Generating Tennis Betting Odds Based On Player Features

Jonas Röser, Cornelius Schramm, Leonidas Graf v. Bothmer, Laura Walser

December 3, 2018

1 Introduction

The world wide web contains a wide range of betting websites nowadays. Bookmakers all over the world offer, for a given sports game, separately calculated odds, taking into account commissions and margins. Our project targets on calculating our very own betting odds based on our algorithm's predicted probabilities of each player winning the game.

2 Project Selection

When looking for projects we discussed four possible categories: politics, health, consumer goods and sports. In the end we decided to go for the sports topic, because that provided us with a wide range of available data sets and therefore a lot of possibilities. We picked tennis as the sport of our choice because we have special expertise in this field and are all passionate about it. Furthermore it was an advantage that there are only two possible match outcomes. Anything with more than two outcomes (e.g. a draw as a third possible outcome) would have complicated constructing the classifier and calculating the betting odds.

3 Data

3.1 Data Selection

After a bit of research we found the data sets we needed on Datahub [2]. The listed data sets are based on web scraped data from the official ATP world tour website [5] which is frequently updated. From the available sets, we decided to work with the following ones:

- Match Scores and Stats from 1991 to 2016
- Match Scores and Stats from 2017
- Player Overviews
- Rankings from 1973 to 2017
- Tournaments from 1877 to 2017

Since they were all provided by the same source, the structures of the data sets were already quite similar, which made working with them and merging them much easier.

3.2 Data Cleaning

After having decided on what data to use, we started cleaning and merging the data into a useful form. Since our *Match Scores* and *Match Stats* data sets only contained data from 1991 to 2017 we took that as a reference and cut all other data sets such as they would correspond to that time span. In a second step we tried to merge our sets together and make one big data set out of it. In order to achieve this we had to undergo quite a bit of data cleaning such as transforming date format and using different identifiers to merge the data. A major issue was, that our *Rankings* data set contained the weekly updated rankings of each player from 1991 to 2017 and that the match-date and the date of the update mostly wasn't the same. So in order to match *Match Scores* with *Rankings* we had to match *Match Scores* with *Tournaments* first. After that we were able to merge the sets by seeking for the most current ranking available on the match date and assigning it to each player. A big issue was the fact that our Dataset was structured in a way that the first column always contained the winner and the second column the loosing one. For us to be able to properly train a classification algorithm we had to cleverly reshuffle and create a column with the corresponding numerical value (0 if player 0 won, 1 if player 1 won).

4 Features

4.1 Feature Selection

Looking at our cleaned data set we looked at the already available features and decided to use the features *age*, *height* and *rank*. We decided not to use features like: *handedness* or *when turned pro* because of too many missing values.

4.2 Feature Construction

We've also constructed new features based on the information in our data set. The feature *weighted form* has been constructed by looking at the player's wins from the last ten, five or one matches and assigning the weighted sum of these to the column that corresponds to the player. If at the date in question the player hasn't played enough matches yet, a "NA" was assigned. Constructing the *head to head* feature, we tried three different versions to minimize the variance and to prevent a math error:

- **Version One:** Assigning 0 to both of the players if they've never played against each other before and calculating the winning rate for each player by the following equation:

$$\text{head to head} = \frac{\text{number of wins player x}}{\text{total of matches against player y}} \tag{1}$$

  But with this method we encountered situations where a player who's played only once against another player and lost, got the same value as a player who's played 20 times against another player and lost all the time.

- **Version Two:** Adding 1 to the number of wins of each player, adding 2 to the total number of matches played against each other and then use the calculation from version one. Like this we were able to prevent the situations that we've encountered above and keep the probability format as it was.

- **Version Three:** Adding 1 to the number of wins of each player and calculating the form by the following equation:

$$\text{head to head} = \frac{(\text{number of wins player x}) + 1}{(\text{number of wins player y}) + 1} \tag{2}$$

We got the best results using the third version, which was why we settled for that one.
Other features we constructed are: *condition wins*, which takes into account the number of total past wins under the respective condition (indoor or outdoor) and the court surface (clay, carpet, grass or hard court); *fatigue*, which adds up the durations of the player's previous two games in the tournament and weights the game(t-1) with 1 and the game(t-2) with $\frac{1}{3}$; *titles*, which shows how many titles the player has already won up to this point; *home game*, which indicates whether the match was taking place in the home country of a player or not. The latter meant manually looking up and typing in the translation of each of the 206 location abbreviations contained in the data set. This took quite some time and nerves.
Finally we had nine features to feed our algorithms with. We soon noticed, that in general we

Table 1: Final Feature Selection

	Final Features
1	rank
2	rank move
3	form
4	head to head
5	condition wins
6	home town
7	age difference
8	height difference
9	fatigue

get better accuracies by using differences of feature values rather than using absolute values for each player, so we started using them wherever it made sense.

5 Data Analysis

In order to get the highest accuracy, we trained and tested four methods on our standardized final data set. It was not feasible for our machines to optimize the hyper-parameters and the feature selection simultaneously, which would have allowed us to fully maximize the performance of our algorithms by trying out all possible feature combinations.
So we were left with the issues of which optimization to do first and of how to treat the interaction of those optimizations with the k-fold cross validation. In order to avoid a too positive bias when optimizing and to achieve a slightly negative one we used the method shown in Figure 1. Our machines couldn't compute the k-fold cross validation for the random forests for the whole 70% of our data at once though, so we had to split it up into three parts and then take the average as our final measure. We applied this method similarly to the other models for better comparability. That came with the additional benefit of further reducing the risk of a positive bias.

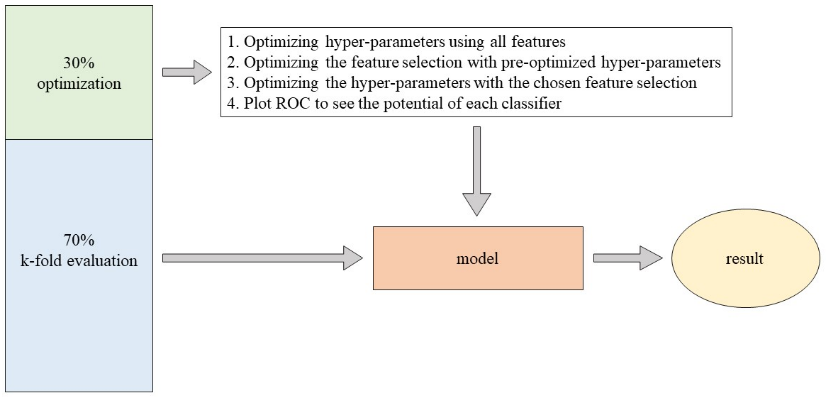
30%
optimization
70%
k-fold evaluation
1. Optimizing hyper-parameters using all features
2. Optimizing the feature selection with pre-optimized hyper-parameters
3. Optimizing the hyper-parameters with the chosen feature selection
4. Plot ROC to see the potential of each classifier
model
result

Figure 1: Hyper-Parameter Optimization and Feature Selection

5.1 Random Forests

5.1.1 Bagging

First we tried a random forest with bagging, which was based on Marc Schöni's recommendation to use it in order to get a glance on what is possible with a particular data set. When creating a random forest you take a bootstrapped version of your data, which has the same length as your original set and feed the algorithm with it. Certain rows can appear multiple times and some won't appear at all. The algorithm will then build a large number of decision trees and stops creating branches for them once there is no further improvement. To get the model's accuracy the algorithm feeds it with the points, that have not been featured in the random subset. The percentage of then misclassified points is called the *out of bag error rate (OOB)* and can be used as a measure.[6]
For Bagging we optimized the hyper-parameters: *number of feature columns to be randomly considered for each internal node* and *number of trees* (see figure 2 and 3). Furthermore, we got the best testing accuracy using all features except *fatigue* for training (see table 2).

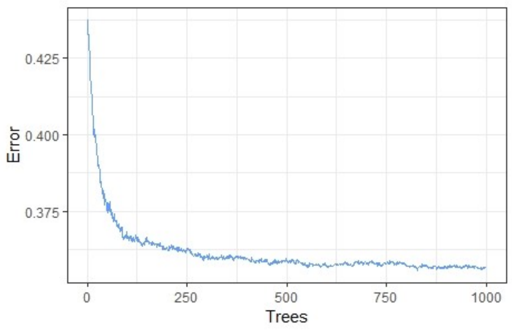

Figure 2: OOB Random Forest 1

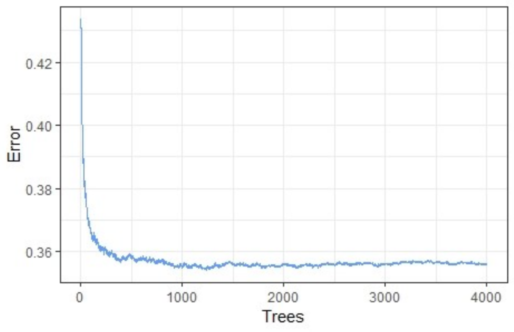

Figure 3: OOB Random Forest 2

5.1.2 Boosting

In addition to bagging we performed a boosting method on our data, which targets on increasing the model accuracy by converting weak learners into strong ones. First, the algorithm assigns a weight of one to all data points, after running the first model M_0 it then increases the weights of the misclassified points and additionally normalizes the weights such as they add up to one. For the next run M_1, the selection of points is influenced by the weights. These steps are repeated several times, creating n models and weighing them according to their performance (higher weight for better performance). At the end a vote is made based on the model weights. [4]
With boosting and also using all features except *fatigue* we got a slightly higher testing accuracy than we did with bagging (see table 2).
We also found, that both random forests performed better with a large number of trees (2000) and using only the selected features than when we used a smaller number of trees (500) but all of the features.

5.2 Logistic Regression

We used the logistic regression to try out all of the possible 511 features combinations. We then picked the best ten for the feature optimization of the other models. This model calculates an output between 0 and 1 using the given variables. For every variable a coefficient *beta* is created additionally to the intercept. Using these betas it's possible to calculate the probability of predicting player 0 or player 1 as a winner. We got the highest training accuracy by using all features but *head to head*. This gave us the testing accuracy shown in table 2. As you see it turned out to be lower than with both versions of random forests. [3]

5.3 Neural Networks

Our neural network performed its optimization using three hidden layers with 6, 5 and 6 neurons respectively and minimizing its loss function via back propagation. We got the best accuracy using all features but *home game*. The mean accuracy we got out of the neural network was better than the one we got out of the Logistic Regression but worse than for both random forest methods (see table 2). [1]

6 Observations

In the end the boosted random forest gave us the highest testing accuracy. When looking at the area under the models' ROC curves, which were computed with 30% of the data and the model accuracies, which were computed with 70% of the data, we noticed, that for the biggest area under ROC we also got the highest accuracy. This made us confident that the boosted random forest was in fact the best model, so naturally we took this as the final one.

While optimizing our algorithms we found, that the algorithms maximized their performance

Table 2: Area under ROC and Accuracies

	Random Forest Bagging	Random Forest Boosted	Logistic Regression	Neural Network
Area under ROC	0.7085	0.7327	0.7101	0.7158
Testing Accuracy	0.6529	0.6541	0.6458	0.6499

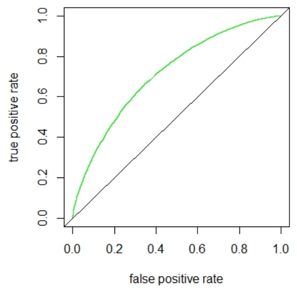

Figure 4: NN

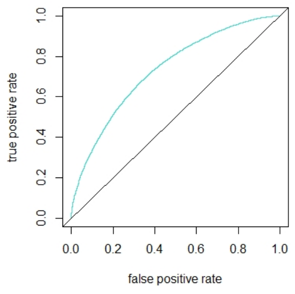

Figure 5: RFBst

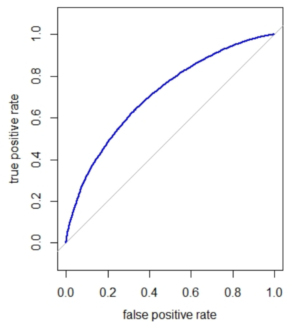

Figure 6: GLM

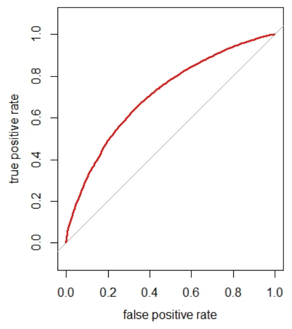

Figure 7: RFBag

under different sets of features but all of them tended to perform better with most of the available features used. We also observed, that when using the non-shuffled data set, training with older dates and testing with newer dates, we got a higher testing accuracy. This may be due to the ranking reforms in 1997 and 2009, making the newer rankings a better reflection of the player's level of play. As a consequence we randomly shuffled the rows in our data set.

6.1 Betting Odds

The final model provided us with the probabilities of each player winning. Recalling our main target, we had to transform said probabilities into betting odds. There are three types of betting odds: *American Odds*, *Fractional Odds* and *Decimal Odds*. We decided to use the latter because on the one hand those are quite easy to understand and on the other hand they are widely spread in central Europe. Their format is a decimal number greater than one. The total payout is calculated by just multiplying the stake with the odd itself. Calculating the decimal odd from a given probability is done as follows: $\text{odd} = \frac{1}{\text{probability}}$. If we calculated the odds in that way, we'd have a fair game and winnings would on either side equal zero in the long term. But as bookmakers, we also want to earn money with our betting website, so we included a 5% margin and changed our odds calculation to: $\text{odd} = \frac{1}{\text{probability}*(1+\text{margin})}$. This way we could make sure to earn 100% of every lost bet and 5% of every won bet.

7 Conclusion

Though a 65.5% accuracy may not seem too overwhelming, there are a couple of things to consider. The first thing is, that this may be a reflection of our negative bias, which is due to the method we have chosen. Furthermore we are fairly confident that if we had a greater timespan and more computational resources available it would have been feasible to push that close to 70% (by constructing more features and tuning all parameters more rigorously) but not much further beyond that. The reason for that is a certain degree of randomness and unpredictability in match outcomes. If over the long run people's bets are distributed evenly on both players we should generate a 5% average return. That is, of course, unless someone is able to make better predictions than our algorithms.

References

[1] Jiri Pospichal Daniel Svozil Vladimir Kvasnicka. “Introduction to multi-layer feed-forward neural networks”. In: *Chemometrics and intelligent laboratory systems* (1997).

[2] Datahub. *ATP World Tour tennis data*. 2018. URL: https://datahub.io/sports-data/atp-world-tour-tennis-data.

[3] C. Mitchell Dayton. *Logistic Regression Analysis*. 1992.

[4] Yohei Mishina Ryuei Murata Yuji Yamauchi Takayoshi Yamashita Hironobu FujiYoshi. *Boosted Random Forest*. 2015.

[5] ATP Tour Inc. *ATP World Tour website*. 2017. URL: https://www.atpworldtour.com/.

[6] Leo Breiman. *Random Forest*. 2001