Erasmus University Rotterdam

Bachelor's Thesis 2019

Final Version

Estimating Football Match Outcomes

Author

M. A. Kulakowski (455325)

Supervisor

MSc N.F.S. Dijkstra

First reader

Prof. dr. P.J.F. Groenen

Date

July 7, 2019

Abstract

Many gamble hoping to get rich and they wish they could maximize returns and minimize the risks. This paper aims to examine whether it is possible to create a method that predicts football match outcomes accurately and to make stable profits through online betting. I look at the Bundesliga games. In the estimation I use data related to characteristics of every team, past match and season outcomes, travelling distances, regional features, other (international) games and betting odds publicly available before every game. Three machine learning techniques are considered, Ordered Forest Estimator (OFE), XGBoost and Kernel Support Vector Machine (SVM). I train the models using seasons 2008/9 through 2017/18 and test them on season 2018/19. I predict the probabilities of home team win, draw and away team win for each game. I estimate betting returns based on 4 betting strategies. Only the XGBoost technique provides profits. The highest returns are from the XGBoost model that considers all the variables available and uses a betting strategy known as the Kelly Criterion. These returns are as high as 6.64% and are delivered from the closing odds of the bookmaker Pinnacle through making 276 bets in the whole season.

Keywords— machine learning, betting, random forest, classification, football

Contents

1 Introduction

Predicting the outcomes of football matches has been an intriguing topic ever since the game had existed. There were many pieces of research conducted pursuing an accurate way to do so (for instance, Leitner, Zeileis, and Hornik (2010), Nakamura et al. (2018) and many more). Aside from the scientific world, there is also a lot of money to be made, since every modern gambler is using some kind of programme in order to best predict the odds for every outcome. Using a more accurate method can increase the profits from betting. With the ongoing research in fields of mathematics and computer science and the introduction of machine learning, new techniques arise to approach this issue.

One such method is called the Random Forest Estimator, first developed by Breiman (2001). It is an ensembling technique that works through bagging a set of Decision Trees (Breiman et al. (1984)) in order to return the list of probabilities for every possible outcome. A further description is provided in section 3.1.2. This method has already been used in making football outcome predictions (e.g. Groll et al. (2018)). However, the football outcomes are an ordered set, since a win is better than a draw, and a draw is better than a defeat. The basic random forest estimator was not created to account for that additional fact. It can only be applied to continuous outcomes (regression random forest) or binomial outcomes (classification random forest). That is why Goller et al. (2018) have tried to predict the football outcomes by the use of an Ordered Forest Estimator (OFE), as described by Lechner & Okasa (2018). It works similarly to the random forest, but it also accounts for the fact that the outcomes are ordered, which is useful for this case. The method was tested by Goller et al. (2018) and showed to perform better than many odds-counting systems used by online bookmakers.

When one examines the various known machine learning methods, random forest performs relatively well, however, there are some other algorithms that are praised even further. In Goller et al. (2018) they were already able to generate a small positive return by simulating the placing of bets on various websites. It therefore seems intriguing to see, whether we can create an algorithm that could provide an accurate forecast of the match outcomes, so that betting on the games would bring a strictly positive return. Therefore, the aim of my paper is to replicate the method used by Goller et al. (2018) and then to evaluate the effectiveness of the OFE through betting. At the same time, I compare OFE with the other methods, namely the Kernel Support Vector Machine (Kernel SVM) and XGBoost, and see if any of them can bring positive returns. This leads to the following research question:

Is it possible to make a stable profit, through betting on football outcomes predicted by machine learning algorithms?

For this, I use the Bundesliga game outcomes. The outcomes considered are whether there has been a win, a draw or a loss for the home team in each game. Each algorithm is used to predict outcome probabilities for the season 2018/19 and these are applied to make (simulated) bets, using four different betting strategies. The best technique for making bets is the XGBoost, it is the only one that brings positive returns.

My methodology is described in section 3. The results are displayed in section 4. The results

are discussed and summarized in section 5. Section 6 contains some suggestions on possible further research.

2 Data

Since one of the goals of this paper is to replicate the method from Goller et al. (2018), I will attempt to use a similar set of variables. However, they claim to be using about 300 variables in their estimation, but they only describe around 120 of them. Therefore, for this research, only a subset of these 300 variables will be used, while making sure that every category is included and the most relevant data is accounted for. My data consists of over 3300 observations with 90 independent variables. The data is for Bundesliga games, from season 2008/09 to 2018/19 and contains various variables that might have influenced the game outcomes.

The variables include, for every game, the information on the last game outcomes and points for both teams that play a given game. There is also information about the previous season points for every team, as well as additional information, such as goals, corners and cards, taken from `www.football-data.co.uk/germanym.php`. Next, there is also information regarding the team characteristics, such as the teams' market values, average age of players, the proportion of left-legged players etc., which were taken from `www.transfermarket.com`. Then, the data set also includes economic information, such as the teams' brand values, regional GDP and unemployment rates, taken from `www.regionalstatistik.de`. It also contains the television revenues from transmitting the matches. These were taken from `www.fernsehgelder.de`. Then, there are the variables used for the number of kilometres the away team has to travel, and the total time it takes to get there. These were retrieved from Google Maps. The data set also contains information about each stadium's capacity. These were taken from their corresponding Wikipedia pages. Next, the schedules for Champion's league and World cup were retrieved from `www.wikipedia.org`. The betting odds of a group of leading bookmakers were taken from `www.football-data.co.uk/germanym.php` and contain the odds for every outcome, set before each game has started. Nine sets of odds are used in predictions, providing additional 27 variables to the model, to a total sum of 117. For simulating the returns, seven sets of odds are used.

Unfortunately, since every year the some teams from Bundesliga get demoted and the top teams from Bundesliga 2 are promoted, there were cases of missing variables for the previous season outcomes. I coped with that in the following way. First, I computed the previous season outcomes for the three demoted teams. Then, I averaged these outcomes. Lastly, I assigned the averaged outcomes as the previous season outcomes for the teams that were promoted. This method allowed an accurate approximation of the outcomes, as the three promoted teams can be expected to perform similarly to the ones that were demoted.

3 Methodology

This section describes the methods used in this research. Firstly, the machine learning techniques used to predict outcome probabilities are described. Then, the possible extensions to the data used

are presented. Afterwards, the betting strategies applied are summarized. The section ends with describing the measures used to evaluate the predictions.

3.1 Ordered Forest Estimator

The Ordered Forest Estimator (OFE) is a technique developed in Lechner & Okasa (2018). It is based on the random forest estimator, which in turn is made of decision trees.

3.1.1 Decision Trees

In order to understand the random forest estimator, it is crucial to know what decision trees are. The algorithm was developed in Breiman et al. (1984) and is based on a set of decisions, taken in a tree structure. A tree is comprised of nodes and branches. The final nodes, the ones without any children, are called leaves. The first node is called the root node. The other nodes are called internal nodes. The number of moves required to reach the furthest leaf from the root is called the depth of the tree.

In a decision tree, there is a decision taken at every node except for the leaves. We consider a sample of observations, N, a set of explanatory variables, X, and the dependent variable Y. Every leaf contains an outcome o from the possible set of outcomes O. In this research, I only consider binomial outcomes in decision trees, therefore $O = \{0, 1\}$ and $|O| = 2$. At each node, the decisions can be regarding both numerical and classification variables, such as age or marital status. Every node has exactly two children, therefore the decisions can be regarded as yes/no questions. A node splits the current sample of observations S, in two subsamples S_1 and S_2. For every node, the decision how to split is based on a variable $x \in X$. The aim of the decision is to split the sample in a way that best separates the two classes of outcomes, so that most of the observations in each child only belong to one class. The splits are made until one of the stopping criteria is reached. There are three criteria used, reaching one of them is enough for the node to stop further splits. The first criterion is the maximum depth of the tree, no leaves can have depths higher than this value. Another one is the minimum number of observations required to make a further split. If there too few observations in a node, that node becomes a leaf. Last criterion is the minimum information gain. The information gain is defined as

$$IG = \frac{Gini_{Split} - Gini_{Node}}{Gini_{Split}},$$

where $Gini$ is a Gini impurity of a sample. The Gini impurity of a split is defined as

$$Gini_{Split} = \frac{S_1}{S_1 + S_2} * (1 - S_{1,0}^2 - S_{1,1}^2) + \frac{S_2}{S_1 + S_2} * (1 - S_{2,0}^2 - S_{2,1}^2),$$

where S_1 and S_2 are the samples sizes after splitting, $S_{1,0}$ and $S_{2,0}$ represent the number of 0 outcomes in each child, while $S_{1,1}$ and $S_{2,1}$ stand for the number of 1 outcomes. The Gini impurity of a node is defined as

$$Gini_{Node} = 1 - S_{0,0}^2 - S_{0,1}^2,$$

where $S_{0,0}$ and $S_{0,1}$ represent respectively the number of 0 and 1 outcomes in the node. If the information gain is high enough, the node will conduct the split. Else, it will stop the splitting and

become a leaf. Such a decision tree can then be used to classify new observations with unknown outcomes into one of the possible classes. An illustration is available in Figure 1.

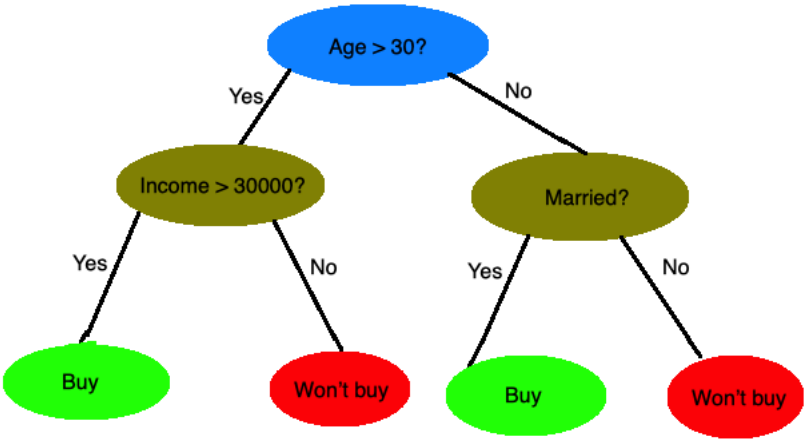


Figure 1: Decision tree illustration.

In figure 1 the previous outcomes have been used to classify a purchase decision of an individual, based on their age and marital status.

3.1.2 Random Forest Estimator

Knowing how the decision trees work, we can now understand the random forest. It is a tree ensemble method, introduced in Breiman (2001). The main issue with trees is that they have low bias but high variance (overfitting). This means that a model is great at predicting the outcomes that were used to build it (training set), but if the outcomes of some new observations need to be predicted (the test set), it may not be so accurate anymore. Random forest decreases the variance by bagging the trees. Bagging, short for bootstrap aggregating, is a procedure where a bootstrapped sample of observations L is generated from sample N, such that $|L| = |N|$ and then L is used to create a single decision tree. This process is done T times, creating a set of T trees. Moreover, while building a tree, each node considers only a subset $K \subset X$ of variables, such that a variable $k \in K$ is used to divide the sample. K is randomly determined at each node. This way ensures that the trees are different enough and prevents overfitting. Then, to make a forecast for new observations, we run each observation through all the trees in the forest. Then, each tree votes for one of the outcomes. The fraction of votes on each class is the outcome probability predicted by the random forest algorithm. Each tree has an equal vote.

3.1.3 Ordered Forest Estimator

Knowing the basics of random forest, it is time to introduce the ordering. The OFE algorithm used in this paper is based on the one used in Goller et al. (2018) and has been illustrated in Figure 2. It works by using the principles of continuous probability distribution. The core assumption is that we run the binomial random forest $M - 1$ times, where M is the number of classes of outcomes and

that we change the dependent variable in each of them. The football case contains three possible outcomes, therefore the random forest needs to be performed twice. The first random forest uses Y_1 as the dependent variable, which is one when the home team wins and zero otherwise. It then computes the probabilities of this outcome for every observation in the forecasted sample. These probabilities are $\hat{Y}_{1,i}$ for every $i \in N$. Then, the second random forest is built, this time using Y_2 as the dependent variable. Y_2 means either the home win or draw. Then, using the estimated model, the probabilities $\hat{Y}_{2,i}$ are computed. A $\hat{Y}_{2,i}$ means the probability of either home win or a tie for match i. Lastly, since the probability of either a home win or a draw or an away team win is equal to 1 (as these are all the possible outcomes), we know that $\hat{Y}_{3,i} = 1$ for every i. From these we can compute the probabilities of each particular outcome, $P^{tot}_{m,i}$ by subtraction: $\hat{Y}_{m,i} - \hat{Y}_{m-1,i}$ for every i. Then, we make sure that no probabilities are below 0 and and fix the issues if necessary. We end up with probabilities $\hat{P}_{m,i}$ for every outcome m and observation i.

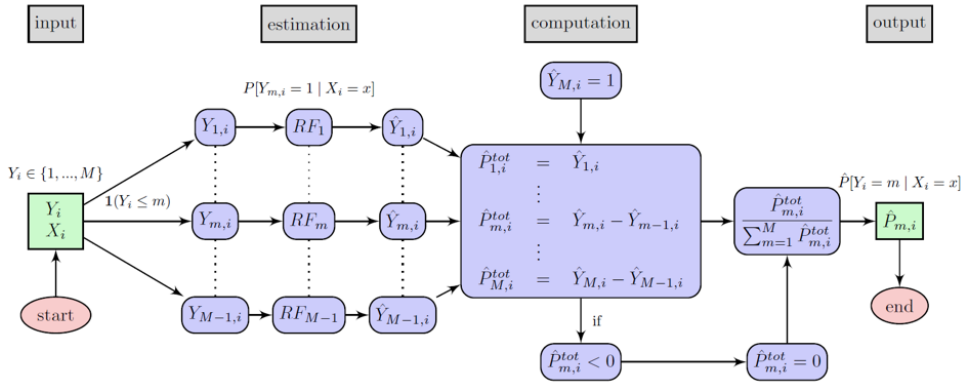


Figure 2: Ordered Forest Estimator Algorithm

In figure 2 the OFE algorithm is described. In our case M = 3, therefore two random forest classifiers are built. $Y_{1,i}$ is the home win, $Y_{2,i}$ is home win or draw. Source: Goller et al. (2018).

3.2 XGBoost

XGBoost is an algorithm, described in Chen & Guestrin (2016), that uses a technique called extreme gradient boosting. It is an ensemble method applied to a number of decision trees. To understand how it works, we first need to know what boosting is.

3.2.1 Boosting

According to `www.analyticsvidhya.com` the term 'Boosting' refers to a family of algorithms which converts weak learners to strong learners. In this research, boosting is an ensemble method applied to a number of decision trees. Therefore, here the weak learners are the single decision trees, while the strong learner is the ensemble of them. The boosting algorithm was developed in Freund & Schapire (1996). An illustration can be seen below (based on `www.analyticsvidhya.com`)

1. Initially, an equal weight is assigned to each observation in the training sample. Then the first decision tree is built.

2. If there is any prediction error caused by the first decision tree, then we pay higher attention (add higher weight) to observations that were incorrectly predicted. Then, we build the next

decision tree.

3. Iterate step 2 until the limit of iterations is reached or higher accuracy is achieved.

Afterwards, the outputs predicted by every tree are combined together to form the estimated probabilities for each outcome. A key difference from bagging is that the trees have varying weights when it comes to voting, so that a tree that better contributes to variance reduction will have a higher weight. Furthermore, unlike in bagging, for boosting the order of creating trees is important. Every new tree is made in a way that could best account for the errors of its predecessors. Then, a weight is assigned, depending on the tree's contribution to the increase in accuracy of the model. Higher weights are also assigned to observations in the sample that are more difficult to classify.

3.2.2 Gradient Boosting

The concept of gradient boosting was first developed in Breiman (1997), while Chen & Guestrin (2016) used it to develop the XGBoost technique. A clear explanation is provided on `www.analyticsvidhya.com`. It is a modified version of boosting, that trains new models sequentially. The new models gradually minimize the following loss function using the gradient descent method. The weights are assigned in the following way (based on **www.analyticsvidhya.com**)

1. First, assume an initial weight distribution across the sample, D_1, such that $D_{1,i} = 1/\mathsf{N}$ for all $i \in N$.

2. We assume some α_t.

3. Create a new weak classifier c_t (the new tree).

4. Update the weight distribution in a following way

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i c_t(x_i))}{Z_t},$$

where

$$Z_t = \sum_{i=1}^{\mathsf{N}} D_t(i)\exp(-\alpha_t y_i c_t(x_i)),$$

where α_t is the learning rate, y_i is the actual outcome and $c(x)$ will be the class predicted by learner. The independent variables are represented by x. We calculate α_t in the following way

$$\alpha_t = 0.5\log(\frac{1-\epsilon_t}{\epsilon_t}),$$

where ϵ_t is the current error.

5. Repeat steps 1-4 until no significant improvements are possible or if the maximum number of iterations has been reached. In this research, there is a maximum of 60 iterations, as more would lead to overfitting.

6. Compute the final prediction by using the weighted average of the outputs of all trees. Use α_t as the weight for each tree t.

The above algorithm is the basis of gradient boosting. However, in extreme gradient boosting, more techniques are used. One of the extensions is the algorithm used to make the splits at every node of the trees. There are three splitting algorithms commonly used, the Basic Exact Greedy algorithm, the Approximate algorithm and the Sparsity-Aware split finding algorithm. This research, however, only focuses on the Greedy algorithm to make splits. That is because the other two were designed solely for handling some specific issues that the data could cause. The Approximate algorithm is only needed if the data set used is too large for the system to handle the Greedy algorithm efficiently (in cases with millions of observations, and/or hundreds of variables, or when low computation time is essential). The sparsity-aware algorithm, as the name would suggest, is used when there is missing data in the majority of observations, as the algorithm contains features that can deal with that. Since neither case applies to the data used in this research (we deal with missing data in a different manner, see section 2), the Basic Exact Greedy algorithm is used.

3.2.3 Basic Exact Greedy Algorithm

The Greedy algorithm is used by the decision trees, at every node of the tree, in order to make the most optimal split of the sample based on one variable. It is described by Chen & Guestrin (2016) and illustrated in algorithm 1.

Input : I the set of indices of observations considered by the node
Input : d_i the dimension of features used at the split
$gain \leftarrow 0$;
$G \leftarrow \sum_{i \in I} g_i$, $H \leftarrow \sum_{i \in I} h_i$
for $k = 1$ **to** $|X|$ **do**
  $G_{Left} \leftarrow 0, H_{Left} \leftarrow 0$;
  for every j in $sorted(I,$ by $x_{jk})$ **do**
    $G_{Left} \leftarrow G_{Left} + g_j$,$H_{Left} \leftarrow H_{Left} + h_j$;
    $G_{Right} \leftarrow G_{Right} + g_j$,$H_{Right} \leftarrow H_{Right} + h_j$;
    $score \leftarrow \max(score, \frac{G_{Left}^2}{H_{Left} + \lambda} + \frac{G_{Right}^2}{H_{Right} + \lambda} - \frac{G^2}{H + \lambda})$;
  end
end
Output : Split with max score

Algorithm 1: Exact Greedy Algorithm for Split Finding

Where g_i and h_i are the first and second order gradient statistics on the loss function, λ is the learning rate parameter, which is used to determine the maximum weight that we can attach to each new decision tree. Here, the loss function l is differentiable and convex. It measures the difference between the prediction $\hat{y}_i$ and the true observation y_i.

3.2.4 System Design

Another aspect of XGBoost that makes it useful is the system design, with some features applied into the technique, in order to decrease the total computation time. One of those features is called †he column block for parallel learning. Every time a decision tree makes a split, it needs to evaluate which variable is the best to split and what is the best value of the variable, by which to split. In order to do so, the observations need to be sorted by every variable considered in the split. Since a method can do thousands of those evaluations, repeating the sort can be very time consuming. That

is why XGBoost uses the column block for parallel learning, which basically stores a block of sets of observations, every set in the block contains the same observations (the observations needed for the split), but they are sorted by one of the variables. Thanks to that, the technique can just access the right set, instead of sorting them every time. This can decrease the computation time severely, proportionately to log of the size of the trees.

Another problem that might slow down the XGBoost while using the Greedy algorithm is that it requires computation of large number of gradient statistics in each step, which can slow down the split if all the gradient statistics computed do not fit into CPU cache memory and misses occur. To cope with that XGBoost uses the cache-aware prefetching algorithm. It works by saving the computed gradient statistics and allocating a caller method, so that the appropriate statistics can be called and used whenever necessary. According to Chen & Guestrin (2016), this can double the execution speed of the Greedy algorithm.

Lastly, the algorithm optimises how it uses the disk space. It uses two blocks for out-of-core computation. The first one is block compression, where the block of data is compressed by columns and then when needed it is decompressed by an independent thread while being loaded into main memory. The block sharding requires multiple disks present, which is not available, therefore it is of no interest in this research.

3.2.5 K-fold Cross Validation

Lastly, while training itself, the XGBoost model uses a technique called k-fold cross validation in order to best determine its current accuracy. It divides the data sample into a number of folds, in general 10 folds. Then, it uses 9 of the folds to build the model and tests the forecast accuracy on the tenth fold. The process is repeated so that the accuracy can be tested for each fold. The accuracies are then averaged to achieve the total current prediction accuracy.

In our case there are three classes of outcomes. Unlike the other methods, XGBoost can predict those directly, therefore the ordered algorithm, as described in section 3.1.3 is not necessary. This method is considered one of the best methods in machine learning, due to its extraordinary predicting abilities. According to the CEO of Kaggle, an online community for data scientists which organises challenges for prediction making, XGBoost was the technique associated with over 50% of recent winners (`https://www.import.io/post/how-to-win-a-kaggle-competition/`). I therefore expect it to perform the best.

3.3 Kernel Support Vector Machine

Kernel SVM is a machine learning technique that combines the SVM method and some Kernel mapping function.

3.3.1 Support Vector Machine

The Support Vector Machine (SVM) algorithm, first described in Vapnik & Chervonenkis (1974), is a machine learning technique that aims at using all variables in X to make a partition in the data sample, such that each side has different outcome class. The best way to explain it is through the

following example. The SVM is illustrated in Figure 3, in the example, there are two (continuous) variables, X_1 and X_2, therefore $|X| = 2$. They are the age and income of each individual. The outcome has two classes, a yes/no decision to buy a product, the 'yes' answers are marked green, while the 'no' answers are red. It can be seen that here, both outcomes are in two distinctive groups. The aim of the algorithm is to find, for each class of outcomes, the points that are the closest to the other group. These are the points marked A and B on Figure 3. The SVM then creates a line that partitions the sample in two parts, so that the outcomes get separated. The choice of the line's slope and location is chosen in such a way as to maximize the sum of orthogonal distances from both points, but also the line should be equidistant from them. The distances are marked d_1 and d_2, it can be seen that the partitioning line is perpendicular to them as that maximizes their sum. Because points A and B are so important and since they are vectors, we call them the support vectors. Hence the SVM method's name. The partition has the amount of dimensions corresponding to the amount of variables, for 2 variables it is a line, for 3 it is a plane and so on. Unfortunately, sometimes it may happen that we are not able to linearly separate the data. The best way to solve such issues is through the use of a mapping function.

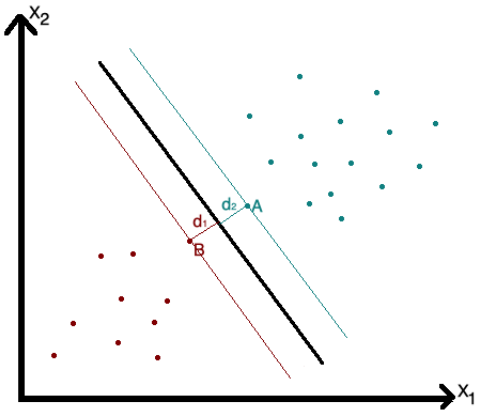
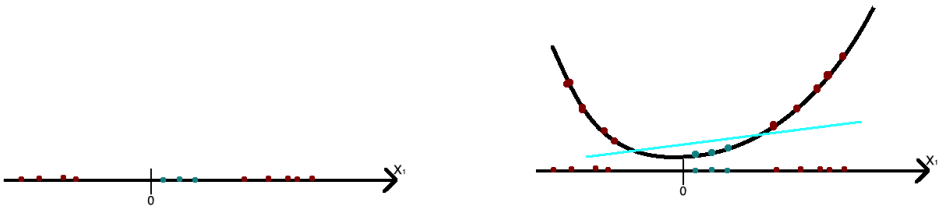


Figure 3: Support Vector Machine

In figure 3 the SVM method is illustrated. X_1 and X_2 are the age and income of each individual. The outcome has two classes, a yes/no decision to buy a product, the 'yes' answers are marked green, while the 'no' answers are red.

3.3.2 Mapping Functions

The SVM method works well for cases where it is easy to make a clear partition of the classes, however in practice it is often hard to draw a line. An example is illustrated in Figure 4a, where every observation only has one explanatory variable, X_1. The green points (outcomes) are in the middle, while the red are on both sides. It can be seen that it is impossible to put one dot there that would partition the sample well. This issue can be solved by the use of mapping functions. The mapping functions transform the data to a higher dimension, so that now a border can be drawn between the outcome classes. An example follows in Figure 4b, where the original values were squared, therefore moving from a one- to a two-dimensional space. Now, a clear partition of the outcomes can be made.



(a) Before mapping, no feasible partition.

(b) After mapping, there is a feasible partition, marked with the blue line.

Figure 4: The demonstration of how a mapping function works. Here, the function is $f(x) = x^2$

3.3.3 The Kernel

The Kernel SVM, as described in Boser et al. (1992), is an SVM algorithm that uses a specific mapping function. It is called the Kernel function. There are many types of Kernel functions used in research, however I will only focus on one of them, the radial base Kernel function. It is defined in the following way

$$K(A, B) = \exp(\frac{-\|A - B\|^2}{2\sigma^2})$$

where $K(A, B)$ is the kernel function, A and B are two samples, represented as feature vectors. σ is a free parameter, determined during programming phase for most optimal performance.

3.3.4 Ordered Kernel SVM

Since Kernel SVM is also traditionally used only for binomial cases, the same ordered algorithm as in section 3.1.3 needs to be applied. This is done in the exactly same way as for the random forest.

3.4 Betting Odds as Covariates

Goller et al. (2018) write that they did not use the betting odds as explanatory variables in their research. I think that they may be an interesting addition, for the following reasons. Firstly, all the odds are available many hours before the games, therefore using them for daily predictions is feasible. Moreover, they already contain outputs of some programmes that estimate the outcome probabilities, therefore they may be a valuable addition to the machine learning models. Nine sets of odds are used (for home win, draw and away win), providing additional 27 variables. The odds are from Bet365, Bet&Win, Gamebookers, Ladbrokes, Interwetten, Sportingbet, Stan James, VC Bet and Blue Square, they are collected on the day of the game. I record the performance of models with and without the betting odds, to see if the returns can be increased.

3.5 Betting Strategies

This section describes the four betting strategies used in this research to estimate the returns that can be achieved.

3.5.1 Proportionate Betting

The first strategy used is the same as the one in Goller et al. (2018). Namely, for all games I look at the estimated probability of every outcome and place the bets proportionately to each of them. This method has some flows, as it may not always be worth it to bet on highly unlikely outcomes, especially if the estimated probability is lower than the one implied by the odds. Section 3.5.4 contains a method that deals with this issue.

3.5.2 Value Bet

This strategy also comes from Goller et al. (2018), where they examine the probabilities reflected by the odds. They only bet on those outcomes where the predicted probability is higher than the one reflected by the odds. Moreover, they only bet on one website and one outcome per match, choosing the bookmaker/outcome combination that has the highest difference. In my research, however, I do not limit myself to one bookmaker per match. Here, I bet separately for each bookmaker and I also allow more than one bet per match. For example, if my predicted probabilities for home win and draw are both higher than the probabilities implied by the betting odds, I will bet on both outcomes.

3.5.3 The Most Likely Outcome

This strategy bets the the amount of 1 on the most probable outcome of every match, provided that the probability is higher than 0.5.

3.5.4 The Kelly Criterion

The last betting strategy used is about using the kelly criterion to determine the optimal size of the bet. The method is described on `www.pinnacle.com`. The formula is defined in a following way

$$Kelly = \frac{(B \cdot P - Q)}{B}$$

where B is the decimal odds of the outcome - 1, P is the estimated probability of the outcome and $Q = 1 - P$. $Kelly$ returns a fraction of the budget that should be placed on a bet. If it is negative, I do not place a bet. If it is positive, I place a bet with the amount of money equal to $10 \cdot Kelly$. This method looks promising, as it only bets on those matches and outcomes where the predicted probability is larger than the one reflected by the odds.

3.6 Evaluation Measures

In this section, the measures used to evaluate the performance of the models are defined.

3.6.1 Returns

The primary measure are the returns on betting from every strategy used. A return on bet is defined in the following way

$$R = (\frac{W}{B} - 1) \cdot 100\%$$

where R is the return, W represents the total winnings from the betting, and B is the total sum of bets placed. If the winnings are higher than the bets, the return is positive.

3.6.2 Control Measures

While the main purpose of this research is to maximize the return on betting, a few additional evaluation measures are introduced. That is in order to give the reader a better idea of prediction abilities of each model. The following measures are based on the Marketing Models lecture notes by prof. Paap (2019). First, I analyse the computed probabilities and make a prediction, using the most likely outcome. Then, I use the predicted and the actual outcomes to create a prediction-realization table for every technique analysed. It is a table where every row corresponds to a class of an actual outcome and every column to the predicted outcome. For example, if for observation i the predicted outcome is home win and the actual is draw, we assign this match to row 2, column 1. The final table contains the number of outcomes assigned to each cell. The numbers in diagonal cells are correctly predicted outcomes. A measure called the hit rate is computed by summing all the numbers in diagonal cells and dividing them by the total number of outcomes. Then, using the hit rates I will conduct a test called Better than Random Prediction (BRP). The null hypothesis is that the model predictions are not better than random predictions. The following test statistic is used

$$BRP = \frac{h - q}{\sqrt{q(1-q)/\mathsf{N}}},$$

where h denotes the hit rate, q is the random hit rate and N is the size of the predicted sample. The value of q is computed by $\sum_{m=1}^{M} p_m^2$, where p_m denotes the proportion of outcomes covered by the outcome m. We can reject the null hypothesis in favour of the prediction, at 5% significance level, if BRP is larger than 1.645. The test statistic follows normal distribution (single-tailed).

4 Results

In the following section the results are displayed. Every technique used is evaluated, based on its prediction accuracy, as well as the returns from betting, by using the four betting strategies.

4.1 Model Evaluations

Here, the models are evaluated based on their predictive accuracy and hit rates. Then we test if the predictions are better than random.

4.1.1 Evaluations OFE

Before looking at returns, I evaluate the performance of each model in predicting the outcomes. I apply the control measures as described in section 3.6.2. Tables 1(a) and 1(b) display prediction-realization tables for forecasts made using OFE. Table 1(a) contains predictions without using the odds, table 1(b) shows predictions that use them. The forecasts are quite accurate, with hit rates being 52.61% and 55.88% respectively. Variable q has a constant value of 0.36 for the entire 2018/19

season. This gives BRP values of 6.19 and 7.38, therefore both predictions are better than random. Moreover, it seems that the odds indeed help in improving the OFE's performance. It is interesting to note that this method never predicts any draws. It may be due to the fact that a draw is in practice the rarest outcome, therefore the estimated probabilities are never the highest for this outcome class, and we only make predictions based on the most likely option.

Table 1: Prediction-realization tables OFE

Actual	**Predicted**		
	Home	Draw	Away
Home	115	0	23
Draw	56	0	17
Away	49	0	46

(a) OFE prediction-realization table

Actual	**Predicted**		
	Home	Draw	Away
Home	117	0	21
Draw	48	0	25
Away	41	0	54

(b) OFE with odds, prediction-realization table

4.1.2 Evaluations XGBoost

Tables 2(a) and 2(b) display prediction-realization tables for forecasts made using XGBoost. Table 2(a) contains predictions that do not use the odds, while table 2(b) shows predictions that use them. The forecasts are quite accurate, with hit rates being 44.12% and 46.41% respectively. This gives BRP test statistic values of 3.09 and 3.92, therefore both predictions are better than random. Moreover, the odds are also valuable in improving the performance of XGBoost, improving the hit rate by over 2%.

Table 2: Prediction-realization tables XGBoost

Actual	**Predicted**		
	Home	Draw	Away
Home	86	26	26
Draw	36	12	25
Away	35	23	37

(a) XGBoost prediction-realization table

Actual	**Predicted**		
	Home	Draw	Away
Home	82	32	24
Draw	38	20	15
Away	36	19	40

(b) XGBoost with odds, prediction-realization table

4.1.3 Evaluations Kernel SVM

Tables 3(a) and 3(b) display prediction-realization tables for forecasts that use the Kernel SVM technique. Table 3(a) contains predictions that do not use the odds, while table 3(b) shows predictions that use them. The forecasts are quite accurate, with both hit rates being 47.39 % and 47.39%.This gives BRP values of 4.28 for both predictions, therefore they are better than random. In Kernel SVM using the odds as covariates yields no improvement on the accuracy of prediction of outcomes. However, this does not prevent the odds from having an effect on the probability forecasting abilities of the method.

4.2 Betting Returns

This section contains the returns achieved through betting by using the estimated probabilities computed with every method used, via the four betting strategies considered.

Table 3: Prediction-realization tables Kernel SVM

Actual	Predicted		
	Home	Draw	Away
Home	115	16	7
Draw	57	8	8
Away	53	20	22

(a) Kernel SVM prediction-realization table

Actual	Predicted		
	Home	Draw	Away
Home	115	16	7
Draw	55	7	11
Away	54	18	23

(b) Kernel SVM with odds, prediction-realization table

4.2.1 Returns XGBoost

This section contains the returns from betting by using the probabilities predicted by the XGBoost model. Betting strategies used as described in section 3.5. Seven sets of odds were used. Tables 4 and 5 contain values from betting on the matches by using the probabilities estimated without odds and with odds respectively. In Table 4 there were three cases of positive returns. All were for the Pinnacle odds. The highest returns of 2.88% were returned through the value bet strategy.

Table 4: XGBoost technique: Returns per betting strategy and bookmaker. The positive returns are marked with green colour.

Strategy	Bookmaker						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	-3.91	-4.26	-3.88	+0.09	-4.50	-1.61	+0.90
Value bet	-3.36	-4.05	-2.91	-1.37	-5.22	-1.11	+2.88
Most likely outcome	-12.87	-12.75	-11.78	-10.47	-12.66	-11.76	-10.33
Proportional bet	-4.86	-4.67	-5.09	-2.19	-5.07	-3.40	-1.69

Table 5 contains the returns gained from using the XGBoost technique that also uses odds as explanatory variables. This technique provides a lot of positive returns. It can be seen that using the Kelly Criterion betting strategy provides profits for every bookmaker used. The highest returns are again for the Pinnacle website, yielding as much as 6.64%. This is not a result of few lucky bets, as 276 of them were placed. Aside from the fully green Kelly Criterion, the strategy of betting on the most likely outcome provides profits in two cases.

Table 5: XGBoost technique with odds: Returns per betting strategy and bookmaker. The positive returns are marked with green colour.

Strategy	Bookmaker						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	+2.44	+2.59	+2.19	+5.11	+2.56	+4.09	+6.64
Value bet	-4.99	-2.79	-5.41	-3.48	-3.69	-3.58	-1.08
Most likely outcome	-2.16	-1.34	-1.14	+0.32	-2.00	-0.40	+0.41
Proportional bet	-4.12	-3.69	-3.89	-1.59	-4.23	-2.67	-1.33

Since using the XGBoost with odds as covariates seems to return noticeable profits for the season 2018/19, it is interesting to see if this could be successfully repeated for another season. Table 6 contains the returns from betting using probabilities estimated with XGBoost with odds as covariates, for season 2017/18. Unfortunately, the returns are much lower for this season. There are three cases of positive returns, of which two cases return profits above 1%. These are +2.04% and +1.55%, they

both are for Pinnacle closing odds. The successful strategies are the most likely outcome and Kelly Criterion respectively.

Table 6: XGBoost technique with odds, season 2017/18. Returns per betting strategy and bookmaker. The positive returns are marked with green colour.

Strategy	**Bookmaker**						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	-4.20	-5.65	-8.19	-0.41	-5.23	-2.76	+1.55
Value bet	-9.90	-10.04	-11.57	-5.44	-9.95	-7.65	-5.63
Most likely outcome	-2.82	-3.84	-5.11	0.00	-3.84	-2.19	+2.04
Proportional bet	-4.11	-4.64	-6.01	-1.40	-4.73	-2.90	+0.03

4.2.2 Returns OFE

This section contains the returns from betting by using the probabilities predicted by the OFE model. Betting strategies used as described in section 3.5. Seven sets of odds are used. Tables 7 and 8 contain values from betting on the matches by using the probabilities estimated without odds and with odds respectively. Unfortunately, there are no cases of positive returns.

Table 7: OFE technique: Returns per betting strategy and bookmaker

Strategy	**Bookmaker**						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	-21.10	-20.35	-18.89	-10.66	-29.52	-28.89	-9.63
Value bet	-13.10	-12.14	-16.86	-10.94	-14.67	-12.51	-8.60
Most likely outcome	-8.16	-7.49	-5.90	-6.58	-8.03	-7.03	-6.36
Proportional bet	-6.80	-6.57	-6.82	-4.37	-7.06	-5.62	-4.17

Table 8: OFE technique with odds: Returns per betting strategy and bookmaker

Strategy	**Bookmaker**						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	-15.44	-18.61	-16.15	-9.87	-26.43	-21.43	-12.24
Value bet	-12.15	-11.44	-11.72	-7.38	-12.75	-10.68	-8.63
Most likely outcome	-5.46	-4.64	-2.79	-3.76	-5.18	-4.12	-3.71
Proportional bet	-6.07	-5.86	-5.99	-3.65	-6.37	-4.97	-3.42

4.2.3 Returns Kernel SVM

This section contains the returns from betting by using the probabilities predicted by the Kernel SVM model. Betting strategies used as described in section 3.5. Seven sets of odds are used. Tables 9 and 10 contain values from betting on the matches by using the probabilities estimated without odds and with odds respectively. Unfortunately, there are no cases of positive returns.

Table 9: Kernel SVM technique: Returns per betting strategy and bookmaker

Strategy	**Bookmaker**						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	-17.75	-20.90	-24.87	-14.84	-23.72	-16.64	-6.73
Value bet	-5.29	-4.81	-8.87	-0.41	-5.79	-5.14	-2.57
Most likely outcome	-7.85	-7.04	-5.39	-6.38	-7.66	-6.71	-5.95
Proportional bet	-5.51	-5.34	-6.10	-2.85	-5.76	-4.17	-2.25

Table 10: Kernel SVM technique with odds: Returns per betting strategy and bookmaker

Strategy	**Bookmaker**						
	Bet365	Bet&Win	Interwetten	Pinnacle	William Hill	VC Bet	Pinnacle (closing odds)
Kelly Criterion	-17.45	-22.07	-23.42	-13.76	-22.55	-18.77	-7.88
Value bet	-4.20	-5.71	-10.83	-2.25	-6.46	-4.85	-1.93
Most likely outcome	-7.71	-6.93	-5.24	-6.22	-7.54	-6.57	-5.91
Proportional bet	-5.47	-5.31	-6.05	-2.82	-5.74	-4.15	-2.23

5 Conclusion

Looking at the returns in section 4.2, it is clear that the XGBoost method is the most precise predictor of the probabilities of each outcome. It is the only technique considered that actually managed to provide reliable positive returns. The highest return for season 2018/19 was achieved by using the XGBoost with betting odds as covariates, combined with the Kelly Criterion betting strategy. This delivered positive returns for every bookmaker used, ranging from 2.19% to 6.64%. Moreover, the XGBoost method with odds as covariates also provided profits in some cases from season 2017/18. Betting through using Pinnacle closing odds, with strategies of Kelly Criterion and most likely outcome, returned consistent profits for both seasons. This answers the research question; it is possible to use a machine learning technique to gain consistently positive returns from betting on Bundesliga match outcomes.

Looking at sections 4.2 and 4.1 we can see that using the betting odds as covariates improves the predictive ability of OFE and XGBoost models, but it does not have any noticeable positive effect on the Kernel SVM hit rate. The effects of using odds are particularly important for maximizing returns, and since that is the main goal of this research, they are a valuable addition to the model.

From section 4.1, it is clear that the OFE provides the best forecast of the most likely match outcomes. The XGBoost and Kernel SVM both have lower hit rates, with XGBoost estimated without odds having the worst outcome predictions of all. This means that if someone is interested in predicting the final league table, OFE is the best technique to use. It is a surprising result, as one might suspect that a single technique should be best at predicting both outcomes and probabilities. This result may be caused by the way that the predicted outcomes are computed in this research. The probabilities are first computed and then the most likely outcome is picked. For XGBoost, the probabilities predicted are often closer to each other, than in the case of OFE probabilities. For example if for XGBoost $\hat{P}(Y_i = H) = 0.45$, $\hat{P}(Y_i = A) = 0.449$, while for OFE $\hat{P}(Y_i = H) = 0.20$, $\hat{P}(Y_i = A) = 0.75$ then XGBoost will "predict" a home team win, while OFE will "predict" an away

team win. If the away team wins, the OFE prediction would be right, but if the home team wins then we would loose a lot of money from betting through OFE. For XGBoost on the other hand, since the probabilities are more "conservative", it may be harder to predict an outcome, but in the long run the losses from placing wrong bets are simply much lower, therefore it would deliver higher returns. This is further confirmed by the OFE's performance in using the Kelly Criterion strategy, which yields the lowest returns for OFE, while being the strategy that is the most highly dependent on precisely estimated probabilities. For the XGBoost hit rate, if instead of always predicting that the most likely outcome will happen, we would predict one randomly based on the computed probabilities, the hit rates might improve.

To summarise, it is possible to gain profits from betting on match outcomes, by using the XGBoost method with odds as additional covariates. It is important to provide as many relevant explanatory variables as possible to optimize the predictions. The betting odds are a useful extension to the set of covariates. Unfortunately, Kernel SVM and OFE did not bring positive returns from betting. OFE, however, is more useful in making outcome predictions, and therefore the final league table, as the hit rates it got were by far the highest.

6 Limitations and Further Research

It is important to note that for the estimations I used at most 117 explanatory variables. Compared with over 300 variables used by Goller et al. (2018), 117 is relatively low. Some returns are already as high as 6.64%. Getting more explanatory variables could further improve the profits from betting. Moreover, Kernel SVM and OFE did not bring positive returns from betting. This may be due to the fact that the amount of variables used is relatively small. For Goller et al. (2018), who used over 300 variables in their research, the OFE returned profits in some cases. I am fascinated by the possible outcomes from using their data with XGBoost, once their paper is officially published and the data used becomes publicly available.There were also some limitations to the models used in this research. It can be seen that each model has its downsides, with XGBoost being worse in predicting outcomes, while OFE struggling with predicting precise probabilities that could yield positive returns. All the models also have other issues that need to be considered.

For XGBoost, it is really difficult to choose the right amount of iterations. Too many iterations can lead to overfitting, while too few may lead to low accuracy. Both issues can contribute to worse prediction performance. It is impossible to fix this without knowing the actual outcomes of the predicted sample, which will never be the case in practice.

The Kernel SVM also has its limitations. For this model it is also difficult to choose the right parameters and the appropriate Kernel function. This can again lead to overfitting, as I build the model in a way that could most accurately predict the training sample outcomes, but not necessarily the test sample outcomes. Moreover, the Kernel SVM model is difficult to interpret and visualize for large sets of variables (in case of 117 variables, the model uses 118 dimensions), therefore making small improvements to the model can be cumbersome.

In case of Random Forests, the first disadvantage lies again in the possible issue of overfitting, since the model is built and the parameters are chosen based on the training sample. The model can

therefore be very accurate for predicting observations within the training sample, but the predictions it makes for test sample observations may not be so good. Another issue both for Random Forests and Kernel SVMs is that they are not very good at predicting the multiclass outcomes. We therefore needed an algorithm for making the ordered predictions, like the one described in Figure 2. Lastly, the machine learning models are hard to interpret as they do not show the significance and coefficients of every parameter used. Knowing exactly which variables best contribute to accurate predictions could facilitate further research and model improvements.

References

Boser, B. & Guyon, I. & Vapnik, V. (1992). *A training algorithm for optimal margin classifiers".* Proceedings of the fifth annual workshop on Computational learning theory – COLT '92. p. 144.

Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J. (1984). *Classification and Regression Trees.* Wadsworth, Belmont, CA.

Breiman, L. (1997). *Arcing The Edge.* Technical Report 486. Statistics Department, University of California, Berkeley.

Breiman, L. (2001). *Random Forests.* Machine Learning, 45(1), pp. 5-32.

Chen, T. & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System.* In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ACM, New York, NY, USA, 785-794.

Freund, Y. and Schapire, R.E. (1996) *Experiments with a New Boosting Algorithm.* International Conference on Machine Learning, Bari, 3-6 July 1996, 148-156.

Friedman, J. & Hastie T. & Tibshirani, R. & et al. (2000). *Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors).* The annals of statistics, 28(2), 337–407.

Friedman, J. (2001). *Greedy function approximation: a gradient boosting machine.* Annals of Statistics, pp. 1189–1232.

Goller, D. & Knaus, M. & Lechner, M. & Okasa, G. (2018). *Predicting Match Outcomes in Football by an Ordered Forest Estimator.* Electronic Publication at http://www.seps.unisg.ch

Groll, A. & Ley, Ch. & Schauberger, G. & Eetvelde, H. (2018). *Prediction of the FIFA World Cup 2018 - A random forest approach with an emphasis on estimated team ability parameters.*

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning.* Elements, Volume 1.

Lechner, M. & Okasa, G. (2018). *Random Forest Estimation of the Econometric Ordered Choice Model.* Unpublished Manuscript.

Leitner, C., Zeileis, A. & Hornik, K. (2010). *Forecasting sports tournaments by ratings of (prob)abilities: A comparison for the EURO 2008.* International Journal of Forecasting.

McCullagh, P. (1980). *Regression Models for Ordinal Data. Journal of the Royal Statistical Society.* Series B (Methodological),42(2), 109-142.

Nakamura, L. R. et al. (2018). *A New Continuous Distribution on the Unit Interval Applied to Modelling the Points Ratio of Football Teams.* Journal of Applied Statistics.

Paap, R. (2019). Lecture 3: *Chapter 5, Part I An Unordered Multinomial Dependent Variable*, lecture notes, Marketing Models, FEB23002, Erasmus University Rotterdam, delivered 22 January 2019.

Vapnik, V. & Chervonenkis, A. (1974). *The theory of pattern recognition.* Nauka, Moscow

Wooldridge, J. M. (2010). *Econometric Analysis of Cross Section and Panel Data.* MIT Press Books, The MIT Press, edition 2, volume 1, number 0262232588, March.

Appendix

R code used

In this section, the R code is displayed. Every script presented here uses the data file "modata.xlsx".

The XGBoost model:

```
# Applying XGBoost:

# Getting the dataset
library(readxl)
modata = read_excel("modata.xlsx")

# Splitting into training and test samples:
training_sample = modata[modata$SeasID<11,]
test_sample = modata[modata$SeasID==11,]
## For Season 2017/18:
training_sample2 = modata[modata$SeasID<10,]
test_sample2 = modata[modata$SeasID==10,]

# Getting rid of unnecessary columns:
training_set = training_sample[, 6:98]
test_set = test_sample[, 6:98]

## For Odds as covariates:
 training_set_odds = training_sample[, 6:125]
 test_set_odds = test_sample[, 6:125]
 ### Season 2017/18:
 training_set_odds2 = training_sample2[, 6:125]
 test_set_odds2 = test_sample2[, 6:125]

training_set = training_set[, -2:-3]
test_set = test_set[, -2:-3]

training_set_odds = training_set_odds[, -2:-3]
test_set_odds = test_set_odds[, -2:-3]

training_set_odds2 = training_set_odds2[, -2:-3]
test_set_odds2 = test_set_odds2[, -2:-3]



# Setting the dependent variable as factor and then as a numeric and subtract 1,
# to ensure that there are three classes which start from 0:
# OutC = outcomes; 0=HomeWin, 1=Draw, 2=AwayWin
thelabel = as.numeric(as.factor(training_set$OutC))-1
```

```
thelabel_odds = as.numeric(as.factor(training_set_odds$OutC))-1
thelabel_odds2 = as.numeric(as.factor(training_set_odds2$OutC))-1
## Preparing and executing the xgboost:
library(xgboost)

# Preparing the data matrix, defining the dependent variable
xgbdata = xgb.DMatrix(data.matrix(training_set[, -1]),
                      label=thelabel, missing=NA)
xgbdata_odds = xgb.DMatrix(data.matrix(training_set_odds[, -1]),
                           label=thelabel_odds, missing=NA)
xgbdata_odds2 = xgb.DMatrix(data.matrix(training_set_odds2[, -1]),
                            label=thelabel_odds2, missing=NA)

xgboo = xgboost(data = xgbdata,
                objective = "multi:softprob",
                nrounds = 60,
                num_class = 3)
xgboo_odds = xgboost(data = xgbdata_odds,
                objective = "multi:softprob",
                nrounds = 60,
                num_class = 3)
xgboo_odds2 = xgboost(data = xgbdata_odds2,
                      objective = "multi:softprob",
                      nrounds = 60,
                      num_class = 3)


# Predicting probabilities of outcomes from the test set
# The following line returns a list of 891 probabilities, each three rows correspond to
# one match's probabilities of HomeWin, Draw and AwayWin respectively. Further processing
# in MATLAB.
y_pred = predict(xgboo, newdata = as.matrix(test_set[-1]))

y_pred_odds = predict(xgboo_odds, newdata = as.matrix(test_set_odds[-1]))
y_pred_odds2 = predict(xgboo_odds2, newdata = as.matrix(test_set_odds2[-1]))

# Saving the probabilities to an excel file:
library("openxlsx")
write.xlsx(y_pred, file = "OutcomeXGB.xlsx",
           sheetName = "Sheet1", append = FALSE)
write.xlsx(y_pred_odds, file = "OutcomeXGBOdds.xlsx",
           sheetName = "Sheet1", append = FALSE)
write.xlsx(y_pred_odds2, file = "OutcomeXGBOdds2.xlsx",
           sheetName = "Sheet1", append = FALSE)
```

The Ordered Forest Estimator model:

```
# OFE application:

# Getting the dataset
library(readxl)
modata = read_excel("modata.xlsx")

# Splitting into training and test samples:
training_sample = modata[modata$SeasID<11,]
test_sample = modata[modata$SeasID==11,]


# Getting rid of unnecessary columns:
training_set = training_sample[, 7:98]
test_set = test_sample[, 7:98]

## For the case using odds as covariates:
  training_set_odds = training_sample[, 7:125]
  test_set_odds = test_sample[, 7:125]


# Dividing the dataset for M-1 sets of binary outcomes (H and H&D)
#     For outcome 1 = only H
training_set_h = training_set[, -2]
test_set_h = test_set[, -2]
training_set_h_odds = training_set_odds[, -2]
test_set_h_odds = test_set_odds[, -2]
#     For outcome 2 = H or D
training_set_hd = training_set[, -1]
test_set_hd = test_set[, -1]
training_set_hd_odds = training_set_odds[, -1]
test_set_hd_odds = test_set_odds[, -1]

# Setting the dependent variables as factors:
training_set_h$H_only = factor(training_set_h$H_only, levels = c(0, 1))
training_set_hd$H_D = factor(training_set_hd$H_D, levels = c(0, 1))
training_set_h_odds$H_only = factor(training_set_h_odds$H_only, levels = c(0, 1))
training_set_hd_odds$H_D = factor(training_set_hd_odds$H_D, levels = c(0, 1))

# Looking for the optimal parameters for random forest, using grid search:
library(caret)
gs_rf_h = train(form = H_only ~ ., data = training_set_h, method = 'rf')
gs_rf_hd = train(form = H_D ~ ., data = training_set_hd, method = 'rf')
gs_rf_h_odds = train(form = H_only ~ ., data = training_set_h_odds, method = 'rf')
```

```
gs_rf_hd_odds = train(form = H_D ~ ., data = training_set_hd_odds, method = 'rf')

gs_rf_h$bestTune
gs_rf_h_odds$bestTune
# Return for H:
# Accuracy: 0.620
# mtry: 2
## For Betting odds as covariates:
# Accuracy: 0.621
# mtry: 2

gs_rf_hd$bestTune
gs_rf_hd_odds$bestTune
# Return for H&D:
# Accuracy: 0.706
# mtry: 2
## For Betting odds as covariates:
# Accuracy: 0.708
# mtry: 2

# Creating the Binomial Random Forest Estimators
library(randomForest)
set.seed(420)
ran_for_h = randomForest(x = training_set_h[,-1],
                         y = training_set_h$H_only,
                         ntree = 1000, mtry = 2)

ran_for_hd = randomForest(x = training_set_hd[-1],
                          y = training_set_hd$H_D,
                          ntree = 1000, mtry = 2)
ran_for_h_odds = randomForest(x = training_set_h_odds[,-1],
                         y = training_set_h_odds$H_only,
                         ntree = 1000, mtry = 2)

ran_for_hd_odds = randomForest(x = training_set_hd_odds[-1],
                          y = training_set_hd_odds$H_D,
                          ntree = 1000, mtry = 2)


# Computing the probabilities for possible outcomes
y_pred_h = predict(ran_for_h,
                   newdata = test_set_h[,-1],
                   type = "prob")
y_pred_hd = predict(ran_for_hd,
```

```
                    newdata = test_set_hd[,-1],
                    type = "prob")
y_pred_h_odds = predict(ran_for_h_odds,
                   newdata = test_set_h_odds[,-1],
                   type = "prob")
y_pred_hd_odds = predict(ran_for_hd_odds,
                    newdata = test_set_hd_odds[,-1],
                    type = "prob")

# Counting the probabilities of each ordered outcome:
HomeWin = y_pred_h[, 2]
Draw = y_pred_hd[, 2] - HomeWin
a = length(Draw)
fulik = replicate(a,1)
AwayWin = fulik - y_pred_hd[, 2]
probs = cbind(HomeWin, Draw, AwayWin)
## For odds:
HomeWin_odds = y_pred_h_odds[, 2]
Draw_odds = y_pred_hd_odds[, 2] - HomeWin_odds
a_odds = length(Draw_odds)
fulik_odds = replicate(a_odds,1)
AwayWin_odds = fulik_odds - y_pred_hd_odds[, 2]
probs_odds = cbind(HomeWin_odds, Draw_odds, AwayWin_odds)

# Changing the negative probabilities to 0 and making sure each row sums to 1
probs = ifelse(probs < 0, 0, probs)
probs = probs/rowSums(probs)

probs_odds = ifelse(probs_odds < 0, 0, probs_odds)
probs_odds = probs_odds/rowSums(probs_odds)

# Saving the probabilities to an excel file:
library("openxlsx")
 write.xlsx(probs, file = "OutcomeOFE.xlsx",
            sheetName = "Sheet1", append = FALSE)
write.xlsx(probs_odds, file = "OutcomeOFEOdds.xlsx",
           sheetName = "Sheet1", append = FALSE)
```

The Kernel Support Vector Machine model:

```
# Ordered Kernel SVM application:

# Getting the dataset
library(readxl)
modata = read_excel("modata.xlsx")

# Splitting into training and test samples:
training_sample = modata[modata$SeasID<11,]
test_sample = modata[modata$SeasID==11,]

# Getting rid of unnecessary columns:
 training_set = training_sample[, 7:98]
 test_set = test_sample[, 7:98]
## For the case using odds as covariates:
training_set_odds = training_sample[, 7:125]
test_set_odds = test_sample[, 7:125]

# Scaling the continuous variables, since Kernel SVM requires that:
library(robustHD)
training_set[,-1:-16] = standardize(training_set[,-1:-16])
test_set[,-1:-16] = standardize(test_set[,-1:-16])

training_set_odds[,-1:-16] = standardize(training_set_odds[,-1:-16])
test_set_odds[,-1:-16] = standardize(test_set_odds[,-1:-16])

# Dividing the dataset for M-1 sets of binary outcomes (H and H&D)
#     For outcome 1 = only H
training_set_h = training_set[, -2]
test_set_h = test_set[, -2]
training_set_h_odds = training_set_odds[, -2]
test_set_h_odds = test_set_odds[, -2]
#     For outcome 2 = H or D
training_set_hd = training_set[, -1]
test_set_hd = test_set[, -1]
training_set_hd_odds = training_set_odds[, -1]
test_set_hd_odds = test_set_odds[, -1]

# Setting the dependent variables as factors:
training_set_h$H_only = factor(training_set_h$H_only, levels = c(0, 1))
training_set_hd$H_D = factor(training_set_hd$H_D, levels = c(0, 1))
training_set_h_odds$H_only = factor(training_set_h_odds$H_only, levels = c(0, 1))
training_set_hd_odds$H_D = factor(training_set_hd_odds$H_D, levels = c(0, 1))
```

```
# # Applying the grid search to find optimal parameters:
library(caret)
# Method: Radial
gs_svm_h1 = train(form = H_only ~ ., data = training_set_h, method = 'svmRadial')
gs_svm_h1_odds = train(form = H_only ~ ., data = training_set_h_odds, method = 'svmRadial')
gs_svm_h1$bestTune
gs_svm_h1_odds$bestTune
# returns: sigma = 0.00720, C = 0.25
# accuracy: 0.620
## For odds:
# returns: sigma = 0.00643, C = 0.25
# accuracy: 0.631
gs_svm_hd1 = train(form = H_D ~ ., data = training_set_hd, method = 'svmRadial')
gs_svm_hd1_odds = train(form = H_D ~ ., data = training_set_hd_odds, method = 'svmRadial')
gs_svm_hd1$bestTune
gs_svm_hd1_odds$bestTune
# returns: sigma = 0.00731, C = 0.25
# accuracy: 0.718
## For odds:
# returns: sigma = 0.00628, C = 0.25
# accuracy: 0.719

# Creating the Binomial Kernel SVM estimators:
set.seed(420)
library(e1071)
ksvm_h = svm(formula = H_only ~ .,
             data = training_set_h,
             type = 'C-classification',
             kernel = 'radial',
             probability = TRUE,
             sigma = 0.00720,
             C = 0.25)
ksvm_hd = svm(formula = H_D ~ .,
              data = training_set_hd,
              type = 'C-classification',
              kernel = 'radial',
              probability = TRUE,
              sigma = 0.00731,
              C = 0.25)

# For odds as covariates:
ksvm_h_odds = svm(formula = H_only ~ .,
             data = training_set_h_odds,
             type = 'C-classification',
```

```
              kernel = 'radial',
              probability = TRUE,
              sigma = 0.00643,
              C = 0.25)
ksvm_hd_odds = svm(formula = H_D ~ .,
               data = training_set_hd_odds,
               type = 'C-classification',
               kernel = 'radial',
               probability = TRUE,
               sigma = 0.00628,
               C = 0.25)

# Computing the probabilities for possible outcomes
y_pred_h = predict(ksvm_h,
                   newdata = test_set_h[,-1],
                   probability = TRUE)
y_pred_hd = predict(ksvm_hd,
                    newdata = test_set_hd[,-1],
                    probability = TRUE)
probs_h = attr(y_pred_h,"probabilities")
probs_hd = attr(y_pred_hd,"probabilities")

y_pred_h_odds = predict(ksvm_h_odds,
                   newdata = test_set_h_odds[,-1],
                   probability = TRUE)
y_pred_hd_odds = predict(ksvm_hd_odds,
                    newdata = test_set_hd_odds[,-1],
                    probability = TRUE)
probs_h_odds = attr(y_pred_h_odds,"probabilities")
probs_hd_odds = attr(y_pred_hd_odds,"probabilities")

# Counting the probabilities of each ordered outcome:
HomeWin = probs_h[, 2]
Draw = probs_hd[, 1] - HomeWin
a = length(Draw)
fulik = replicate(a,1)
AwayWin = fulik - probs_hd[, 1]
probs = cbind(HomeWin, Draw, AwayWin)

HomeWin_odds = probs_h_odds[, 2]
Draw_odds = probs_hd_odds[, 1] - HomeWin_odds
a_odds = length(Draw_odds)
fulik_odds = replicate(a_odds,1)
AwayWin_odds = fulik_odds - probs_hd_odds[, 1]
```

```
probs_odds = cbind(HomeWin_odds, Draw_odds, AwayWin_odds)

# Changing the negative probabilities to 0 and making sure each row sums to 1
probs = ifelse(probs < 0, 0, probs)
probs = probs/rowSums(probs)
probs_odds = ifelse(probs_odds < 0, 0, probs_odds)
probs_odds = probs_odds/rowSums(probs_odds)

# Saving the probabilities to an excel file:
library("openxlsx")
write.xlsx(probs, file = "OutcomeKSVM.xlsx",
           sheetName = "Sheet1", append = FALSE)
## For odds as covariates:
write.xlsx(probs_odds, file = "OutcomeKSVMOdds.xlsx",
          sheetName = "Sheet1", append = FALSE)
```

Matlab code used

This section contains the matlab codes used for processing of the estimated probabilities. Three scripts are provided.

Script 1: editXGB.m

This script changes the one-column output of the R XGBoost into three columns, one for each probability:

```
% Fixing the XGB output:

% 1) import the OutcomeXGB.xlsx file
% Or: Import the OutcomeXGBOdds.xlsx file

%  rawout = OutcomeXGB;
 rawout = OutcomeXGBOdds2;

a = size(rawout,1)/3;

home = zeros(a,1);
draw = zeros(a,1);
away = zeros(a,1);

for i = 1:a
home(i,1) = rawout{3*i-2,1};
draw(i,1) = rawout{3*i-1,1};
away(i,1) = rawout{3*i,1};
end

probs = table(home,draw,away)
```

```
 % writetable(probs,"OutcomeEdXGB.xlsx",'Sheet',1)
% Or, for the estimation using Odds:
 writetable(probs,"OutcomeEdXGBOdds.xlsx",'Sheet',1)
```

Script 2: BettingStrategies.m

This script computes the predicted returns based on a set of estimated probabilities and the set of odds.

```
% Making bets through the Kelly Criterion

% 1) Import the file Odds.xlsx
% 2) Import the file OutcomeEdXGB.xlsx/OutcomeOFE.xlsx/OutcomeKSVM.xlsx, or
% the respective outcomes including Odds,
% depending on which one we wish to use.
% Remember to edit the OutcomeXGB.xlsx file first with editXGB.m code

%Taking the odds:
dejta = Odds;


% PR is the set of probabilities for each outcome
PR = OutcomeEdXGBOdds2;


% OD is the current set of odds used from the data, with H, D and A odds in
% each column respectively
od1 = dejta(:,8:10);
od2 = dejta(:,11:13);
od3 = dejta(:,14:16);
od4 = dejta(:,17:19);
od5 = dejta(:,20:22);
od6 = dejta(:,23:25);
od7 = dejta(:,26:28);

% Picking the set of odds that will be analysed:
OD = od7;

% Positive returns:
%XGBoost: od4 and od7 for strategy 2
%XGBoostOdds: [Strategy 1: all sites]; od4: str 1 & 3;  od7 for str 1-3;

% The table of dummy variables for each outcome (actual):
outC = dejta(:,4:6);

z = size(PR,1);

ddddd=0;
betsum = 0;
betsum2 = 0;
```

```
wins = 0;
wins2 = 0;
betsum3 = 0;
wins3 = 0;
betsum4 = 0;
wins4 = 0;
prh = PR(:,1);
prd = PR(:,2);
pra = PR(:,3);

hodds = OD(:,1);
dodds = OD(:,2);
aodds = OD(:,3);

hot = outC(:,1);
dot = outC(:,2);
aot = outC(:,3);
% Kelly Criterion:
for i = 1:z
    for j = 1:3
        odds = OD{i,j};
        outcome = outC{i,j};
        prob = PR{i,j};
        q = 1-prob;
        b = odds-1;

        kelly = (b*prob-q)/b;
        if(kelly > 0.1)
            ddddd=ddddd+1
            betsize = kelly*10;
            betsum = betsum+betsize;
            if(outcome>0)
            wins = wins+(odds*betsize);
            end
        end
    end
end

% Value Bet:
for i = 1:z
    for j = 1:3
        odds = OD{i,j};
        outcome = outC{i,j};
        prob = PR{i,j};
```

```
        if(~isnan(odds) && (odds~=0))
        prodds = 1/odds;
        else
            prodds = 1;
        end
        if(prob > prodds)
            betsize = 1;
            betsum2 = betsum2+betsize;
            if(outcome>0 && ~isnan(odds))
            wins2 = wins2+(odds*betsize);
            end
        end
    end
end
% Bet only if prob > 0.5
for i = 1:z
    for j = 1:3
        odds = OD{i,j};
        outcome = outC{i,j};
        prob = PR{i,j};

        if(prob > 0.5)
            betsize = 1;
            betsum3 = betsum3+betsize;
            if(outcome>0 && ~isnan(odds))
            wins3 = wins3+(odds*betsize)
            end
        end
    end
end
% Proportional betting
for i = 1:z
    for j = 1:3
        odds = OD{i,j};
        outcome = outC{i,j};
        prob = PR{i,j};


            betsize = prob;
            betsum4 = betsum4+betsize;
            if(outcome>0 && ~isnan(odds))
            wins4 = wins4+(odds*betsize)
            end
```

```
    end
end
returns1 = (wins/betsum - 1) * 100
returns2 = (wins2/betsum2 - 1) * 100
returns3 = (wins3/betsum3 - 1) * 100
returns4 = (wins4/betsum4 - 1) * 100
```

Script 3: CMOR.m

This code computes confusion matrices and odds ratios of every prediction, based on the estimated probabilities:

```
% Computing Confusion Matrices and Odds Ratios

% Step 1: Import the appropriate outcome set (remember to
% edit the XGB outcome files first in EditXGB.m)
probs = OutcomeEdXGB;
% Step 2: Import the set Odds.xlsx (as it contains actual outcomes)
outcomes = Odds(:,4:6);

% Computing predicted outcomes, based on the most likely class:
z = size(probs,1);
predo = zeros(z,3);
for i = 1:z
    m = 1;
    for j = 1:3
        if probs{i,m} < probs{i,j}
            m = j;
        end
    end
    predo(i,m) = 1;
end

% Confusion Matrix:
% Rows: Actual, Columns: predicted
CM = zeros(3);
for i = 1:z
    for j = 1:3
        for h = 1:3
            if predo(i,j) == 1
                if outcomes{i,h} == 1
                    CM(h,j) = CM(h,j)+1;
                end
            end
        end
    end
end

% Hit Rate:
HR = (CM(1,1)+CM(2,2)+CM(3,3))/306

% Number of each outcome:
homes = CM(1,1)+CM(1,2)+CM(1,3);
```

```
draws = CM(2,1)+CM(2,2)+CM(2,3);
aways = CM(3,1)+CM(3,2)+CM(3,3);

% "True" probabilities estimations:
prh = homes/306;
prd = draws/306;
pra = aways/306;
% Better-than-random Prediction test statistic:
q = prh^2 + prd^2 + pra^2
ST = (HR -q)/sqrt(q*(1-q)/306)
if ST > 1.645
resul = "reject␣(better)";
else
    resul = "Don't␣reject"
end
```

Other codes

There are three other files, all written in Java, that are too large to include in this paper, therefore they were attached separately.

The java package considered is called OrderedForestEstimation. The files with test data were included. The test data is comprised of 500 observations with 19 independent variables, a subsample taken from the dataset used in the rest of the paper.

The package contains the following .java files:

1. DecisionTree.java is a file that contains the manually programmed decision tree algorithm.
2. RandomForest.java is a file that contains the manually programmed random forest estimator.
3. OrderedForestEstimator.java is a file that contains the implementation of the OFE technique.