INFS 692 Data Science Final Project Model2

Ivan Huang

2022-12-07

Model 2

first thing first, don't forget to import essential libraries

```
library(keras)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

Step1: Create a neural network-based classification model.

```
# read the csv file to data frame
data_m2 <- read.csv('./radiomics_completedata.csv')

index <- createDataPartition(data_m2$Failure.binary,p=0.8,list=F)

#Test labels in the Species column (column 5)
Train_Features <- data.matrix(data_m2[index,-2])
Train_Labels <- data_m2[index,2]
Test_Features <- data.matrix(data_m2[-index,-2])
Test_Labels <- data_m2[-index,2]

#convering the labels into categorical
to_categorical(as.numeric(Train_Labels))[,c(-1)] -> Train_Labels
```

```
## Loaded Tensorflow version 2.9.2
```

```
to_categorical(as.numeric(Test_Labels))[,c(-1)] -> Test_Labels

summary(Train_Labels)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0000  0.0000  0.3291  1.0000  1.0000
```

```
str(Train_Features)
```

```
##  num [1:158, 1:430] 1 1 1 1 1 1 1 1 1 1 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:158] "2" "3" "4" "6" ...
##   ..$ : chr [1:430] "Institution" "Failure" "Entropy_cooc.W.ADC" "GLNU_align.H.PET" ...
```

```
#converting the features into matrix
as.matrix(apply(Train_Features, 2, function(x) (x-min(x))/(max(x) - min(x)))) -> Train_Features
as.matrix(apply(Test_Features, 2, function(x) (x-min(x))/(max(x) - min(x)))) -> Test_Features
```

Step 2: Create five hidden layers with 256, 128, 128, 64 and 64 neurons, respectively with activation functions of Sigmoid

```
#building the model
model <- keras_model_sequential()

#model training
model <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "sigmoid", input_shape = ncol(Train_Features)) %>%
  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') %>%

  #compiling the model
  compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_rmsprop(),
    metrics = c("accuracy")
    )

summary(model)
```

```
## Model: "sequential_1"
## _________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =========================================================================
##  dense_5 (Dense)                    (None, 256)                     110336
##  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: 172,290
## Trainable params: 172,290
## Non-trainable params: 0
## --------------------------------------------------------------------------------
```

Step 4: Copy the slide 33 model compiler approach.

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

Step 5: Train the model with epoch = 10, batch size = 128 and validation split = 0.15 (reference slide 33).

```
history <- model %>%
  fit(Train_Features, Train_Labels, epochs = 10, batch_size = 128, validation_split = 0.5)
```

Step 6: Evaluate the trained model using the testing dataset.

```
model %>% evaluate(Test_Features,Test_Labels)
```

```
##      loss  accuracy
## 0.6662458 0.6153846
```

Step 7: Get the model prediction using the testing dataset.

```
model %>% predict(Test_Features)
```

```
##            [,1]      [,2]
##  [1,] 0.6103910 0.3896090
##  [2,] 0.6102262 0.3897738
##  [3,] 0.6102059 0.3897941
##  [4,] 0.6102496 0.3897505
##  [5,] 0.6102636 0.3897364
##  [6,] 0.6103192 0.3896807
##  [7,] 0.6104610 0.3895390
```

```
##  [8,] 0.6104718 0.3895281
##  [9,] 0.6102268 0.3897732
## [10,] 0.6101736 0.3898264
## [11,] 0.6104292 0.3895708
## [12,] 0.6103175 0.3896825
## [13,] 0.6101081 0.3898919
## [14,] 0.6098777 0.3901223
## [15,] 0.6102908 0.3897093
## [16,] 0.6098870 0.3901130
## [17,] 0.6103286 0.3896714
## [18,] 0.6102271 0.3897729
## [19,] 0.6103230 0.3896771
## [20,] 0.6103951 0.3896049
## [21,] 0.6102831 0.3897169
## [22,] 0.6102976 0.3897025
## [23,] 0.6101804 0.3898196
## [24,] 0.6105156 0.3894844
## [25,] 0.6100547 0.3899453
## [26,] 0.6104928 0.3895072
## [27,] 0.6103169 0.3896831
## [28,] 0.6101460 0.3898540
## [29,] 0.6107951 0.3892049
## [30,] 0.6105409 0.3894590
## [31,] 0.6095138 0.3904862
## [32,] 0.6094935 0.3905064
## [33,] 0.6093163 0.3906837
## [34,] 0.6089234 0.3910767
## [35,] 0.6094825 0.3905176
## [36,] 0.6096620 0.3903380
## [37,] 0.6093158 0.3906843
## [38,] 0.6094611 0.3905390
## [39,] 0.6094196 0.3905804
```