Hyper-parameter_Tuning

Milton Candela

8/2/2021

Packages used

The following packages were used:

```
library(R.matlab)
library(dplyr)
library(caret)
library(randomForest)
library(RColorBrewer)
library(lattice)
```

Loading data and training model

Three new, processed CSV files were created using *createCSV.R*:

- RF_MARS_None.csv
- RF_MARS_Down.csv
- RF_MARS_Up.csv

Each file already has a sampling technique applied to reduce the class imbalance, given by the third word on the name of the file (*None*, *Down*, *Up*). The dataset is based on the document that contains data from 4 minutes (*WFM.4*), and combines both *FeatureMatrices.mat* and *FeatureMatrices_pre.mat*. It only has 11 columns (10 features and a Class column), although, a different feature selection technique was used in order to obtain the best features.

The LDA feature selection technique was removed, and instead, the RF's GINI Importance was used, filtering the number of features to 20. That is the only modification on the previous pipeline, the MARS model is still being used as a second feature selection technique using these 20 features to obtain the best 10 features. The previous change was done because the RF algorithm showed promising results, and thus a technique that was more related to it, could improve its performance. Due to the RF algorithm having a factor of randomness, 20 different seeds were used to compute the best features using the hybrid feature selection method, afterwards, the frequency of each feature was computed to obtain the most predominant features.

A new 80:20 split is being applied to train the model with more data, now that we found the optimal parameters. We now use **RandomForest** package to tune its parameters more precisely and thus get a more efficient algorithm according to our data and objective. The hyper-parameters used were selected due a grid search performed in the script *gridSearch.R*, the script outputs a CSV file with the grid values and the models accuracy on training and testing dataset. The best ratio between testing and training accuracy was chosen, and so its parameters are being applied on the final model that is being fitted.

```
df <- read.csv('Data/Processed/RF_MARS_Up.csv')
df <- mutate(df, Clas = as.factor(Clas))

set.seed(1002)
trainIndex <- createDataPartition(df$Clas, p = 0.8, list = FALSE)
training <- df[trainIndex,]
testing <- df[-trainIndex,]

model <- randomForest(Clas ~ ., data = training, mtry = 9, ntree = 60,
                      maxnodes = 1300, nodesize = 8)
writeLines(paste('Accuracy of the model on the training dataset',
                 round(confusionMatrix(training$Clas, predict(model, training))$overall[1], 5)))
```

```
## Accuracy of the model on the training dataset 0.97296
```

```
writeLines(paste('Accuracy of the model on the testing dataset',
                 round(confusionMatrix(testing$Clas, predict(model, testing))$overall[1], 5)))
```

```
## Accuracy of the model on the testing dataset 0.90353
```

As it can be seen, there is not so many difference between training and testing accuracy, and so the main issue of random forest: overfitting, is solved. The hyper-parameters selected are local and so they must be changed when new data is used, although they are useful for the current purposes.

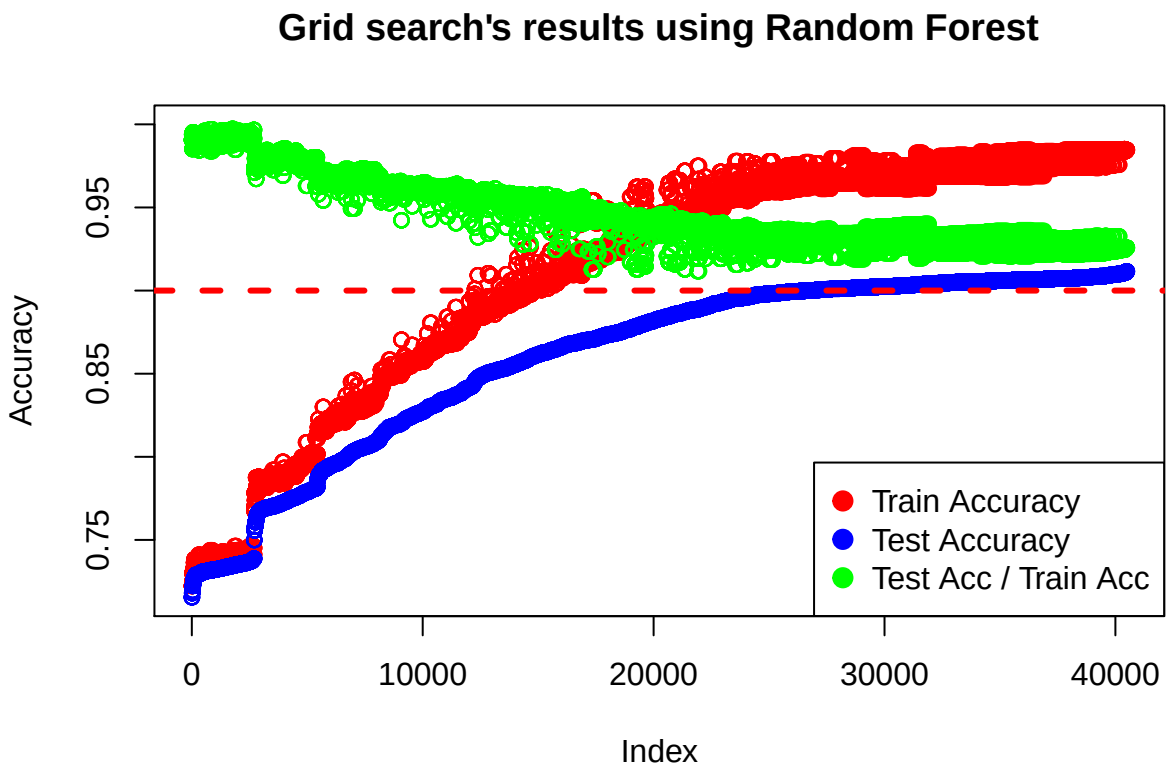
Hyper-parameters justification

Using the previously describe script *gridSearch.R*, a plot of training and testing accuracies could be achieve to better visualize the relationship between these factors. The next plot also includes another variable, which is the ratio of the testing and training accuracy, this represents the trade-off between giving more priority to training dataset rather than the testing dataset, an additional red line was also included, which represents the threshold of 0.90 accuracy.

It is worth nothing that the accuracies between the 20000 and 30000 index have an "elbow", which means that their accuracy gradient is lowering as the index increases, and so these parameters are an optimal solution because they represent the inflection point. This reduces overfitt within the function, as well as underfitting, because the model is balancing both training and testing accuracy.

```
hyper_grid <- read.csv('Data/Processed/rfGrid.csv')
hyper_order <- hyper_grid[order(hyper_grid$accTe),]
hyper_order[,'accProm'] <- hyper_order$accTe / hyper_order$accTa

plot(hyper_order$accTa, col = 'red', ylab = 'Accuracy',
     ylim = c(hyper_order[1,'accTe'], 1),
     main = "Grid search's results using Random Forest")
points(hyper_order$accTe, col = 'blue')
points(hyper_order$accProm, col = 'green')
abline(h = 0.90, lty = 2, col = 'red', lwd = 3)
legend('bottomright', pch = 20, col = c('red', 'blue', 'green'), pt.cex = 2,
       legend = c('Train Accuracy', 'Test Accuracy', 'Test Acc / Train Acc'))
```



Model's confusion matrix

In order to validate the model's performance, a confusion matrix would be created using the **lattice** package. The values are manually normalized and so 1 represents perfect accuracy, while 0 represents poor prediction power. The current dataset is balanced, and so the mean confusion matrix's accuracy can be used as a reliable matrix, that is free from bias.

```
Pred <- predict(model, testing)
confm <- confusionMatrix(reference = testing$Clas, data = Pred)$table

sum_row <- apply(t(confm), FUN = sum, MARGIN = 1)
conf_centage <- round(sweep(confm, MARGIN = 2, sum_row, FUN = '/'), 2)
row.names(conf_centage) <- c('No Fatigue', 'Moderate\nFatigue', 'Extreme\nFatigue')
colnames(conf_centage) <- c('No Fatigue', 'Moderate\nFatigue', 'Extreme\nFatigue')
color <- brewer.pal(n = 9, name = 'Greys')

levelplot(conf_centage, col.regions = color,
          xlab = 'Prediction', ylab = 'Reference', colorkey = FALSE,
          main = "Final model's confusion matrix on testing dataset",
          panel = function(x, y, z, ...) {
               panel.levelplot(x, y, z, ...)
               panel.text(x, y, conf_centage[cbind(x,y)])
          })
```

Final model's confusion matrix on testing dataset