NbaProject

Javier Esteban Aragoneses

Project part 2

Firt, we will check if our sistem is close to be linear.

```
model=gam(Pos ~s(AST)+s(PTS)+s(DRB)+s(ORB)+s(DRB)+s(STL)+s(BLK)+s(FT)+s(F
G.)+s(FT.)+s(TOV),
    data = my_data,
    family = binomial,
    method = "REML",
    select=TRUE

    )
summary(model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## Pos ~ s(AST) + s(PTS) + s(DRB) + s(ORB) + s(DRB) + s(STL) + s(BLK) +
##     s(FT) + s(FG.) + s(FT.) + s(TOV)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   2.4443     0.2297   10.64   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##              edf Ref.df Chi.sq  p-value
## s(AST) 6.497e-05      9  0.000 0.402856
## s(PTS) 9.491e-01      9 16.942 7.44e-06 ***
## s(DRB) 3.763e-01      9  0.583 0.200910
## s(ORB) 1.178e+00      9 11.510 0.000191 ***
## s(STL) 9.308e-01      9 12.678 0.000187 ***
## s(BLK) 2.025e+00      9 46.084 1.41e-12 ***
## s(FT)  9.531e-06      9  0.000 0.782415
## s(FG.) 1.244e+00      9 21.505 1.46e-06 ***
## s(FT.) 6.870e-01      9  2.007 0.085061 .
## s(TOV) 7.150e-05      9  0.000 0.397991
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## R-sq.(adj) =  0.596   Deviance explained = 57.9%
## -REML = 132.88  Scale est. = 1         n = 540
```

All edf variables are close to one except TOV and FT so this variables have not a linear relationship with the response.

Now I will create a data partition with 80% por our data to train, because we have only 540 observations

```
in_train <- createDataPartition(my_data$Pos, p = 0.8, list = FALSE)
training <- my_data[ in_train,]
testing <- my_data[-in_train,]
nrow(training)

## [1] 433

nrow(testing)

## [1] 107
```

Then I will create a cost function. In order to create that, I have to take in account that classify a center like a guard and viceversa are worse that classify a foward like a center or a guard and viceversa. with this data, I calculate the naive cost results, taking into account that there are more fowards than center and guards and there are 41 guards and 23 centers in the data .

```
print("number of players by position in the data testing set ")

## [1] "number of players by position in the data testing set "

table(testing$Pos)

##
##  C  F  G
## 23 43 41

cost.unit <- c(0, 0.5, 1, 0.5, 0, 0.5,1, 0.5, 0)
relative.cost=cost.unit
print("cost")

## [1] "cost"

cost=0.5*(23+41)
cost

## [1] 32
```

The economic profit and control function is:

```
EconomicProfit <- function(data, lev = NULL, model = NULL)
{
  y.pred = data$pred
  y.true = data$obs
```

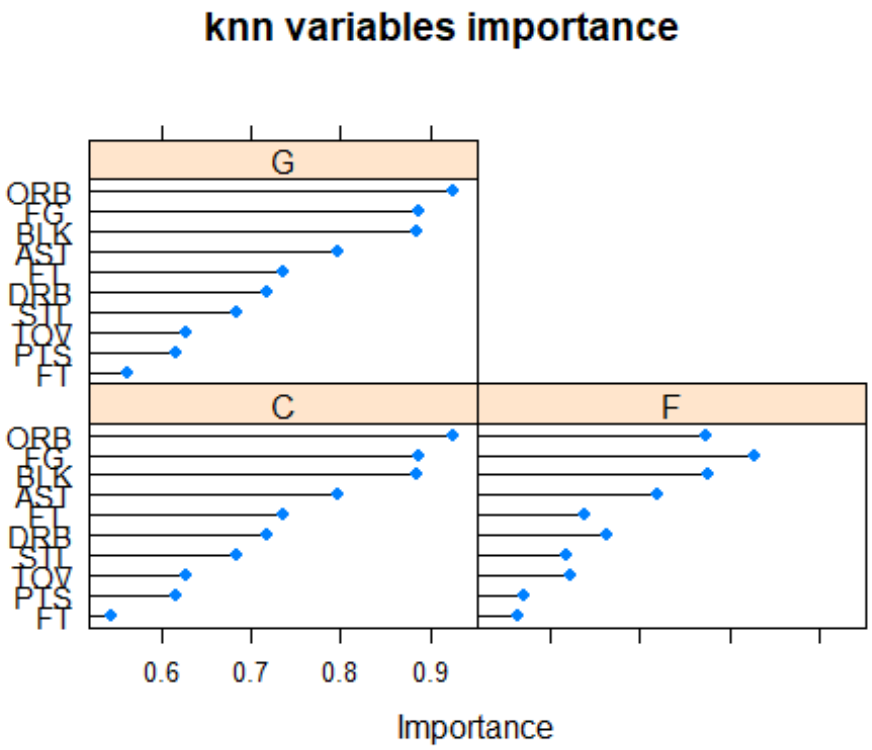
```
  CM = confusionMatrix(y.pred, y.true)$table
  out = sum(cost.unit*CM)/sum(CM)
  names(out) <- c("EconomicProfit")
  out
}
ctrl <- trainControl(method = "cv", number = 5,
                     classProbs = TRUE,
                     summaryFunction = EconomicProfit,
                     verboseIter=T)
```

First I will use the knn classification tool.This method have a good perfomance with highly no linear relationship and with modearte p/n. However, our model is slightly linear and have no enough n/p. So this is not the better tools to our problem.Nevetheless, this tools wotks good with less than 5 classification groups and in this case we have only three. The hyperparameter tunning will be use to know the optimal k.

```
knnFit <- train(Pos ~ .,
                method = "knn",
                data = training,
                preProcess = c("center", "scale"),
                tuneLength = 5,
                metric = "EconomicProfit",
                maximize=FALSE,
                trControl = ctrl)
print(knnFit)
```

The better result is with k=9.The importance of the variables for knn are:

```
knn_imp <- varImp(knnFit, scale = F)
plot(knn_imp,main="knn variables importance", scales = list(y = list(cex
= .95)))
```



We can see that the most important variables to classify centers and guards are blocks and offensive rebounds, statics that are higher for the tallest players(centers). Then assistance,field Goal Percentage and defensive rebounds helps us to classify because guards usually give more assistance than forwards and forwards more than centers. With FG. and DRB we have the opposite situation. Checking the predictions:

```
knnPred = predict(knnFit, testing)
confusionMatrix(knnPred,testing$Pos)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 15  4  0
##          F  8 31  4
##          G  0  8 37
##
## Overall Statistics
##
##                Accuracy : 0.7757
##                  95% CI : (0.6849, 0.8507)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : 3.744e-15
##
##                   Kappa : 0.6491
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.6522   0.7209   0.9024
## Specificity            0.9524   0.8125   0.8788
## Pos Pred Value         0.7895   0.7209   0.8222
## Neg Pred Value         0.9091   0.8125   0.9355
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1402   0.2897   0.3458
## Detection Prevalence   0.1776   0.4019   0.4206
## Balanced Accuracy      0.8023   0.7667   0.8906
```

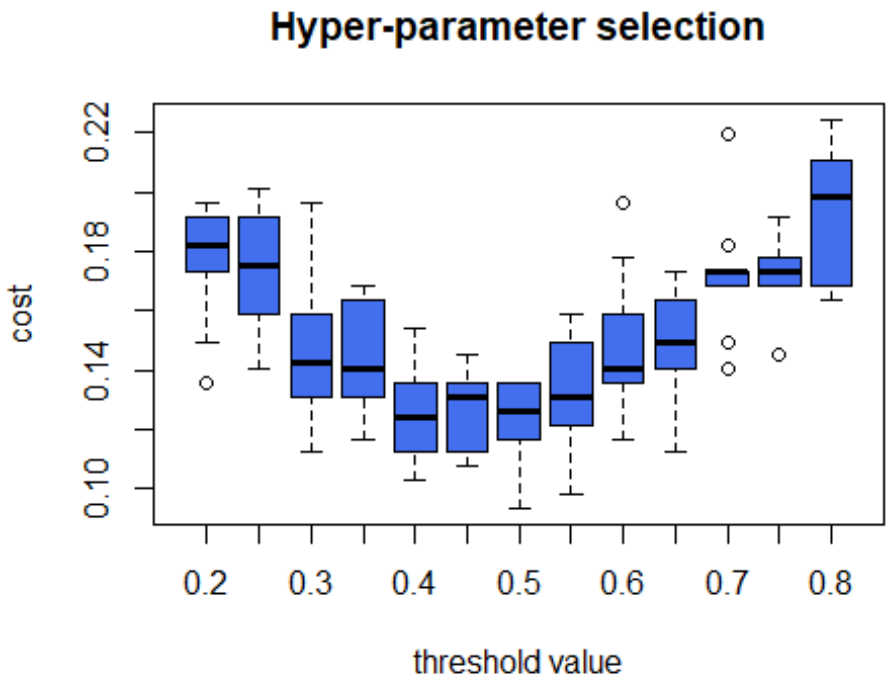
```
cost=sum(cost.unit*confusionMatrix(knnPred,testing$Pos)$table)
cost
```

```
## [1] 12
```

The accuracy is acceptable and the cost is reduce from 32 of the the naive classification to 10.The kappa is also high. Now I will check the best threshold

```
j=0;
cost.i = matrix(NA, nrow = 10, ncol = 13)
for (threshold in seq(0.2,0.8,0.05)){

  j <- j + 1
  cat(j)
  for(i in 1:10){


    d <- createDataPartition(my_data$Pos, p = 0.8, list = FALSE)

    train<-my_data[d,]
    test <-my_data[-d,]

    knnfit <- train(Pos ~ .,
                    method = "knn",
                    data = train,
                preProcess = c("center", "scale"),
                tuneLength = 5,
                maximize = F,
                metric = "EconomicProfit",
                trControl = ctrl)
    knnProb = predict(knnfit, test, type="prob")

    knnPred = rep("F", nrow(test))
    knnPred[which(knnProb[,1] > threshold)] = "C"
    knnPred[which(knnProb[,3] > threshold)] = "G"
    knnPred = factor(knnPred)
```

```
    CM = confusionMatrix(knnPred, test$Pos)$table

    cost.i[i,j] <- sum(relative.cost*CM)/nrow(test) # unitary cost

  }
}
```



We get the best results with 0.45 threshold

```
threshold = 0.45
knnProb = predict(knnfit, newdata=testing, type="prob")
knnPred = rep("F", nrow(testing))
knnPred[which(knnProb[,1] > threshold)] = "C"
knnPred[which(knnProb[,3] > threshold)] = "G"
knnPred = as.factor(knnPred)
confusionMatrix(knnPred, testing$Pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 17  1  0
##          F  6 36  5
##          G  0  6 36
##
## Overall Statistics
##
##                Accuracy : 0.8318
```

```
##                  95% CI : (0.7472, 0.8971)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7359
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.7391   0.8372   0.8780
## Specificity            0.9881   0.8281   0.9091
## Pos Pred Value         0.9444   0.7660   0.8571
## Neg Pred Value         0.9326   0.8833   0.9231
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1589   0.3364   0.3364
## Detection Prevalence   0.1682   0.4393   0.3925
## Balanced Accuracy      0.8636   0.8327   0.8936
```

```
CM = confusionMatrix(knnPred, testing$Pos)$table
print(" cost")
```

```
## [1] " cost"
```

```
sum(relative.cost*CM)
```

```
## [1] 9
```

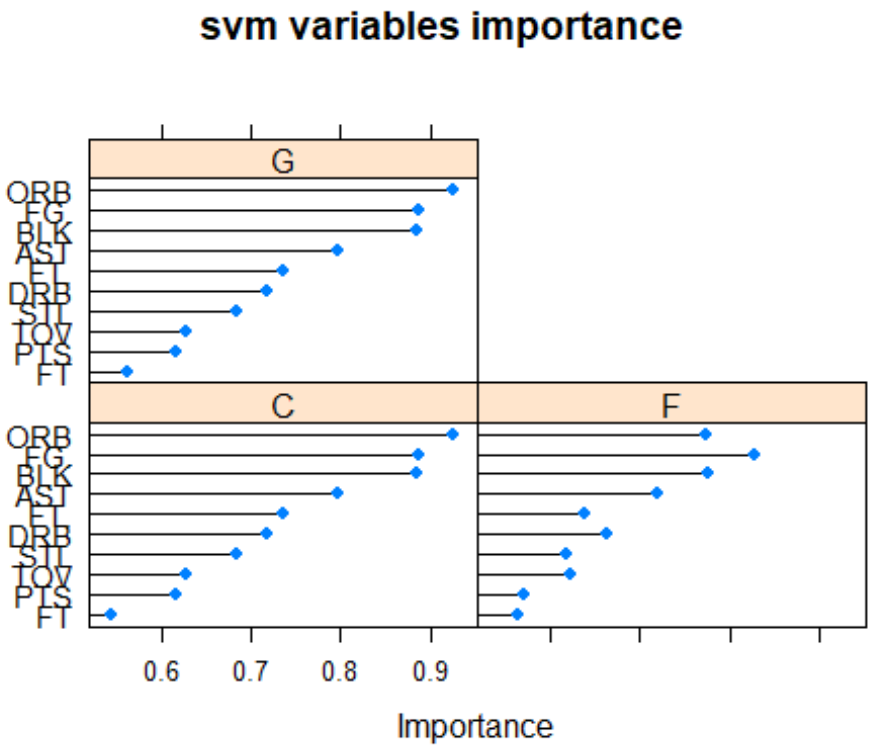
The total cost is reduced by one unit and the accuracy is much better.

Lets continue with svm. This tool is better with no linear predictors. However we have seen that there are some linear variables. This tool should be better because doesnt need a moderate p/n ratio. i Wil make hyperpameter tuning with the same control function.

```
svmFit <- train(Pos ~., method = "svmRadial",
                data = training,
                preProcess = c("center", "scale"),
                tuneGrid = expand.grid(C = seq(from = 0.05, to = 1, by =
0.1),
                                       sigma = seq(from = 0.05, to = 1, by
= 0.1)),
                metric = "EconomicProfit",
                maximize= F,
                trControl = ctrl)
```

The best parameters are sigma=0.15 and c=0.95. It means that we penalize the error more than the gapp. Now we check the imporance of the variables

```
svm_imp <- varImp(svmFit, scale = F)
plot(svm_imp,main="svm variables importance", scales = list(y = list(cex
= .95)))
```



The variables have similar importance than with knn. The cost is:

```
svmPred = predict(svmFit, testing)
confusionMatrix(svmPred,testing$Pos)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 16  2  0
##          F  7 33  6
##          G  0  8 35
##
## Overall Statistics
##
##                Accuracy : 0.785
##                  95% CI : (0.6951, 0.8586)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : 7.1e-16
##
##                   Kappa : 0.6626
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.6957   0.7674   0.8537
## Specificity            0.9762   0.7969   0.8788
## Pos Pred Value         0.8889   0.7174   0.8140
## Neg Pred Value         0.9213   0.8361   0.9062
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1495   0.3084   0.3271
## Detection Prevalence   0.1682   0.4299   0.4019
## Balanced Accuracy      0.8359   0.7822   0.8662
```

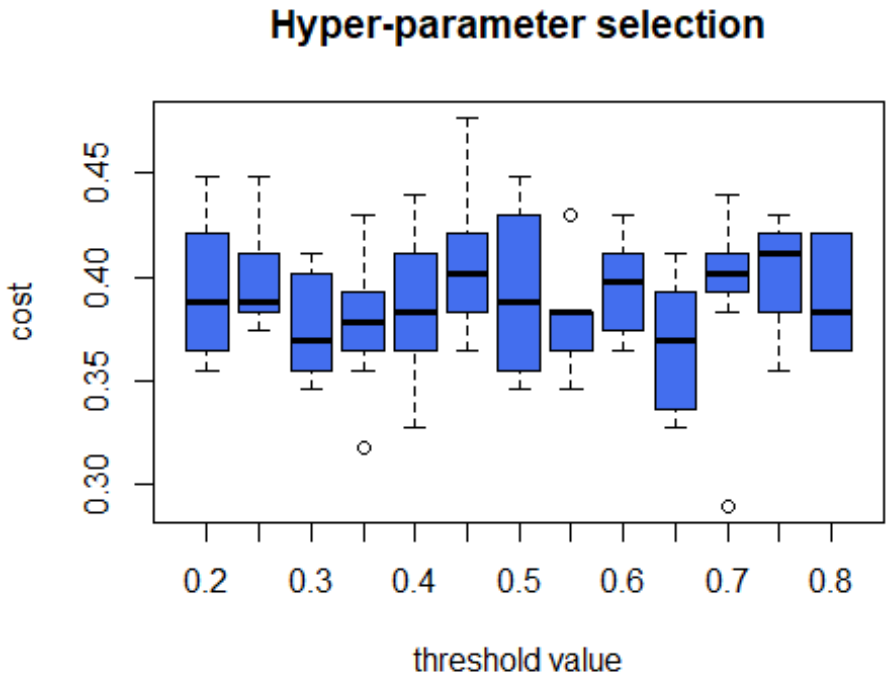
```
cost=sum(cost.unit*confusionMatrix(svmPred,testing$Pos)$table)
cost
```

```
## [1] 11.5
```

Svm have very similar results. Now i will check the best threshold with the optimal parameters. Accuracy and kappa are good.

```
j=0;
cost.i = matrix(NA, nrow = 10, ncol = 13)
for (threshold in seq(0.2,0.8,0.05)){

  j <- j + 1
  cat(j)
  for(i in 1:10){


    d <- createDataPartition(my_data$Pos, p = 0.8, list = FALSE)

    train<-my_data[d,]
    test <-my_data[-d,]

    svmfit <- train(Pos ~ .,
                    method = "svmRadial",
                    data = train,
                preProcess = c("center", "scale"),
                tuneLength = 5,
                maximize = F,
                tuneGrid = expand.grid(C = 0.95,
                sigma = 0.15),
                metric = "EconomicProfit",
                trControl = ctrl)
    svmProb = predict(svmfit, test, type="prob")

    svmPred = rep("F", nrow(test))
    svmPred[which(svmProb[,1] > threshold)] = "C"
    svmPred[which(svmProb[,3] > threshold)] = "G"
    svmPred = factor(svmPred)
```

```
    CM = confusionMatrix(knnPred, test$Pos)$table

    cost.i[i,j] <- sum(relative.cost*CM)/nrow(test) # unitary cost

  }
}

boxplot(cost.i, main = "Hyper-parameter selection",
        ylab = "cost",
        xlab = "threshold value",names = seq(0.2,0.8,0.05),col="royalblue
2")
```



The best threshold is 0.55. The cost with it is:

```
threshold = 0.5
svmProb = predict(svmFit, newdata=testing, type="prob")
svmPred = rep("F", nrow(testing))
svmPred[which(svmProb[,1] > threshold)] = "C"
svmPred[which(svmProb[,3] > threshold)] = "G"
svmPred = as.factor(svmPred)
confusionMatrix(svmPred, testing$Pos)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 16  2  0
```

```
##          F  7 34  7
##          G  0  7 34
##
## Overall Statistics
##
##                Accuracy : 0.785
##                  95% CI : (0.6951, 0.8586)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : 7.1e-16
##
##                   Kappa : 0.6624
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.6957   0.7907   0.8293
## Specificity            0.9762   0.7812   0.8939
## Pos Pred Value         0.8889   0.7083   0.8293
## Neg Pred Value         0.9213   0.8475   0.8939
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1495   0.3178   0.3178
## Detection Prevalence   0.1682   0.4486   0.3832
## Balanced Accuracy      0.8359   0.7860   0.8616
```

```
confusionMatrix(svmPred, testing$Pos)$table
```

```
##           Reference
## Prediction  C  F  G
##          C 16  2  0
##          F  7 34  7
##          G  0  7 34
```

```
CM
```

```
##           Reference
## Prediction  C  F  G
##          C  2  9  7
##          F 12 16 19
##          G  9 18 15
```

```
print(" cost")
```

```
## [1] " cost"
```

```
sum(relative.cost*CM)
```

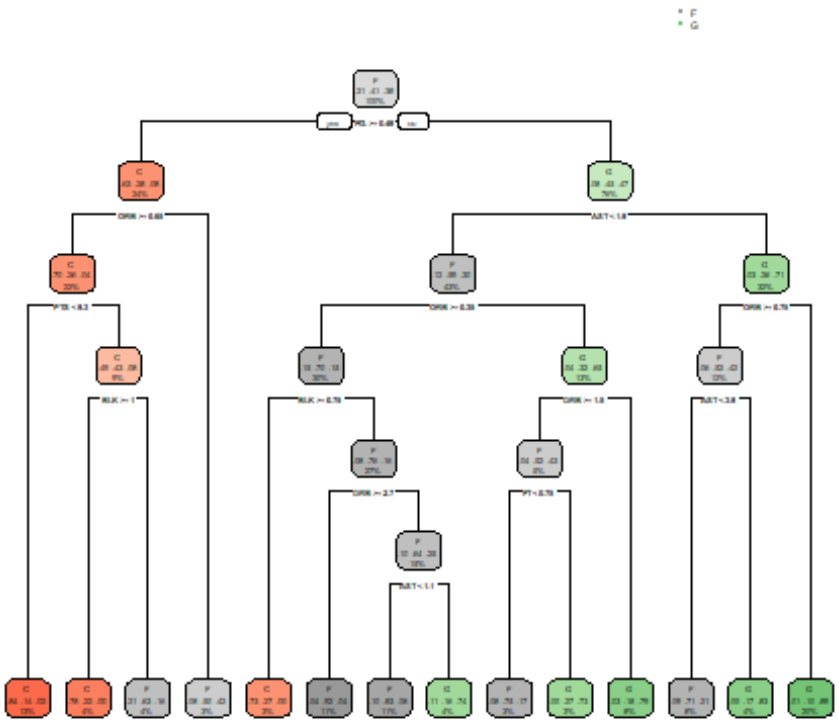
```
## [1] 45
```

With this threeshold we have a better performance that with knn.

Then, I will use Decision tree. In this case I will not make hyperàrameter tuning because i will use this method only to interpret, in order to predict it would be better to use random forest.

```
set.seed(1)
tree.fit <- train(Pos~.,
                  data = training,
                  method = "rpart",
                 trControl = ctrl,
                 metric = "EconomicProfit",
                 maximize=FALSE,
                  tuneLength=10)

## + Fold1: cp=0
## - Fold1: cp=0
## + Fold2: cp=0
## - Fold2: cp=0
## + Fold3: cp=0
## - Fold3: cp=0
## + Fold4: cp=0
## - Fold4: cp=0
## + Fold5: cp=0
## - Fold5: cp=0
## Aggregating results
## Selecting tuning parameters
## Fitting cp = 0.00389 on full training set

tree.fit

## CART
##
## 433 samples
##  10 predictor
##   3 classes: 'C', 'F', 'G'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 347, 347, 345, 347, 346
## Resampling results across tuning parameters:
##
##   cp           EconomicProfit
##   0.000000000  0.1525731
##   0.003891051  0.1525731
##   0.013618677  0.1709260
##   0.015564202  0.1663535
##   0.019455253  0.1709381
##   0.021400778  0.1754836
##   0.027237354  0.1859487
##   0.033073930  0.1859487
##   0.073929961  0.2068258
##   0.194552529  0.2403872
```

```
##
## EconomicProfit was used to select the optimal model using the smallest
value.
## The final value used for the model was cp = 0.003891051.

rpart.plot(tree.fit$finalModel)
```



```
treeProb = predict(tree.fit, newdata=testing, type="prob")
treePred = rep("F", nrow(testing))
treePred[which(treeProb[,1] > threshold)] = "C"
treePred[which(treeProb[,3] > threshold)] = "G"
treePred = as.factor(treePred)
CM = confusionMatrix(treePred, testing$Pos)$table
cost = sum(as.vector(CM)*cost.unit)
cost

## [1] 19.5
```

The cost is high but it does not matters because we use this technique to interpret. We can see how it works: When the number of offensive rebounds is bigger than 1.2, and the number of assistance are less than 1.4, the algorithm classify the observation like a center. However, if the player have more of 1.4 assistance per game, only classify like center if the player has more than 0.76 blocks per game. We can see all the classification criteria following the scheme.

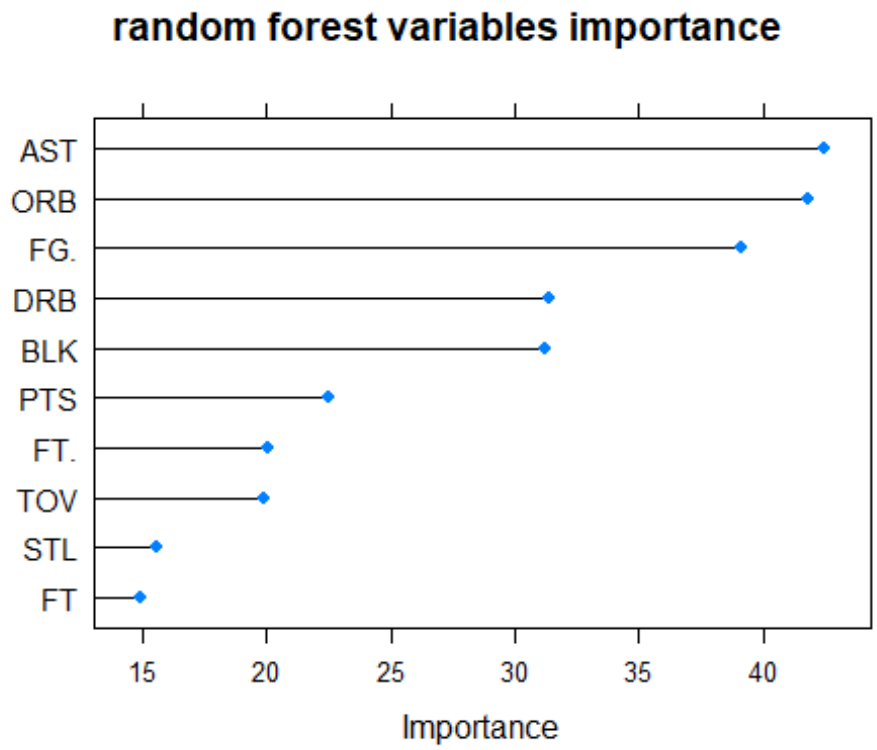
Now I will use random forest. I will try, 2,3,4,5 and 6 variables by chance per node. Futhermore, we know that is more likely to be a guard or forward than center. We

reflect that in the cutoff.I will use 10000 trees because is the bigger number with witch my computer could work quickly.

```
rf.train <- train(Pos ~.,
                  method = "rf",
                  data = training,
                  preProcess = c("center", "scale"),
                  ntree = 10000,
                  cutoff=c(2/9,7/18,7/18),
                  tuneGrid = expand.grid(mtry=c(2,3,4,5,6)),
                  metric = "EconomicProfit",
                  maximize = F,
                  trControl = ctrl)
```

Is better to use only two variables by chance.

```
rf_imp <- varImp(rf.train, scale = F)
plot(rf_imp,main="random forest variables importance", scales = list(y =
list(cex = .95)))
```



The variable importance is also similar.

```
rfPred = predict(rf.train, testing)
confusionMatrix(rfPred,testing$Pos)

## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction  C  F  G
##          C 17  7  0
##          F  6 28  4
##          G  0  8 37
##
## Overall Statistics
##
##                Accuracy : 0.7664
##                  95% CI : (0.6747, 0.8427)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : 1.874e-14
##
##                   Kappa : 0.6394
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.7391   0.6512   0.9024
## Specificity            0.9167   0.8438   0.8788
## Pos Pred Value         0.7083   0.7368   0.8222
## Neg Pred Value         0.9277   0.7826   0.9355
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1589   0.2617   0.3458
## Detection Prevalence   0.2243   0.3551   0.4206
## Balanced Accuracy      0.8279   0.7475   0.8906
```

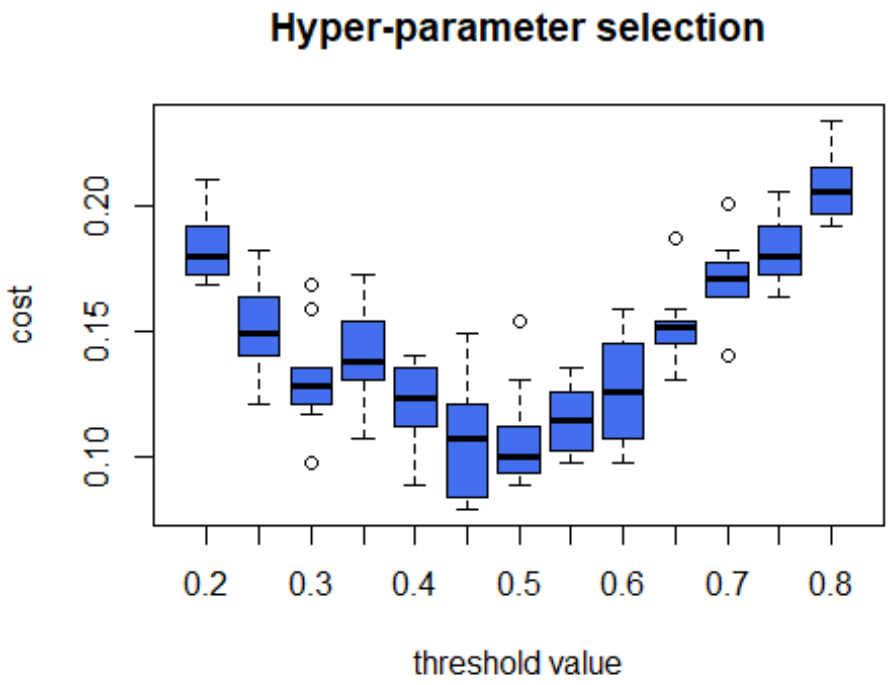
```
cost=sum(cost.unit*confusionMatrix(rfPred,testing$Pos)$table)
cost
```

```
## [1] 12.5
```

The accuracy,kappa and the cost are the worst. Now I will search the best theshold

```
j=0;
cost.i = matrix(NA, nrow = 10, ncol = 13)
for (threshold in seq(0.2,0.8,0.05)){

  j <- j + 1
  cat(j)
  for(i in 1:10){


    d <- createDataPartition(my_data$Pos, p = 0.8, list = FALSE)

    train<-my_data[d,]
    test <-my_data[-d,]

    rffit <- train(Pos ~ .,
```

```
                    method = "rf",
                    data = train,
                preProcess = c("center", "scale"),
                 ntree = 1000,
                cutoff=c(2/9,7/18,7/18),
                  tuneGrid = expand.grid(mtry=2),
                  metric = "EconomicProfit",
                  maximize = F,
                  trControl = ctrl)
    rfProb = predict(rffit, test, type="prob")

    rfPred = rep("F", nrow(test))
    rfPred[which(rfProb[,1] > threshold)] = "C"
    rfPred[which(rfProb[,3] > threshold)] = "G"
    rfPred = factor(rfPred)

    CM = confusionMatrix(rfPred, test$Pos)$table

    cost.i[i,j] <- sum(relative.cost*CM)/nrow(test) # unitary cost

  }
}

boxplot(cost.i, main = "Hyper-parameter selection",
        ylab = "cost",
        xlab = "threshold value",names = seq(0.2,0.8,0.05),col="royalblue
2")
```



The best threshold is 0.5

```
threshold = 0.5
svmProb = predict(svmFit, newdata=testing, type="prob")
svmPred = rep("F", nrow(testing))
svmPred[which(svmProb[,1] > threshold)] = "C"
svmPred[which(svmProb[,3] > threshold)] = "G"
svmPred = as.factor(svmPred)
CM = confusionMatrix(svmPred, testing$Pos)$table
CM
```

```
##            Reference
## Prediction  C  F  G
##          C 16  2  0
##          F  7 34  7
##          G  0  7 34
```

```
print(" cost")
```

```
## [1] " cost"
```

```
sum(relative.cost*CM)
```

```
## [1] 11.5
```

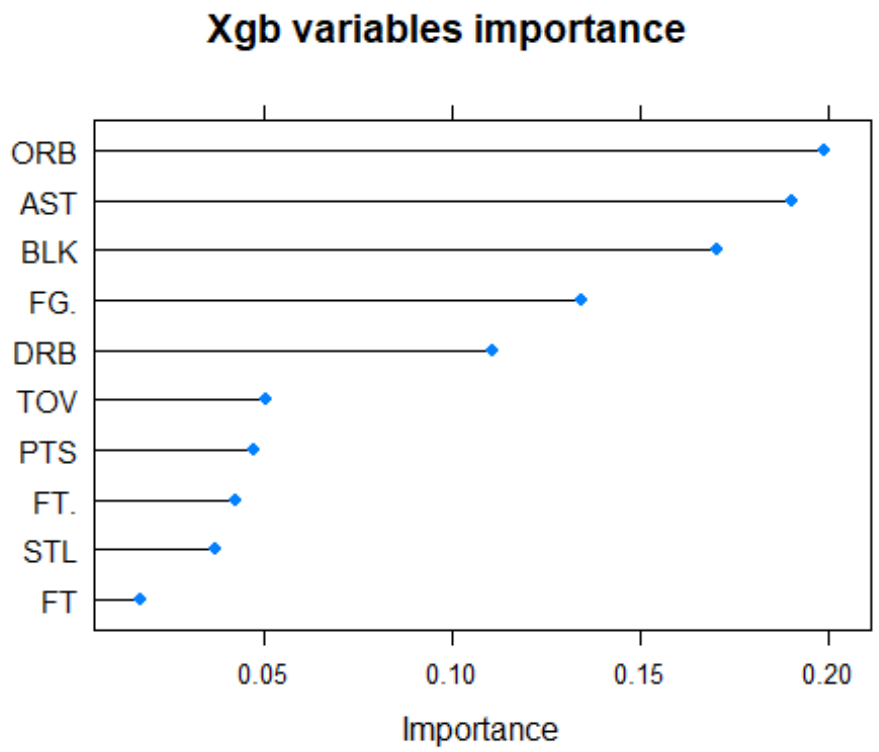
This is the worse tool for our problem because have the biggest cost

Then , I will use gradient boosting. This should have a great perfomance. The problem would be the computational cost. However, I select a range of parameters enough small to my computer can deal with it.

```
xgb_grid = expand.grid(
  nrounds = c(500,1000),
  eta = seq(from = 0.04, to = 0.16, by = 0.04),
  max_depth = c(2, 4, 6),
  gamma = 1,
  colsample_bytree = seq(from = 0.1, to = 0.5, by = 0.5),
  min_child_weight = seq(from = 0.5, to = 5, by = 1),
  subsample = 1
)

xgb.train = train(Pos ~ .,  data=training,
                  trControl = ctrl,
                  metric="EconomicProfit",
                  maximize = F,
                  tuneGrid = xgb_grid,
                  preProcess = c("center", "scale"),
                  method = "xgbTree"
)
```

The most importance variables are:

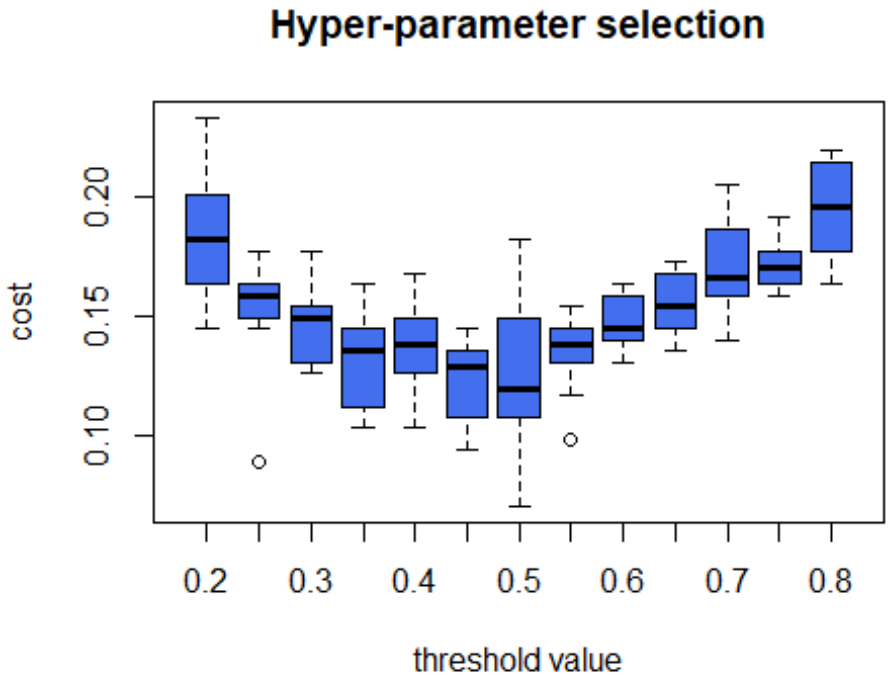
```
xgb_imp <- varImp(xgb.train, scale = F)
plot(xgb_imp,main="Xgb variables importance", scales = list(y = list(cex
= .95)))
```



In this case, FG are most importance than blocks, unlike the others tools. with the best parameters, I will search the best threshold

```
xgb_grid = expand.grid(
  nrounds = 500,
  eta = 0.12,
  max_depth = 2,
  gamma = 1,
  colsample_bytree = 0.1,
  min_child_weight = 1.5,
  subsample = 1
)
j=0;
cost.i = matrix(NA, nrow = 10, ncol = 13)
for (threshold in seq(0.2,0.8,0.05)){

  j <- j + 1
  cat(j)
  for(i in 1:10){


    d <- createDataPartition(my_data$Pos, p = 0.8, list = FALSE)

    train<-my_data[d,]
    test <-my_data[-d,]

    xgb.train = train(Pos ~ .,  data=train,
```

```
                    trControl = ctrl,
                    metric="EconomicProfit",
                    maximize = F,
                    tuneGrid = xgb_grid,
                    preProcess = c("center", "scale"),
                    method = "xgbTree"
)
    xgbProb = predict(xgb.train, test, type="prob")

    xgbPred = rep("F", nrow(test))
    xgbPred[which(xgbProb[,1] > threshold)] = "C"
    xgbPred[which(xgbProb[,3] > threshold)] = "G"
    xgbPred = factor(xgbPred)

    CM = confusionMatrix(xgbPred, test$Pos)$table

    cost.i[i,j] <- sum(relative.cost*CM)/nrow(test) # unitary cost

  }
}

boxplot(cost.i, main = "Hyper-parameter selection",
        ylab = "cost",
        xlab = "threshold value",names = seq(0.2,0.8,0.05),col="royalblue
2")
```



```
threshold = 0.45
xgbProb = predict(xgb.train, newdata=testing, type="prob")
xgbPred = rep("F", nrow(testing))
xgbPred[which(xgbProb[,1] > threshold)] = "C"
xgbPred[which(xgbProb[,3] > threshold)] = "G"
xgbPred = as.factor(xgbPred)
confusionMatrix(xgbPred, testing$Pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 15  2  0
##          F  8 34  5
##          G  0  7 36
##
## Overall Statistics
##
##                Accuracy : 0.7944
##                  95% CI : (0.7054, 0.8664)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6764
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.6522   0.7907   0.8780
## Specificity            0.9762   0.7969   0.8939
## Pos Pred Value         0.8824   0.7234   0.8372
## Neg Pred Value         0.9111   0.8500   0.9219
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1402   0.3178   0.3364
## Detection Prevalence   0.1589   0.4393   0.4019
## Balanced Accuracy      0.8142   0.7938   0.8860
```

```
CM = confusionMatrix(xgbPred, testing$Pos)$table
print(" cost")
```

```
## [1] " cost"
```

```
sum(relative.cost*CM)
```

```
## [1] 11
```

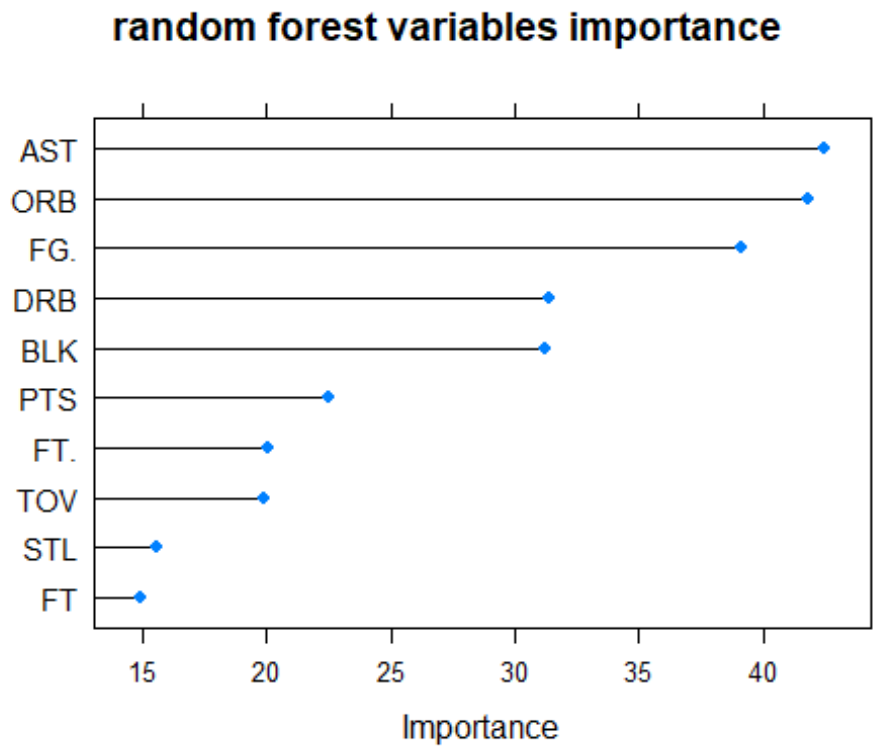
The results with this tools are the best by far. The accucary and kappa are amazing and the cost is very low.

The last method to use is neural network. We will use shallow learning beacuse I will use only one hidden layer

```
nn.train <- train(Pos ~.,
                  method = "nnet",
                  data = training,
                  preProcess = c("center", "scale"),
                  MaxNWts = 1000,
                  maxit = 100,
                  tuneGrid = expand.grid(size=seq(from = 2, to = 10, by =
20),decay= seq(from = 0.001, to = 0.01, by = 0.001)),
                  metric = "EconomicProfit",
                  maximize = F,
                  trControl = ctrl)
```

We get the best performance with size=2, decay=0.002. Now we will see the variable importance

```
rf_imp <- varImp(rf.train, scale = F)
plot(rf_imp,main="random forest variables importance", scales = list(y =
list(cex = .95)))
```



For this tool, the second most importance variable is asitnce and FG is better than blocks. Now I check the accuracy

```
nnPred = predict(nn.train, testing)
confusionMatrix(nnPred,testing$Pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 19  3  1
##          F  3 32  5
##          G  1  8 35
##
## Overall Statistics
##
##                Accuracy : 0.8037
##                  95% CI : (0.7158, 0.8742)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6962
##
##  Mcnemar's Test P-Value : 0.875
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.8261   0.7442   0.8537
## Specificity            0.9524   0.8750   0.8636
## Pos Pred Value         0.8261   0.8000   0.7955
## Neg Pred Value         0.9524   0.8358   0.9048
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1776   0.2991   0.3271
## Detection Prevalence   0.2150   0.3738   0.4112
## Balanced Accuracy      0.8892   0.8096   0.8586
```

```
cost=sum(cost.unit*confusionMatrix(nnPred,testing$Pos)$table)
cost
```

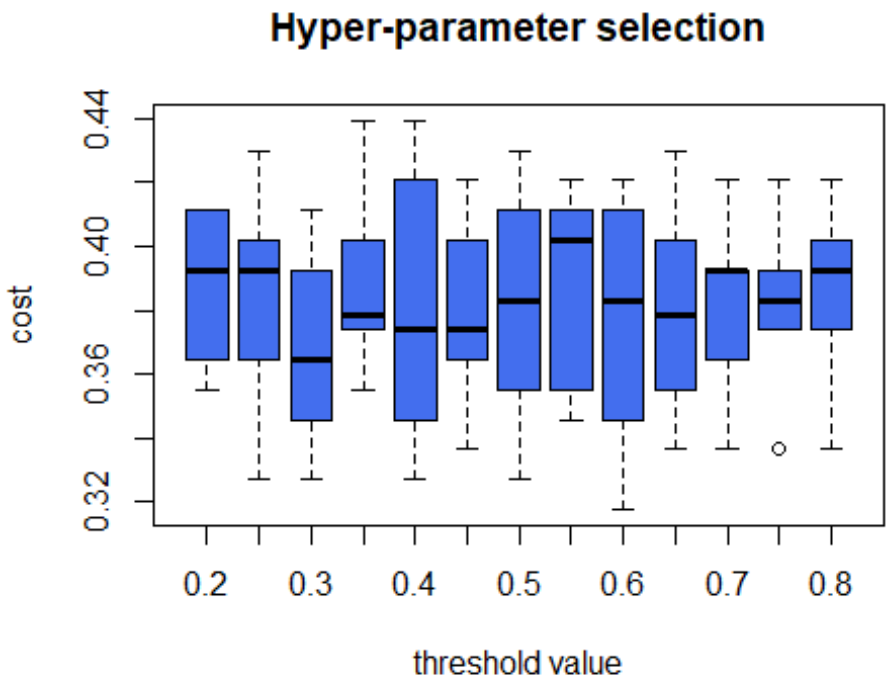
```
## [1] 11.5
```

Accuracy, kappa and cost are similar than svm and knn

Now I search the best threshold

```
j=0;
cost.i = matrix(NA, nrow = 10, ncol = 13)
for (threshold in seq(0.2,0.8,0.05)){

  j <- j + 1
  cat(j)
  for(i in 1:10){


    d <- createDataPartition(my_data$Pos, p = 0.8, list = FALSE)
```

```
    train<-my_data[d,]
    test <-my_data[-d,]

nn.train <- train(Pos ~.,
                  method = "nnet",
                  data = train,
                  preProcess = c("center", "scale"),
                  MaxNWts = 1000,
                  maxit = 100,
                  tuneGrid = expand.grid(size=2,decay= 0.02),
                  metric = "EconomicProfit",
                  maximize = F,
                  trControl = ctrl)
    nnProb = predict(nn.train, test, type="prob")

    nnPred = rep("F", nrow(test))
    nnPred[which(nnProb[,1] > threshold)] = "C"
    nnPred[which(nnProb[,3] > threshold)] = "G"
    nnPred = factor(nnPred)

    CM = confusionMatrix(xgbPred, test$Pos)$table

    cost.i[i,j] <- sum(relative.cost*CM)/nrow(test) # unitary cost

  }
}

boxplot(cost.i, main = "Hyper-parameter selection",
        ylab = "cost",
        xlab = "threshold value",names = seq(0.2,0.8,0.05),col="royalblue
2")
```



The best threshold is 0.35

```
threshold = 0.35
nnProb = predict(nn.train, newdata=testing, type="prob")
nnPred = rep("F", nrow(testing))
nnPred[which(nnProb[,1] > threshold)] = "C"
nnPred[which(nnProb[,3] > threshold)] = "G"
nnPred = as.factor(nnPred)
confusionMatrix(nnPred, testing$Pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 21  3  0
##          F  1 32  2
##          G  1  8 39
##
## Overall Statistics
##
##                Accuracy : 0.8598
##                  95% CI : (0.7793, 0.9194)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7838
##
##  Mcnemar's Test P-Value : 0.1328
```

```
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.9130   0.7442   0.9512
## Specificity            0.9643   0.9531   0.8636
## Pos Pred Value         0.8750   0.9143   0.8125
## Neg Pred Value         0.9759   0.8472   0.9661
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1963   0.2991   0.3645
## Detection Prevalence   0.2243   0.3271   0.4486
## Balanced Accuracy      0.9387   0.8487   0.9074
```

```
CM = confusionMatrix(nnPred, testing$Pos)$table
print(" cost")
```

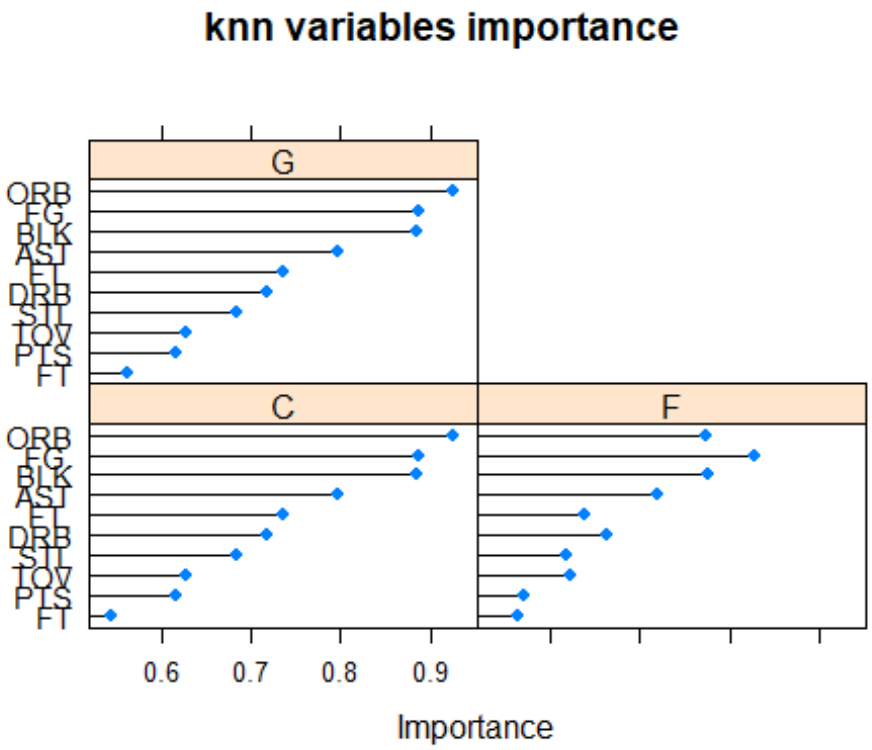
```
## [1] " cost"
```

```
sum(relative.cost*CM)
```

```
## [1] 8
```

The cost is the highest and accuracy and kappa are not good enough. Probably we need more hidden layers so now i will check deep learning with three hidden layers and one hidden_dropout in ourder to avoid over fitting. The problem could be that we dont have enough data to feed our nets

```
dnn.train <- train(Pos ~.,
                   method = "dnn",
                   data = training,
                   preProcess = c("center", "scale"),
                   numepochs = 20, # number of iterations on the whole tra
ining set
                   tuneGrid = expand.grid(layer1 = 2:10,
                                          layer2 = 2:10,
                                          layer3 = 2:10,
                                          hidden_dropout = 1,
                                          visible_dropout = 0),
                   metric = "EconomicProfit",
                   maximize = F,
                   trControl = ctrl)

dnn_imp <- varImp(dnn.train, scale = F)
plot(dnn_imp,main="knn variables importance", scales = list(y = list(cex
= .95)))
```



The variable importance is similar to knn and svm

```
DnnPred = predict(dnn.train, testing)
confusionMatrix(DnnPred,testing$Pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C  0  0  0
##          F 23 43 41
##          G  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.4019
##                  95% CI : (0.3082, 0.5011)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : 0.5367
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity             0.000   1.0000   0.0000
```

```
## Specificity             1.000   0.0000   1.0000
## Pos Pred Value            NaN   0.4019      NaN
## Neg Pred Value          0.785      NaN   0.6168
## Prevalence              0.215   0.4019   0.3832
## Detection Rate          0.000   0.4019   0.0000
## Detection Prevalence    0.000   1.0000   0.0000
## Balanced Accuracy       0.500   0.5000   0.5000
```

```
cost=sum(cost.unit*confusionMatrix(DnnPred,testing$Pos)$table)
cost
```

```
## [1] 32
```

The accuracy, kappa and cost are like a naive classifier. I discard this method because the problem is that we need a big data data set in order to make that deep learning works enough good.

My best model by far is xglm, so I decide to build a ensemble mode with knn, svm , and shallow learning and see how it works and if it can get the results of

```
ensemble.prob1 = (knnProb[,1]+ svmProb[,1] + rfProb[,1]+nnProb[,1])/4
ensemble.prob3 = (knnProb[,3]+ svmProb[,3] + rfProb[,3] + nnProb[,3])/4
totalPred = rep("F", nrow(test))
totalPred[which(ensemble.prob1 > 0.5)] = "C"
totalPred[which(ensemble.prob3 > 0.5)] = "G"
totalPred = factor(totalPred)
confusionMatrix(totalPred, testing$Pos)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  C  F  G
##          C 15  2  0
##          F  8 37  7
##          G  0  4 34
##
## Overall Statistics
##
##                Accuracy : 0.8037
##                  95% CI : (0.7158, 0.8742)
##     No Information Rate : 0.4019
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6907
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: C Class: F Class: G
## Sensitivity            0.6522   0.8605   0.8293
```

```
## Specificity            0.9762   0.7656   0.9394
## Pos Pred Value         0.8824   0.7115   0.8947
## Neg Pred Value         0.9111   0.8909   0.8986
## Prevalence             0.2150   0.4019   0.3832
## Detection Rate         0.1402   0.3458   0.3178
## Detection Prevalence   0.1589   0.4860   0.3551
## Balanced Accuracy      0.8142   0.8130   0.8843
```

```
CM = confusionMatrix(totalPred, testing$Pos)$table
print(" cost")
```

```
## [1] " cost"
```

```
sum(relative.cost*CM)
```

```
## [1] 10.5
```

The accuracy, cost and kappa are good but doesnt have the peromace of xgb_grid

```
```