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')
featuredMatches <- extractMatchResultFeatures(playerStats = stats,
                                              matches = matches,
                                              priceAssignedPositions = positions,
                                              functs = c('min', 'max', 'avg', 'sum'),
                                              lineupAssignments)
# Select the relevant predictors
filteredFeatureMatches <- filterFeaturedMatches(featuredMatches)
```

Used Features:

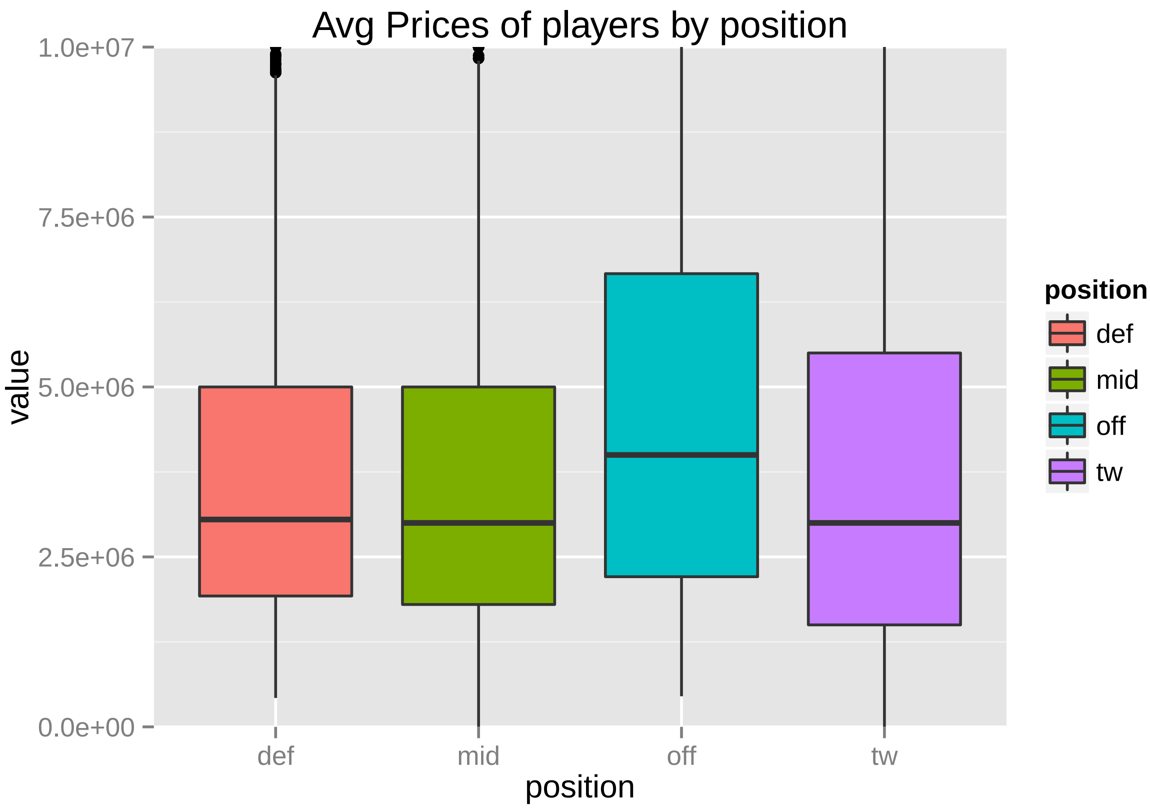
```
explMatches <- dplyr:::select(filteredFeatureMatches, -matchId, -matchResult, -goalsHome, - goalsVisito
colnames(explMatches)
```

```
##  [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 and tuning

The target of the model tuning process is to find a model which maximizes the simulated profit [%]. This is a custom metric and implemented in the function betMetricsSummary. This function is integrated in the caret resampling and tuning process (caret is so great!!).

Model algorithms shown here are: POLR, Gradient boosting and extreme gradient boosting (tree). I tryed different models like: random forests, support vector machienes, C5.0, neural nets and knn. These method performances were bad in comparison, so they are not shown here.

Remarks on custom metrics:
GainPerc is a custom metric, which calculates the simulated profit against Booky odds, if the model would have been applyed consistently. A bet is simulated, if the predicted probability of an outcome divided through the booky probability > 1.1. ValueDiffPerc is a custom metric, which calculates the mean difference in percentage points of the predicted probability and the booky probability of the real outcome.

Configuration of the fitting process

I use 5-fold cross validation first. 10-fold would be probably better, but calculation would take much longer. A static seed is set to make the results reproduceable.

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

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.

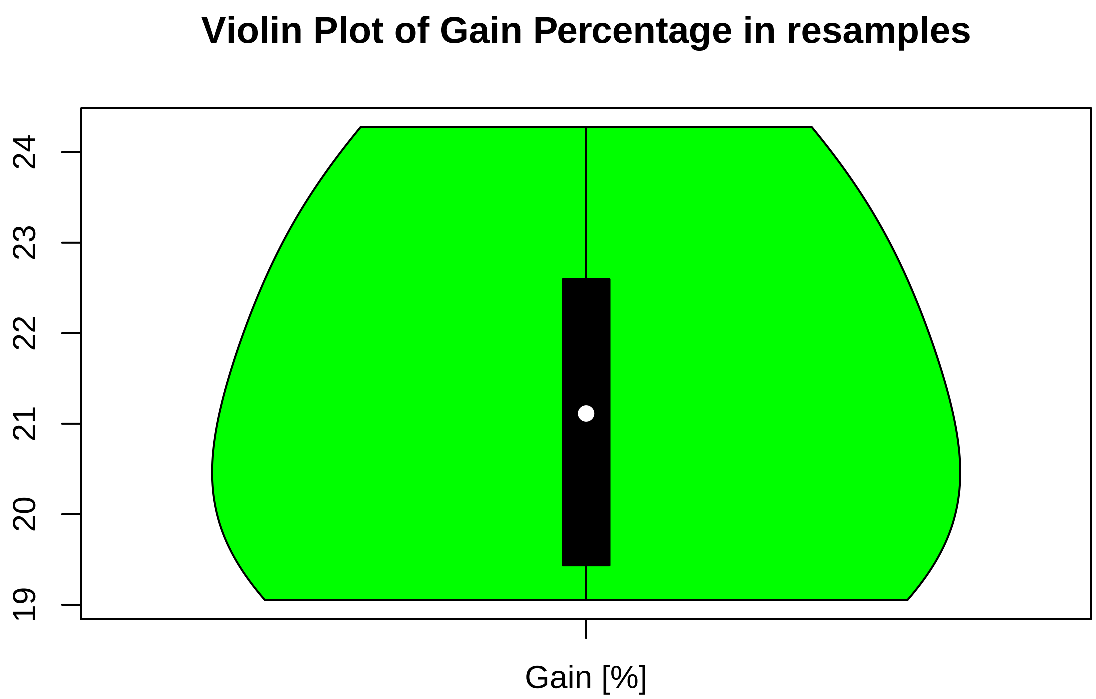
```
set.seed(seed)
polrModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'polr',
                   preProcess = c('center', 'scale'), trControl = customCvContr)
polrModel
```

```
## Ordered Logistic or Probit Regression
##
## 3060 samples
##   29 predictor
##    3 classes: 'VisitorsVictory', 'Draw', 'HomeVictory'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2448, 2448, 2447, 2449, 2448
## Resampling results
##
##   Accuracy  Kappa      BookyAccuracy  BookyKappa  GainPerc  ValueDiffPerc
##   0.522226  0.1911971  0.5075206      0.1676443   21.29483  -1.902575
##   Accuracy SD  Kappa SD  BookyAccuracy SD  BookyKappa SD  GainPerc SD
##   0.008875975  0.013606  0.008687744       0.01428432     2.185257
##   ValueDiffPerc SD
##   0.3159565
##
##
```

```
confusionMatrix(polrModel)
```

```
## Cross-Validated (5 fold) Confusion Matrix
##
## (entries are percentages of table totals)
##
##                  Reference
## Prediction        VisitorsVictory Draw HomeVictory
##   VisitorsVictory            13.1  7.6         5.9
##   Draw                        0.0  0.0         0.0
##   HomeVictory                16.4 17.8        39.2
```

```r
# POLR Resampling exploration
vioplot(polrModel$resample$GainPerc, names = 'Gain [%]', col = 'green')
title('Violin Plot of Gain Percentage in resamples')
```




```r
summary(polrModel$resample$GainPerc)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   19.05   19.44   21.11   21.29   22.60   24.28
```

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

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

A profit of 21% is huge!! Model Accuracy and kappa are both much better than the booky metrics.

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

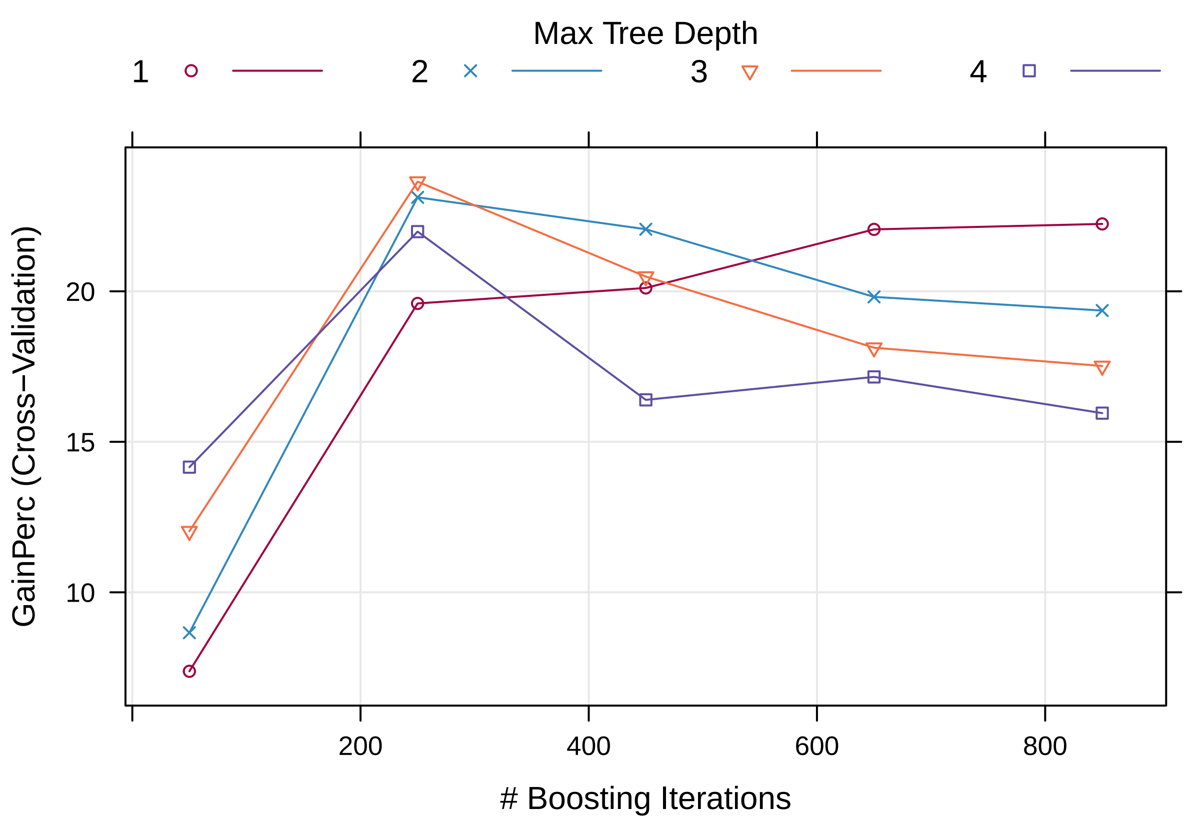
TODO Explore correlations of metrics like GainPerc ~ I(Accuracy - BookyAccuracy) + I(Kappa - BookyKappa) GainPerc ~ ValueDiffPerc

Gradient Boosting

```
gbmGrid <- expand.grid(.interaction.depth = c(1, 2, 3, 4),
                       .n.trees = seq(50, 1000, by = 200),
                       .shrinkage = c(.05),
                       .n.minobsinnode = c(10))
set.seed(seed)
gbmModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'gbm',
                  trControl = customCvContr, verbose = FALSE,
                  tuneGrid = gbmGrid, distribution = 'multinomial',
                  metric = 'GainPerc')
# Best Tune
gbmModel$results[as.integer(rownames(gbmModel$results)) == as.integer(rownames(gbmModel$bestTune)), ]
```

```
##    shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 12      0.05                 3             10     250 0.5189436 0.2176345
##    BookyAccuracy BookyKappa GainPerc ValueDiffPerc AccuracySD    KappaSD
## 12     0.5075206  0.1676443 23.64336    -0.8587156 0.01447617 0.02313875
##    BookyAccuracySD BookyKappaSD GainPercSD ValueDiffPercSD
## 12     0.008687744   0.01428432   5.800862        0.437723
```

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

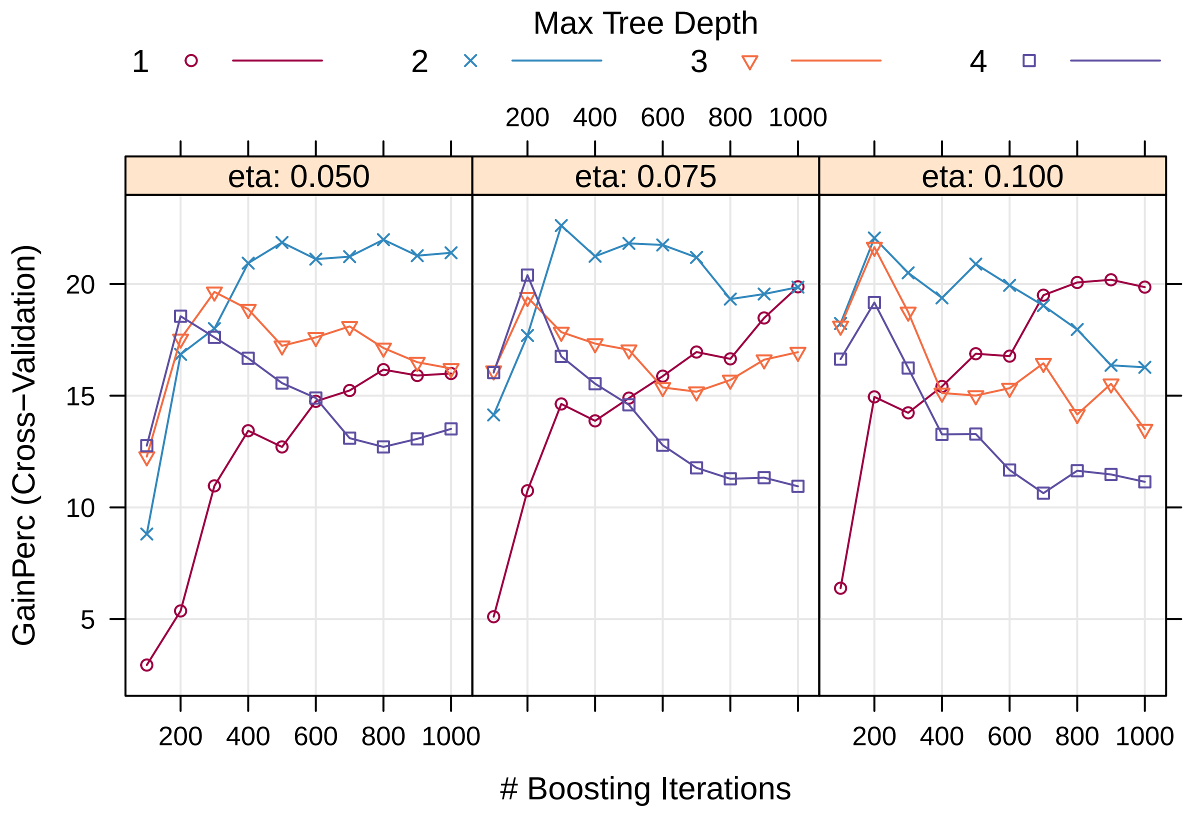


There is a slide increase in model performance (GainPerc and ValueDiffPerc). Here is space for some further fine tuning, but I encounter memory problems with gbm, so I try extreme gradient boosting.

Extreme Gradiant Boosing

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)

```
extrBoostGrid <- expand.grid(nrounds = (1:10)*100,
                             eta = c(.05, .075, .1),
                             max_depth = 1:4)
set.seed(seed)
extrBoostModel <- train(form = resultFormula, data = filteredFeatureMatches, method = 'xgbTree',
                        trControl = customCvContr, tuneGrid = extrBoostGrid, metric = 'GainPerc',
                        objective = 'multi:softprob', num_class = 3,
                        colsample_bytree = 1, min_child_weight = 1)
trellis.par.set(caretTheme())
plot(extrBoostModel)
```




```
extrBoostModel$results[as.integer(rownames(extrBoostModel$results)) == as.integer(rownames(extrBoostMode
```

```
##      eta max_depth nrounds  Accuracy     Kappa BookyAccuracy BookyKappa
## 53 0.075         2     300 0.5202556 0.2100217     0.5075206  0.1676443
```

```
##    GainPerc ValueDiffPerc AccuracySD    KappaSD BookyAccuracySD
## 53 22.61473     -1.807299 0.00977219 0.01714942     0.008687744
##    BookyKappaSD GainPercSD ValueDiffPercSD
## 53   0.01428432   5.322177       0.2925239
```

```
# Non-resampled training performance
testPred <- predict(extrBoostModel, filteredFeatureMatches)
confMatrix <- confusionMatrix(testPred, reference = filteredFeatureMatches$matchResult)
confMatrix$overall[1:2]
```

```
##  Accuracy     Kappa
## 0.6241830 0.3802949
```

Predictions with just the starting lineup

I don't trust the huge expected profit of ca. 25%. So, I reevaluate my predictions without an information advantage. This means I integrate just the players in the starting lineup (They are known before a match). Additionally I extract features for the players on bench.

```
# Integrate starting lineup and additional features for players on bench
realLineupAssignments <- c('DURCHGESPIELT', 'AUSGEWECHSELT')
benchFuncts <- c('max', 'avg')
realFeaturedMatches <- extractMatchResultFeatures(playerStats = stats,
                                                  matches = matches,
                                                  priceAssignedPositions = positions,
                                                  functs = c('min', 'max', 'avg', 'sum'),
                                                  realLineupAssignments,
                                                  benchFuncts = benchFuncts)

realFilteredFeatureMatches <- filterFeaturedMatches(realFeaturedMatches)
```

Used Features:

```
explMatches <- dplyr:::select(realFilteredFeatureMatches, -matchId, -matchResult, -goalsHome, - goalsVis
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"
## [27] "def_Bench_Home_max"     "def_Bench_Home_avg"
## [29] "mid_Bench_Home_max"     "mid_Bench_Home_avg"
## [31] "off_Bench_Home_max"     "off_Bench_Home_avg"
```

```
## [33] "def_Bench_Visitors_max" "def_Bench_Visitors_avg"
## [35] "mid_Bench_Visitors_max" "mid_Bench_Visitors_avg"
## [37] "off_Bench_Visitors_max" "off_Bench_Visitors_avg"
```

Tuning Models for a realistic and applicable approach

I tune POLR, GBM and extreme gradient boosting models.

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

```
## Ordered Logistic or Probit Regression
##
## 3060 samples
##   41 predictor
##    3 classes: 'VisitorsVictory', 'Draw', 'HomeVictory'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2448, 2448, 2447, 2449, 2448
## Resampling results
##
##   Accuracy   Kappa     BookyAccuracy  BookyKappa  GainPerc
##   0.4937919  0.132722  0.5075206      0.1676443   -0.6268271
##   ValueDiffPerc  Accuracy SD  Kappa SD    BookyAccuracy SD  BookyKappa SD
##   -3.685424      0.009676879  0.01825154  0.008687744       0.01428432
##   GainPerc SD  ValueDiffPerc SD
##   8.747762     0.1588822
##
##
```

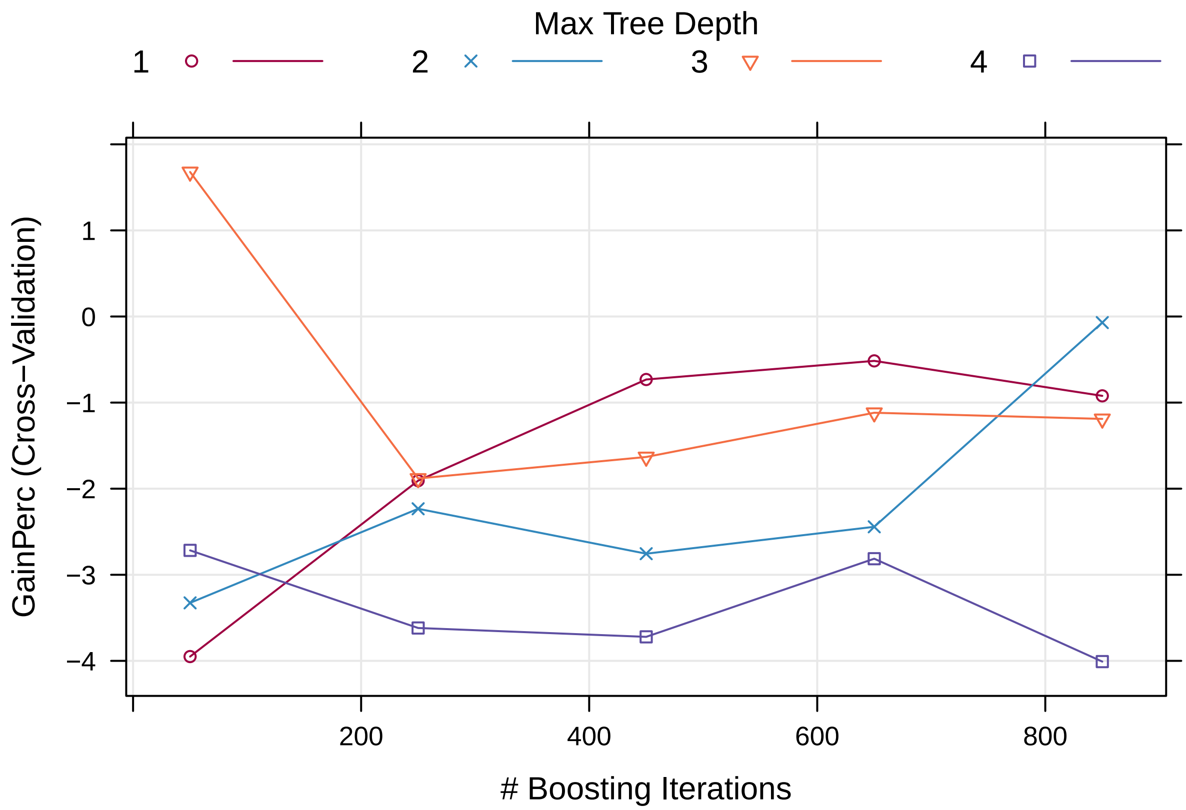
Wow, what a huge difference. The expected small information advantage, is big in regard of the expected profit. Lets see if other models perform better.

GBM

```r
gbmGrid <- expand.grid(.interaction.depth = c(1, 2, 3, 4),
                       .n.trees = seq(50, 1000, by = 200),
                       .shrinkage = c(.05),
                       .n.minobsinnode = c(10))
set.seed(seed)
gbmModel <- train(form = resultFormula, data = realFilteredFeatureMatches, method = 'gbm',
                  trControl = customCvContr, verbose = FALSE,
                  tuneGrid = gbmGrid, distribution = 'multinomial',
                  metric = 'GainPerc')
# Best Tune
gbmModel$results[as.integer(rownames(gbmModel$results)) == as.integer(rownames(gbmModel$bestTune)), ]
```

```
##    shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 11      0.05                 3             10      50 0.4947766 0.1528949
##    BookyAccuracy BookyKappa GainPerc ValueDiffPerc  AccuracySD    KappaSD
## 11     0.5075206  0.1676443 1.679064     -3.629954 0.008330952 0.01498598
##    BookyAccuracySD BookyKappaSD GainPercSD ValueDiffPercSD
## 11     0.008687744   0.01428432   12.38585       0.2202334
```

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



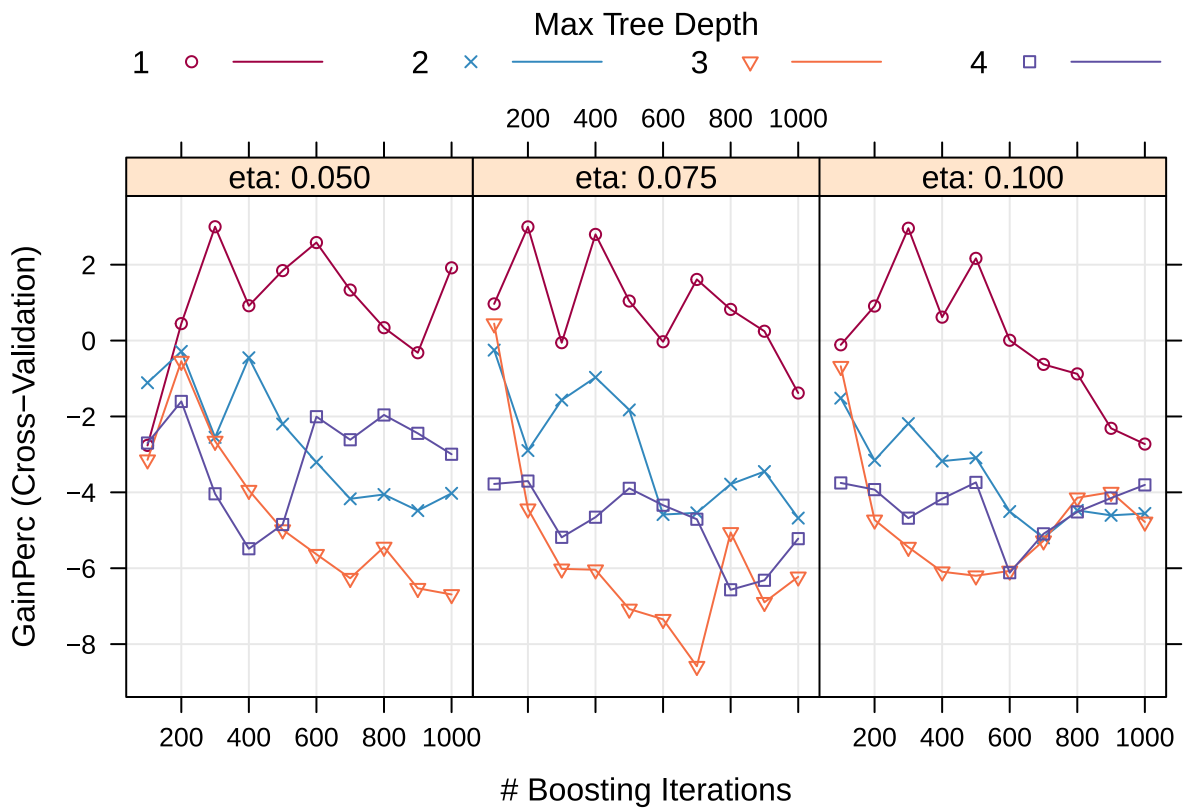
Extreme Boosting

```
extrBoostGrid <- expand.grid(nrounds = (1:10)*100,
                             eta = c(.05, .075, .1),
                             max_depth = 1:4)
set.seed(seed)
extrBoostModel <- train(form = resultFormula, data = realFilteredFeatureMatches, method = 'xgbTree',
                        trControl = customCvContr, tuneGrid = extrBoostGrid, metric = 'GainPerc',
                        objective = 'multi:softprob', num_class = 3,
                        colsample_bytree = 1, min_child_weight = 1)
extrBoostModel$results[as.integer(rownames(extrBoostModel$results)) == as.integer(rownames(extrBoostMode
```

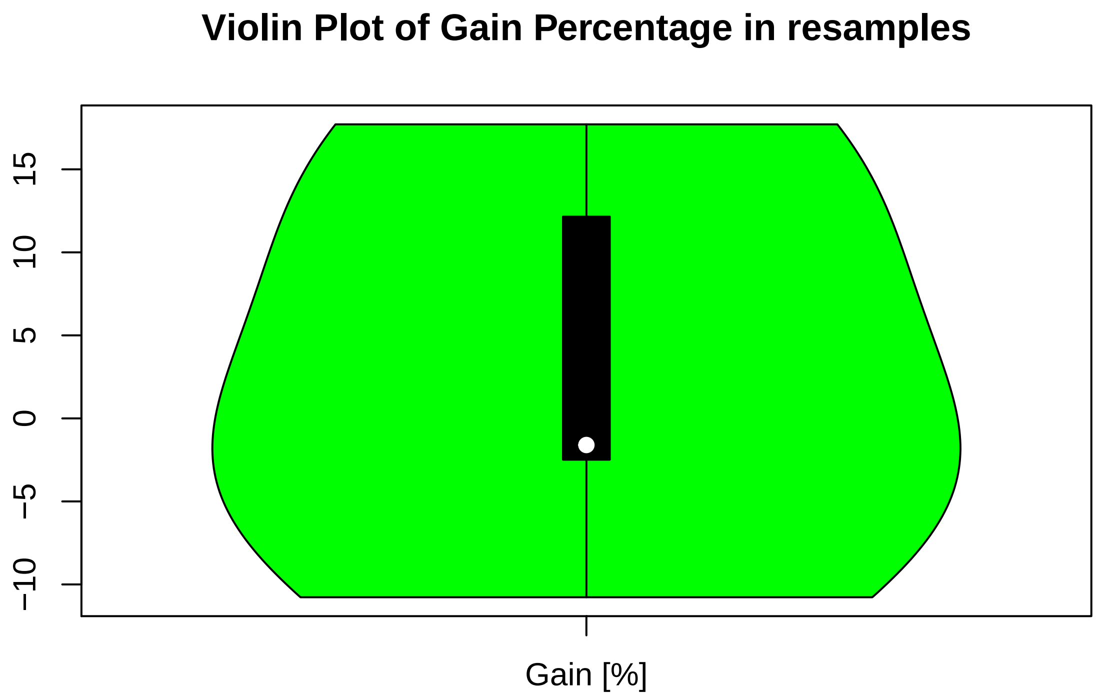
```
##    eta max_depth nrounds  Accuracy     Kappa BookyAccuracy BookyKappa
```

```
## 3 0.05         1     300 0.5029417 0.1615302     0.5075206  0.1676443
##   GainPerc ValueDiffPerc  AccuracySD    KappaSD BookyAccuracySD
## 3 2.999922     -3.677135 0.006898669 0.01344465     0.008687744
##   BookyKappaSD GainPercSD ValueDiffPercSD
## 3   0.01428432   11.63051       0.1414778
```

```
trellis.par.set(caretTheme())
plot(extrBoostModel)
```




```
vioplot(extrBoostModel$resample$GainPerc, names = 'Gain [%]', col = 'green')
title('Violin Plot of Gain Percentage in resamples')
```




```r
summary(extrBoostModel$resample$GainPerc)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -10.770  -2.487  -1.603   3.000  12.150  17.710
```

Results and Forecast

The results are not overwhelming. The final model is very unstable concerning the profit (high SD). The minimum requirement for applying the model on real bets is a positive GainPerc over all folds and at least > 5%. To achieve this, I think it is necessary to integrate new features. Imaginable features could describe team formations or distance of the visiting team to travel. A disadvantage of featured based on marketprices of players is, that the marketprices are created twice a year, so they describe the long term quality of a player. A way to integrate short term form predictors of players is to engineer features based on the kicker.de grade associated to players in past matches.