The Betlab Project

Tobias Diederich

Saturday, October 31, 2015

Abstract

The target of the Betlab project is to find the best model predicting the outcome (HomeVictory, Draw, VisitorsVictory) of football matches. The best predicted probability simulates the highest percentage profit in relation to booky odds (Value Betting). The fundamental predictors are the aggregated marketprices of the participating players (parsed on transfermarkt.de).

I show the high potential of my approach, even if it is not yet practicable.

Data

The match, team and player data are collected from Transfermarkt. Booky odds are collected from Sfstats. The parsers are written in Java and are not part of this paper.

Relevant data will be of germanies 1. Bundesliga from season 2005-2006 to 2014-2015. I included the english premier league too, but focus here for simplicity on BL1.

```
source(file = 'production/loadData.R', echo = FALSE, encoding = 'UTF-8')
toMatchday <- 34
seasons <- c('2005-2006', '2006-2007', '2007-2008', '2008-2009', '2009-2010',
             '2010-2011', '2011-2012', '2012-2013', '2013-2014', '2014-2015')
leagues <- c('BL1')
trainingRaw <- loadTrainingData(toMatchday = toMatchday, seasons = seasons, leagues = leagues)
matches <- trainingRaw$matches
odds <- trainingRaw$odds
stats <- trainingRaw$stats
```

Datasets

matches -> contains all matches
odds -> contains booky odds and probabilies for all matches
stats -> an observation contains information for one player in one match

Here is a brief exploration of the raw data:

```
describe(dplyr:::select(matches, goalsHome, goalsVisitors, matchResult))
```

```
## dplyr:::select(matches, goalsHome, goalsVisitors, matchResult)
##
##  3  Variables      3060  Observations
## ---------------------------------------------------------------------------
## goalsHome
##        n missing  unique    Info    Mean     .05     .10     .25     .50
##     3060       0      10    0.94   1.619       0       0       1       1
##      .75     .90     .95
##        2       3       4
```

```
##
##             0   1   2   3   4  5  6 7 8 9
## Frequency 647 957 779 403 182 63 20 6 2 1
## %          21  31  25  13   6  2  1 0 0 0
## ---------------------------------------------------------------------------
## goalsVisitors
##       n missing  unique    Info    Mean
##    3060       0       9    0.92   1.255
##
##             0    1   2   3   4  5  6 7 8
## Frequency 928 1055 647 285 104 28 11 1 1
## %          30   34  21   9   3  1  0 0 0
## ---------------------------------------------------------------------------
## matchResult
##       n missing  unique
##    3060       0       3
##
## VisitorsVictory (901, 29%), Draw (779, 25%)
## HomeVictory (1380, 45%)
## ---------------------------------------------------------------------------
```

```r
describe(dplyr:::select(odds, HomeVictory, VisitorsVictory, Draw))
```

```
## dplyr:::select(odds, HomeVictory, VisitorsVictory, Draw)
##
##  3  Variables      6859  Observations
## ---------------------------------------------------------------------------
## HomeVictory
##       n missing  unique    Info    Mean     .05     .10     .25     .50
##    6859       0     623       1  0.4829  0.1681  0.2257  0.3663  0.4762
##     .75     .90     .95
##  0.6024  0.7407  0.8065
##
## lowest : 0.03774 0.05882 0.06329 0.06557 0.06835
## highest: 0.90090 0.90909 0.91743 0.92593 0.94340
## ---------------------------------------------------------------------------
## VisitorsVictory
##       n missing  unique    Info    Mean     .05     .10     .25     .50
##    6859       0     863       1  0.3157 0.08764 0.11587 0.19608 0.29499
##     .75     .90     .95
## 0.40000 0.56497 0.64103
##
## lowest : 0.03150 0.03968 0.04000 0.04274 0.04310
## highest: 0.82645 0.83333 0.84034 0.85470 0.86957
## ---------------------------------------------------------------------------
## Draw
##       n missing  unique    Info    Mean     .05     .10     .25     .50
##    6859       0     389       1  0.2766  0.1757  0.2078  0.2604  0.2941
##     .75     .90     .95
##  0.3077  0.3125  0.3155
##
## lowest : 0.08598 0.09174 0.09434 0.10060 0.10194
## highest: 0.32895 0.33003 0.33113 0.33223 0.33445
## ---------------------------------------------------------------------------
```

```r
describe(dplyr:::select(stats, fitPrice, position, playerAssignment, formation))
```

```
## dplyr:::select(stats, fitPrice, position, playerAssignment, formation)
##
##  4  Variables      109552  Observations
## ---------------------------------------------------------------------------
## fitPrice
##        n  missing   unique     Info     Mean      .05      .10      .25
##   109219      333      137        1  4021380   275000   500000  1000000
##      .50      .75      .90      .95
##  2400000  4500000  9000000 13000000
##
## lowest :        0    25000    40000    50000    75000
## highest: 42000000 45000000 48000000 50000000 55000000
## ---------------------------------------------------------------------------
## position
##       n missing  unique
##  109552       0      13
##
## Torwart (12209, 11%), Innenverteidiger (18817, 17%)
## Linker Verteidiger (8595, 8%)
## Rechter Verteidiger (8155, 7%)
## Defensives Mittelfeld (12953, 12%)
## Zentrales Mittelfeld (5907, 5%)
## Linkes Mittelfeld (3522, 3%)
## Rechtes Mittelfeld (3379, 3%)
## Offensives Mittelfeld (7366, 7%)
## Haengende Spitze (2421, 2%)
## Mittelstuermer (15189, 14%)
## Linksaussen (5562, 5%), Rechtsaussen (5477, 5%)
## ---------------------------------------------------------------------------
## playerAssignment
##       n missing  unique
##  109552       0       4
##
## AUSGEWECHSELT (16932, 15%), BENCH (25235, 23%)
## DURCHGESPIELT (50385, 46%)
## EINGEWECHSELT (17000, 16%)
## ---------------------------------------------------------------------------
## formation
##       n missing  unique
##  108379    1173      25
##
## lowest : 3-1-4-2     3-3-3-1     3-4-2-1     3-4-3       3-4-3 flach
## highest: 4-5-1       4-5-1 flach 5-3-2       5-4-1       5-4-1 flach
## ---------------------------------------------------------------------------
```

Feature Engineering

The features I extract are the marketprices of participating players aggregated by team (Home, Visitors), grouped position (TW, DEF, MID, OFF) and aggregation method (min, max, avg, sum). My first analysis is on including the players who played the whole match, who got substituted from bench and to bench. This is not practicable because I have an unrealistic information advantage in comparison to the booky. I stick to this approach at first, because I don't expect the advantage as big and I want to show the potential of this approach.

```
source('./production/positionFeatureExtraction.R',
       echo = FALSE, encoding = 'UTF-8')
### Preparation
#[1] "Torwart"               "Innenverteidiger"      "Linker Verteidiger"    "Rechter Verteidiger"   "D
#[6] "Zentrales Mittelfeld"  "Linkes Mittelfeld"     "Rechtes Mittelfeld"    "Offensives Mittelfeld" "H
#[11] "Mittelstuermer"        "Linksaussen"           "Rechtsaussen"
positions <- c('tw', 'def', 'def', 'def', 'mid', 'mid', 'mid', 'mid', 'off', 'off', 'off', 'off', 'off'
lineupAssignments <- c('DURCHGESPIELT', 'AUSGEWECHSELT', 'EINGEWECHSELT')
#benchFuncts = c('max', 'avg')
featuredMatches <- extractMatchResultFeatures(playerStats = stats,
                                              matches = matches,
                                              priceAssignedPositions = positions,
                                              functs = c('min', 'max', 'avg', 'sum'),
                                              lineupAssignments)
# Selects the relevant predictors and outcomes
filteredFeatureMatches <- filterFeaturedMatches(featuredMatches)
```

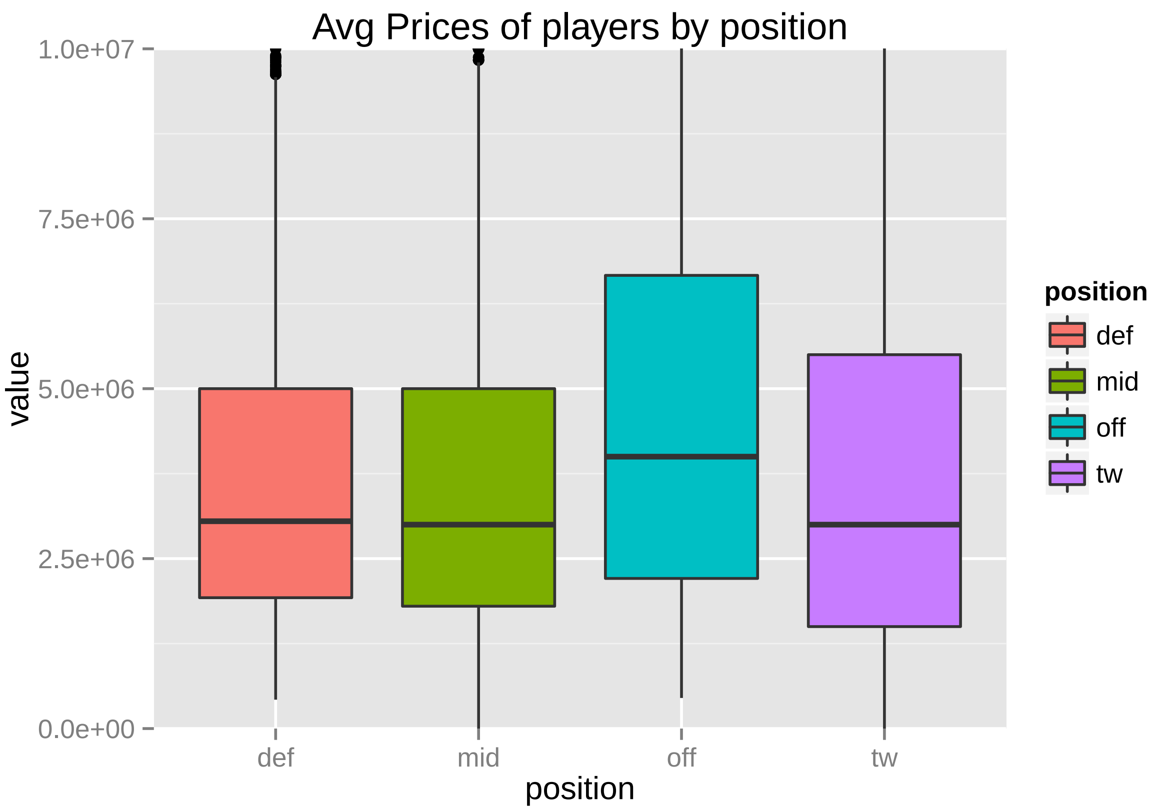
```
explMatches <- dplyr:::select(filteredFeatureMatches, -matchId, -matchResult, -goalsHome, - goalsVisito
# Features:
colnames(explMatches)
```

```
##  [1] "tw_Price_Home_avg"      "def_Price_Home_min"
##  [3] "def_Price_Home_max"     "def_Price_Home_avg"
##  [5] "def_Price_Home_sum"     "mid_Price_Home_min"
##  [7] "mid_Price_Home_max"     "mid_Price_Home_avg"
##  [9] "mid_Price_Home_sum"     "off_Price_Home_min"
## [11] "off_Price_Home_max"     "off_Price_Home_avg"
## [13] "off_Price_Home_sum"     "tw_Price_Visitors_avg"
## [15] "def_Price_Visitors_min" "def_Price_Visitors_max"
## [17] "def_Price_Visitors_avg" "def_Price_Visitors_sum"
## [19] "mid_Price_Visitors_min" "mid_Price_Visitors_max"
## [21] "mid_Price_Visitors_avg" "mid_Price_Visitors_sum"
## [23] "off_Price_Visitors_min" "off_Price_Visitors_max"
## [25] "off_Price_Visitors_avg" "off_Price_Visitors_sum"
```

```
library(magrittr)
library(tidyr)
explGathered <- explMatches %>% gather(feature, value)

getGroupStr <- function(feature, group) {
    charList <- strsplit(as.character(feature), '_')
    charFrame <- data.frame(do.call(rbind, charList))
    if(group == 'func') {
        return(charFrame[, 4])
    } else if(group == 'pos') {
        return(charFrame[, 1])
    } else {
        return(NA)
    }
```

```
}
groupedMatches <- mutate(explGathered, funct = factor(getGroupStr(feature, 'func')),
                         position = factor(getGroupStr(feature, 'pos')))
avgPlot <- ggplot(filter(groupedMatches, funct == 'avg'), aes(x = position, y = value, fill = position))
    geom_boxplot() +
    ggtitle('Avg Prices of players by position') +
    coord_cartesian(ylim = c(0, 10000000))
avgPlot
```



No surprise here, offensive players are the most expensive.

Model fitting

Now I fit several models with several different configurations. The fitting process is devided into two parts. First different models are tuned for optimizing common metrics like accuracy, kappa, ROC. Second, the best model configurations are used to resample again to predict the target performance metric 'percentage gain'. The first step is necessary by now, because to discard it I have to integrate this custom percentage metric into carets fitting process.

Configuration of the fitting process

I use 5-fold cross validation for calculation time performance first, later 10-fold would be probably better.

```r
source(file = './production/models.R',
       echo = FALSE, encoding = 'UTF-8')
seed <- 16450
cvContr = trainControl(method = 'cv', number = 5, classProbs = TRUE,
                       summaryFunction = multiClassSummary)

resultFormula <- as.formula('matchResult ~ . -matchId -goalsHome -goalsVisitors')
```

POLR model

First I fit a linear POLR model for simplicity and because it regards the outcome as an ordered factor.

```r
set.seed(seed)
polrModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'polr',
                   preProcess = c('center', 'scale'),
                   trControl = cvContr)

# Resampled Training Performance
dplyr:::select(polrModel$results, Accuracy, Kappa, Sensitivity, Specificity, ROC, logLoss)
```

```
##   Accuracy     Kappa Sensitivity Specificity       ROC   logLoss
## 1 0.522226 0.1911971   0.4373544   0.7281924 0.6668224 0.5816057
```

```r
testPred <- predict(polrModel, filteredFeatureMatches)
confMatrix <- confusionMatrix(testPred, reference = filteredFeatureMatches$matchResult)
# Training Performance without resampling
confMatrix$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.5254902 0.1972013
```

The gap between training and resampled training accuracy and kappa is small, thus I will use more complex models with lower bias.

Random forest model

RF models give good predictions as well as reducing variance because it decreases tree correlations by introducing additional randomnes. I set the number of trees to 1000, because it is a good initial value. Accuracy is used as performance metric.

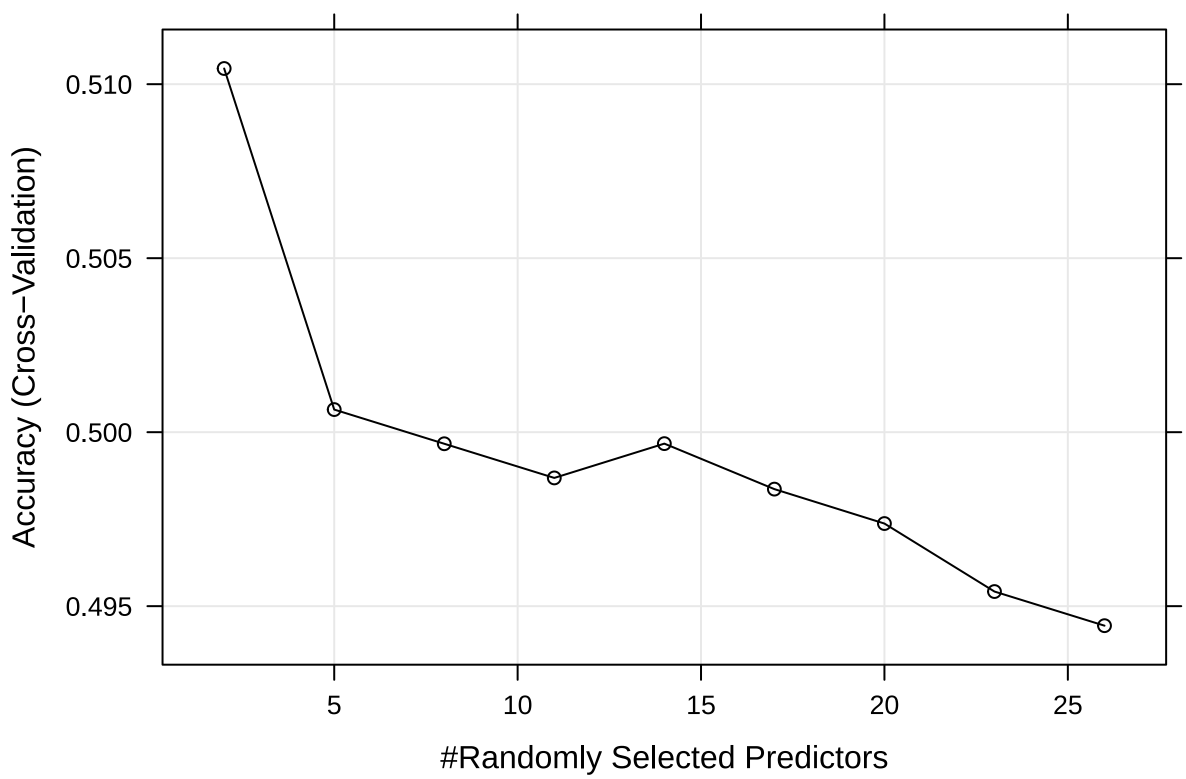
```r
rfNtree <- 1000
rfGrid <- expand.grid(.mtry = seq(2, ncol(filteredFeatureMatches) - 4, by = 3))
set.seed(seed)
rfModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'rf',
                 trControl = cvContr, ntree = rfNtree, importance = TRUE, tuneGrid = rfGrid,
                 metric = 'Accuracy')
bestRf <- dplyr:::select(rfModel$results, mtry, Accuracy, Kappa, Sensitivity, Specificity, ROC, logLoss)
bestRf <- dplyr:::filter(bestRf, mtry == rfModel$bestTune[1, 1])
# Best model performance
bestRf
```

```
##   mtry  Accuracy     Kappa Sensitivity Specificity       ROC   logLoss
## 1    2 0.5104506 0.1900084   0.4391117    0.729351 0.6424116 0.5869149
```

```
# Training Performance without resampling
testPred <- predict(rfModel, filteredFeatureMatches)
confMatrix <- confusionMatrix(testPred, reference = filteredFeatureMatches$matchResult)
confMatrix$overall[1:2]
```

```
## Accuracy    Kappa
##        1        1
```

```
# Plotting the resampling profile
trellis.par.set(caretTheme())
plot(rfModel)
```



The poor resampled training performance and the perfect prediction on the not resampled training data indicates massive overfitting, thus the model has very high variance and will be discarded.

Gradiant Boosing

I expect better accuracy from gbm models, because it is more complex in case of tuning.

```
gbmGrid <- expand.grid(.interaction.depth = c(1, 5, 9),
                       .n.trees = (1:10)*100, .shrinkage = c(.05, .1), .n.minobsinnode = c(15, 20))
```

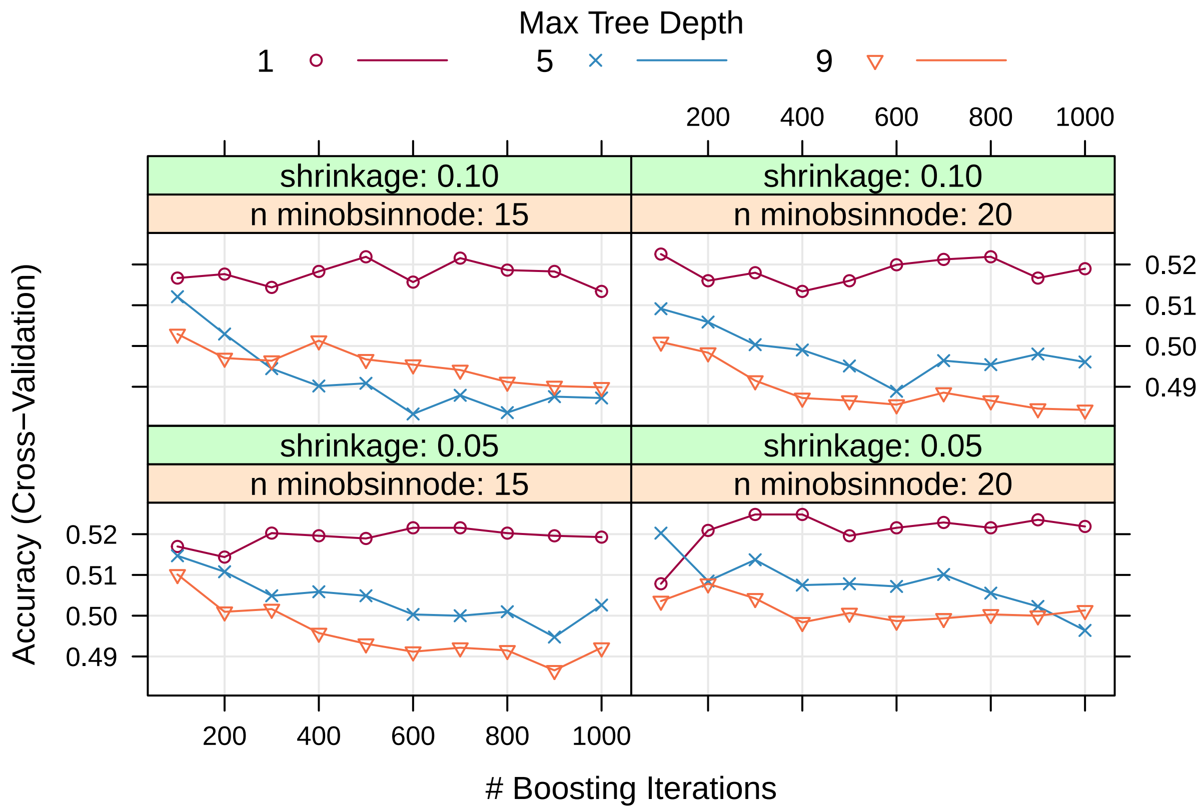
```r
set.seed(seed)
gbmModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'gbm',
                  trControl = cvContr, verbose = FALSE,
                  tuneGrid = gbmGrid, distribution = 'multinomial',
                  metric = 'Accuracy')
bestGbm <- dplyr:::select(gbmModel$results, shrinkage, interaction.depth, n.minobsinnode,
                          n.trees, Accuracy, Kappa, Sensitivity, Specificity, ROC, logLoss)
bestGbm <- dplyr:::filter(bestGbm, shrinkage == gbmModel$bestTune[1, 'shrinkage'],
                          interaction.depth == gbmModel$bestTune[1, 'interaction.depth'],
                          n.minobsinnode == gbmModel$bestTune[1, 'n.minobsinnode'],
                          n.trees == gbmModel$bestTune[1, 'n.trees'])
# Best model performance
bestGbm
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 1      0.05                 1             20     400 0.5248361 0.2184071
##   Sensitivity Specificity       ROC   logLoss
## 1   0.4579463   0.7384993 0.6568442 0.5805858
```

```r
# Training Performance without resampling
testPred <- predict(gbmModel, filteredFeatureMatches)
confMatrix <- confusionMatrix(testPred, reference = filteredFeatureMatches$matchResult)
confMatrix$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.5797386 0.3078711
```

```r
# Plotting the resampling profile
trellis.par.set(caretTheme())
plot(gbmModel)
```



Fine tuning: The graphs show that I should focus on a small number of interaction.depth. The gbm model is slightly more overfitted than POLR.

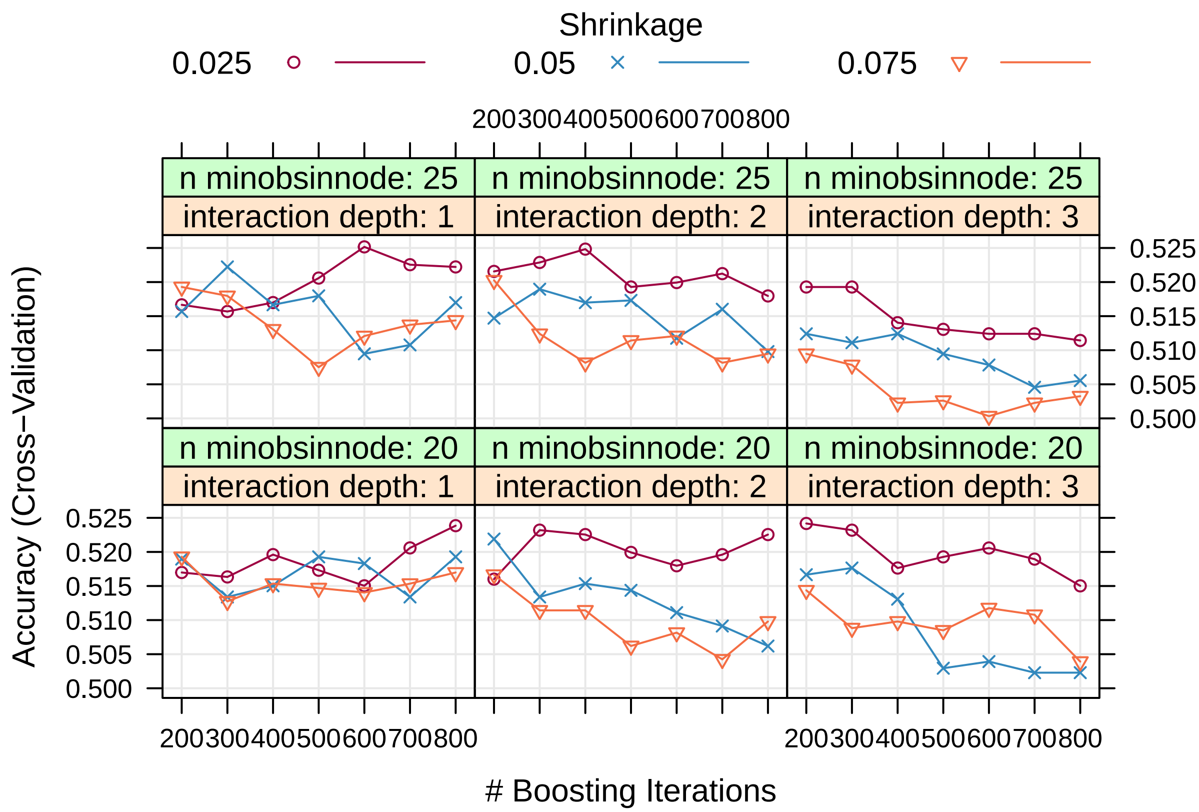
```
gbmGrid2 <- expand.grid(.interaction.depth = c(1:3),
                        .n.trees = (2:8)*100, .shrinkage = c(0.025, .05, .075), .n.minobsinnode = c(20, 
set.seed(seed)
gbmModel2 <- train(form = resultFormula, data = filteredFeatureMatches, method = 'gbm',
                   trControl = cvContr, verbose = FALSE,
                   tuneGrid = gbmGrid2, distribution = 'multinomial',
                   metric = 'Accuracy')
bestGbm2 <- dplyr:::select(gbmModel2$results, shrinkage, interaction.depth, n.minobsinnode,
                           n.trees, Accuracy, Kappa, Sensitivity, Specificity, ROC, logLoss)
bestGbm2 <- dplyr:::filter(bestGbm2, shrinkage == gbmModel2$bestTune[1, 'shrinkage'],
                           interaction.depth == gbmModel2$bestTune[1, 'interaction.depth'],
                           n.minobsinnode == gbmModel2$bestTune[1, 'n.minobsinnode'],
                           n.trees == gbmModel2$bestTune[1, 'n.trees'])
```

So, we found the best model:

```
# Best model performance
bestGbm2
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 1     0.025                 1             25     600 0.5251629 0.2158091
##   Sensitivity Specificity       ROC logLoss
## 1   0.4555455   0.7374405 0.6551336 0.58021
```

```r
# Plotting the resampling profile
trellis.par.set(caretTheme())
plot(gbmModel2)
```




```r
# Setting best configuration for second step predictions
bestGbmConfig <- gbmModel2$bestTune
```

Extreme Gradient Boosting

Tuning Parameters: - nrounds (# Boosting Iterations) - max_depth (Max Tree Depth), - eta (Shrinkage) - gamma (Minimum Loss Reduction) - colsample_bytree (Subsample Ratio of Columns) - min_child_weight (Minimum Sum of Instance Weight)

```r
library(xgboost)
extrBoostGrid <- expand.grid(nrounds = (1:15)*20,
                             eta = c(.025, .05, .075, .1, .15),
                             max_depth = c(1, 2, 3))
set.seed(seed)
extrBoostModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'xgbTree',
                      trControl = cvContr, tuneGrid = extrBoostGrid, metric = 'Accuracy',
                      objective = 'multi:softprob', num_class = 3,
                      colsample_bytree = 1, min_child_weight = 1)

bestExtrBoost <- dplyr:::select(extrBoostModel$results, nrounds, max_depth, eta,
```

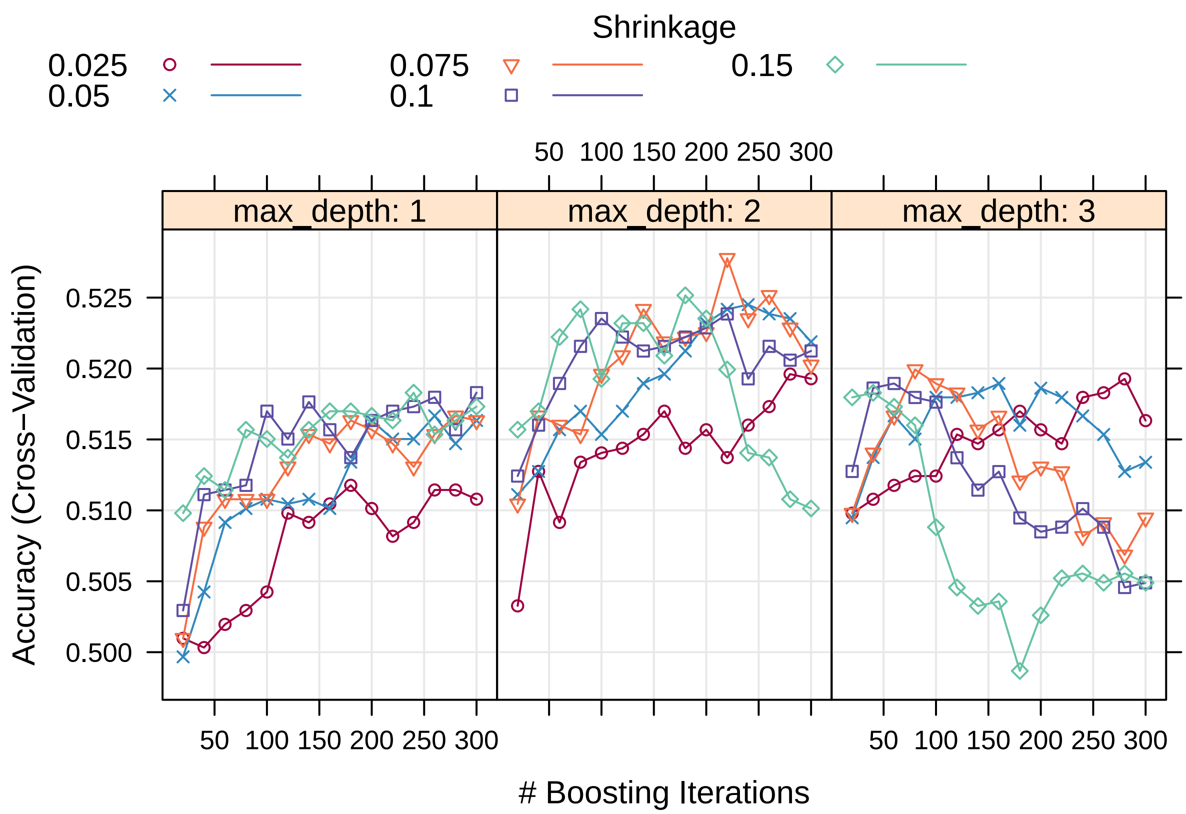
```r
                                 Accuracy, Kappa, Sensitivity, Specificity, ROC, logLoss)
bestExtrBoost <- dplyr:::filter(bestExtrBoost, nrounds == extrBoostModel$bestTune[1, 'nrounds'],
                                 max_depth == extrBoostModel$bestTune[1, 'max_depth'],
                                 eta == extrBoostModel$bestTune[1, 'eta'])
bestExtrBoost
```

```
##   nrounds max_depth   eta  Accuracy     Kappa Sensitivity Specificity
## 1     220         2 0.075 0.5277709 0.2188507   0.4574186   0.7382728
##         ROC   logLoss
## 1 0.6497426 0.5837869
```

```r
# Training Performance without resampling
testPred <- predict(extrBoostModel, filteredFeatureMatches)
confMatrix <- confusionMatrix(testPred, reference = filteredFeatureMatches$matchResult)
confMatrix$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.5993464 0.3349345
```

```r
# Plotting the resampling profile
trellis.par.set(caretTheme())
plot(extrBoostModel)
```



```
bestExtrBoostConfig <- extrBoostModel$bestTune
```

Second Level Prediction

Target is to maximize percent profit in comparison to booky odds. First have a look at the booky performance:

```
source(file = './evaluatePrediction.R',
       echo = FALSE, encoding = 'UTF-8')
bookySummary <- getBookyPerformance(odds = odds, matches = matches)
bookySummary
```

```
##          Accuracy            Kappa     AccuracyLower     AccuracyUpper
##      5.075163e-01     1.676227e-01      4.896392e-01      5.253790e-01
##    AccuracyPValue    McnemarPValue       Sensitivity       Specificity
##      2.041197e-10    2.087837e-192      4.252474e-01      7.205178e-01
##    Pos_Pred_Value   Neg_Pred_Value    Detection_Rate Balanced_Accuracy
##      4.965847e-01     7.519211e-01      1.691721e-01      5.728826e-01
##               ROC          logLoss
##      6.424531e-01     5.863675e-01
```

Now I do resampling again to predict the percentage profit.

```
folds <- 10
noneContr <- trainControl(method = 'none', classProbs = TRUE)

# Split Data
splits <- splitMatches(matchesToSplit = filteredFeatureMatches, splitBy = filteredFeatureMatches$matchRe
                       testingMatches = filteredFeatureMatches, folds = folds, seed = seed)

# Resampling
allPredictions <- data.frame()
for(i in 1:folds) {
    actTrain <- splits[[i]]$train
    actTest <- splits[[i]]$test

    set.seed(seed)
    actPolrFit <- caret:::train(form = resultFormula, data = actTrain,
                         method = 'polr', preProcess = c('center', 'scale'),
                         trControl = cvContr)
    set.seed(seed)
    actGbmFit <- caret:::train(form = resultFormula, data = actTrain,
                                method = 'gbm', trControl = noneContr,
                                verbose = FALSE, distribution = 'multinomial',
                                tuneGrid = bestGbmConfig)

    actExtrBoostFit <- caret:::train(form = resultFormula, data = actTrain, method = 'xgbTree',
                            trControl = noneContr, tuneGrid = bestExtrBoostConfig,
                            objective = 'multi:softprob', num_class = 3,
                            colsample_bytree = 1, min_child_weight = 1)

    models <- list('POLR' = actPolrFit, 'GBM' = actGbmFit, 'EXTRBOOST' = actExtrBoostFit)
```

```
    preds <- predict(models, actTest, type = 'prob')
    preds <- do.call(cbind.data.frame, preds)
    preds <- cbind('matchId' = actTest$matchId,
                   'matchResult' = actTest$matchResult,
                   preds)

    if(nrow(allPredictions) == 0) {
        allPredictions <- preds
    } else {
        allPredictions <- rbind(allPredictions, preds)
    }
}
```

Evaluate Predictions for percentage profit

```
source(file = './evaluatePrediction.R',
       echo = FALSE, encoding = 'UTF-8')
allPredictions <- arrange(allPredictions, matchId)
polrPreds <- dplyr:::select(allPredictions, matchId, matchResult,
                            'HomeVictory' = POLR.HomeVictory,
                            'VisitorsVictory' = POLR.VisitorsVictory,
                            'Draw' = POLR.Draw)
polrEvals <- evaluatePrediction(prediction = polrPreds,
                                comparison = odds,
                                probRatioToBet = 1.1, stake = 1)
printEvaluation(polrEvals)
```

```
## [1] "Stake: 1669"
## [1] "Gain: 373.97"
## [1] "Gain [%]: 22.4068304373877"
## [1] "Value Diff [%]: -1.94418777865"
## [1] "Accuracy [%]: 52.1895424836601"
## [1] "Booky Accuracy [%]: 50.6535947712418"
```

```
gbmPreds <- dplyr:::select(allPredictions, matchId, matchResult,
                           'HomeVictory' = GBM.HomeVictory,
                           'VisitorsVictory' = GBM.VisitorsVictory,
                           'Draw' = GBM.Draw)
gbmEvals <- evaluatePrediction(prediction = gbmPreds,
                               comparison = odds,
                               probRatioToBet = 1.1, stake = 1)
printEvaluation(gbmEvals)
```

```
## [1] "Stake: 1678"
## [1] "Gain: 314.67"
## [1] "Gain [%]: 18.7526817640048"
## [1] "Value Diff [%]: -1.8980004815565"
## [1] "Accuracy [%]: 52.0261437908497"
## [1] "Booky Accuracy [%]: 50.6535947712418"
```

```r
extrBoostPreds <- dplyr::select(allPredictions, matchId, matchResult,
                          'HomeVictory' = EXTRBOOST.HomeVictory,
                          'VisitorsVictory' = EXTRBOOST.VisitorsVictory,
                          'Draw' = EXTRBOOST.Draw)
extrBoostEvals <- evaluatePrediction(prediction = extrBoostPreds,
                              comparison = odds,
                              probRatioToBet = 1.1, stake = 1)
printEvaluation(extrBoostEvals)
```

```
## [1] "Stake: 1662"
## [1] "Gain: 314.46"
## [1] "Gain [%]: 18.9205776173285"
## [1] "Value Diff [%]: -2.14005024164136"
## [1] "Accuracy [%]: 51.1764705882353"
## [1] "Booky Accuracy [%]: 50.6535947712418"
```

Prediction with just the starting lineup

This time only the players in the starting lineup are considered.

TODO