Compulsory Exercise 2: Predicting Football Matches

Espen Bjørge Urheim Lavrans Seierstad Simen Nesland

21 april, 2023

Abstract

In this project we have predicted the outcome of football matches. Our goal was to use methods from statistical learning, find out which one yields the best result, and compare it to simply guessing match result. More precisely, we used Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), K-nearest neighbours with 10-fold cross validation, random forests and Support Vector Machines (SVM) on the European Soccer Dataset, which contains data on several thousand matches and a lot of statistics about each match, such as FIFA player ratings and odds. In general, we found that using these methods yielded a better result than simply guessing - LDA and SVM scored the highest with up to 57% prediction rate, compared to simply guessing 'home win' each time, which would yield a rate of 50% correct.

Introduction

In this paper we aim to analyse the European Soccer Dataset by Hugo Mathien from Kaggle (https://www.kaggle.com/datasets/hugomathien/soccer). This dataset contains data on several thousand football matches, such as the results, goal types, possession, corners, fouls, odds etc., as well as data on player attributes and ratings from FIFA, and different team attributes from the 2008-2016 seasons. Our idea is to use this data in an attempt to classify match results (i.e. home win, draw, away win) based on total team rating, i.e. the sum of FIFA ratings of players on each team, and odds.

The purpose of this is to predict match results, and find which model out of LDA, QDA, KNN, random forests and SVM perform the best for this exact job. Then, we want to find out if this model performs better than simply performing the best guess each time, i.e. guessing 'home win'.

Descriptive data analysis/statistics

First we import the dataset.

```r
## connect to db
con <- dbConnect(drv = RSQLite::SQLite(), dbname = "database.sqlite")
tables <- dbListTables(con)
tables <- tables[tables != "sqlite_sequence"]
lDataFrames <- vector("list", length = length(tables))
for (i in seq(along = tables)) {
    lDataFrames[[i]] <- dbGetQuery(conn = con, statement = paste("SELECT * FROM '",
        tables[[i]], "'", sep = ""))
}

## create a data.frame for each table
for (i in seq(along = tables)) {
    lDataFrames[[i]] <- dbGetQuery(conn = con, statement = paste("SELECT * FROM '",
        tables[[i]], "'", sep = ""))
}
```

Next we separate all 7 data frames into more descriptive, independent data frames, and take a closer look at some of the data graphically. To do so we make a function `getRating()` that extracts overall player ratings, find the sum of ratings of all players on a team in a given match, and take a look at the summary of the new data frame.

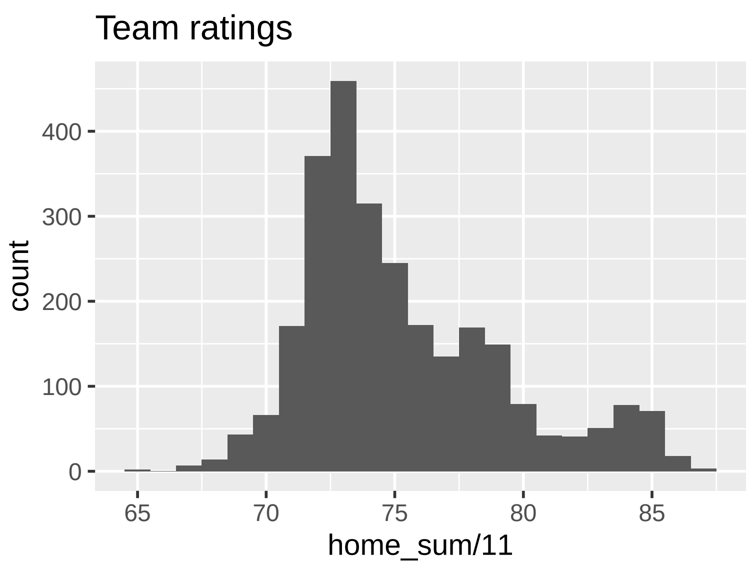
```
# Sorting into different dataframes
country <- as.data.frame(lDataFrames[1])
league <- as.data.frame(lDataFrames[2])
player <- as.data.frame(lDataFrames[4])
player_att <- as.data.frame(lDataFrames[5])
team <- as.data.frame(lDataFrames[6])
team_att <- as.data.frame(lDataFrames[7])

# Returns the most recent rating of a player given a player ID and a date
getRating <- function(player_id, date) {
    plyr <- player[player$player_api_id == player_id, ]
    id <- plyr$player_fifa_api_id
    plyr <- player_att[player_att$player_fifa_api_id == id, ]
    plyr <- plyr[plyr$date < date, ]
    rating <- plyr$overall_rating[1]
    return(rating)
}

# Chooses all games from given league with no missing player ID's nor any
# missing betting odds
league.id <- 21518  #spain
match <- as.data.frame(lDataFrames[3])
match <- match[complete.cases(match[56:77]), ]
match <- match[match$league_id == league.id, ]
match <- match[, -(98:100)]
match <- match[, -(101:103)]
match <- match[, -(104:109)]
match <- match[complete.cases(match[86:103]), ]

# Finds player rating for all home players in the selected matches Adds 11
# columns with each player's rating
time0 <- Sys.time()
for (i in 1:11) {
    col_name <- paste0("home_player_", i)
    match <- match %>%
        mutate(!!paste0(col_name, "_rating") := map2_dbl(!!sym(col_name), match$date,
            getRating))
}

# Finds player rating for all away players in the selected matches Adds 11
# columns with each player's rating
for (i in 1:11) {
    col_name <- paste0("away_player_", i)
    match <- match %>%
        mutate(!!paste0(col_name, "_rating") := map2_dbl(!!sym(col_name), match$date,
            getRating))
}

# Creates new df with only ratings and result of games
match_slim <- match[, 104:125]
ratings <- rowSums(match_slim)
match_slim$home_goals <- match$home_team_goal

# Sums up team rating for each game (home and away), and classifies matches as
# home win (H), draw (D) or away win (A) based on goals scored
home_sum <- c()
away_sum <- c()
home_odds <- c()
away_odds <- c()
draw_odds <- c()
result <- c()
for (i in 1:nrow(match_slim)) {
    home_ind <- seq(86, 103, 3)
    draw_ind <- seq(87, 103, 3)
    away_ind <- seq(88, 103, 3)
    nOdds <- length(home_ind)
    home_sum <- c(home_sum, sum(match_slim[i, 1:11]))
    away_sum <- c(away_sum, sum(match_slim[i, 12:22]))
    home_odds <- c(home_odds, sum(match[i, home_ind])/nOdds)
    draw_odds <- c(draw_odds, sum(match[i, draw_ind])/nOdds)
    away_odds <- c(away_odds, sum(match[i, away_ind])/nOdds)
    if (match$home_team_goal[i] == match$away_team_goal[i]) {
        result <- c(result, "D")
    } else if (match$home_team_goal[i] > match$away_team_goal[i]) {
        result <- c(result, "H")
    } else {
        result <- c(result, "A")
    }
}

# Creates new df, with columns home sum: sum of home team rating away sum: sum
# of away team rating result: factor (H, D, A)
match_data <- as.data.frame(home_sum)
match_data$away_sum <- away_sum
match_data$Hodds <- home_odds
match_data$Dodds <- draw_odds
match_data$Aodds <- away_odds
match_data$result <- as.factor(result)

summary(match_data)
```

```
##     home_sum        away_sum         Hodds            Dodds
##  Min.   :716.0   Min.   :716.0   Min.   : 1.042   Min.   : 2.443
##  1st Qu.:798.0   1st Qu.:797.0   1st Qu.: 1.653   1st Qu.: 3.275
##  Median :816.0   Median :815.0   Median : 2.067   Median : 3.425
##  Mean   :828.1   Mean   :827.6   Mean   : 2.761   Mean   : 4.081
##  3rd Qu.:854.0   3rd Qu.:852.0   3rd Qu.: 2.662   3rd Qu.: 4.068
##  Max.   :952.0   Max.   :960.0   Max.   :28.000   Max.   :17.583
##      Aodds          result
##  Min.   : 1.090   A: 760
##  1st Qu.: 2.622   D: 632
##  Median : 3.542   H:1309
##  Mean   : 5.078
##  3rd Qu.: 5.225
##  Max.   :41.667
```

First of all, we see that 'home win' is the most likely outcome of a match. This supports our initial idea that guessing only 'home win' is the best way to guess result. In coherence with this, home odds has the lowest mean value at 2.6, whereas draw and away odds tend to be higher. Draw has the lowest minimum odds, at about 3, so we would assume that the models we use later predict draw very rarely. Note that the maximum value of home_sum and away_sum are not the same, which is because the FIFA rating used in the dataset is updated several times throughout a season, so the rating of a team can go up or down from one match to the next.

Next we look at the distribution of teams by average player rating in a histogram.

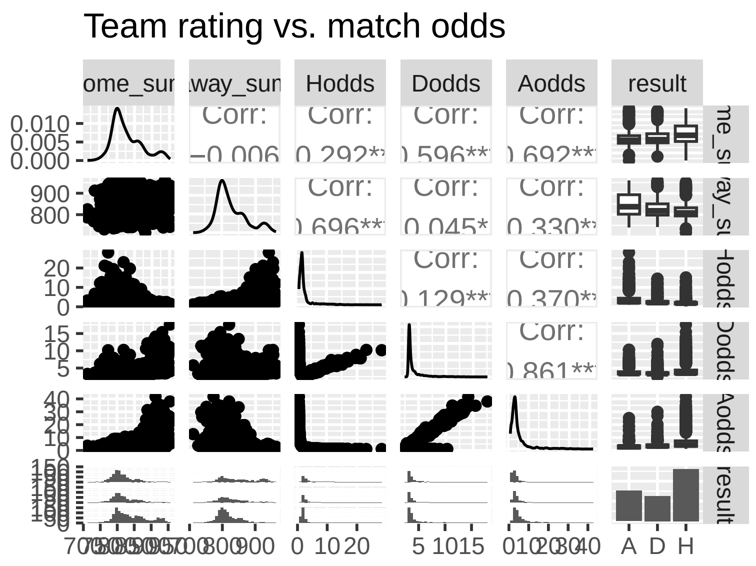
```
ggplot(match_data, aes(x = home_sum/11, title = "Team ratings")) + geom_histogram(binwidth = 1) +
    ggtitle("Team ratings")
```



From this we see that most teams have an average rating of about 75, with the highest rated teams going as high as 85, and the lowest at around 65.

We also compare team rating to odds in a pairs-plot to look at the average rating of a player on the teams we analyse.

```
ggpairs(match_data, title = "Team rating vs. match odds")
```



From the result-chart we see that most matches result in home-win, and the fewest result in draw. Furthermore, we see that there is a correlation between team rating and odds - the higher the home team's rating, the higher the away odds and lower the home odds.

Methods

Here we present 5 different classification models that we will use: linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), k-nearest neighbours (KNN), random forests and support vector machines (SVM). For some of these methods we also use 10-fold cross validation to determine the optimal value of some hyperparameters, which we explain further when we later explain the KNN method. First, we divide the data into a training and a test data set.

```r
set.seed(14)
n <- nrow(match_data)
Ntrain <- as.integer(0.75 * n)
train_ind <- sample(1:n, Ntrain)
match_data_train <- match_data[train_ind, ]
match_data_test <- match_data[-train_ind, ]
```

Linear Discriminant Analysis (LDA)

LDA is a classification method that predicts what class to assign an observation to based on predictors only, using a discriminant function. A new observation is is signed to the class for which the discriminant function is largest. Advantages of LDA is that it is simple and fast to use, and yet often efficient, but a weakness is that it makes assumptions that the observations are drawn from a multivariate normal distribution and that each class has the same covariance matrix, which may not be the case. In this case, the bias is high. Given the size of the European football dataset that we use, the assumption of normal distribution may not be too bad, which we will find out when testing.

To measure the performance of LDA, we make a confusion matrix and calculate the ratio between correct guesses and total guesses. We make the LDA using a training set, and find the error using a test set. This gives us a number that is easy to interpret, and which is easy to compare to other methods. In our case where we try to predict match results, we are only interested in whether we are right or wrong, and thus this is an effective way to measure performance.

We implement LDA by using the function `lda()` from MASS. We use `predict()` to get a factor with predictions and finally create a confusion matrix.

```r
lda.fit <- lda(result ~ ., data = match_data_train)
lda.fit_pred <- predict(lda.fit, newdata = match_data_test)$class
lda_conf <- confusionMatrix(lda.fit_pred, reference = match_data_test$result)$table
```

Quadratic discriminant analysis (QDA)

QDA is similar to LDA, but does not assume that the covariance matrix of each class is the same. Thus it approximates a matrix for each class, which has the advantage of reducing the bias, but at the cost of increasing the variance compared to LDA. Given an observation, QDA calculates the value of the now quadratic discriminant function, for each class, and puts the observation in the class for which the discriminant function is the largest. For our case, QDA may therefore work well if the covariance matrices are different.

We split the data into a training and test set, as for LDA, and calculate the error based on the test set the same way as we did previously.

Using `qda()` from MASS we do exactly the same as for LDA.

```r
qda.fit <- qda(result ~ ., data = match_data_train)
qda.fit_pred = predict(qda.fit, newdata = match_data_test)$class
qda_conf <- confusionMatrix(qda.fit_pred, reference = match_data_test$result)$table
```

K-Nearest Neighbours and 10-Fold Cross Validation

K-nearest neighbours classifies an observation to a class based on the classification of the K nearest neighbours. Here, the nearest neighbours are found by calculating the euclidean distance. This method has the advantages that it makes an intuitive classification of observation - if an observation is similar to previous observations of the one class, we assign it to the same class - in addition to allowing us to test for different values of K, so we have several opportunities to reduce error. A disadvantage is that if the number of dimensions is too large, we may have trouble with the *curse of dimensionality*, which essentially means that there are no nearby neighbours to assign a new observation to. In our case, we only have $p = 5$ dimensions, and thus KNN should be okay to use. The error of KNN is measured in the same way as before, i.e. by making a confusion matrix and calculating the percentage of positive guesses. Thus a low error means a high ratio of correct answers.

In KNN, K is a so-called hyperparameter. To find the optimal value of K that yiels the lowest error, we use 10-fold cross validation. This method works in a way that we split the training set into ten parts, and for each $K = 1, 2, ...$ we iterate over the set 10 times: each time we leave one part out, use KNN on the other nine and test on the one we left out, calculate the number of misclassifications for each of the 10 sets, and add them together. The K with the lowest total misclassifications is the K we prefer.

We use the function `train()` from caret with argument `method = 'knn'` to fit a knn-model. This, using the trControl argument we preform 10-fold cross validation to adjust the hyperparameter K. We look K's between 1 and 500. Finally we use the best parameter to fit a final model and create a confusion matrix based on the test data.

```r
trControl <- trainControl(method = "cv", number = 10)
knn_fit <- train(result ~ ., method = "knn", tuneGrid = expand.grid(k = seq(1, 500,
    5)), trControl = trControl, metric = "Accuracy", data = match_data_train)

knnBest <- knn(train = match_data_train[, 1:5], test = match_data_test[, 1:5], cl = match_data_train$re
    k = knn_fit$bestTune$k, prob = T)
knnBest_conf <- confusionMatrix(knnBest, reference = match_data_test$result)$table
```

Random forests

Random forests for classification is a similar method to bagging, where we make many classification trees based on our training set, and determine what class an observation in the test set belongs to by running it through all trees, and finding the mode class. In contrary to bagging, where we use all predictors when making each tree, we only use the square root of number of predictors, which in the case for our data set with 5 predictors means that we use 2 random predictors in each tree. The advantage of using many trees is that total variance reduce as the number of trees increase, and by using few predictors, the variance of each tree is also reduced. Thus we have low variance in total. A disadvantage is that it is slower the more trees we use, so we have to find a balance between accuracy and time consumption. Random forests fits well to our dataset, as we have only a few predictors that are intuitive to separate into branches. For random forests we calculate the error the same way as before, i.e. by calculating the fraction of correct predictions of the test set, since it makes it easy to understand intuitively and to compare to other methods.

We create a forest using the function `randomForest()` from the randomForest package. We use 500 trees and 2 predictors. We again use the `predict()` function and create a confusion matrix.

```r
rf.result = randomForest(result ~ ., data = match_data, subset = train_ind, mtry = round(sqrt(ncol(matc
    1)), ntree = 500, importance = TRUE)
rf.pred <- predict(rf.result, newdata = match_data_test)
rf_conf <- confusionMatrix(rf.pred, reference = match_data_test$result)$table
```

Support Vector Machines

In support vector machines we imagine a hyperplane (or several) splitting the classes, with a margin M on each side of the hyperplane. We wish to maximize the margin while maintaining some constraints, including a "budget" determined by the hyperparameter C. We determine the best value for C by using 10-fold cross validation. When C is small, fewer support vectors will be present, and we will obtain lower bias but higher variance. If C is large, we have the opposite, i.e. low variance but high bias. In addition, we use the hyperparameter γ since we use radial kernels, which tells something about flex of the circle. We also determine γ using 10-fold CV. An advantage of SVM is that it is flexible, and allows us to use different types of hyperplanes to separate the classes. A downside is that it is time-consuming to train on data, and since our dataset is quite large, this may mean that we need to reduce the amount of data we use. Also here we measure the error by looking at misclassifications/ratio of correct guesses.

We optimize the hyperparameters using the function `tune()` from the 'e1071' package. We use a radial kernel and test for the values given in the code below. We create a confusion matrix like before.

```r
svm.tune <- tune(svm, result ~ ., data = match_data_train, kernel = "radial", ranges = list(cost = c(0.
    0.1, 1, 5, 10, 100, 1000), gamma = c(0.01, 0.1, 1, 10, 100)))

svm.results <- data.frame(accuracy = svm.tune$performances$error, cost = svm.tune$performances$cost,
    gamma = svm.tune$performances$gamma)

svm.pred <- predict(svm.tune$best.model, newdata = match_data_test)
svm_conf <- confusionMatrix(svm.pred, reference = match_data_test$result)$table
```

Results and interpretation

LDA

```
lda_conf
```

```
##           Reference
## Prediction   A   D   H
##          A  72  34  31
##          D   1   1   1
##          H 108 119 309
```

```
sum(diag(lda_conf))/sum(lda_conf)
```

```
## [1] 0.5650888
```

The LDA model achieved an accuracy of 56.5%. A random guess would on average give an accuracy of 33.3%, so our model is definitely doing something. The most striking feature when looking at the covariance matrix is that the model only guessed on a draw 3 times. However, we are using a linear discriminant and thus would expect some bias. Therefore it is not so odd that the model does not guess draw often, as it would on average be a bad guess.

QDA

```
qda_conf
```

```
##           Reference
## Prediction   A   D   H
##          A  52  23  20
##          D 105  97 189
##          H  24  34 132
```

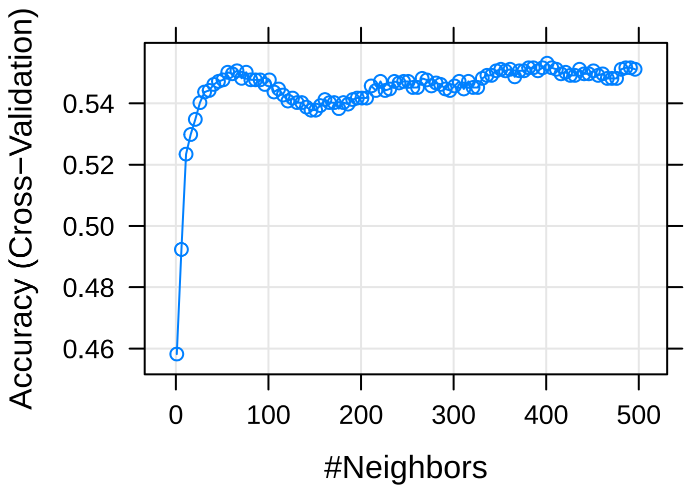
```
sum(diag(qda_conf))/sum(qda_conf)
```

```
## [1] 0.4156805
```

The QDA model does a lot worse than the LDA model with an accuracy of only 41.6%. The model does exactly the opposite of the LDA model, and predicts a lot of draws (and mostly misses). We suspect this model has overfitted quite a bit. There is a lot of variance (i.e. randomness) in football match outcomes, and thus easy for a model to overfit. Hence, in general we suspect a more biased model will do better on the test set. This is coherent with the accuracy of the LDA and the QDA models.

KNN

```
plot(knn_fit)
```



```
knnBest_conf
```

```
##           Reference
## Prediction   A   D   H
##          A  66  27  28
##          D   0   0   0
##          H 115 127 313
```

```
sum(diag(knnBest_conf))/sum(knnBest_conf)
```

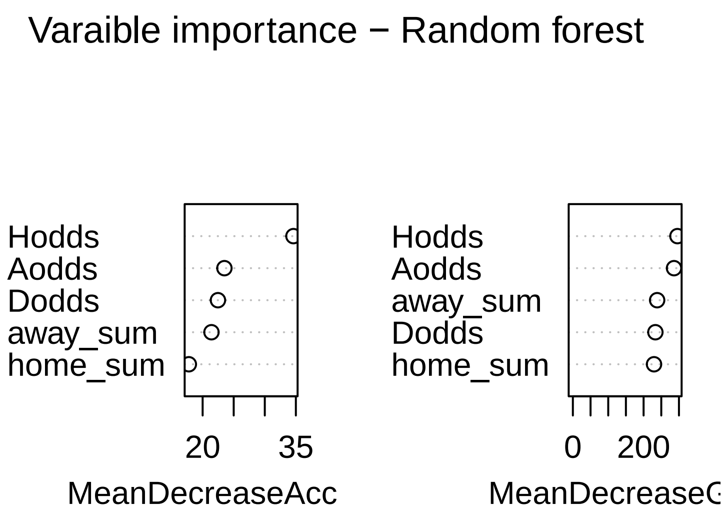
```
## [1] 0.5606509
```

A plot of the cross-validation to optimize K is shown above. It seems it does not matter to much what K is, as long as it is greater then approximatly 50. This is again coherent with our previous analysis of bias-variance. A greater K means more bias, and in this case a relatively large K preformed the best.

The knn model has an accuracy of 56.1%. This model actually predicts 0 draws, and preforms similar to the LDA model. This is not too surprisng as a knn model with a large K is in practice fairly similar to an LDA model.

Random Forest

```
varImpPlot(rf.result, main = "Varaible importance - Random forest")
```



```
rf_conf
```

```
##           Reference
## Prediction   A   D   H
##          A  79  39  51
##          D  27  26  28
##          H  75  89 262
```

```
sum(diag(rf_conf))/sum(rf_conf)
```

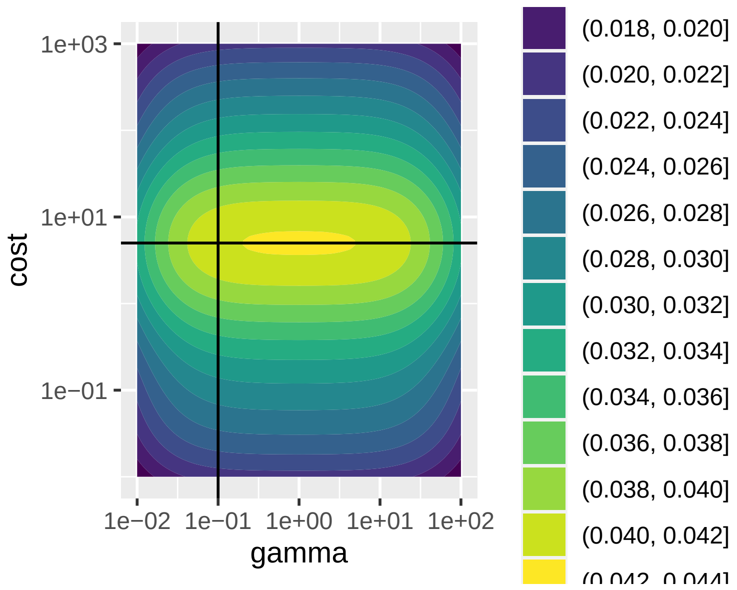
```
## [1] 0.5428994
```

From the variable importance plot we see that the odds are in general the most important predictors. This is not too surprising as the odds are generated using data analysis and similar models.

The random forest model preforms decently with an accuracy of 54.3%. And in contrast to the other models decent models, it actually predicts quite a few draws. However, looking at the draws it predicted, it would actually have preformed even better by guessing all the draws as either home wins or away wins.

SVM

```
ggplot(svm.results, aes(x = gamma, y = cost)) + geom_density2d_filled() + scale_x_continuous(trans = "l
    scale_y_continuous(trans = "log10") + geom_hline(yintercept = svm.tune$best.parameters$cost) +
    geom_vline(xintercept = svm.tune$best.parameters$gamma)
```



```
svm_conf
```

```
##           Reference
## Prediction   A   D   H
##          A  69  30  27
##          D   0   0   0
##          H 112 124 314
```

```
sum(diag(svm_conf))/sum(svm_conf)
```

```
## [1] 0.566568
```

The 10-fold cross validation found cost = 5 and gamma= 0.1 to be the best parameters, as shown in the plot. Using this we got an accuracy of 56.7%, which is the best so far, but very similar to LDA and knn. Looking at the confusion matrices, the similarity becomes even more striking. Again, the model predicts no draws have very similar guesses at home and away wins.

Overall, the LDA, knn and SVM model preformed very similarly with only minor differences. We suspect this is due to the fact we are using odds as a predictor. The model simply predicts home win if that has lower odds and away win otherwise. It is also worth noting that it might not be fair to use 33.3% as comparison since we know that in sports, being the home team is an advantage. Guessing only home wins we actually get an accuracy of 50.4%.

```
home_guess <- factor(rep("H", nrow(match_data_test)), levels = c("A", "D", "H"))
home_guess_conf <- table(home_guess, match_data_test$result)
home_guess_conf
```

```
##
## home_guess   A   D   H
##          A   0   0   0
##          D   0   0   0
##          H 181 154 341
```

```
sum(diag(home_guess_conf))/sum(home_guess_conf)
```

```
## [1] 0.5044379
```

Summary

Using statistical learning techniques we managed to predict the outcome of football matches with an accuracy of 56.7% at best. This was distinctly better than guessing only home wins which resulted in 50.4% accuracy.

Overall, the findings in this project was not too surprising. Due to the great amount of randomness in sports in general, it is very hard to get a good accuracy. Also, the best models never guessed draws, indicating that a more biased model is preferred. This is coherent with the analysis of randomness in sport. I model with high variance would preform. poorly on a test set because the actual result of a match is highly random.

One could also discuss the use of betting odds as a predictor since these numbers come from similar analysis. However, if the goal is solely to predict the outcome, one would be stupid not to include these numbers in the model. This all comes down to the purpose, and in our case, we were looking to predict the outcome.