Configuration Manual

MSc Research Project
Data Analytics

Anamika Chavan
Student ID: x18199950

School of Computing
National College of Ireland

Supervisor: Pierpaolo Dondio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Anamika Chavan
Student ID:	x18199950
Programme:	Data Analytics
Year:	2019
Module:	MSc Research Project
Supervisor:	Pierpaolo Dondio
Submission Due Date:	12/12/2019
Project Title:	Recruitment of Suitable Football Player by using Machine Learning Techniques.
Word Count:	547
Page Count:	8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	11th December 2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Anamika Chavan
x18199950

1 Introduction

A complete guidelines for the implementation of the research ”Recruitment of Suitable Football Player By using Machine Learning Techniques” is given in this document.This research have been developed in R-studio. All the libraries and packages used to develop this project are mentioned in this document.

2 Hardware Requirements

This research study was done on ”DELL Inspiron 13” laptop. Hardware configuration of this laptop is as follows:
Operating System: Windows 10
RAM : 8GB
Processor : Core i5
Storage: 256GBSSD
All above configuration are sufficient to run this project.

3 Software Requirements

For this research study R Studio and PowerBI was used. So below steps will explain installation process for R and PowerBI.

3.1 Download and Install R Studio

- Download the R-studio server for Windows 10 from the below link.
  `https://rstudio.com/products/rstudio/download/`
- After downloading the R-studio, next step is to install the R-studio. To install R-studio, refer the following link which contains clear instruction about installation.
  `http://rprogramming.net/download-and-install-rstudio/`

3.2 Download and Install PowerBI

- Download the PowerBI for Windows 10 from the below link.
  `https://powerbi.microsoft.com/en-us/downloads/`

- After downloading the powerBI, next step is to install the powerBI. To install powerBI, refer the following link which contains clear instruction about installation. `https://www.knowledgehut.com/blog/business-intelligence-and-visualization/how-to-install-power-bi`

4 Implementation of the Models

After installation of all software, implementation of the project can be done as follows.

4.1 Download the dataset

For this research, data is collected from kaggle which contains information about various players. Download the dataset from below link website.
`https://www.knowledgehut.com/blog/business-intelligence-and-visualization/how-to-install-power-bi`

4.2 Import the libraries

For this research, we have used R language to develop machine learning models. To do so, we need to firstly clean the data. Hence, to clean and develop the model, we need to import some R libraries. Below are the R libraries used for this research.

```

library(dplyr)
library(Hmisc)
library(corrplot)
library(caTools)
library(tidyr)
library(e1071)
library(caret)
library(FNN)
library(MASS)
library(rpart)
library(xgboost)

```

Figure 1: Imported Libraries

4.3 Pre-processing of the data

After downloading the dataset,we preprocessed the data before applying it to the machine learning models. Preprocessing steps includes checking missing values,feature engineering, data encoding and data scaling. The screenshot of code snippet is given below.

```
#checking for null values
players_data[players_data==""] <- NA
which(is.na(players_data) == TRUE, arr.ind=TRUE)
#remove unwanted column
players_data[ ,c('Name','Club','Contract_Expiry',
                 'Nationality','Club_Joining','Birth_Date','National_Position',
                 'National_Kit','Club_Position','Club_Kit')] <- list(NULL)
#data Encoding
players_data$Work_Rate_1 <- as.factor(players_data$Work_Rate_1)
players_data$Work_Rate_1 <- factor(players_data$Work_Rate_1,
                                   levels = c("Low ", "Medium ", "High "),ordered = TRUE)

# convert columns character to numeric
cols.num <- c("Height","Weight")
players_data[cols.num] <- sapply(players_data[cols.num],as.numeric)

#convert rating into categories
players_data$Rating <- cut(players_data$Rating, breaks=c(40,45,50,55,60,65,70,75,80,90,100),
                           labels = c("1","2","3","4","5","6","7","8","9","10"))

#separate dataframe for each category
ForwardPlayers<- players_data%>% filter(players_data$Preffered_Position =="Forward")
ForwardPlayers$Preffered_Position <- NULL

MidfielderPlayers<- players_data%>% filter(players_data$Preffered_Position =="Midfielder")

DefenderPlayers<- players_data%>% filter(players_data$Preffered_Position =="Defender")

GoalkeeperPlayers<- players_data%>% filter(players_data$Preffered_Position =="Goalkeeper")
# Feature Scaling
training_set[c(2,3,5,7:41)] <- scale(training_set[c(2,3,5,7:41)])
test_set[c(2,3,5,7:41)] <- scale(test_set[c(2,3,5,7:41)])
```

Figure 2: Preprocessing of the Data

4.4 SVM Model

After pre-processing of the data, now we will apply all the models one by one. We will start with Support Vector Machine(SVM) model.Atter applying the model on the dataset, we have calculated evaluation metrics. The screenshot of all the above process is given below.

```r
#implementation of SVM

library(e1071)
library(caret)
folds <- createFolds(data$Rating, k = 10)

cv <- lapply(folds, function(x) {
  training_fold = data[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = data[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = svm(formula = Rating ~ .,
                   data = training_fold,
                   type = 'C-classification',
                   kernel = 'radial')
  y_pred = predict(classifier, newdata = test_fold[-1])

  cm = table(test_fold$Rating, y_pred)

  return(cm)
})

cm <- Reduce('+', cv)
accuracy_svm <- sum(diag(cm))/sum(cm)
precision_svm <- diag(cm)/colSums(cm)
avg_precision_svm <- mean(as.numeric(precision_svm), na.rm=TRUE)
recall_svm <- diag(cm)/rowSums(cm)
avg_recall_svm <- mean(recall_svm, na.rm=TRUE)
Fmeasure_SVM <- 2 * avg_precision_svm * avg_recall_svm / (avg_precision_svm + avg_recall_svm)
```

Figure 3: Implementation of SVM model

4.5 LDA model

Now we will apply Linear Discriminant Analysis(LDA) model on the data. The screenshot of the process is given below.

```r
#applying LDA model

library(MASS)

cv_lda <- lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = data[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = data[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = lda(Rating ~ .,training_fold)
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-1])

  cm = table(test_fold$Rating, y_pred$class)
  return(cm)
})
cm <- Reduce('+', cv_lda)
accuracy_lda <- sum(diag(cm))/sum(cm)
precision_lda <- diag(cm)/colSums(cm)
avg_precision_lda <- mean(precision_lda, na.rm=TRUE)
recall_lda <- diag(cm)/rowSums(cm)
avg_recall_lda <- mean(recall_lda, na.rm=TRUE)
Fmeasure_lda <- 2 * avg_precision_lda * avg_recall_lda / (avg_precision_lda + avg_recall_lda)
```

Figure 4: Implementation of LDA model

4.6 Naive Bayes model

Developing the Naive Bayes model is given below.

```
#naive_bayes

cv_naive_bayes <- lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = ForwardPlayers[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = ForwardPlayers[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = naiveBayes(formula = Rating ~ .,
                          data = training_fold,
                          probability = TRUE)
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, newdata = test_fold[-1])

  cm = table(test_fold$Rating, y_pred)
  accuracy <- sum(diag(cm))/sum(cm)
  return(cm)
})
cm <- Reduce('+', cv_naive_bayes)
accuracy_naive_bayes <- sum(diag(cm))/sum(cm)
precision_naive_bayes <- diag(cm)/colSums(cm)
avg_precision_naive_bayes <- mean(precision_naive_bayes, na.rm=TRUE)
recall_naive_bayes <- diag(cm)/rowSums(cm)
avg_recall_naive_bayes <- mean(recall_naive_bayes, na.rm=TRUE)
Fmeasure_naive_bayes <- 2 * avg_precision_naive_bayes * avg_recall_naive_bayes /
                            (avg_precision_naive_bayes + avg_recall_naive_bayes)


accuracy_naive_bayes <- mean(as.numeric(cv_naive_bayes))

```

Figure 5: Implementation of Naive Bayes model

4.7 Decision Tree model

Code snippet of building decision tree model is given below.

```r
#decision tree

library(rpart)

cv_rpart <- lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = ForwardPlayers[-x, ]
  # training fold =  training set minus (-) it's sub test fold
  test_fold = ForwardPlayers[x, ]
  # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  classifier = rpart(Rating~., data = training_fold, method = 'class')
  # next step in the loop, we calculate the predictions and cm and we equate the accuracy
  # note we are training on training_fold and testing its accuracy on the test_fold
  y_pred = predict(classifier, test_fold[-1], type = 'class')

  cm = table(test_fold$Rating, y_pred)
  accuracy <- sum(diag(cm))/sum(cm)
  return(cm)
})

cm <- Reduce('+', cv_rpart)
conf_matrix <- confusionMatrix(cm)
accuracy_rpart <- sum(diag(cm))/sum(cm)
precision_rpart <- diag(cm)/colSums(cm)
avg_precision_rpart <- mean(precision_rpart, na.rm=TRUE)
recall_rpart <- diag(cm)/rowSums(cm)
avg_recall_rpart <- mean(recall_rpart, na.rm=TRUE)
Fmeasure_rpart <- 2 * avg_precision_rpart * avg_recall_rpart /
                  (avg_precision_rpart + avg_recall_rpart)
```

Figure 6: Implementation of Decision Tree model

4.8 XGBoost model

Code snippet of building XGBoost model is given below.

```r
#implementation of XGBoost
folds <- createFolds(training_set$Rating, k = 10)
library(xgboost)
cv_xgboost <- lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  ForwardPlayers$Preffered_Foot <- as.numeric(ForwardPlayers$Preffered_Foot)
  training_fold = training_set[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = training_set[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold

  XTrain <- data.frame(lapply(training_fold[,-1],as.numeric))
  XTrain <- as.matrix(XTrain)
  yTrain <- unclass(training_fold$Rating)-1
  m.xg.def <- xgboost(data=XTrain,label=yTrain,objective="multi:softmax",num_class=10,nrounds = 1)

  XTest <- data.frame(lapply(test_set[,-1],as.numeric))
  XTest <- as.matrix(XTest)

  y.xg.def <- predict(m.xg.def,newdata=XTest)+1
  y.gbm.default1 <- factor(y.xg.def, levels = c(1,2,3,4,5,6,7,8,9,10), ordered = TRUE)
  cm <- table(test_set$Rating,y.gbm.default1)
  return(cm)
})
cm <- Reduce('+',cv_xgboost)
accuracy_xgboost <- sum(diag(cm))/sum(cm)
precision_xgboost <- diag(cm)/colSums(cm)
avg_precision_xgboost <- mean(precision_xgboost, na.rm=TRUE)
recall_xgboost <- diag(cm)/rowSums(cm)
avg_recall_xgboost <- mean(recall_xgboost, na.rm=TRUE)
Fmeasure_xgboost <- 2 * avg_precision_xgboost * avg_recall_xgboost / (avg_precision_xgboost + avg_recall_xgboost)
```

Figure 7: Implementation of XGBoost model

4.9 KNN model

Code snippet of building KNN model is given below.

```r
#knnn
library(FNN)
ForwardPlayers$Rating <- as.numeric(ForwardPlayers$Rating)
ForwardPlayers$Work_Rate_1 <- as.numeric(ForwardPlayers$Work_Rate_1)
ForwardPlayers$Work_Rate_2 <- as.numeric(ForwardPlayers$Work_Rate_2)
ForwardPlayers$Preffered_Foot <- as.numeric(ForwardPlayers$Preffered_Foot)
cv_knn <- lapply(folds, function(x) { # start of function
  # in the next two lines we will separate the Training set into it's 10 pieces
  training_fold = ForwardPlayers[-x, ] # training fold =  training set minus (-) it's sub test fold
  test_fold = ForwardPlayers[x, ] # here we describe the test fold individually
  # now apply (train) the classifer on the training_fold
  training_fold <- ForwardPlayers[-x,]
  ##extract testing set
  test_fold <- ForwardPlayers[x,]
  ##extract 5th column of train dataset because it will be used as 'cl' argument in knn function.
  target_category <- ForwardPlayers[-x,1]
  ##extract 5th column if test dataset to measure the accuracy
  test_category <- ForwardPlayers[x,1]
  test_category <- factor(test_category, levels = c(1,2,3,4,5,6,7,8,9,10), ordered = TRUE)
  k <- knn(training_fold, test_fold, training_fold$Rating, k = 8)

  k <- factor(k, levels = c(1,2,3,4,5,6,7,8,9,10), ordered = TRUE)
  cm <- table(k,test_category)
  return(cm)
})
cm <- Reduce('+',cv_knn)
accuracy_knn <- sum(diag(cm))/sum(cm)
precision_knn <- diag(cm)/colSums(cm)
avg_precision_knn <- mean(precision_knn, na.rm=TRUE)
recall_knn <- diag(cm)/rowSums(cm)
avg_recall_knn <- mean(recall_knn, na.rm=TRUE)
Fmeasure_knn <- 2 * avg_precision_knn * avg_recall_knn / (avg_precision_knn + avg_recall_knn)
```

Figure 8: Implementation of KNN model

4.10 Finding the closest match

To find the closest match for the replaced player, we have used knn model.Code snippet of finding the closest match is given below.

```r

k <- knn(training_set[,-1], test_set[,-1], labels, k = 10)
indices = attr(k, "nn.index")
print(indices[156, ])

```

Figure 9: Implementation of closest match

4.11 Evaluation Results by using powerBI

We have compared the two evaluation metrics accuracy and F-measure for different models. Following screenshot shows the comparison between different models in terms of accuracy and F-measure.

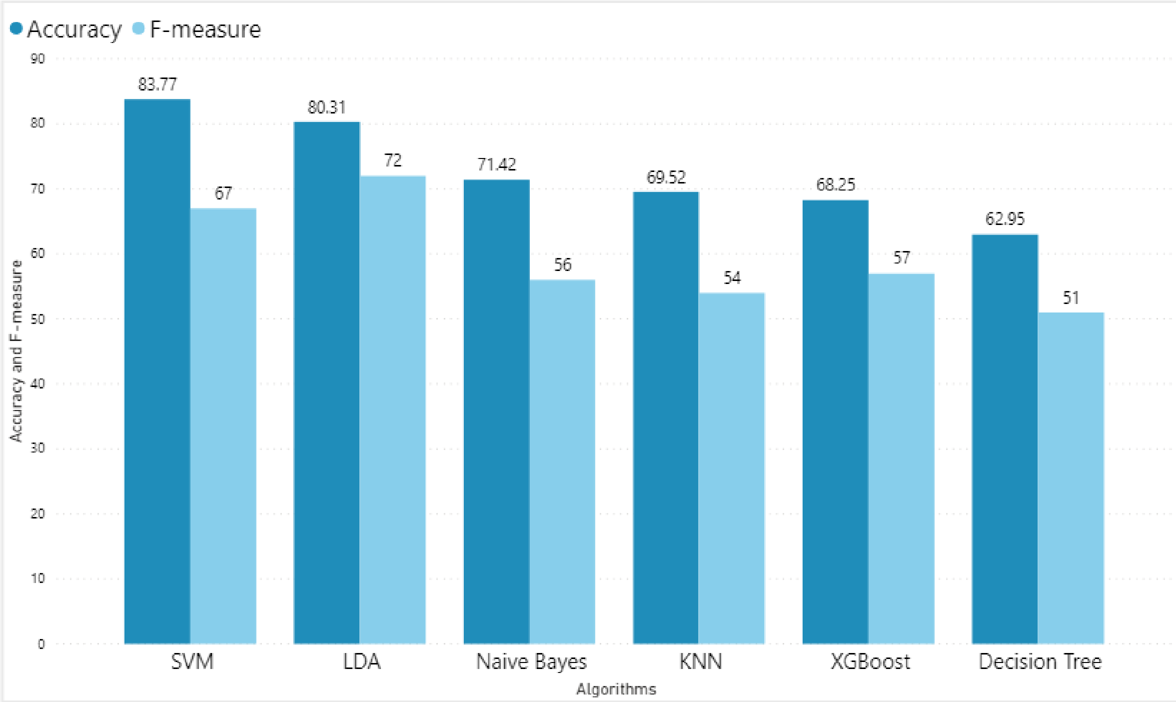

Figure 10: Accuracy and F-measure score by %