0.218  0.0764  -0.153  0.0127
## 5 A                    0   0.787    12.6   0.153  -1.06  -1.15    -1.45  -1.91
## 6 A                    1  -2.80     13.2   0.391  -1.57  -1.91    -1.72  -1.84
## # ... with 422 more variables: Standard_Deviation_hist.PET <dbl>,
## #   Skewness_hist.PET <dbl>, Kurtosis_hist.PET <dbl>, Energy_hist.PET <dbl>,
## #   Entropy_hist.PET <dbl>, AUC_hist.PET <dbl>, H_suv.PET <dbl>,
## #   Volume.PET <dbl>, X3D_surface.PET <dbl>, ratio_3ds_vol.PET <dbl>,
## #   ratio_3ds_vol_norm.PET <dbl>, irregularity.PET <dbl>,
## #   tumor_length.PET <dbl>, Compactness_v1.PET <dbl>, Compactness_v2.PET <dbl>,
## #   Spherical_disproportion.PET <dbl>, Sphericity.PET <dbl>, ...
```

CHECKING FOR NULL AND MISSING VALUES

The result for checking null and missing values is *0* using *sum(is.n())*. Thus, there is no null and missing values.

```r
sum(is.na(radiomicsdt))
```

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

SPLITTING DATASET INTO TRAINING (0.8) AND TESTING (0.2)

```r
radiomicsdt<-radiomicsdt %>%
  mutate(Failure.binary=ifelse(Failure.binary== "No",0,1))
radiomicsdt=radiomicsdt[,-1]

set.seed(123)
split = initial_split(radiomicsdt,prop = 0.8 ,strata = "Failure.binary")
churn_train <- training(split)
churn_test  <- testing(split)

X_train <- churn_train[,-c(1,2)]%>%as.matrix.data.frame()
X_test <- churn_test[,-c(1,2)]%>%as.matrix.data.frame()
y_train <- churn_train$Failure.binary
y_test <- churn_test$Failure.binary
```

RESHAPING DATASET

```r
X_train <- array_reshape(X_train, c(nrow(X_train), ncol(X_train)))
X_train <- X_train

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

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

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

```r
y_test <- to_categorical(y_test, num_classes = 2)
```

The keras package allows us to develop our network with a layering approach. First, we initiated our sequential feedforward DNN architecture with *keras_model_sequential()* and then added some dense layers.Hence, we created five hidden layers with 256, 128, 128, 64 and 64 neurons, we added the *sigmoid* activation function. Followed by an output layer with 2 nodes and specified activation = *softmax*.

BACKPROPAGATION COMPILER APPROACH

To perform backpropagation we need two things: An objective function; An optimizer. First, we established an objective (loss) function to measure performance. For classification problems it is commonly binary and multi-categorical cross entropy. On each forward pass the DNN will measure its performance based on the loss function chosen. To incorporate the backpropagation piece of our DNN we include compile() in our code sequence. In addition to the optimizer and loss function arguments, we can also identify one or more metrics in addition to our loss function to track and report

```r
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "sigmoid", input_shape = c(ncol(X_train))) %>%
  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")%>%
 compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(),
    metrics = c("accuracy")
  )
```

MODEL COMPILER APPROACH

```
 model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = optimizer_adam(),
  metrics = c("accuracy")
)
```

TRAINING THE MODEL

To do so we feed our model into a fit() function along with our training data. We also provide a few other arguments that are worth mentioning: EPOCH = 10, BATCH SIZE = 128 AND VALIDATION SPLIT = 0.15

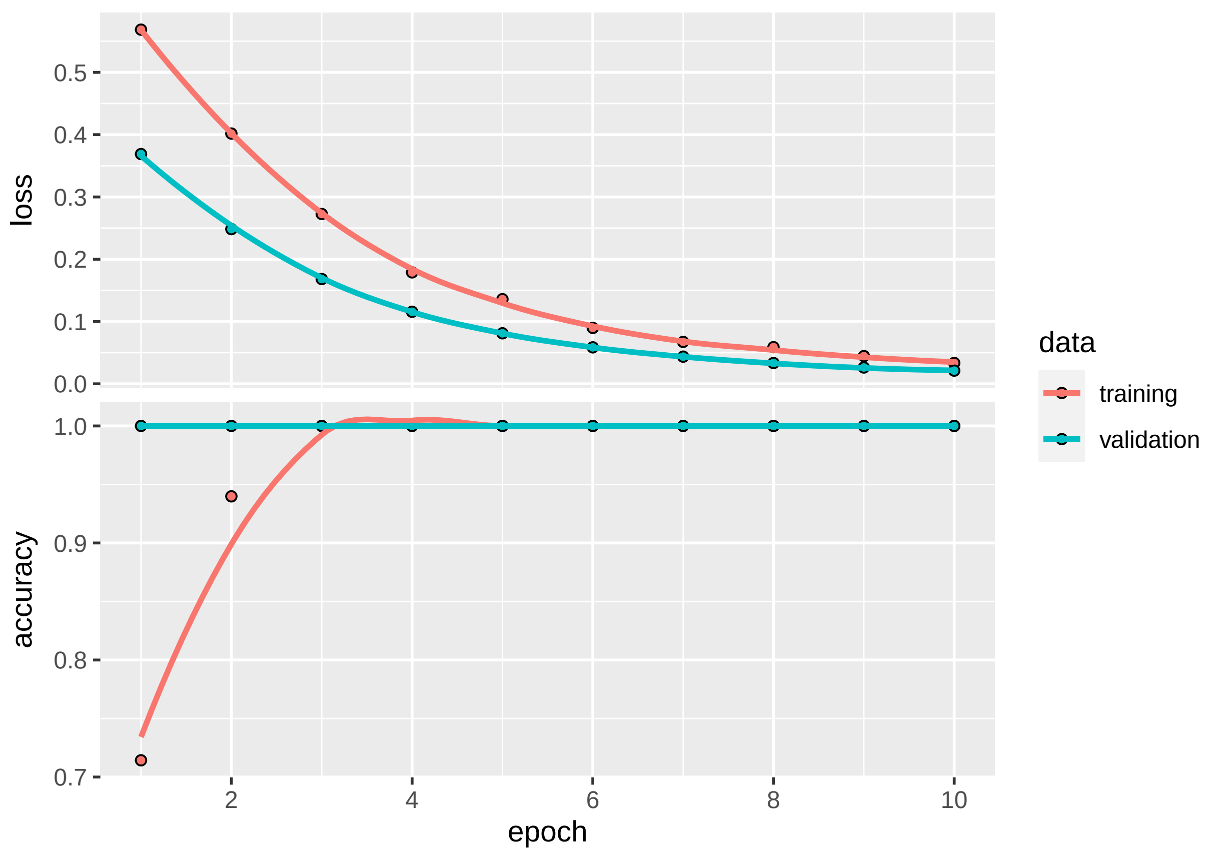
An epoch indicates how many times the algorithm views the entire dataset. Therefore, an epoch has ended whenever the algorithm has viewed all of the samples in the data set. Since a single epoch would be too large to transmit to the computer all at once, we divide it in several smaller batches.

```
trainm <- model %>%
  fit(X_train, y_train, epochs = 10, batch_size = 128, validation_split = 0.15)

trainm
```

```
##
## Final epoch (plot to see history):
##         loss: 0.03348
##     accuracy: 1
##     val_loss: 0.02116
## val_accuracy: 1
```

```
plot(trainm)
```



EVALUATE THE TRAINED MODEL USING THE TESTING DATASET.

```
model %>%
  evaluate(X_test, y_test)
```

```
##       loss   accuracy
## 0.02088225 1.00000000
```

```
dim(X_test)
```

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

```
dim(y_test)
```

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

MODEL PREDICTION USING THE TESTING DATASET

```
model   %>% predict(X_test) %>% `>`(0.8) %>% 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)
```