BACHELOR THESIS

STRATEGISK ANALYSE OG SYSTEM DESIGN

ANALYSIS OF TENNIS MATCHES USING BIG DATA AND MACHINE LEARNING

Thesis by:

Hans Christian Bechsøfft Mikkelsen	s184294
Martin Illum	s184286

Main supervisors: Line Katrine Harder Clemmensen

Co supervisors: Emil Hovad - I Tracket ApS

03-05/2021

Abstract

Tennis is one of the worlds biggest and most popular sports. Multiple researchers have tried to model the outcome of matches using probability modelling or machine learning approaches. The approach presented in this thesis, is to examine whether it is possible to predict the outcomes of points, games and sets in tennis matches, by the sole use historical data from the matches and the players rank. From the models it will be tried to discover important strategic factors for winning points, games and sets. The historical data is from the years 2016 to 2020 in the two Grand Slam tournaments, Wimbledon and US Open resulting in a total 709 matches. Predictions are made by the use of multiple advanced machine learning methods such as, XGBoost, random forest, ADABoost and logistic regression where the results are compared with statistics. An evaluation of the results showed that the models for points only proved to be a fraction better than plain statistics. The average accuracy score for the game models was 76.8% and for set models 70.48%. However, with the applied data, the approach presented is not ideal for examining which factors are important for the outcomes of tennis matches.

Contents

1 Introduction

Tennis is one of the world's most popular sports, with an estimate of a billion fans worldwide (*Top 10 Most Popular Sports in The World* 2021). The Association of Tennis Professionals (ATP) Tour features more than 60 tournaments in over 30 countries with over 1500 professional players each year, which makes the competition really tough. Many players are therefore striving to gain an edge over their opponents. This causes an ever increasing demand for scientific research, mainly through data science, on how players can achieve success. Rafeal Nadal the 20 times Grand Slam winner said just before the Wimbledon final in 2008: *"If I have to hit a ball twenty times to Federer's backhand, I'll hit it twenty times, not nineteen. If I have to wait for the rally to stretch to ten shots or twelve or fifteen to bide my chance to hit a winner, I'll wait. There are moments when you have a chance to go for winning drive , but you have 70 percent chance of succeeding; you wait five shots more and your odds will have improved to 85 percent. So be alert, be patience, don't be rash."* (*Tennis Quotes by Nadal* n.d.).

The baseball team Oakland Athletics consistently appeared in the Mayor League Baseball playoffs in the early 2000's even though, they were amongst the teams with the lowest salary payroll in the entire league. The Oakland Athletics achieved these results, by redefining how they evaluated and scouted players. They used statistics and data science to evaluate players performances and therefore they could scout cheap and high performing players, who other teams would not identify and thereby gaining a edge. This was the birth of the "*Moneyball*" era in baseball. Former Team Djokovic strategy analyst Craig O'Shannessy thinks that the term "*Moneyball*" can be connect with modern tennis (O'Shannessy 2019). Others sport scientist like Stephaine Kovalchik thinks that tennis is lacking behind other sports, like baseball and due to the lack of data describing the actual gameplay, it is to early to say that tennis is in the "*Moneyball*" era (Kovalchik 2021).

This thesis uses data from Jeff Sackmanns github page, who is one of the most recognised promoters of tennis data analysis in the past decade. To investigate if data from Grand Slam tennis matches can be used to predict the outcome of tennis points, games and sets, various machine learning methods will be applied to the data. This approach will lead to the following problem statement:

2 Problem statement

Can the outcome of points, games and sets in men's single Grand Slam tennis matches, be predicted using data from the matches.

In order to answer the problem statement, the following sub-questions have been identified:

- How can the outcome of points, games and sets be modelled using data from matches?
- Which factors are important to determine the winner of points, games and sets in the models?
- How can the models performance be evaluated?

The purpose of this thesis is to examine whether, it is possible to predict the outcome of the points, games and sets in tennis matches, using historical data from tennis matches. A further investigation will be conducted to examine, which factors are the most important for the predictions, in order to

discover potentially strategic advantages. The thesis will focus on men's single Grand Slam matches and therefore the highest level within men's tennis. With inspiration from other literature on the subject, machine learning methods and evaluation metrics will be chosen.

3 Litterateur review & Related works

For tennis match predictions, most studies have aimed at applying statistical models to predict match winners. Klaassen and Magnus (Klaassen & Magnus 2001) showed that points in tennis was not independently and identically distributed (i.i.d). However, they concluded that the deviations from i.i.d was small, and therefore the i.i.d hypothesis would provide a good approximation in many cases. Other studies have used machine learning, to predict the outcome of matches. For example did Clark and Dyte (Clarke & Dyte 2000) fit a logistic regression model to the difference in the ATP rankings of players, to estimate the chance of winning, as a function of the difference in rating points.

Researchers have worked in the fields of predicting the outcome of tennis matches, using past statistical data records. A study by Ghosh, Sadhu, Biswas et al (Ghosh et al. 2019) used past statistical data from Grand Slams, to predict the outcome of tennis singles matches, and evaluated the predictions performance using metrics such as recall, precision, F1 score etc. They found out, that after evaluating three classifier methods: decisions tree, learning vector quantization and support vector machine, with the various metrics that the decisions tree classifier, were superior to the other.

There have not been many studies which explore which factors that relate to victory in tennis. Tennis is a strategic sport and therefore various factors, can influence whether a player wins or losses. A study by Ma, Liu, Tan and Ma (Ma et al. 2013) used a logistic regression model with 16 variables, which were related to player and match characteristics, to predict match outcomes and estimate victory-related factors. Their model explained 79.4% of the variance in match outcomes and had a accuracy of 90.6%. They found out that the most important victory-related factor, was the percentage of first service points won. They also discovered that the positive effect of height, diminishes when players are taller than 186 cm. However, the victory-related factors discovered in the study are not particularly useful for strategic advice. For example the factor first service win percentage does not give a explicit way to improve the first service win percentage.

A recent study by Makino, Odaka, Kuroiwa, Suwa and Shirai (Makino et al. 2020) used the frequency of single shots, two-shots patterns and effective shot patterns from each point rally of ATP matches as features in their L1-regularized logistic regression model to predict point winners and useful tactical factors. All points used in the study was either served by the players: Roger Federer, Rafael Nadal and Andy Murray. The highest accuracy they obtained was 66.5%.

There have also been used machine learning in other sports to predict or improve results. Especially American sports such as baseball, basketball etc. have a long history in collecting and analysing match data. For example, did Oughali, Bahloul and El Rahman (Oughali et al. 2019) use Random Forest and XGBoost models to predict the shooting success of basketball players in the National Basketball Association (NBA). They found out that factors such as, how far away the shot was taken and the time on the shot clock, were important for whether the player made the shot. Their model accuracy for respectively

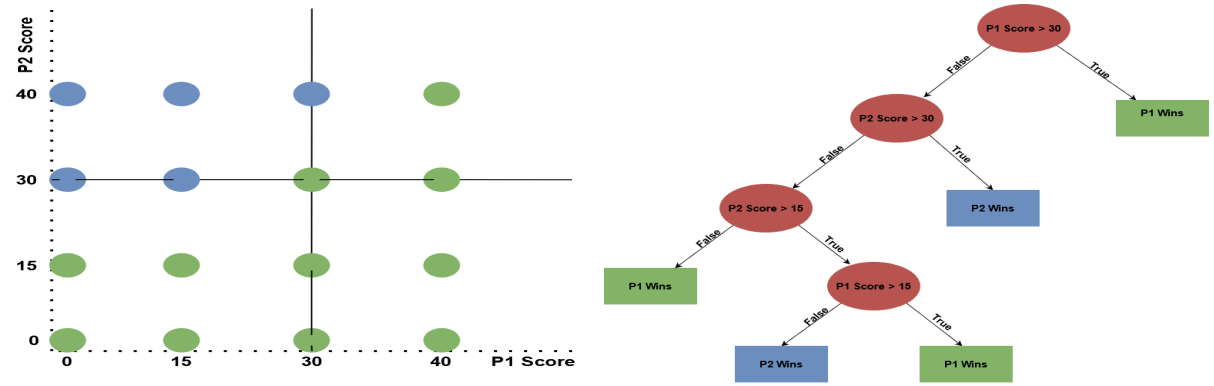
random forest and XGboost were 57% and 62%.

4 Methodology

In this section, a description of the methods used to create and evaluate the models will be given, as well as a description of the software tools used in this thesis.

4.1 Decision Trees

Tree estimators are a family of very interpretable supervised machine learning models. Identically for all tree estimators is the decision trees from which they are built. The decision tree's applied for this thesis are binary tree classifiers, where the goal is to predict a discrete output label y_i for observation i, based on the features of x_i Decision tree models are created using two steps, induction and pruning. Induction is the creation of the tree's, which is done by continuously splitting the input data into two subsets, based upon each attributes condition. Pruning is the process of removing any unnecessary information from the tree, which reduces the complexity of the tree. In Figure 1b a decision tree is illustrated and its feature space in Figure 1a.



(a) Feature space *(b) Decision tree*

Figure 1: Decision tree example and its feature space illustrated, green color represent P1 wins and blue color represents P2 Wins

4.2 Ensemble Methods

Ensemble methods are methods, which trains multiple models and then combines their output to acquire a better predictive performance than the models would have on their own. There are multiple advantages of ensemble methods. To average multiple classifiers will result in a classifier, with reduced variance compared to each unique classifier. This permits the use of classifiers with higher variance, which results in better differentiating of observations, while being less prone to overfitting. The ensemble of classifiers facilitates the possibility for classifiers, which can distinguish between two very alike observations, which the majority of the classifiers would identify as the same class (Herlau et al. 2016).

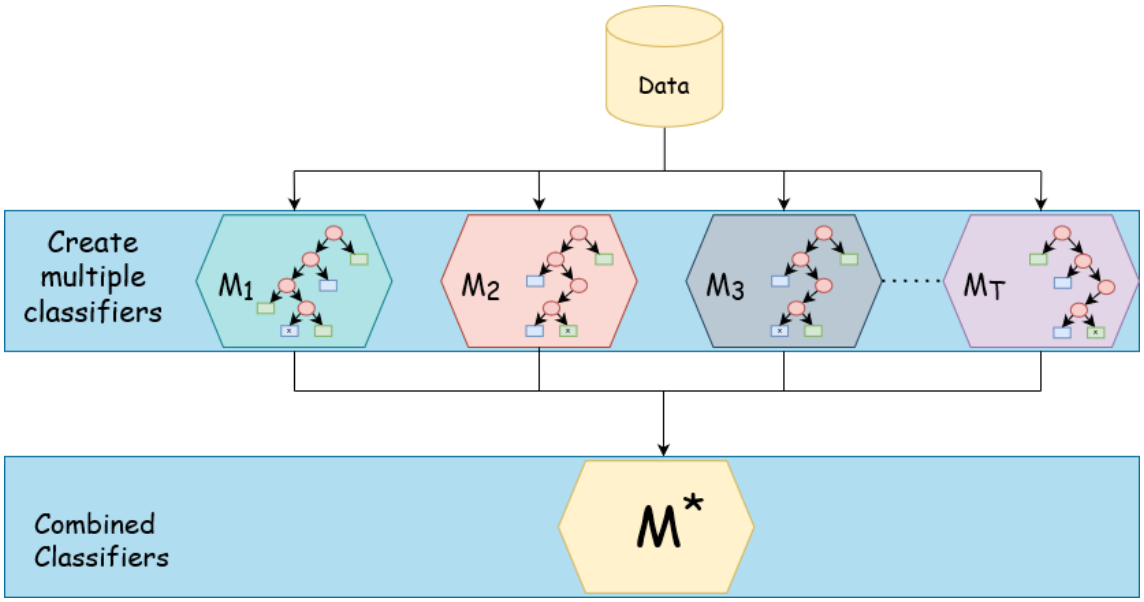


Figure 2: Combining T, M_1, ...M_T models to a single classifier M^*

Unlike ensembles methods for regression models, where the average of the estimates from the classifiers can be taken for an output, classification models uses ”majority voting” to produce an output by selecting the class, which the majority of the classifiers estimates it to be. In the event of equal votes, a random draw between the two classes is made.

4.2.1 Boost and Bagging

Boosting refers to a group of algorithms that converts weak classifiers, to strong classifiers. Boosting focuses on the observations in a data set which are hard to classify, this can be done by representing more hard observations to the classifiers, which results in more classifiers that are suited to solve the difficult observations of the classification problem.

Bagging is an ensemble technique, mainly used for reducing the variance of the predictions, by combining multiple classifiers modelled on different sub-samples of the same data set. The key elements of bagging is to create multiple sub data sets from the original data, for which multiple classifiers are built upon, this is done by building classifiers for each of the sub data sets. Finally the classifiers are combined, which results in a stronger classifier with less variance.

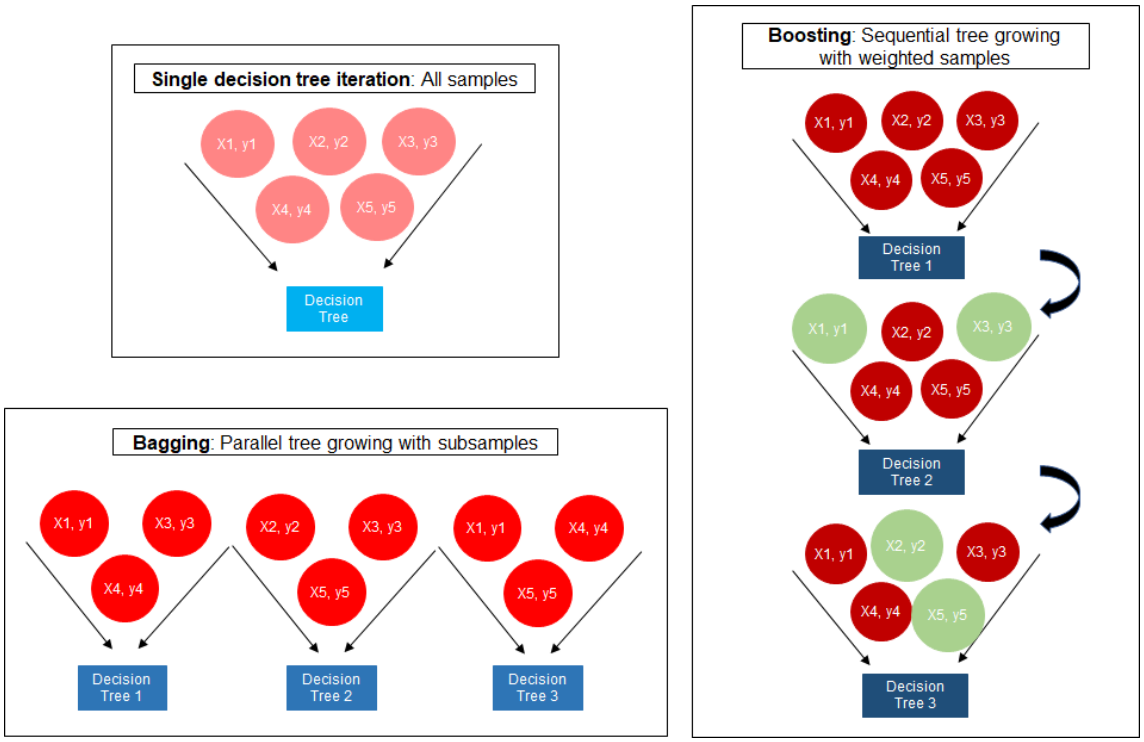


Figure 3: Single classifier, boosting and bagging principles illustrated [3]

4.3 Classification Methods

The following section contains a description of the applied classification machine learning methods used for modelling.

4.3.1 Logistic Regression Classifier

Logistic Regression Classifier is a supervised machine learning technique, that can be used to predict binary classes and their probabilities. Logistic regression is an extended version of linear regression. Traditional linear regression can be described given an input of N observations $x_1, .., x_N$ and N target values $y_1, .., y_N$ tries to predict y given x. The full model can be described as:

$$y = \sum_{i=1}^{N} w_i \cdot x_i = w_0 + w_1 x_1 + w_2 x_2 + ... + w_N x_N + \varepsilon \tag{1}$$

w is a tunable variable, w_0 is the intercept term and ε is the error term. ε

Logistic regression builds upon a re-parameterizing of the Bernoulli distribution, using the sigmoid function to obtain a probability by transforming the output of traditional linear regression to a value between 0 and 1.

$$P(w^t \cdot x) = \frac{1}{1 + e^{-w^T \cdot x}} \tag{2}$$

Where $P(w^t \cdot x)$ is the output between 0 and 1, $-w^t \cdot x$ is the linear combination from the linear regression prediction and e is base natural of log. (Herlau et al. 2016).

4.3.2 Random Forest Classifier

Random Forest is an ensemble tree-based learning algorithm. It is an algorithm which consist of a collection of tree classifiers grown using bagging $M(\mathbf{x}, \Theta_k), k = 1, ...$, where Θ_k are independent identically distributed random vectors, where the final result is found by majority voting of all tree classifiers. Compared to regular decision trees, random forest adds an additional element of randomness, while growing the trees. When trees are split, it finds the most important feature in a random subset of features instead of all features, which results in a more diverse model, compared to a normal decision tree, which searches for the most important feature of all the features.

A few advantages of using random forest in this thesis is, that it handle both categorical and continuous values well, and the random element in the algorithm contributes to less overfitting. The model is however more computational heavy, and its suffers from interpretability, by using weights for the features, it is not possible to determine whether a feature has a negative or positive impact for the result. (Breiman 2001)

4.3.3 Adaboost Classifier

Adaboost is a boosting method, which combines weak learners (classifiers) and assigning them weights depending on how well they classified the observation. The weak learners are made according to how well the previous weak learner classified the observation. Adaboost starts with training a classifier $f_1(\mathbf{x})$ where each observations is assigned an equal weight $\mathbf{w}(t)$, where t is which iteration being performed. An evaluation is performed on the classifier where the weighted error ϵ_t for each observation is calculated by

$$\epsilon_t = \sum_{i=1}^{N} w_i(t)(1 - \delta_{f_t(\mathbf{x}_i, y_i)}) \tag{3}$$

Afterwards the importance of the classifier $\alpha_1(\mathbf{x})$ is calculated by:

$$\alpha_t = \frac{1}{2} log \frac{1 - \epsilon_t}{\epsilon_t} \tag{4}$$

At last the weights $w(t+1)$ are updated by:

$$w_i(t+1) = \begin{cases} w_j(t)e^{-\alpha_t}, & \text{if} f_t(\mathbf{x}_j) = y_j \\ w_j(t)e^{\alpha_t}, & \text{if} f_t(\mathbf{x}_j) \neq y_j \end{cases} \tag{5}$$

The hard to classify observation's weight influence are increased and the easier observations weights influence a lowered.

A new data set D_t is sampled according to the new weights, where the hard to classify observations will be more frequently represented are created. A second classifier $f_2(\mathbf{x})$ for D_t which aims to improve the prediction on the by training on more hard cases. This process is repeated until the training set is predicted perfectly or a maximum number of models are added. The final model is found using majority voting by taking the average each classifiers vote $f_1(\mathbf{x})...f_T(\mathbf{x})$ and their importance, of predictions for each classifier made, with their respective weight importance $\alpha_1(\mathbf{x})...\alpha_T(\mathbf{x})$. This can be described by:

$$f^*(\mathbf{x}) = \underset{y=1,2}{arg\ max} \sum_{t=1}^{T} \alpha_t \delta f_t(\mathbf{x}), y \tag{6}$$

where the delta function $\delta f_t(\mathbf{x}) \in 0, 1$ (Herlau et al. 2016).

4.3.4 XGBoost Classifier

Extreme gradient boosting (XGBoost), builds upon the foundations of classical gradient boosting

Its a scalable machine learning algorithm for tree boosting, which is used both for classification and regression problems. It ensembles multiple weak classifiers, which are added sequentially and each focuses on correcting the mistakes from the prior classifier by minimizing a differentiable loss function $\mathcal{L}$. It is an additive approach which converts weak classifiers $f_1, f_2..f_k$ into a single strong classifier $\boldsymbol{\phi}(x)$. The final prediction of the model is the sum of each tree.

$$\tilde{y}_i = \boldsymbol{\phi}(x) = \sum_{k=1}^{K} f_k(\mathbf{x}_i) \tag{7}$$

The objective is then to grow new trees which combined with the existing trees, lowers the total loss compared to the existing trees alone, while penalizing higher complexity.

The regularized objective function for XGBoost is defined by:

$$\begin{aligned} \mathcal{L}(\phi) &= \sum_i l(y_i, \tilde{y}_i) + \sum_k \Omega(f_k) \\ \text{where } \Omega(f) &= \gamma T + \frac{1}{2}\lambda \|w\|^2 \end{aligned} \tag{8}$$

Where:

- T is the number of leaves in the tree
- l is a differentiable convex loss function
- γ is the minimum loss reduction. required to make a further partition on a leaf node of the tree.
- λ is the learning rate.

here l is a differentiable convex loss function that calculates the difference between the prediction y_i and the target $\tilde{y}_i$. The second term Ω is to penalize the model complexity, the addition regularization term smooths the last updated weights, which makes the model less prone to overfitting.

Unlike tradition gradient boosting, XGboost uses the first and the second order derivative of the gradient. This provides more information to approximate the maximum decrease in the loss function.

XGBoost is significantly less computational expensive than traditional gradient boosting algorithms, while also incorporating a regularization term, it is less prone to overfitting. It is possible to examine how features effects XGBoost, which will be shown by the total gain for each feature normalized. The gain implies the relative contribution of the corresponding feature to the model, calculated by taking each feature's contribution for each tree in the model. A higher value of this metric when compared to another feature implies, that it is more important for generating a prediction

Assume that I_L and I_R are the instance sets of left and right nodes after a split. Letting $I = I_L \cup I_R$, then the loss reduction (gain) after the split is given by:

$$L_{split} = \frac{1}{2}\left[\frac{(\sum_{i\in I_L} g_i^2)}{\sum_{i\in I_L} + h_i + \lambda} + \frac{\sum_{i\in I_R} g_i^2}{\sum_{i\in I_R} + h_i + \lambda} - \frac{\sum_{i\in I} g_i^2}{\sum_{i\in I} + h_i + \lambda}\right] - \gamma \tag{9}$$

where:

- $g_i = \partial_{\tilde{y}^{(t-1)} l(y_i, \tilde{y}^{t-1})}$ are the first order gradient statistics from the loss function.
- $h_i = \partial^2_{\tilde{y}^{(t-1)} l(y_i, \tilde{y}^{t-1})}$ are the second order gradient statistics from the loss function.

(Chen & Guestrin 2016)

4.4 Hyperparameter Tuning

Hyperparameter tuning is the process of determining the ideal combination of hyperparameters, which allows the model to maximize its performance. There are different hyperparameters available for tuning, depending on which machine learning model is applied. The optimal hyperparameters greatly depends on the structure of the data used for the problem, and what type is being solved. There exist several approaches for finding the optimal hyperparameters. It can be done by systematically, testing each possible combination of hyperparameters, which results in the highest model performance, a random approach can be used. In this thesis the two approaches grid search and random search will be considered.

Grid search systematically iterates through every combination of hyperparameters values presented to the algorithm, which is fitted to the model, where the best scoring model and its hyperparameters is returned. The disadvantage with grid search is that it is computational heavy, and it is especially heavy when the number of the hyperparameters grows exponentially. However, since it is an exhausting algorithm working through every combination it should find the optimal values and combination of the hyperparameters presented.

Random search is an approach where the algorithm randomly chooses and applies a combination of the hyperparameters presented, a specified number of times. Since the algorithm does not iterate over all possible combinations, it is not certain the optimal values and combination of the hyperparameters presented are found. However, the algorithm is less computational expensive. Below in Figure 4 a visual representation of both algorithms can be seen(Shahul ES 2021).

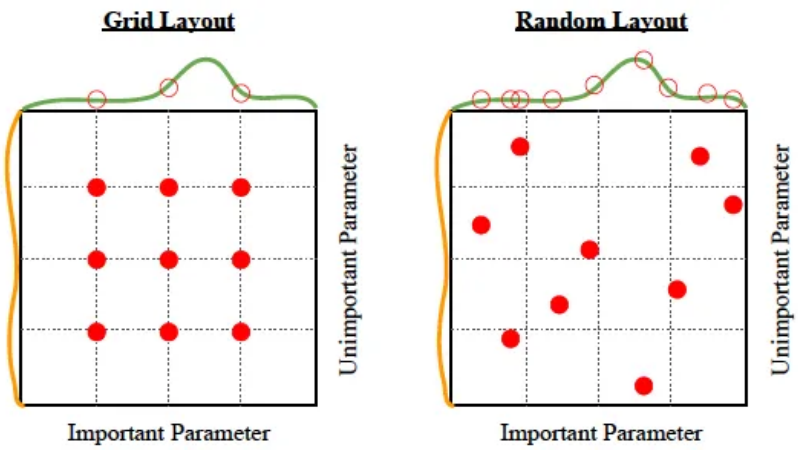


Figure 4: Grid and random search visually represented [2]

4.5 Evaluation Metrics

In order to evaluate the performance of the applied machine learning models in this project, several evaluation metrics will be introduced. Different metrics are suitable for different type of problems and data structures, where the problems faced in this thesis are binary classification problems.

4.5.1 Confusion Matrix

The confusion matrix is a $N \times N$ dimensional matrix, where N is the number of classes to be predicted, which describes the performance of a model. From the confusion matrix multiple metrics can be derived, which are useful to analyze, how a model is performing. In this thesis unbalanced data sets are encountered, therefore the metrics precision, recall, F1 Score and Roc-Auc score will be added in addition to the accuracy score. These metrics are chosen, since they do well with unbalanced data sets, where traditional accuracy can be misleading, accuracy is however kept for an overall score of the model performances.

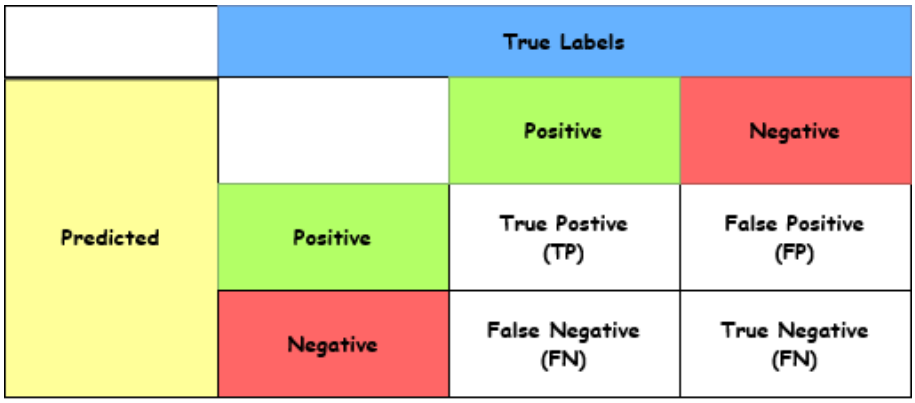


Figure 5: Confusion matrix

Precision is the number of positive predictions divided by the total number of positive class predictions. It can be a measurement for how "exact" a model is, where a low precision could indicate a large number of false positives.

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

Recall is the number of positive class predictions divided by by all positive class values in the data set. It can be measurement of how complete a model is, where a low recall indicates many false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

The F1 is a function of precision and recall, and conveys the balance between them. The advantage of using F1 compared to accuracy score for unbalanced data sets is that a high accuracy score can be a result of an over represented class which the model more often will predict. Here F1 gives a better measure of the incorrectly classified cases.

$$F1 = \frac{2}{Recall^{-1} + Precision^{-1}} \tag{12}$$

Receiver Operating Characteristic (ROC) is an evaluation metric for binary classification problems that summarizes the behavior of a model. It is a probability, which plots the curve of the true positive rate against the false positive rates at various thresholds. The area under the ROC-curve can be found, the higher the AUC, the better the performance of the model at distinguishing between the positive and negative classes.

The accuracy is a measurement of the percentage of times the predicted values $\tilde{y}_i$ matched the actual values y_i. It is a metric suitable for determine how well a classification model performs, which has balanced classes.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{13}$$

(Srivastava 2019)

4.5.2 Cross-validation

Cross-validation is a method of comparing and validating multiple models. This project will only apply K-fold one-layer cross-validation, but there are other cross-validation methods such as hold-out and leave-one-out. K-fold cross-validation splits the data set into K pieces, each with $\frac{N}{K}$ observations. Then one piece k is taken out as the test set and the other $K-1$ sets as training set. The test error of each of these K splits gives K estimates of the error $E^{test}_{\mathcal{M},1}, ..., E^{test}_{\mathcal{M},K}$ for the $\mathcal{M}$ models. The weighted average of the test errors is an approximation of the generalization error, which is a measure of how well a model predicts for unseen data. It is calculated as follows:$E^{gen}_{\mathcal{M}} \approx \sum_{k=1}^{K} \frac{N^{test}_k}{N} E^{test}_{\mathcal{M},k}$ K-fold cross-validation is generally more precise than hold-out cross-validation, because it uses each data point once in the test set. The disadvantage of K-fold cross-validation is that it requires K times more training and testing of models than hold-out cross-validation (Herlau et al. 2016).

4.6 Software Tools

The programming language used for processing data and modelling is Python and Jupyter Notebook is used to structure the code. Jupyter Notebook is an open-source software for Python and other programming languages. In Python, the main libraries used for the machine learning models and visualisations are Matplotlib, Numpy, Pandas, Scikit-learn and Seaborn.

5 Empirical Field

In this section, the empirical field of tennis will be described. This will include a brief summary of what tennis is, the fundamental rules and tendencies in the game.

5.1 What is Tennis

Tennis is a world renowned racket sport, which can be played individually against a single opponent (singles) or between two teams of two players each (doubles). Tennis matches are played at tennis courts of the dimensions 23.8 by 8.2 metres for singles and 23.8 by 11.0 metres for doubles. In the center

splitting the tennis court in two is a net with a height of 0.91 metres. The surface material in tennis courts varies between grass, clay, hard courts and artificial grass. Each surface has different playing characteristics which will affect the style of play and a players natural playing ability. The objective in a match is to hit the ball over the net landing the ball within the margins of the court and in a way that results in your opponent being unable to return the ball. This thesis studies male Grand Slam matches, whereas the following elaboration of the rules will be specific to these.

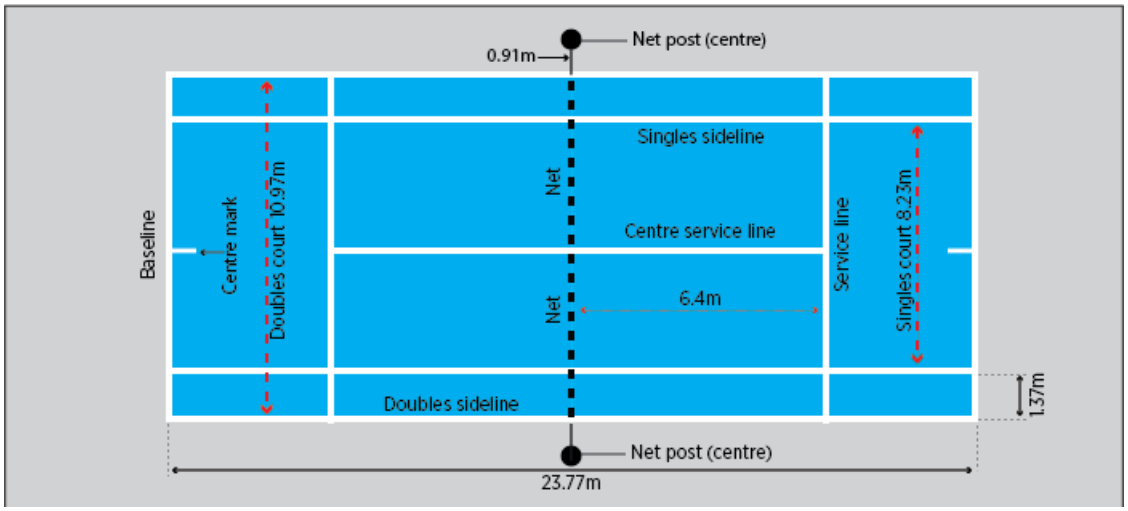


Figure 6: Overview and dimensions of a tennis court for singles and doubbles [3]

5.2 Rules & Point System

Tennis uses a unique scoring system, using points, games and sets. A player has to win enough points to win a game, enough games to win a set and enough sets to win a match. Tennis always have a winner and a loser, there are no ties.

Points are awarded to a player when the opponent fails to return the ball within the players court. The points are known as 15 (1 point), 30 (2 points) and 40 (3 points). A game is decided by the first player to win four points, unless the score is equal at 40-40 (deuce), then the game continues till a player has a 2 point advantage. A set is decided by the first player to win 6 games and have a two game win advantage. If the opponent wins 5 games the player must win the set 7-5. If the set goes to 6-6 then a tiebreak is played, which is decided by the player who first reaches 7 points. Matches are decided by the player who first win 3 of a total 5 sets.

Table 1: Tennis scoring system overview

Unit	**Composed of**	**Requirements**
Points	base unit	single point
Games	points	4 points & 2-point advantage
Sets	games	6 games & 2-games advantage
Match	sets	3 to 5 sets

A match starts with a coin toss to determine which player must serve first, and which side they want to

serve from. The serve alternates between the two players for each game. The server must serve from alternating sides on the baseline.

5.3 Tennis Terms

In order to understand the features in the data, several important events of a match has their own respective terms.

First Serve Every point starts with a serve from one of the players, the player has two attempts to land a successful serve for each point. The server has a major advantage at the first serve, since they can perform more risky shots that are harder to return without getting punished by missing.

Second Serve If the first serve fails to land within their opponents service box, the serve is invalid, and the server must serve again. The second serve is generally less risk prone than the first, since an invalid serve will result in the returner winning the point. Therefore the advantage of the server is reduced, which results in the second serve points to be less one sided.

Double Fault If both serves of the serving player are deemed invalid, a double fault is made, and the point is awarded to the returner without any gameplay between the players.

Ace If the server manages to perform a valid serve without the returner touches the ball it is an ace. This is usually the result of a very well placed and fast serve.

Break Point In tennis the server has a clear advantage, therefore games which are won by the returner are of special interest. Break points are when the returner has the opportunity to break the serve, and win the game. Break points occurs when two conditions are met: 1) a player is missing one point win the game, 2) their opponents is serving.

Net Point A net point is a point won or lost were one player approaches the net, opposite to win or lose a point from a stroke at the baseline.

Rally The rally of a tennis point is defined by the sum of strokes performed by each player in a point.

Unforced Error An unforced error is when the missed shot was the fault of the player who made the shot.

Winner is when a player has performed a shot which the opponent is not able to return, and thereby outplaying the opponent, opposed to winning the point by a personal error from the opponent.

Tiebreak is a system in traditional tennis used to decide a set when the score is tied, 6-all. An important aspect of tiebreaks is the serves alternates every two points, opposed to every game.

5.4 Tournaments, Ranking and Surfaces

There exist multiple ranking system for tennis, where the official one is the ATP-tour ranking. Ranking points awarded for the best 19 results in ranked tournaments over the last 52 weeks.

In tennis the Grand Slam tournaments also called majors, are the most important events of the year. There are four Grand Slam tournaments: French Open, US Open, Australian Open and Wimbledon. The Grand Slams offer the most ranking points, prize money, media attention and sets for men are played best-of-five. They serve as benchmarks for the players and victory's are a necessity if a players wants to be considered one of the greatest. A total of 128 players participate in the Grand Slams. The top 104 ranked players are automatically qualified to the Grand Slams, 16 places are obtained through qualifying matches and 8 places is obtained through wild cards. This ensure that the highest level of tennis is played, and surprise losses where a high rated player losses to a much lower ranked player are less frequent.

Each Grand Slam has their own unique courtyard surface, which has a major influence on how the game is played. The Grand Slams included in this study are Wimbledon and US Open. The surface played on at Wimbledon is a grass surface, which is far more slippery compared to other surfaces at the Grand Slams. With grass there is less friction between the ball, which makes the balls bounce faster and lower to the ground. Therefore players who excels in serving, has an advantages in this tournament. The surface played on at US Open is a hard court. At hard courts the balls bounces a bit slower than a grass courts, but bounces higher, which makes the bounce of the ball more predictable, therefore more controlled and longer points are typically seen here. (Perfect Tennis 2020)

6 Exploratory Data Analysis

This section will explore the content of the data from two public available resources: **https://github.com/JeffSackmann/tennis_slam_pointbypoint** and **https://github.com/JeffSackmann/tennis_atp**. This data will be used to derive the variables needed for modelling. At the end of the section, an evaluation will conclude which aspects of the data are suitable for predicting the outcome of the match. All Figures and statistics are made in the Jupyter Notebook "Exploratory_Data_Analysis.ipynb".

6.1 Initial data cleaning

Before exploring the data an initial data cleaning will be performed. The code for the data cleaning can be seen in the Jupyter Notebook "Data_Preparation.ipynb".

6.1.1 Gender separation

In the obtained data both male and female matches are presented, which causes some complications. The male matches are in a best-of-five set format and female matches are in a best-of-three set format. Another complication is that there is a significant difference between the male and female tennis gameplay. This can especially be seen in the serve, which is a more deciding factor for male tennis than for female tennis (Carboch 2017). Therefore, it is chosen to exclude all female matches.

6.1.2 Tackling NaN-values

The data contained multiple of NaN-values in the features *ServeWidth*, *ServeDepth* and *ReturnDepth*. First and foremost matches in the data without a single value in *ServeWidth*, *ServeDepth* and *ReturnDepth* was

removed.

There are three possible scenarios for when NaN-values are present for *ReturnDepth*. The first is when the server makes a double fault. The second is when the server makes an ace, it should be mentioned that a serve which the returner touches but is not returned properly, will be counted as an ace. The reasoning behind this is because it is assumed, that winning with the serve has the same meaning for the model if it is an ace or not. The third is when the returner returns the ball but the server can hit it without letting it hit the ground. The NaN-values in *ReturnDepth* for the first instance is replaced by ”DoubleFault”. For the second instance the NaN-values are replaced by ”ServeAce” and for the third they are replaced by ”Service box”.

When *ServeWidth* and *ServeDepth* took the value of NaN it could only be because of the server making a double fault. And therefore, the instance were the server makes a double fault the NaN-value is replaced by ”DoubleFault”.

The previous assumptions and actions are based on manual checking points in a full match. The match was the US Open finale 2020 between Alexander Zverev and Dominic Thiem. The match can be found at **https://www.youtube.com/watch?v=Mac4XI3ka5I**.

6.2 Tennis_slam_pointbypoint data

The tennis_slam_pointbypoint data contains information for Grand Slams matches from 2011 to 2020. Only data from Wimbledon and US Open from 2016 to 2020 will be used, because these years and tournaments have additional information about the serves and returns. A description of the features can be seen in Table 2.

Table 2: tennis_slam_pointbypoint data feature description

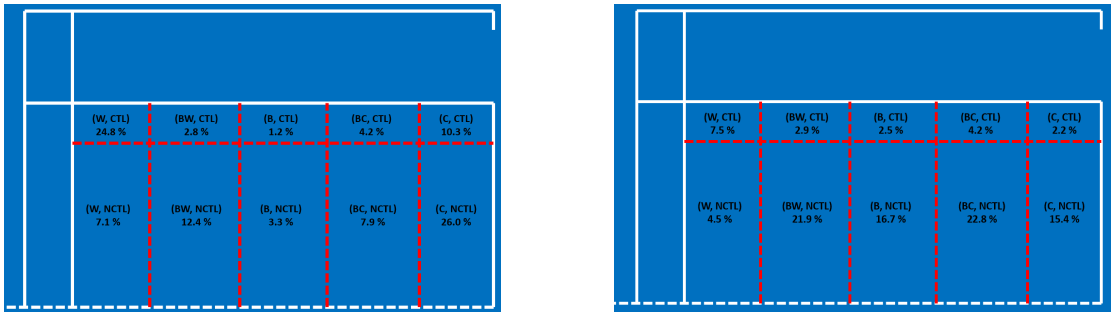
Feature	**Type**	**Explanation**
Match_id	Discrete and nominal	A unique number identifying a given match.
ElapsedTime	Discrete and interval	The start time of a point.
SetNo	Discrete and ratio	The set that is being played for.
P1GamesWon	Discrete and ratio	Number of games player 1 has won.
P2GamesWon	Discrete and ratio	Number of games player 2 has won.
SetWinner	Discrete and Ordinal	Which player won the set.
GameNo	Discrete and ratio	The game that is being played for in the given set.
GameWinner	Discrete and Ordinal	Which player won the game.
PointNumber	Discrete and ratio	What point is being played for.
PointWinner	Discrete and Ordinal	Which player won the point.
PointServer	Discrete and Ordinal	Which player serves.
Speed_KMH	Continuous and ratio	Speed of the serve in kilometers per hour.
P1Score	Discrete and Ordinal	Player 1's score when the point start .
P2Score	Discrete and Ordinal	Player 2's score when the point start .
P1PointsWon	Discrete and ratio	Number of points player 1 have won.
P2PointsWon	Discrete and ratio	Number of points player 2 have won.
P1Ace	Discrete and nominal	Indicator if player 1 has made an ace.
P2Ace	Discrete and nominal	Indicator if player 2 has made an ace.
P1Winner	Discrete and nominal	Indicator if player 1 has made a winner.
P2Winner	Discrete and nominal	Indicator if player 2 has made a winner.
P1DoubleFault	Discrete and nominal	Indicator if player 1 has made a double fault.
P2DoubleFault	Discrete and nominal	Indicator if player 2 has made a double fault.
P1UnfErr	Discrete and nominal	Indicator if player 1 has made an unforced error.
P2UnfErr	Discrete and nominal	Indicator if player 2 has made an unforced error.
P1NetPoint	Discrete and nominal	Indicator if player 1 has approached the net.
P2NetPoint	Discrete and nominal	Indicator if player 2 has approached the net.
P1NetPointWon	Discrete and nominal	Indicator if player 1 has won the point by approaching the net.
P2NetPointWon	Discrete and nominal	Indicator if player 2 has won the point by approaching the net.
P1BreakPoint	Discrete and nominal	Indicator if player 1 has a break point.
P2BreakPoint	Discrete and nominal	Indicator if player 2 has a break point.
P1BreakPointWon	Discrete and nominal	Indicator if player 1 wins a break point.
P2BreakPointWon	Discrete and nominal	Indicator if player 2 wins a break point.
Speed_MPH	Continuous and ratio	Speed of the serve in miles per hour.
P1BreakPointMissed	Discrete and nominal	Indicator if player 1 lose a break point.
P2BreakPointMissed	Discrete and nominal	Indicator if player 2 lose a break point.
Serveindicator	Discrete and Ordinal	Which player serves. Player 1 or player 2.
ServeNumber	Discrete and Ordinal	Which serve the serving player succeed. First or second.
WinnerType	Discrete and Ordinal	Indicator if the serving player won with the serve.
WinnerShotType	Discrete and Ordinal	Indicator if the player which won the point won with backhand or forehand.
P1DistanceRun	Continuous and ratio	Length the player 1 covered in the point in meters.
P2DistanceRun	Continuous and ratio	Length the player 2 covered in the point in meters.
RallyCount	Discrete and ratio	Number of times the ball crosses the net in a point.
ServeWidth	Discrete and Ordinal	Indicator at how wide the successful serve was. The feature can take the values: B (Body), BC (Body/center), BW (Body/wide), C (Center) or W (Wide).
ServeDepth	Discrete and Ordinal	Indicator at how deep the successful serve was. The feature can take the values: CTL (Close to line) or NCTL (Not close to line).
ReturnDepth	Discrete and Ordinal	Indicator at how deep the successful return ball was. The feature can take the values: D (Deep) or ND (Not deep).

Every match in the data has a unique match id. The match id consists of the tournament it is in, the year the match is played and a number that determine the gender of the players, if over 2000 then the players are women and otherwise men. Therefore, it is possible to track different stats for each point in each match.

Given our objective of predicting point, game and set winners in the Grand Slam tournaments US Open and Wimbledon, and from this explore which factors in a game are the most important, it is relevant to explore the patterns in matches.

6.2.1 Match patterns

The importance of the serve in tennis is well known. So it would be interesting to see where players tend to serve for their first and second serve. In Figure 7a the frequency in percentage of all successfully first serve placements is shown. It can be seen that 68.2% of the successful serves were served close to the sidelines of the service box. If compared to the second serve, seen in Figure 7b, where only 29.6% of the serves were served close to the sidelines of the service box. Furthermore it can be seen in Figure 7b that four of the most occurring serve positions had a serve depth of NCTL. This is most likely because if the second serve is unsuccessful the returner will get the point. Therefore the server is not willing to take risks when serving the second serve. It is important to mention that Figure 7a and 7b is an estimation of the placements for the combinations of the *ServeWidth* and *ServeDepth*, since the data does not specify the exact coordinates of the serve and return.



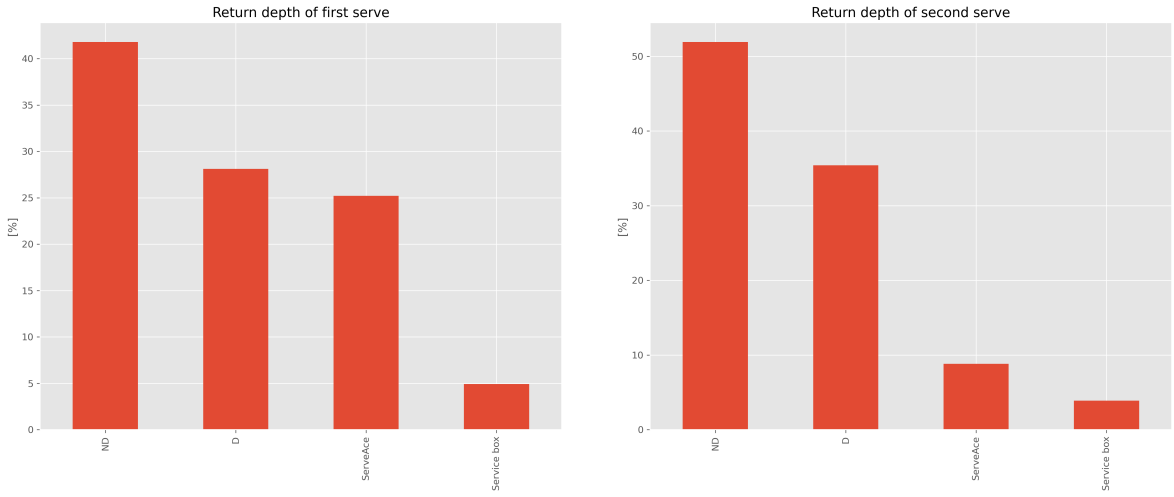
(a) 1st serve *(b) 2nd serve*

Figure 7: Images of one-quarter of a tennis court with combinations of ServeWidth and ServeDepth. Image (a) shows the combinations of ServeWidth and ServeDepth for the 1st serve, and image (b) shows the combinations of ServeWidth and ServeDepth for the 2nd serve

It can be seen that there was a difference between the placement of the serves from first to second. The second serves placement were often worse than the first serve. This affects the win percentage for the server. In the data the server wins 73.2% of the points where the first serve is successful. On the other hand will the server only win 57.2% of the points were the second serve is successful. A comparison to the overall server win percentage at 64.8%, shows the importance of hitting the first serve.

Now that the tendencies of the server have been explored. It would only be natural to look at the returner. In Figure 8a the frequency in percentage of all successful return placements for the first serve is shown. It can be seen that around 43% of the returns are classified as not deep and around 27% were deep. If compared with return ball placements for the second serve, seen in Figure 8b, around

52% and 35% of the return balls return depth, were not deep and deep. Both percentage of deep and not deep return balls increase from the first to second serve. This is mainly because the percentage of aces decrease with around 15% from the first serve to the second serve. So it can be assumed that it is generally easier to return a second serve.



(a) 1st serve *(b) 2nd serve*

Figure 8: Bar plots of the percentage of the ReturnDepth. Plot (a) shows the percentage of the different return depths for the 1st serve, and plot (b) shows the percentage of the different return depths for the 2nd serve

Looking at the win percentage of each of the return balls, seen in Table 3, return balls which were not returned deep had a higher win percentage then returns balls which were returned deep. A explanation for this could be that a ND return can have an more acute angle, which makes the return ball go wider. This hypothesis can not be explored with the current data because their is no feature which indicates the width of the return ball.

Table 3: Win percentage of each of the return balls

	D	**ND**	**ServiceAce**	**Service box**	**Returner win**
Percentage	38.2	41.7	0	40.9	35.2

For a better understanding of the match patterns, the rally count of the points are explored. Figure 9 shows the number of points given a rally count. The graph only shows the rallies with the rally count from zero to 15, as there were a small amount of points with an larger rally count. One example is a point with 46 in rally count. The rally count seems to follow a right skewed normal distribution.

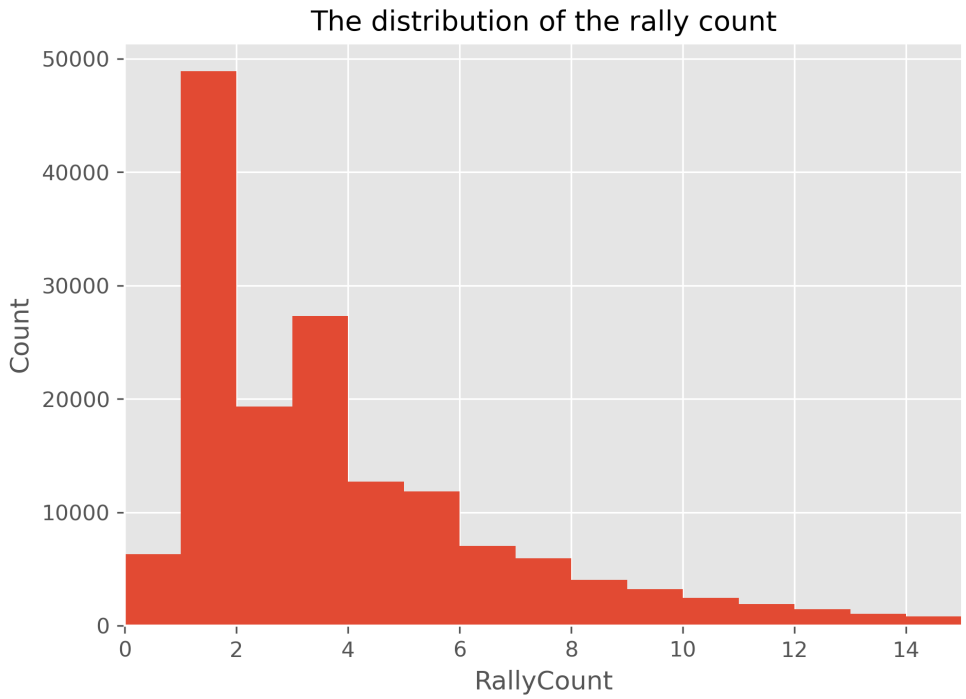


Figure 9: The distribution of the rally count

Another relevant exploration is the win percentage of the returner the longer the rallies are. In Figure 10, the win percentage for the returner the longer a rally is is shown. A rally with an odd number as length means that the server won the point, otherwise the returner won the point. Therefore, the bins in Figure 10 consists of an even and odd number. The graph indicate that server have a clear advantage in first four shorts of the point. Afterwards the returner wins over 50 % of the points.

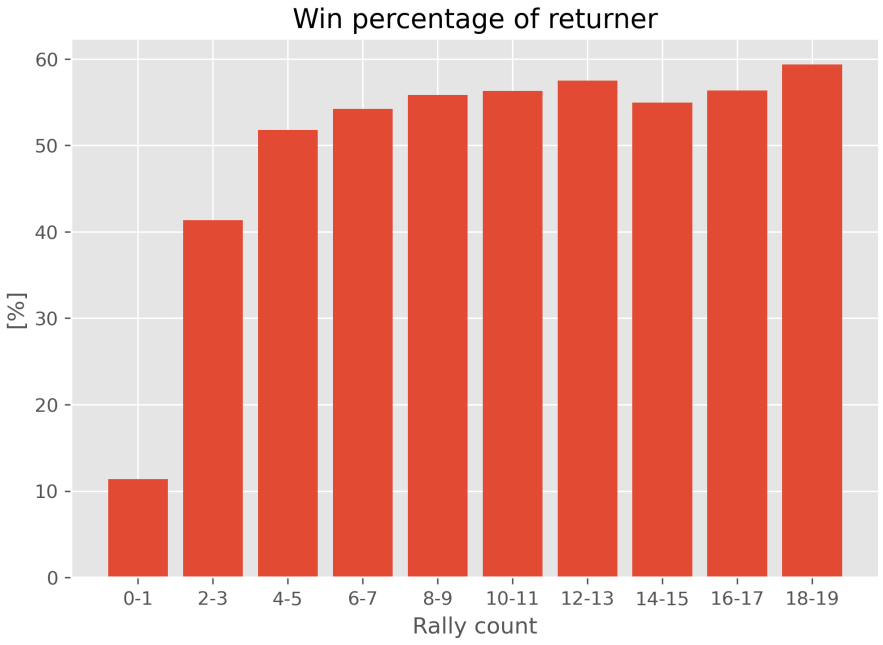


Figure 10: The win percentage of the returner relative to the rally length

6.3 Tennis_atp data

The tennis_atp data contains information for tour-level main draw atp matches from 1968 to 2021. Only data from Wimbledon and US Open from 2016 to 2020 will be used, because these years and tournaments can be merged with the tennis_slam_pointbypoint data. A description of the features there will be used can be seen in Table 4.

Table 4: tennis_atp data feature description

Feature	Type	Explanation
tourney_name	Discrete and nominal	A Name of the tournament.
surface	Discrete and nominal	The surface the match is played on.
match_num	Discrete and nominal	The match number of the given match.
winner_name	Discrete and nominal	The name of the player which won the match.
loser_name	Discrete and nominal	The name of the player which lost the match.
winner_rank	Discrete and ordinal	The rank of the winner when the match started.
loser_rank	Discrete and ordinal	The rank of the loser when the match started.

The main reason for including tennis_atp data is for the rank of the players, at the time the match was played and the surface of the court. The players rank will be explored in the next section.

6.3.1 Player rank

The Grand Slams tournaments are the four most important annual tennis events. The reason for this is because they give the most ranking points. In Table 5 descriptive statistics can be seen for the features *winner_rank* and *loser_rank*.

Table 5: Descriptive statistics for winner_rank and loser_rank

	winner_rank	loser_rank
count	157385	157222
mean	35.388862	78.362106
std	50.031736	111.603561
min	1	1
25%	7	23
50%	21	53
75%	44	95
max	589	1415

When examining the Table, the first thing which is notet, is the different in count between *winner_rank* and *loser_rank*. In the data, one player have not played for the last year and therefore doesn't have a rank, meaning he get a NaN-value in the data. So he will get an rank of 501, When looking at the mean it can be seen that *winner_rank* has a lower value than *loser_rank*. The better a player is the lower is his rank. Also looking at the three quartiles, the *winner_rank* is more "compact" than the *loser_rank*. Players with a better rank wins more games. Both features have a max-value which is a lot higher than the 75% quartile. This could indicate outliers, but because lower ranked players either have to compete

in qualifiers, and therefore deserve their spot, or get a wild card from the tournament committee to participate.

Looking at the win percentage on the court surfaces hard and grass, seen in Table 11, it can be seen that on the hard surface there is almost no difference between win percentage for the player with the highest and lowest rank. On grass there is a clear difference, it can be seen that the lower ranked player wins 25% of the matches on grass, where the higher ranked player wins 75% of the times. A reason for this could be that grass is a faster surface and the ball has a lower bounce than hard court. This means that the grass court favor players with good serve and technique, which is most a characteristic for high ranked players.

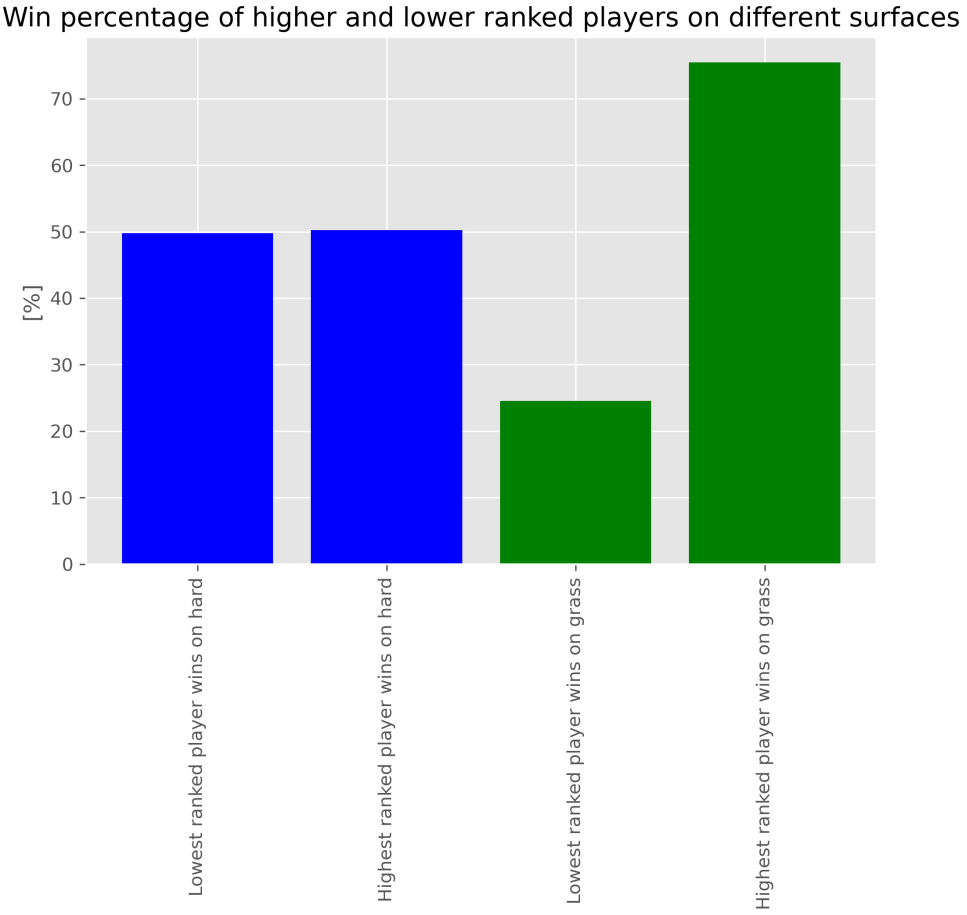


Figure 11: Bar plot of the win percentage for how many time the highest and lowest ranked players wins a match on either grass (green) or hard court (blue) surface

6.4 Evaluation of the exploratory analysis

The exploratory analysis have given information of the data and which aspects of the game could yield important for the models. The importance of the serve was already well known. The analysis indicate the same, with a win percentage of over 60% for the server, the different between first and second serve and the placement of the serve. The serve relevant features should therefore be used in the models.

Another observation from the analysis is the difference between the court surfaces and player ranks.

The analysis shows that the grass court favored the higher ranked player more than the hard court did. Therefore the models should have the court type and player ranks as input variables.

7 Data Preparation

In this section the preparation of the data for the classification models will be explained and the reasons behind the decisions. The code for the data preparation can be seen in the Jupyter Notebook "Data_Preparation.ipynb".

7.1 Tiebreaks

Points which accrue in tiebreaks will be removed when used in the GameWinner models. The reason is that in tiebreaks the players take turns serving after every other point. Therefore it is assumed that this will confuse the models more than giving them extra and more useful information.

7.2 Defining the server and returner

To make it easier to analyze and interpret the results of the models, the serving player will always be identified as player 1 and the returning player as player 2. This will make the data sets features focus more on the server/returner rather than having identical classes inform of player 1 and 2. So throughout a match all player specific features such as *P1PointsWon* and *P2PointsWon* etc. will swap between each other each time the server change.

7.3 Accumulated features

Multiple features such as *P1UnfErr*, *P2DoubleFault*, *P1Winner* etc. directly indicates which player wins the point and sometimes the game or set and therefore will these features not be used in the models. There will instead be made an accumulated feature for each feature which directly indicate which player wins the point. This is done because it can be assumed that events like a double fault, serve ace etc. can change the flow of a match. The Figures in this section show the data before making player 1 always be the serving player, and before making some of the categorical binary. A slice of the data can be seen in Figure 12. Where two accumulated features have been made for *P1UnfErr* and *P2UnfErr*. It can be seen in index 0 that player 1 makes a unforced error and therefore the row after and till player 1 makes another unforced error *P1UnfErrA* is equal to one. The same can be seen for *P2UnfErr* and all other features which directly indicate which player wins the point. The full list of the new added features can be found in the Jupyter Notebook "Data_Preparation.ipynb" under section Accumulated features.

	match_id	ElapsedTime	SetNo	GameNo	GameWinner	PointWinner	P1Score	P2Score	P1UnfErr	P2UnfErr	P1UnfErrA	P2UnfErrA
0	2016-usopen-1101	0:00:00	1	1	0	2	0	0	1	0	0.0	0.0
1	2016-usopen-1101	0:00:30	1	1	0	1	0	15	0	0	1.0	0.0
2	2016-usopen-1101	0:00:52	1	1	0	1	15	15	0	1	1.0	0.0
3	2016-usopen-1101	0:01:33	1	1	0	1	30	15	0	1	1.0	1.0
4	2016-usopen-1101	0:01:59	1	1	0	2	40	15	1	0	1.0	2.0
5	2016-usopen-1101	0:02:19	1	1	1	1	40	30	0	1	2.0	2.0
6	2016-usopen-1101	0:03:25	1	2	0	1	0	0	0	0	2.0	3.0
7	2016-usopen-1101	0:03:56	1	2	0	2	15	0	0	0	2.0	3.0
8	2016-usopen-1101	0:04:26	1	2	0	1	15	15	0	0	2.0	3.0
9	2016-usopen-1101	0:05:00	1	2	0	1	30	15	0	1	2.0	3.0

Figure 12: A slice of P1UnfErr and P2UnfErr and their accumulated features

Two new features will be made, *SetWinnerA* and *GameWinnerA*, which either take the value of one or two for which player who wins the current set and game. These will be the target variables for the game and set models. The reason behind this is that the models should predict who will win the game/set at any point in the game/set. And not only when one of the players have the opportunity to win the game/set. A slice of the data with *GameWinner* and *GameWinner* side by side can be seen in Figure 13. You can see that *GameWinnerA* is one till index six were game number two starts. And it can be seen that player 2 wins game number two.

	match_id	ElapsedTime	GameNo	GameWinner	GameWinnerA	PointWinner	P1Score	P2Score
0	2016-usopen-1101	0:00:00	1	0	1	2	0	0
1	2016-usopen-1101	0:00:30	1	0	1	1	0	15
2	2016-usopen-1101	0:00:52	1	0	1	1	15	15
3	2016-usopen-1101	0:01:33	1	0	1	1	30	15
4	2016-usopen-1101	0:01:59	1	0	1	2	40	15
5	2016-usopen-1101	0:02:19	1	1	1	1	40	30
6	2016-usopen-1101	0:03:25	2	0	2	1	0	0
7	2016-usopen-1101	0:03:56	2	0	2	2	15	0
8	2016-usopen-1101	0:04:26	2	0	2	1	15	15
9	2016-usopen-1101	0:05:00	2	0	2	1	30	15
10	2016-usopen-1101	0:05:40	2	0	2	2	40	15
11	2016-usopen-1101	0:06:17	2	0	2	2	40	30
12	2016-usopen-1101	0:06:48	2	0	2	2	40	40
13	2016-usopen-1101	0:07:12	2	2	2	2	40	50

Figure 13: A slice of GameWinner and GameWinnerA

Another slice of the data with *SetWinner* and *SetWinnerA* side by side can be seen in Figure 14. You can see that *SetWinnerA* change from one to two in index 64. This means that player 1 won set one and player 2 won set two.

	match_id	ElapsedTime	SetNo	SetWinner	SetWinnerA	PointWinner	P1Score	P2Score
60	2016-usopen-1101	0:41:16	1	0	1	1	15	40
61	2016-usopen-1101	0:41:35	1	0	1	1	30	40
62	2016-usopen-1101	0:42:02	1	0	1	1	40	40
63	2016-usopen-1101	0:42:50	1	1	1	1	50	40
64	2016-usopen-1101	0:45:27	2	0	2	2	0	0
65	2016-usopen-1101	0:46:10	2	0	2	2	0	15
66	2016-usopen-1101	0:46:29	2	0	2	1	0	30
67	2016-usopen-1101	0:47:05	2	0	2	2	15	30
68	2016-usopen-1101	0:47:26	2	0	2	2	15	40
69	2016-usopen-1101	0:48:26	2	0	2	1	0	0

Figure 14: A slice SetWinner and SetWinnerA

7.4 Shifting data

The classification models will have the target variables *SetWinnerA*, *GameWinnerA* and *PointWinner*. The problem is that these features does not match with the *P1Score* and *P2Score*. It is more advantageous if *P1Score* and *P2Score* is the score when point is served and not after. The Figures in this section show the data before making player 1 always be the serving player and before making some of the categorical binary. A slice of the unshifted data can be seen in Figure 15. It can be seen in index one that player 1 makes an unforced error and therefore *P2Score* get the value of 15. So the score gives who wins the point away.

	match_id	ElapsedTime	SetNo	GameNo	GameWinner	PointWinner	P1Score	P2Score	P1UnfErr	P1Winner
0	2016-usopen-1101	0:00:00	1	1	0	0	0	0	0	0
1	2016-usopen-1101	0:00:00	1	1	0	2	0	15	1	0
2	2016-usopen-1101	0:00:30	1	1	0	1	15	15	0	0
3	2016-usopen-1101	0:00:52	1	1	0	1	30	15	0	0
4	2016-usopen-1101	0:01:33	1	1	0	1	40	15	0	0
5	2016-usopen-1101	0:01:59	1	1	0	2	40	30	1	0
6	2016-usopen-1101	0:02:19	1	1	1	1	0	0	0	0
7	2016-usopen-1101	0:03:25	1	2	0	1	15	0	0	1
8	2016-usopen-1101	0:03:56	1	2	0	2	15	15	0	0
9	2016-usopen-1101	0:04:26	1	2	0	1	30	15	0	1

Figure 15: A slice of the unshifted data

Therefore a large amount of features will be shifted one row up. The list of features can be found in the Jupyter Notebook "Data_Preparation.ipynb". A slice of the shifted data can be seen in Figure 16. It can be seen in index zero that *PointWinner* and *P1UnfErr* matches the score.

	match_id	ElapsedTime	SetNo	GameNo	GameWinner	PointWinner	P1Score	P2Score	P1UnfErr	P1Winner
0	2016-usopen-1101	0:00:00	1	1	0	2	0	0	1	0
1	2016-usopen-1101	0:00:30	1	1	0	1	0	15	0	0
2	2016-usopen-1101	0:00:52	1	1	0	1	15	15	0	0
3	2016-usopen-1101	0:01:33	1	1	0	1	30	15	0	0
4	2016-usopen-1101	0:01:59	1	1	0	2	40	15	1	0
5	2016-usopen-1101	0:02:19	1	1	1	1	40	30	0	0
6	2016-usopen-1101	0:03:25	1	2	0	1	0	0	0	1
7	2016-usopen-1101	0:03:56	1	2	0	2	15	0	0	0
8	2016-usopen-1101	0:04:26	1	2	0	1	15	15	0	1
9	2016-usopen-1101	0:05:00	1	2	0	1	30	15	0	0

Figure 16: A slice of the shifted data

7.5 Feature scaling

In order to optimize the model performance on the data, all features are standardized. This is primarily done for the sake of logistic regression which is a gradient descent based model. Different magnitudes and ranges of different features will result in varying step sizes for each feature, instead of the gradient descent moving smoothly towards a minimum. The tree based models are fairly insensitive of varying scales for each feature, since splits in trees are only depend on one feature, and different scaling features will therefore not have an impact on each other (Bhandari 2020).

8 Results

In this section the creation of the PointWinner, GameWinner and SetWinner models will be outlined and and their results will be analyzed. Two different approaches has been used to create these models. For the PointWinner models, the approach is to be able to predict who wins the next point, from any given point in the match. For games and sets which consist of multiple points or games, it will be attempted to predict the winner at specific timestamps, instead at any moment in the match. The four classification methods outlined in the Methodology section will be applied to each problem. These machine learning methods are chosen based on their different levels of complexity, and their different approaches of classifying. The hyperparameters are optimized for the best performing models in the validation data, by random search, ideally the more exhaustive approach grid search had been applied however due to time constrains it was not possible. At last an analysis of the performance of the final models will is conducted, examining the importance of the features used and evaluation metrics.

8.1 Procedure for Model Creation

The general approach to the model creation can be seen in Figure 17. First the data is collected, then the data preparation conducted by dealing with NaN-values, creating new features, one-hot encoding and data standardization. The data is then split into three subsets before any modelling is done. The test data consist of 10% of all the matches randomly chosen, and is saved to test the final models. The validation and training data consist of the rest of the 90% of the data, which is split by the ratio 80%

/ 20%. The four machine learning methods are then trained and validated upon these using 10-fold cross-validation. The performance of each model is then compared on evaluation metrics presented in the Methodology section. The best overall performing model is then chosen and its hyperparameters are optimized using random search and is applied to the test data for a final evaluation.

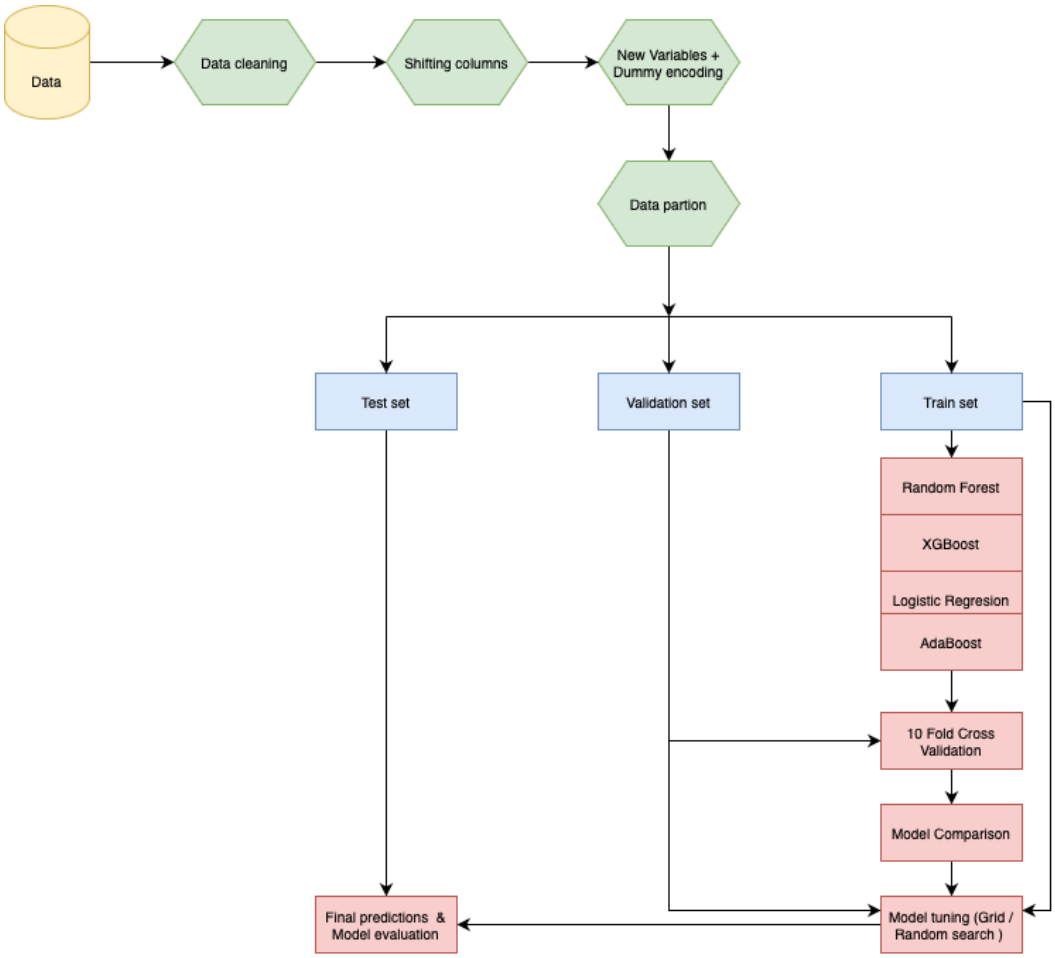


Figure 17: General approach for model creation

There are 57 features used as input for the models. These features includes accumulated features which feeds the model the historical data of a match, general features which e.g. describes the surface which the match is being played upon or the ranking of the players, and finally match specific features, e.g whether the point being predicted is part of a tiebreak. The full list of features can be found in appendix 11.A. Multiple features from the original data has been removed due to almost perfect correlation with the used ones, in order to keep the complexity down. The final reason for choosing these features is that none of them contains any specific information, that would let the models know the outcome. All player specific features is shifted in relative to the point server, which have resulted in it is always player 1 (P1) which is being presented as the server for the models. This is done for the model to be able to differentiate between the server (P1) and the returner (P2), instead of having two almost equal classes, and hereby examine which features are most important for either the server or the returner. However, as mentioned in the exploratory analysis, the server has a major advantage, and therefore the class server will be represented more often than returner. The evaluation metrics F1 score, recall score and precision

score will therefore be for the underrepresented class (returner) in the PointWinner and GameWinner models. For the SetWinner models, the same evaluation metrics will be the averaged for the two classes because the serve advantage is neutralised, as result of both players are serving in a set.

8.2 Hyperparamter tuning

All of the described machine learning algorithms in the Methodology section contains hyperparameters which can be adjusted to optimize the performance of the algorithms. The initial approach for all models will be to tune relative to accuracy score. This is done to achieve the overall most accurate predictions.

XGboost contains multiple parameters for tuning, choosing the parameter values is a trade off between bias and variance. Since the problems in this thesis are binary classification problems, the objective function is to "*Binary Logistic*" which outputs a probability of an observation belonging to a class. *Learning_rate* can be adjusted, which determines the step size shrinkage used in updates of the weights to prevents overfitting (default=0.3). The hyperparameter *max_depth* controls the depth of the trees grown, where high values makes the model more complex and more prone to overfitting (default=6). The parameter *gamma* determines what the minimum loss reduction required to split a leaf node in a tree, the higher *gamma* is the more conservative will the model be (default=0). The parameter *min_child _weight* determines the minimum sum of instance weight in a child needs to continue splitting the tree, the larger *min_child_weight* is the more conservative the model will be (default=1). The paramter *colsample_bytree* is the fraction of features (randomly selected) that will be used to train each tree (default = 1) (*XGBoost Parameters* — n.d.).

Random forrest contain fewer parameters which can be tuned than XGBoost. *N_estimators* indicates the number of tree's in the forest (default=100). *Max_depth* determines the maximum depth of trees, higher values will make the model more complex and more prone to overfitting (default=none). *Min_samples_split* determines the minimum required number of observations in nodes for splitting, higher values help prevent overfitting be minimizing the complexity of the trees due to less splits of nodes (default=2). *Min_sample_leafs* determines the minimum number of samples that should be present in the leaf node after splitting a node, which helps controlling the growth of tree's and reducing overfitting (default=2) (Sexena 2020).

Adaboost can by tuned two hyperparamters. *N_estimators* which is the maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early (default=50). *Learning_rate* shrinks the contribution of each classifier by *learning_rate*. There is a trade-off between *learning_rate* and *n_estimators*. (*sklearn.ensemble.AdaBoostClassifier* n.d.)

Logistic regression will be tuned by two hyperparameters, the first is the regularization penalty, which describe which cost function is used (default=L2). The second hyperparamter is *C* which controls the penalty strength (default=1) (Brownlee 2019).

8.3 PointWinner Models

The first two models presented are the PointWinner models, where the objective is to predict the winner of the next point played, at any instance during a match from the first and second serve. Two models are

made due to the big difference between the first and second serve illustrated in the exploratory analysis. The target feature for both models is *PointWinner*, which states if the server or the returner wins the point. The approach for both models consist of applying the mentioned machine learning methods in the Methodology section with 10-fold cross-validation using default hyperparamters, on the train and validation data.

8.3.1 PointWinner first serve

For the first serve PointWinner model, there is a class imbalance of roughly 0.73/0.27, whereas the baseline is calculated by the number of times the server wins the point in percentage. Below in Table 6 are the average metric scores from 10-fold cross-validation applied to the described machine learning methods using standard parameters.

Table 6: Average values of 10-fold-cross-validation for the evaluation metrics for the first serve PointWinner models, on training and validation data

Model	**Accuracy score [%]**	**Recall**	**Precision**	**F1 score**	**Roc-Auc**
Baseline	73.2	-	-	-	-
XGBoost	73.2	0	0.168	0.001	0.573
Adaboost	73.2	0	0	0	0.573
Random Forrest	73.2	0	0.1	0	0.576
Logistic Regression	73.2	0	0.31	0.002	0.576

From Table 6 it can be seen that the general model performance is almost identical. All the models scores roughly equally to the baseline in the accuracy score, however performs very poorly in F1, recall and precision score, which means the models rarely or never actually predicts the returner as the winner. Logistic Regression does however score higher in precision, however due to low recall score the result is quite unreliable. XGBoost is chosen for further evaluation and hyperparamter tuning, due to the historical performance of XGBoost (Nielsen 2016), and its multiple hyperparamters which can be configured.

The optimal hyperparamters are found by random search and can be seen in appendix 11.B.1. Here it can be noted that the *scale_pos_weight* is 1.3 which scales the the gradient for the under represented class and encourages the loss function to focus on correcting mistakes of predicting for this class. Finally the tuned model is trained on the training and validation data and tested on the test data giving the following results:

Table 7: Tuned XGBoost for first serve PointWinner predictions on the test data

Model	**Accuracy score [%]**	**Recall**	**Precision**	**F1 score**	**Roc-Auc**
Baseline	73.2	-	-	-	-
XGBoost	73.4	0.012	0.437	0.023	0.569

The model achieves a slight gain in accuracy score compared to the baseline, but in general the overall

model performance is questionable in terms of the evaluation metrics. The model performs overall extremely poorly at predicting when the returner wins however, the Roc-Auc still indicates that it is slightly better than a random prediction. In order to examine how the different features contributes to the model, the gain of each feature is presented in figure 18, which describes the relative contribution of the corresponding feature to the model calculated by taking each feature's contribution for each tree in the model. It can be seen that numerous features has an impact on the model, resulting in the average individual contribution being low. A closer examination of the most contributing features shows that some of the more important features are *P1SetsWon* and *P2SetsWon* which describes the number of sets the server and returner has won at the point being predicted, and thereby their overall performance in the match. The another feature describing number of breakpoints won or missed for the returner given by *P2BreakPointMissedA* and *P2BreakPointWonA* is also relative important, which unlike the *P2SetsWon*, describes how the returner has played in critical points of the match. The rank of each player given by the features *P1Rank* and *P2Rank* is also presented as important features, which indicates the rank of a player has impact on the outcome.

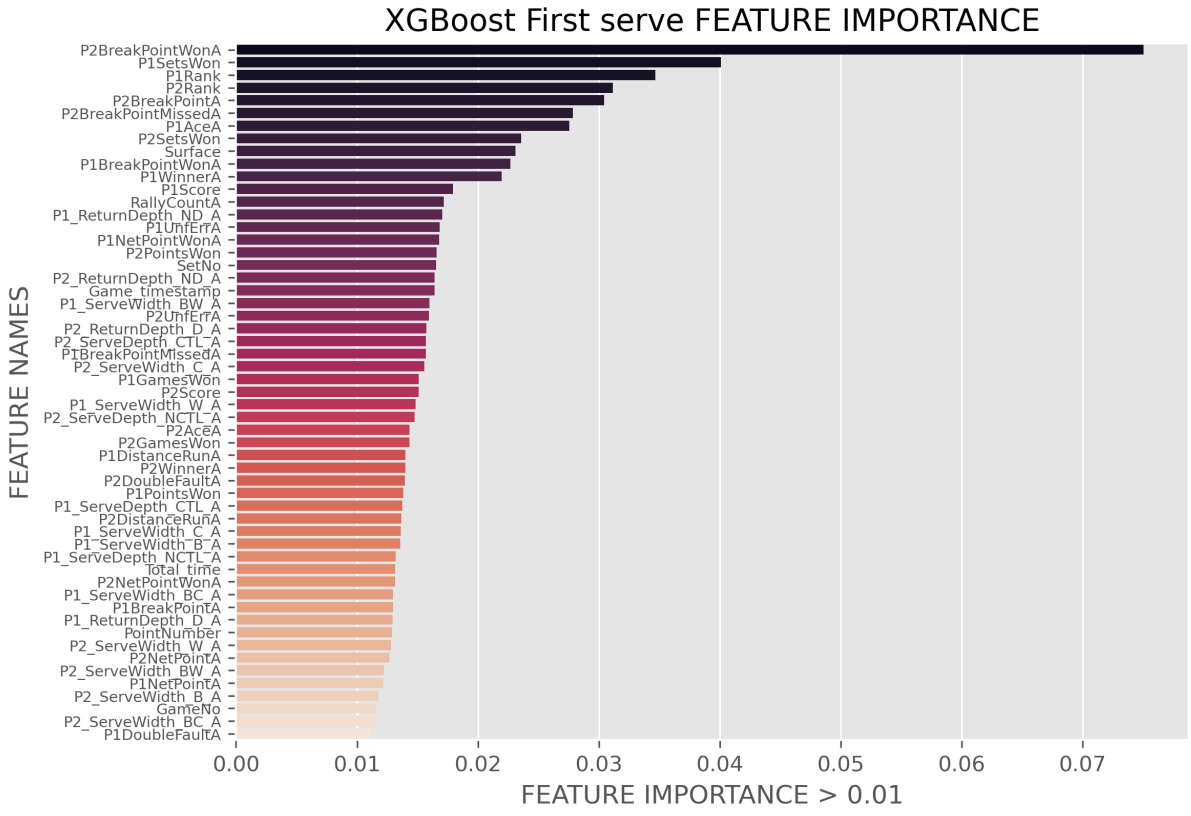


Figure 18: Feature importance (Gain) > 0.01 for tuned XGBoost for first serve PointWinner model, sorted by value

8.3.2 PointWinner second serve

The second serve PointWinner model, is trained on all observations decided from the second serve. The class representation is roughly equal at 57% P1 and 43% P2, whereas the baseline is calculated by the number of times the server wins the point from the second serve in percentage. Below in Table 8 the average metric scores from 10-fold cross-validation applied to the described machine learning methods

using standard parameters can be seen.

Table 8: Second serve PointWinner models performance, metrics are averages from 10-fold cross-validation on training and validation data

Model	Accuracy score [%]	Recall	Precision	F1 score	Roc-Auc
Baseline	57.2	-	-	-	-
XGBoost	53	0.448	0.523	0.483	0.541
AdaBoost	52.8	0.447	0.52	0.481	0.541
Random Forest	53	0.396	0.526	0.457	0.543
Logistic Regression	52.7	0.395	0.523	0.452	0.538

It shows that the model performances are almost identical. All models scores roughly 4% lower in accuracy than the baseline. The recall, precision and F1 scores are significantly higher for the second serve than with the first serve, which indicates that it is easier to predict when the returner wins in the second serve compared to the first serve. However, the class distribution is more balanced for the second serve and an examination of Roc-Auc values shows that the models are slightly better than random guessing. The two best performing models are XGBoost and random forest, which performs almost equally in all metrics, but XGBoost is slightly better at predicting the returner. Therefore XGBoost is chosen as the best model for further evaluation and hyperparamter tuning.

Likewise for the PointWinner model for the first serve the hyperparamters are found by random search and can be seen in appendix 11.B.2. The tuned XGBoost model is trained on the training and validation data and tested on the test data giving the following results:

Table 9: Tuned XGBoost for second serve PointWinner predictions on the test data

Model	Accuracy score [%]	Recall	Precision	F1 score	Roc-Auc
Baseline	57.2	-	-	-	-
XGBoost	53.1	0.447	0.506	0.475	0.535

It can be seen that the tuned XGBoost model performs 5.1% lower in accuracy score compared to the baseline. In general, the overall model performance is questionable in terms of the evaluation metrics. The Roc-Auc score indicates that the model is slightly better than a random guess. The model predicts 45% of the points where the returner wins correctly out of all possible points where the returner wins. And of all points which the returner is predicted win, 50% is correct. In order to examine which features which influence the model, the feature importance is shown in Figure 19.

Likewise the first serve model, as for the second serve model it can be seen that numerous features has an impact on the model, resulting in the average individual contribution being quite low. Some of the most important features based on gain are the accumulated features describing the score *P2BreakPointWona*, *P1SetsWon*, *P2SetsWon* are strong indicators how well the given player has performed during the match. Match describing features like *P1Rank* ,*P2Rank* and *Surface* also has a slightly higher impact on the model

than the average feature. It can be seen that the numerous serve and return accumulated features contributes to the model, however none of them have a significant impact.

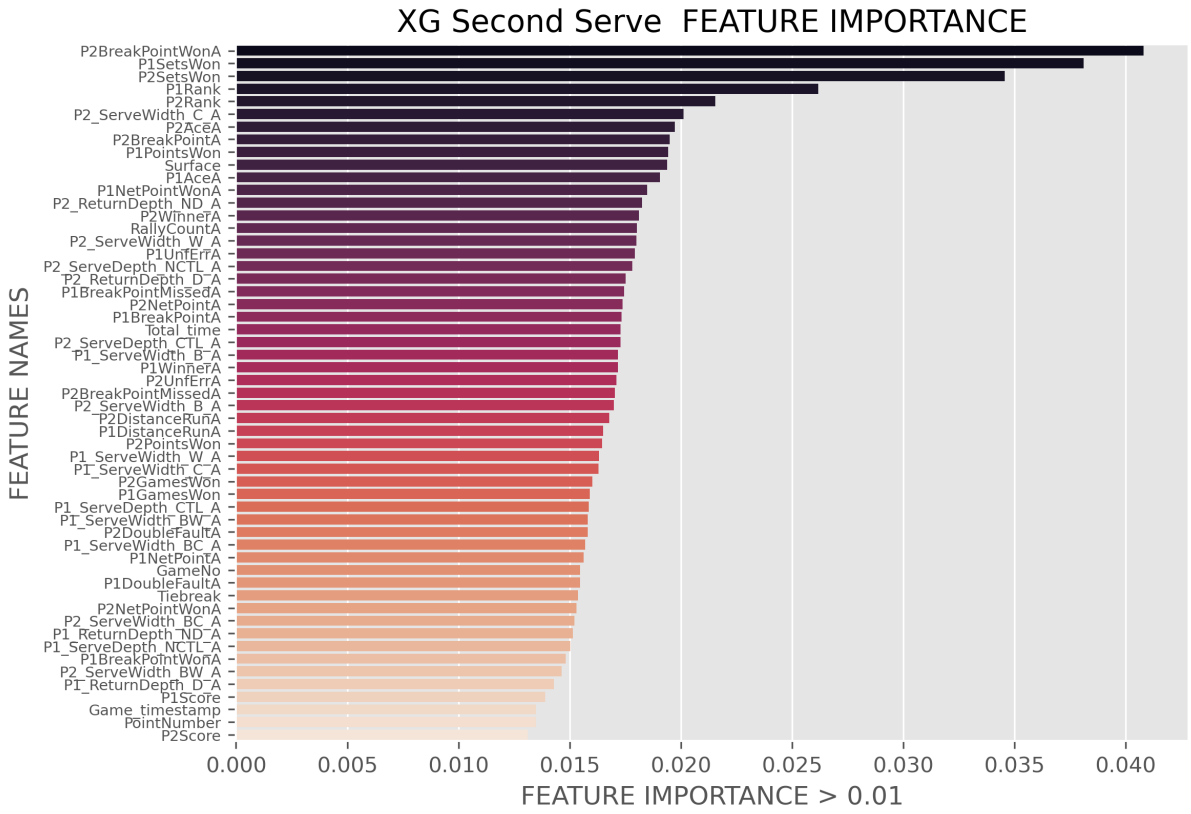


Figure 19: Feature importance (Gain) > 0.01 for tuned XGBoost for second serve PointWinner model, sorted by value

8.4 Revised PointWinner Models

The PointWinner models offered almost no useful strategic factors for wining a point, from the examination of the features. The models were purely based on historical information from the match being played. In order to further examine which actions within a point that are important for its outcome, information about the serve, return and length of the point will be included. The additional serve and return information includes the depth and the width of the serve, and the depth of the return. In order to not feed the models any specific information about which player that wins the point, and to focus more on points where the actions of player 2 has a slightly larger impact, points decided by aces or double faults will be excluded from the data. The length in seconds between each point played is included as well, to feed the models a rough estimate of the length of the point. This is done to examine whether the length of a point has an impact on the outcome of the point. Again will there be distinguished between first and second serve.

8.4.1 Revised first serve PointWinner Model

For the revised first serve PointWinner model the baseline is 64.2% which is calculated by the number of times the server wins the point in percentage. This is a decrease from the 73.2% in the previous first serve PointWinner model, which also states the class imbalance is 0.64/0.36. This is due to serve aces

and double faults being removed from the data and therefore the win percentage of the server have decreased. Below in Table 10 are the average metric scores from 10-fold cross-validation applied to the same machine learning methods.

Table 10: Average values of 10-fold-cross-validation for the evaluation metrics for the revised first serve PointWinner models, on training and validation data

Model	**Accuracy score [%]**	**Recall**	**Precision**	**F1 score**	**Roc-Auc**
Baseline	64.2	-	-	-	-
XGBoost	68.6	0.144	0.871	0.247	0.725
Adaboost	65	0.128	0.545	0.208	0.663
Random Forrest	64.2	0.0	0.15	0.0	0.644
Logistic Regression	64.2	0.022	0.508	0.042	0.575

In Table 10 it can be seen that XGBoost performs the best on all evaluation metrics. Therefore XGBoost is chosen as the best model for further evaluation and hyperparameter tuning. The optimal hyperparameters are found by random search and can be seen in appendix 11.B.3. The tuned XGBoost model is trained on the training and validation data and tested on test data. The result can be seen in Table 11.

Table 11: Tuned XGBoost for revised first serve PointWinner prediction on the test data

Model	**Accuracy score [%]**	**Recall**	**Precision**	**F1 score**	**Roc-Auc**
Baseline	64.2	-	-	-	-
XGBoost	70.7	0.421	0.632	0.506	0.750

Table 11 shows that the tuned XGBoost model has an accuracy score of 70.7%, which is well over the baseline. From the Roc-Auc score it can be seen that it is significantly better than a random guess. The F1 score indicates that is does a reasonably well job at predictions for the returner, which was the imbalanced class.

An examining of figure 20 shows that *P1Score* and *P2Score* are the most important features by far. This indicates that current score of each player has major influence on the outcome of the point. Features which describe the actual gameplay of the point are also represented. *Point_lenght_sec* and *ServeWidth_W* can be seen to have a larger impact on the model.

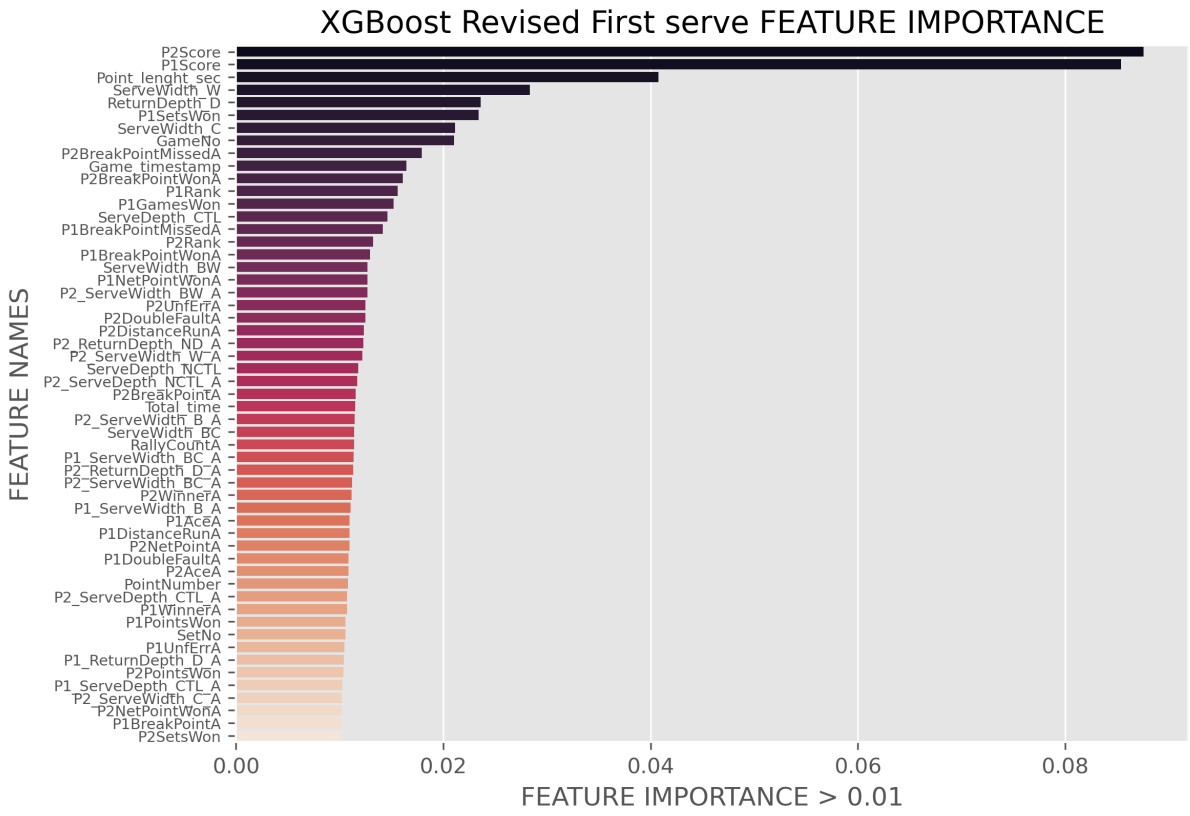


Figure 20: Feature importance (Gain) > 0.01 for tuned XGBoost for revised first serve PointWinner model, sorted by value

8.4.2 Revised second serve PointWinner Model

For the revised second serve PointWinner model the baseline is 53.3% which is calculated by the number of times the server wins the point in percentage. This is a decrease from the 57.2% in the previous second serve PointWinner model, the baseline states the class balance is roughly equal at 0.53/0.47. Below in Table 10 are the average metric scores from 10-fold cross-validation applied to the same machine learning methods.

Table 12: Average values of 10-fold-cross-validation for the evaluation metrics for the revised second serve PointWinner models, on training and validation data

Model	**Accuracy score**	**Recall score**	**Precision score**	**F1 score**	**ROC-AUC**
Baseline	53.1	-	-	-	-
AdaBoost	57.09	0.456	0.533	0.499	0.609
XGBoost	62.4	0.515	0.62	0.562	0.696
Logistic Regression	54.6	0.348	0.525	0.418	0.561
Random Forest	57	0.264	0.597	0.365	0.621

It can be seen in Table 12 that XGBoost again is the best performing model in all metrics. Therefore XGBoost is chosen as the best model for further evaluation and hyperparameter tuning. The optimal hyperparameters are found by random search and can be seen in appendix 11.B.4. The tuned XGBoost

model is trained on the training and validation data and tested on the test data. The can be seen in Table 13.

Table 13: Tuned XGBoost for revised second serve PointWinner prediction on the test data

Model	Accuracy score	Recall score	Precision score	F1 score	ROC-AUC
Baseline	53.1	-	-	-	-
XGBoost	62.4	0.559	0.62	0.588	0.695

Table 13 shows that the tuned XGBoost model has an accuracy score of 62.4 %, which is well over the baseline. From the Roc-Auc it can be seen that the model is significantly better than a random guess. The F1 score shows that the model does reasonable well at at predicting when the returner wins. When examining Figure 21, it can be seen that the most influential features are the same for the revised first and second serve PointWinner models. Again, it is *P1Score* and *P2Score* that are the most important features by far.

Some of the new added features are also important for the model, but instead of feature *ServeWidth_W* it is the feature *ReturnDepth_D* which is the most important serve/return related feature for the second serve model. This could indicate that the serve positions are more important for the first serve and the return positions are more important for the second serve. This can be one reason why the second serve model is better at classifying the returner than the first serve model.

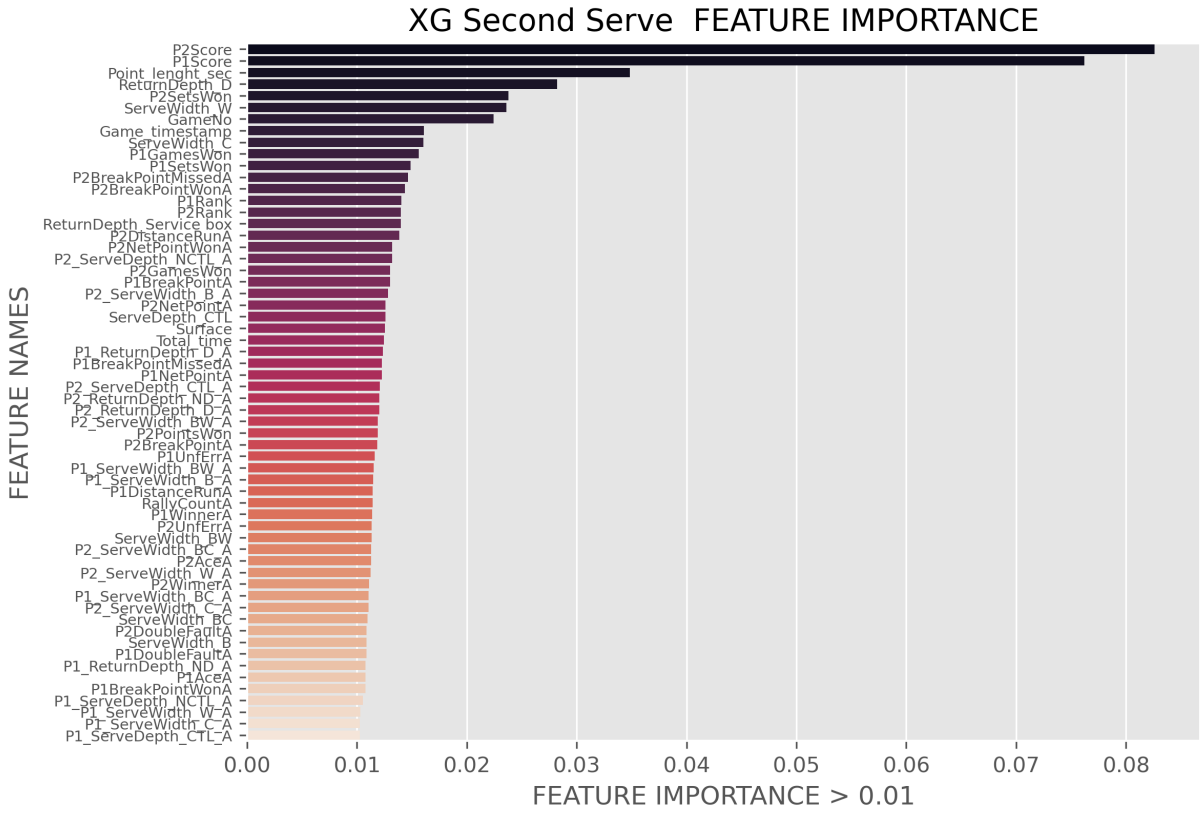


Figure 21: Feature importance (Gain) > 0.01 for tuned XGBoost for revised second serve PointWinner model, sorted by value

8.5 GameWinner Model

The GameWinner model consist of multiple models. The objective is to be able to predict who wins the next game at each point played in a game, which is given by the feature *GameWinnerA*. This is done by creating a variable *Game_timestamp*, which describes what point of a game is being played. The data is split into multiple subsets depending on its *Game_timestamp*, where tiebreak games are excluded due to how the serve advantage is switched. Overall this results in a reduction in data for the models for each *Game_timestamp* compared to a general model, especially at higher *Game_timestamp*'s. A timestamp model will therefore only be created as long as there is above 300 observations with the given timestamp. The advantage for creating a model for each *Game_timestamp* is that its possible to examine, which features is most important during a game and to see how the accuracy of the predictions differs during a game. The input features are the same as for the PointWinner model. The data distribution for the classes is more imbalanced for games than for points at 80% P1 and 20% P2, this is due to the server will serve in all points of a game. The imbalance evens out from *Game_times* 4, as there begins to be relative more cases of the returner winning. The baseline is calculated by the percentage of times the server wins a game, which is 80.5 % of the time. The four machine learning methods are applied with 10-fold cross-validation using default hyperparamters, on the train and validation data for each *Game_timestamp* subset. Below the average metric values for all *Game_timestamp*, which are the averages of results for 10-fold cross-validation for each *Game_timestamp*

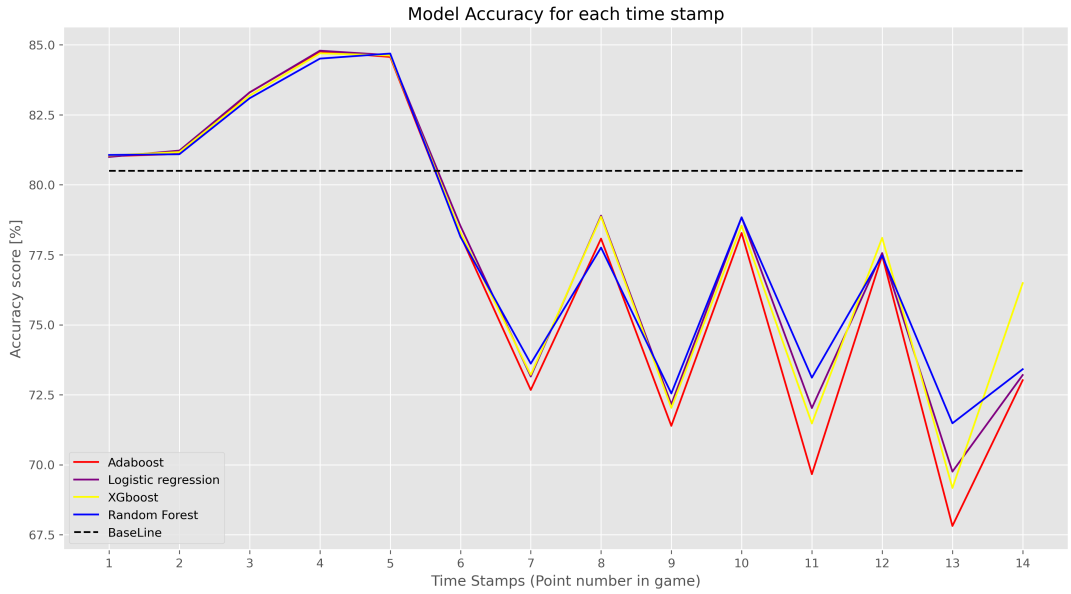


Figure 22: Machine learning methods average accuracy of 10-fold cross-validation for each Game_timestamp

Table 14: Game_timestamp average of the average of 10-fold cross-validation for each classification model

Model	Accuracy score	Recall score	Precision score	F1 score	ROC-AUC
Baseline	80.5	-	-	-	-
AdaBoost	77.4	0.297	0.519	0.355	0.724
XGBoost	77.9	0.307	0.523	0.345	0.729
Logistic Regression	77.8	0.325	0.541	0.366	0.74
Random Forest	77.8	0.275	0.47	0.223	0.737

In Figure 22 shows that the models performs quite similarly in accuracy for the first six *Game_timestamp*. There after random forest, logistic regression and XGBoost are less extreme in their predictions and Adaboost is a bit more erratic. Overall in Table 14 it can seen that XGBoost just barely scores highest in the average accuracy score, even though the score is roughly 0.6 % lower than the baseline. However, it shows that within the first six *Game_timestamps* all models scores higher than the baseline. Logistic regression performs just slightly better on average than the other models when predicting for the returner to win. Due to XGBoost generally high scores, its historical performance with complex data (Nielsen 2016) and its multiple hyperparamters that can be configured, it is chosen for hyperparamter tuning and further evaluation. For hyperparamter tuning random seach for each *Game_timestamp* model have been made. The parameters obtained can be found in appendix 11.C. The final XGBoost model performances for each *Game_timestamp* can be seen in Figure 23, which is trained on the training and validation data and is tested on the test data. The standard error is found by running the model 10 times with reshuffled data for each *Game_timestamp*.

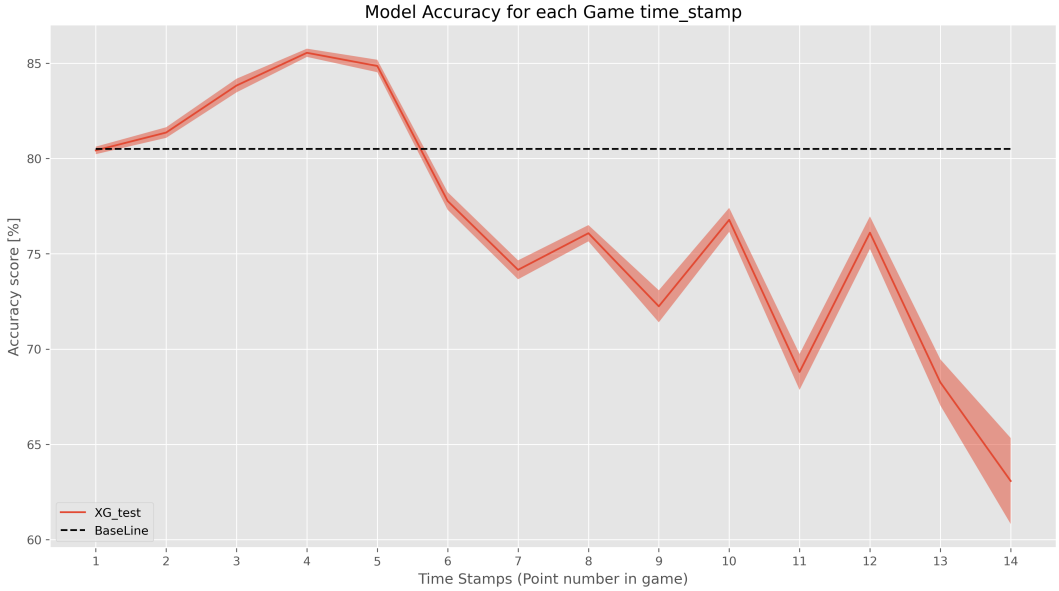


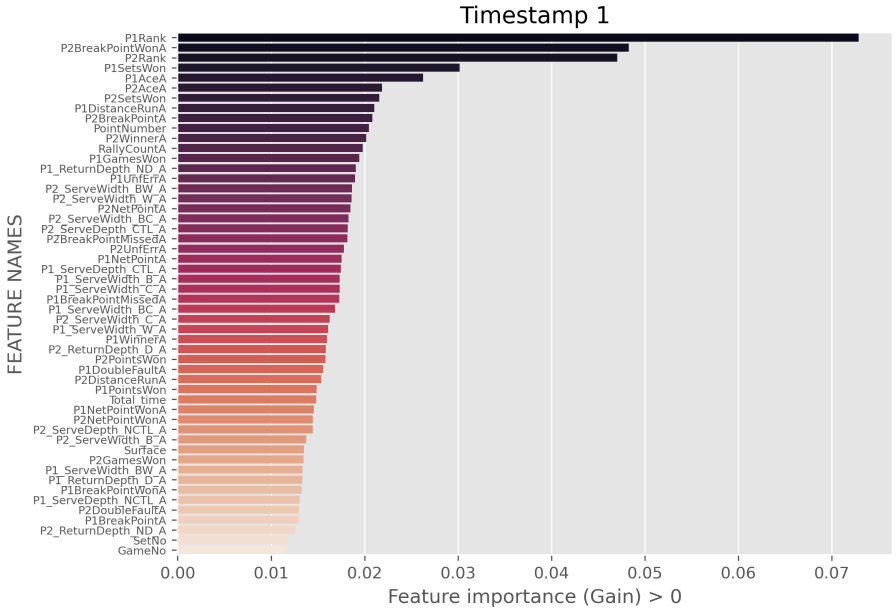
Figure 23: Accuracy score for each Game_timestamp for the tuned XGBoost model, plotted with standard error

Table 15: Table of performance metrics for the tuned XGBoost GameWinner models

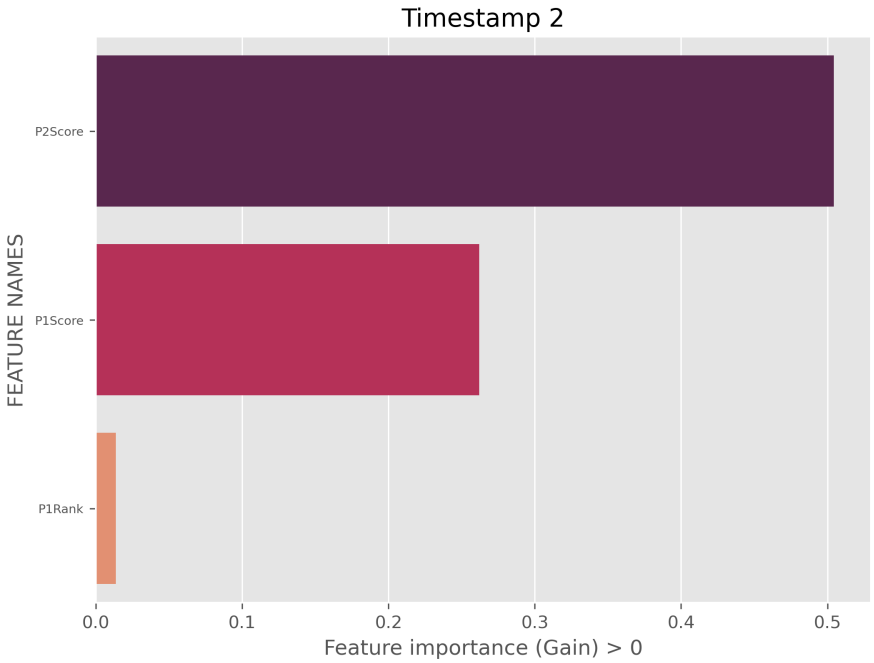
Game_timestamp	**Accuracy Score [%]**	**Recall score**	**Precision score**	**F1 score**	**ROC-AUC**
1	80.4	0.002	0.5	0.004	0.634
2	81.39	0.053	0.407	0.094	0.717
3	83.8	0.334	0.614	0.433	0.812
4	85.5	0.365	0.668	0.472	0.874
5	84.89	0.511	0.727	0.6	0.866
6	77.8	0.487	0.555	0.519	0.792
7	74.2	0	0	0	0.596
8	76.1	0.541	0.541	0.541	0.760
9	72.2	0.055	0.5	0.99	0.619
10	76.8	0.638	0.543	0.586	0.763
11	68.8	0.025	1	0.049	0.606
12	76.1	0.567	0.548	0.557	0.753
13	68.3	0.056	0.25	0.091	0.384
14	63.1	0.364	0.444	0.4	0.624
Average	76.8	0.286	0.521	0.381	0.700

The XGBoost models for each *Game_timestamp* scores higher than the baseline in accuracy for timestamp 2 till 5. However, the average accuracy at 76.8% is worse than the baseline. This could be due to lack of data for the higher *Game_timestamps*, and maybe the reason behind the big drop in accuracy at *Game_timestamp* 14. which also can be seen as the standard error increases with the *Game_timestamps* The models for *Game_timestamp* 1, 2, 7, 9, 11 and 13 all scores zero or almost zero in precision, recall and F1 score. This means that the models are predicting the server to win in all or all most all the observations. The common denominator for these models expect *Game_timestamp* 2, is that the point score of each player is equal. It can be seen that that the accuracy score takes a drop from timestamp 5, as the possible combinations for a lead in point score for the server or the returner is reduced to one point or equal scores. This could also explain the general trend of the accuracy from *Game_timestamps* 6 going up for even numbers and down for uneven, since at even *Game_timestamp* a player will have a point advantage and will be ”closer” to winning the game. An examination of the F1 scores shows that the models generally are performing better at predicting when the returner wins at the *Game_timestamp* 3 to 6 and the even *Game_timestamp*'s larger than 6. An examination of the ROC-auc shows that the models for all *Game_timestamps* except 13 are relatively competent in their prediction compared to a random guess.

In order to examine how the features impact the different models, the feature importance for specific *Game_timestamps* will be showed. The first two models features that will be examined is for *Game_timestamps* 1 and 2, which can be seen in Figure 24.



(a) Feature importance for Game_timestamp model 1



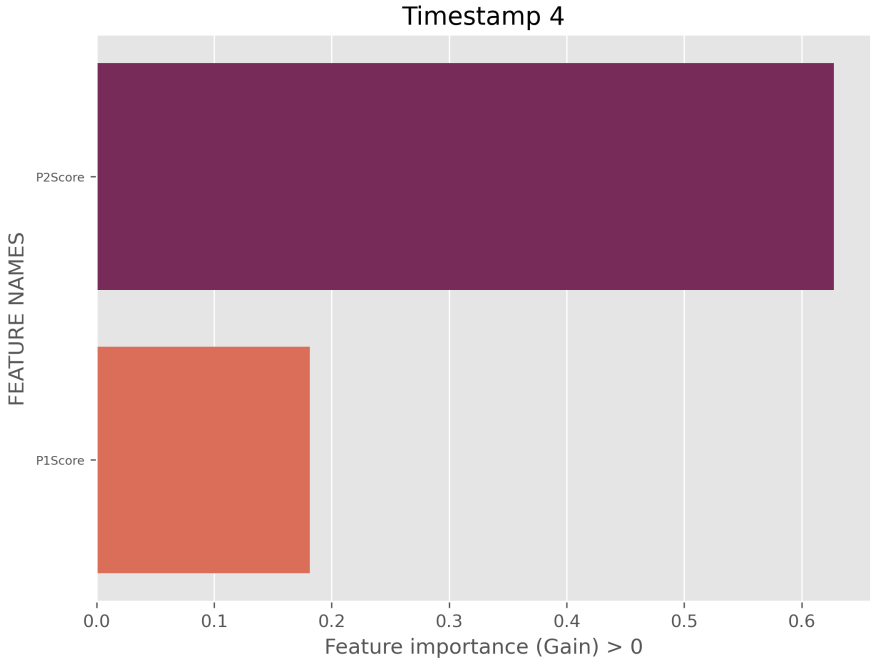
(b) Feature importance for Game_timestamp model 2

Figure 24: Feature importances for first two Game_timetamp models

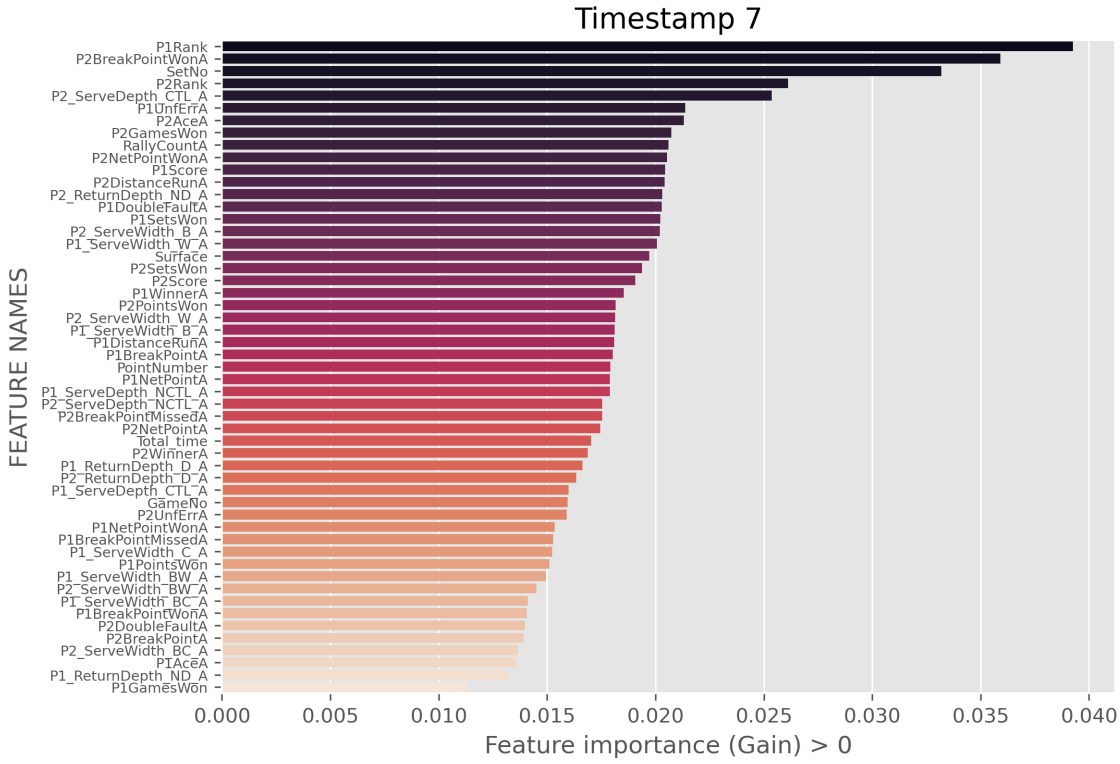
For *Game_timestamp* model 1, the point score will be 0-0 and therefore neither player has an lead in points in the game. The most influential features are the rank of the server (*P1Rank*), how many breakpoints the returner has won in the match (*P2BreakPointWonA*) which is a clear indication of how well he has performed in the match while their opponent has a serving advantage, and the rank of the returner (*P2rank*). Overall the important features for *Game_timestamp* model 1, are quite identical for the point winner models. It can however be seen that the amount of gain for each feature is quite low, which could indicate that the majority of the features have a significant impact on the model. The number

of influential features for *Game_timestamp* model 2, which score almost equal in F1, recall and accuracy score have significantly fewer. Here it is *P2Score* is the most influential based on gain followed by *P1Score*, which could indicate that as soon as a player obtains an advantage in points the models does not have to utilize all the features for accurate predictions.

The accuracy score for the models are largest at *Game_timestamp* model 4, which is where a player can obtain the biggest lead (40-0) against his opponent, and thereby obtain the highest chance of winning the point. It can be seen in Figure 25a that only two features are utilized in the model for *Game_timestamp* model 4, which are the scores for each player. A general drop off in the accuracy can be seen from *Game_timestamp* model 4 to *Game_timestamp* model 7. *Game_timestamp* model 7 is important since the players for the first time, besides at the start of the game, only can have an equal score (40-40) and therefore will start playing for the advantage. In 25b the feature importance's can be seen for *Game_timestamp* model 7. Likewise as for *Game_timestamp* model 1, multiple features have an influence on the model, which makes the average feature importance low. The most important features are the same as for *Game_timestamp* model 1, *P1Rank* and *P2BreakPointWonA*.



(a)



(b)

Figure 25: Feature importance for Game_timestamp models 4 (a) and 7 (b)

8.6 SetWinner Model

The final model is the SetWinner model and like the GameWinner model it consist of multiple models. Here the objective is to predict who wins the next set, from the first point in each game in each set. This is done by only using points where *Game_timestamp* is equal to one. As in the GameWinner models the data in the SetWinner models is split into multiple subsets depending on the feature *GameNo*, which states what game in a set is being played. This results in significant reduction in data compared to a general model, especially for higher *GameNo's* as the data is rapidly declining as *GameNo* increases, models will therefore only be created for the first 13 *GameNo's*. The advantage of creating this type of model is the same as for the GameWinner model, that its makes it possible to examine which features are important at the different *GameNo's*. The reason only the first point within a game is chosen, is to avoid that the models are trained on almost identical data.

The target variable is *SetWinnerA* and the input variables are the as the GameWinner models. The baseline is calculated by the number of times the highest ranked player wins the set in percentage, which is 61.1% of the time. The four machine learning methods are applied with 10-fold cross-validation using default hyperparamters, on the train and validation data for each *GameNo* subset. Below the average metric values for all *GameNo*'s, which are the averages of results for 10-fold cross-validation for each *GameNo* can be seen in Table 16.

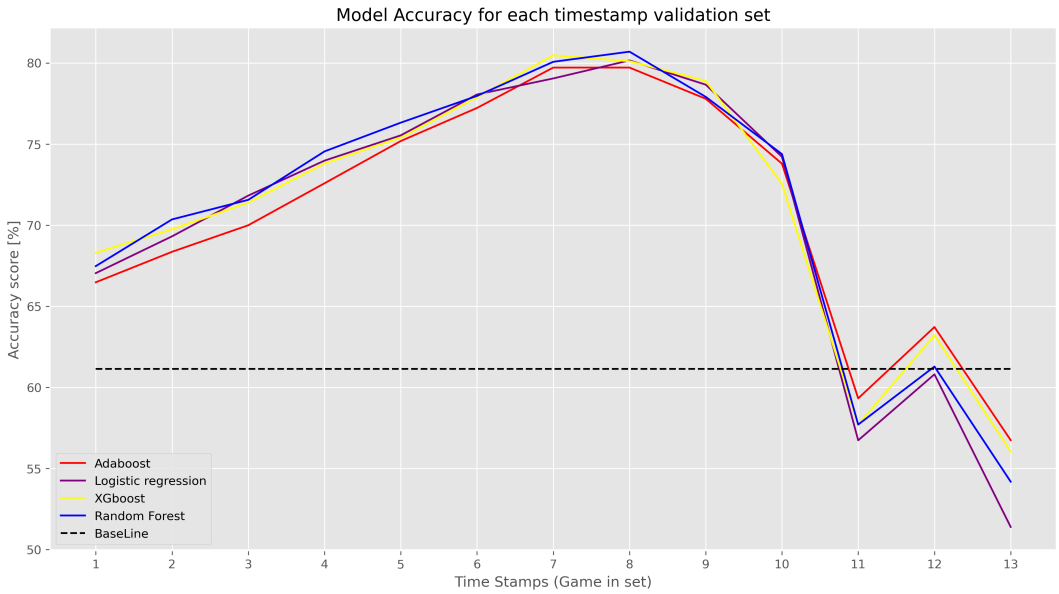


Figure 26: Average validation accuracy-score of each model

Table 16: Machine learning methods average accuracy of 10-fold cross-validation for each

Model	**Accuracy score**	**Recall score**	**Precision score**	**F1 score**	**ROC-AUC**
Baseline	61.4	-	-	-	-
AdaBoost	70.82	0.706	0.707	0.704	0.778
XGBoost	71.19	0.709	0.71	0.708	0.784
Logistic Regression	70.52	0.711	0.704	0.702	0.776
Random Forest	71.11	0.718	0.712	0.706	0.782

In Figure 26 can the four validation models for each *GameNo* average accuracy be seen. All machine learning models perform worse than the baseline in *GameNo* 11 and 13, which is when the standings in the set is either 5-5 or there is a tiebreak. On the other hand does all the models perform best on *GameNo* 8, which is when the standing in the set is 3-4 or 2-5.

Table 16 shows the average accuracy of each machine learning models 13 *GameNo* models. All the machine learning models performs better on average than the baseline, but the XGBoost and random forest models performs better than the rest on average. The random forest models performs on average better in recall and precision score, but the XGBoost performs on average better in Roc-Auc and only a little worse in recall and precision score. The two models performs almost identically, so chosen either would be acceptable. However, XGBoost is chosen because it have more hyperparameters to tune. After hyperparameter tuning, will the XGBoost models hyperparameters for each *GameNo* be set to specific values, which can be seen in appendix 11.D.

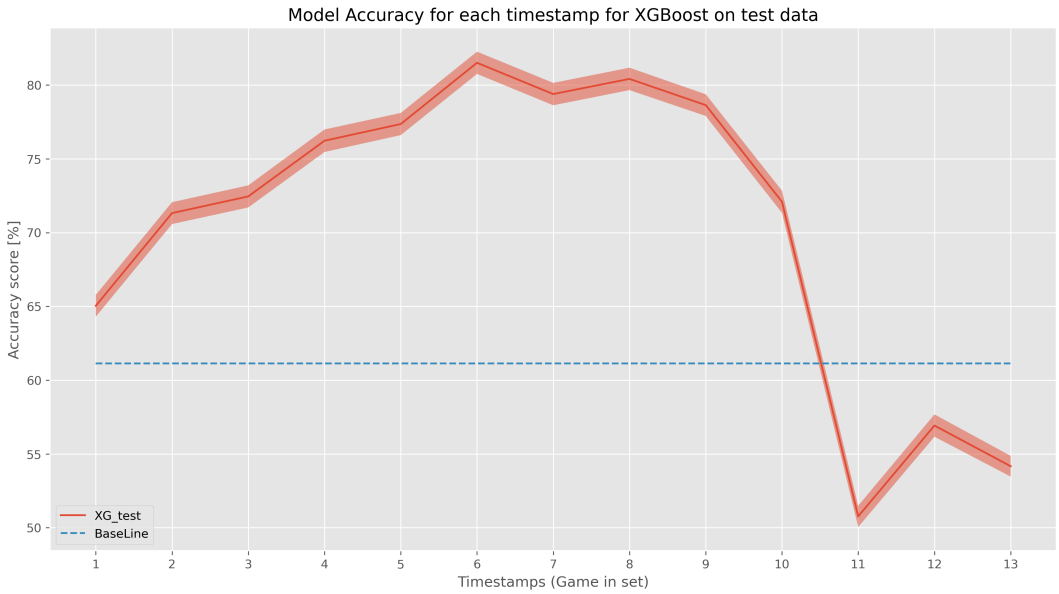


Figure 27: Accuracy score for the tuned XGBoost model for each GameNo tested on the test data

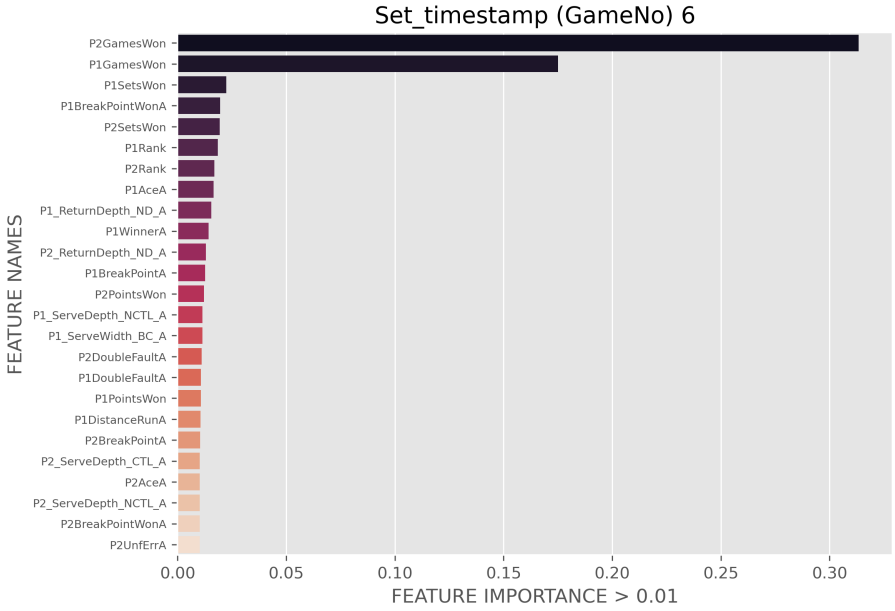
The tuned XGBoost models average accuracy-score on the test data is 70.48%. Looking at the tuned XGBoost models for each *GameNo* tested on the test data in Figure 27, *GameNo* model 6 is the best performing. At *GameNo* 6 the game score in the set can be 3-2, 2-3, 4-1, 1-4, 5-0 and 0-5. The worst performing *GameNo* models are number 11 and 13. They both performs worse than the baseline, that is when the game score in the set is either 5-5 or 6-6 (tiebreak). This is most likely because that these *GameNo* accrue less than *GameNo* below 10 and therefore is there less training data.

When examining Table 17, which contains the metrics of each *GameNo* model and the average of all *GameNo* models, it can be seen that most of the *GameNo* models perform quite well. Looking at the same *GameNo* from before, the *GameNo* 6 model have high scores in all evaluation metrics and this pulls the average for all metrics up. On the other hand does the *GameNo* models 11, 12 and 13 hurt the average scores.

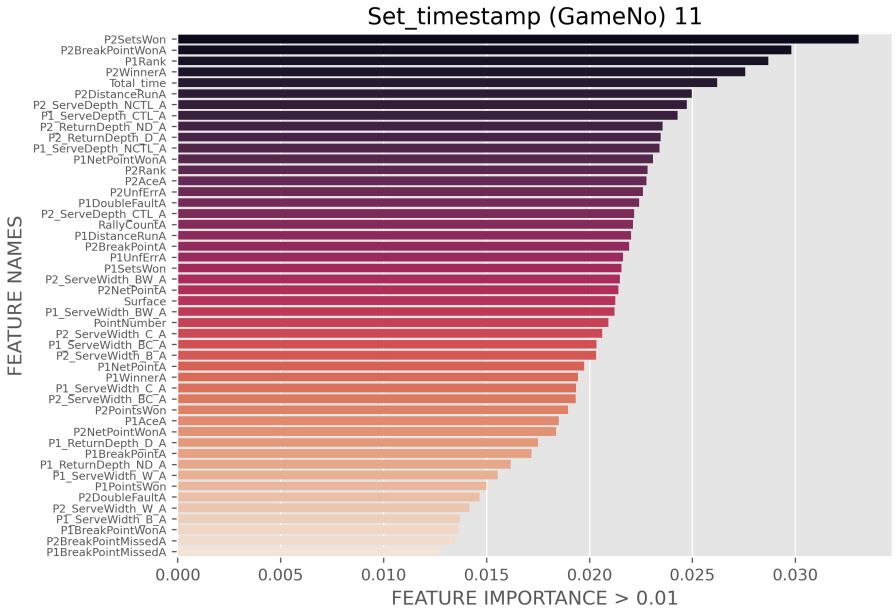
Table 17: The average evaluation metrics for both classes for the tuned XGBoost SetWinner models

GameNo	**Accuracy Score [%]**	**Recall score**	**Precision score**	**F1 score**	**ROC-AUC**
1	65.0	0.649	0.659	0.645	0.752
2	71.3	0.713	0.714	0.713	0.768
3	72.5	0.724	0.725	0.724	0.780
4	76.2	0.762	0.764	0.762	0.830
5	77.4	0.773	0.783	0.772	0.862
6	81.5	0.815	0.815	0.815	0.879
7	79.4	0.793	0.796	0.793	0.882
8	80.4	0.802	0.804	0.803	0.877
9	78.6	0.786	0.786	0.786	0.879
10	72.1	0.693	0.693	0.693	0.776
11	50.8	0.492	0.491	0.487	0.536
12	56.9	0.594	0.616	0.557	0.689
13	54.2	0.524	0.529	0.511	0.517
Average	70.48	0.747	0.705	0.697	0.771

When examining the feature importance for the 13 different *GameNo* models, which can be seen in the JupyterNotebook "SetWinner_Model.ipynb", it can be seen that all models have features which repeats. The features with high importance which repeats most often are *P1Rank*, *P2Rank*, *P1GamesWon* and *P2GamesWon*. This means that the average model agree with the assumption behind the baseline, which is that player rank have influence on who wins the set. However, it use extra information, which makes it predicting better than the baseline. Looking at Figure 28a, where the features importance of *GameNo* model 6, which performs the best is shown, it can be seen that *P1GamesWon* and *P2GamesWon* are the most important features. The reason for this could be that in two of the possible standing (5-0 or 0-5), one of the players wins the set if they win the game. And therefore the model weight how many games the player have high. Looking at *GameNo* model 11's important features, seen in Figure 28b, it can be seen that there are a lot of features with importance over 0.01. This can indicate that there is not a feature which is paramount for determine which player who wins the set in the first point in the game when the standing in games are 5-5.



(a) Feature importance for the GameNo model 6



(b) Feature importance for the GameNo model 11

Figure 28: Feature importance for the GameNo model 6 and 11

9 Discussion

From the analysis, it is clear, that all the models have their own challenges. The PointWinner models performed equal or worse than their baselines, and the first serve PointWinner model had significant difficulties making predictions for the returner. The revised PointWinner models performed better than their baseline, but most importantly they were much better at predicting when the returner wins, they gave more information about which factors are important to win a point however nothing groundbreaking. The GameWinner models only primary utilized the score features at some timestamps, and was also

having difficulties predicting the returner at certain timestamps where the score of the players are equal, however they did performed well in terms of accuracy of the predictions. The SetWinner models performed on average better their baselines, however they gave no groundbreaking insight in how to win a set. There can be many reasons for why this is happening and plenty of possible changes or additions, which could be tested in order to improve the models. This will be explored in the following section.

9.1 Model discussion

The first serve PointWinner model primary challenge was that it did not predict the returner as the winner in any of the points. Therefore the accuracy score was almost the same as the baseline. When the important features were examined could it be seen that multiple attributes had influence on the model, however the general contribution was very low. This could indicate that none of the attributes actually having any influence on the model, and its predictions are almost solely based on the class distribution. The second serve PointWinner model did a far better job at predictions for the returner, however the overall accuracy of the model was roughly 5% lower than its baseline. The Roc-Auc was just above 0.5 which indicated that model was not performing much better than a random guess. An examination of the features showed, as the first serve PointWinner model, that average feature contribution was very low, which could indicate that none of the features actually have an impact on the models performance.

A comparison of the revised PointWinner models with the PointWinner models show that there were an improvement in regards to discovering point specific factors for winning a point and predicting the returner. The reason for the improvement in discovering point specific factors is probably because the revised PointWinner models had input features which gave some information of the actual gameplay, which the PointWinner models did not have. It was however seen in the revised Pointwinner models that the score had a major impact on the predictions, which aligns with the findings of (Klaassen & Magnus 2001), that points in tennis are not independently and identically distributed. The revised PointWinner models improvement in predicting the unfavored class (returner) is likely because of two things. The first is that the extra input features could give the models better circumstances for predicting the returner. The second is that all serve aces and double faults were removed, which is where the server performed the best or worst. Therefore the number of points were the server wins are decreased and the models were less likely to predict the server as winner. It could also be a combination of the two. When comparing the two revised PointWinner models, it could be seen that they had many of the same important features. However, they differentiated from each other in that the first serve model had a serve position feature (*ServeWidth_W*) as the most important serve/return feature and the second serve model had a return position feature (*ReturDepth_D*) as most important serve/return feature. The reason for this could be that the model catches the tendency in tennis were the server has the advantage, especially in the first serve and less in the second serve.

The Gamewinner models did well at describing the overall trend of when the winner of a game was most likely to be decided. It was seen that at higher *Game_timestamp*'s for equal point scores the models performed generally worse at predicting the overall winner and especially predicting the returner. The reason for this might be that the models for these *Game_timestamp*'s suffer from the same as first serve PointWinner model, where roughly all features had a small influence on the model, which could indicate that none of the features actually had an influence on the model. This could explain why the models

replicate the statistical baseline based on times the server wins. In terms of predicting who actually wins the game it was seen that the score of either the server or returner could be solely used to make accurate predictions for *Game_timestamp's* where there was a score difference. An ideal approach would be to model every score combination in a game, instead of basing the models depending on how many points were played in a game. Looking at the model for *Game_timestamp* 4, there can be multiple possible score combinations, 40-0, 30-15, 15-30 and 0-40, where the probability for the server and returner to win can be expected to vary a lot. Using this approach would get a more precise model for predictions and would make it possible to examine which features has the biggest impact at the different combinations, other then the point score of the players. There is however a possibility that the feature influence for the additional models would be similar to *Game_timestamp* models with equals points scores, and thereby not provide any insight-full information.

Looking into the SetWinner models, the biggest problem and source of error was how the models were designed. Using only the first point in each game for all sets for training and testing resulted in a significant reduction in data. At most one match can have 5 sets and 13 games in each set. And with the 709 matches in the data will there at most be 46085 points of the total 157385 points for the model to be trained and tested upon. Still on average performed the models better than the baseline. However, the feature importance showed that were not any gameplay specific features which were important for the models, but instead features that tells how a player in general wins a set in *P1GamesWon* and *P2GamesWon*, and how good a player is, in *P1Rank* and *P2Rank*.

The best performing models throughout this thesis were the tree based model XGBoost. The advantages of the tree based models are that they can capture non-linear relationships in data, high order interactions and automatically performs feature selection which makes them great for the prediction objective of this thesis. The tree based models are however not ideal to examine how different features impacts the model, due to the way the models are constructed. The performance parameter used for XGBoost were gain, which implies the relative contribution of the corresponding feature to the model, but not how they are important for the predictions. Examining feature importance for tree based models leaves therefore a lot of room for interpretation, compared to models like logistic regression, where positive coefficients represent features for one class and negative features for the other class.

9.2 Future Research

Based on the experience obtained during this thesis, an approach for further development of the different models could be adding additional data sources. These sources could be full descriptions of how points were played and emotion capture of players. Possible sources of this data could be the hawk-eye system, which is implemented for the Grand Slam tournaments. The systems records data for every action taken on the tennis court, however the hawk-eye data is not publicly available yet. The emotion capture could come from a kind of image recognition algorithm used on tennis video footage to capture the emotional state of the players. The reason this could be interesting is because player performance can vary because of their mental and physical state. With the additional data sources of hawk-eye and emotion capture it would be possible to further examine which factors are important for the outcome of a point, game and set. The data would contribute with specific information of the gameplay and the state of mind of players, contrary to the approach in this thesis which is more orientated towards how

the historical data of a match is important for predicting the outcome of a point, game and set.

Another approach which could improve the model performances, would be adding player specific data. This is due to the individual players strength and physical appearance vary, which the models therefore could account for. Players like John Isner, which is said to be the best serving player in the game, would the model recognize and therefore most likely predict as winner of the point/game when serving.

The use of more complex models could also improve the results. Implementing a recurrent neural network would introduce a more complex model, which could maybe perform better than those used in this thesis. However, neural networks are described as "black-box" models, which means that information about how each feature effects the model would be limited. This would provide little information on how players could improve their game.

A mentioned in the model discussion, the current GameWinner model could be further developed by making models for each score combination. It would be possible with the available data sources. By making a model for each score, the most important attributes for the majority of *Game_timestamps*, *P1Score* and *P2Score* would be redundant. It would here by be possible to further investigate what are the most important features for making predictions during a game.

10 Conclusion

The purpose of this thesis was to examine if the outcome of men's single Grand Slam tennis matches could be predicted using data from the match. The data used was supplied from Jeff Sackmann's github. The data contained matches from two Grand Slam tournaments, Wimbledon and US Open from 2016 to 2020, which after data cleaning and preparation consisted of 709 matches.

The problem statement was approached with the use of machine learning methods to predict the outcome of tennis matches, and hereby examine the important factors in tennis matches. This approach was chosen on the basis of previous literature on the subject, which had shown to be a feasible way to analyse tennis and sports in general. The approach presented has primarily been focused on how the historical events of matches can be used to predict the outcome of a given match, as a result of the available data.

The models performance were evaluated by evaluation metrics derived from the confusion matrix, and statistics of from the data. Overall it was seen that it was possible to predict the outcome of points, games and sets, however with varying performance. The PointWinner models only proved to be a fraction better than plain statistics, due to difficulties predicting when the returner won. The revised PointWinner models, where serve and return specific features for the actual point were added, only provided limited information about how the serve and return were important for the outcome of the point. However, they were better to predict the returner as the winner. The GameWinner models proved to be relative accurate in terms of accuracy in the predictions, especially when a player had the opportunity for a point lead. The SetWinner models performed on average better than the baseline and had a average score around 70% in all metrics.

The features which proved to influence the models the most was either score related or the rank of the players. For all the models the accumulated features for the serve and return position had no significant

influence. However, in the revised PointWinner models, it was seen that the placement of a serve and return ball offer some insight on the outcome of a point, but the player score was still by far the most important. It had therefore not been possible to identify factors which are gives a strategic advances in tennis.

The approach presented in this thesis can be used to predict the outcome of points, games and sets, however it is not an ideal approach for examining which factors are important for the outcome of tennis matches.

References

Bhandari, A. (2020), 'Feature Scaling — Standardization Vs Normalization'.
URL: *https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/*

Breiman, L. (2001), Random Forests, Technical report.

Brownlee, J. (2019), 'Tune Hyperparameters for Classification Machine Learning Algorithms'.
URL: *https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/*

Carboch, J. (2017), 'Comparison of game characteristics of male and female tennis players at grand-slam tournaments in 2016', *Trends in sport sciences* **4**, 151–155.

Chen, T. & Guestrin, C. (2016), XGBoost: A Scalable Tree Boosting System, Technical report.
URL: *https://github.com/dmlc/xgboost*

Clarke, S. R. & Dyte, D. (2000), 'Using official ratings to simulate major tennis tournaments', *International Transactions in Operational Research* **7**(6), 585–594.

Ghosh, S., Sadhu, S., Biswas, S., Sarkar, D. & Sarkar, P. P. (2019), 'A comparison between different classifiers for tennis match result prediction', *Malaysian Journal of Computer Science* **32**(2), 97–111.

Herlau, T., Schmidt, M. N. & Mørup, M. (2016), 'Introduction to machine learning and data mining', *Lecture notes of the course of the same name given at DTU (Technical University of Denmark)* .

Klaassen, F. J. & Magnus, J. R. (2001), 'Are points in tennis independent and identically distributed? Evidence from a dynamic binary panel data model', *Journal of the American Statistical Association* **96**(454), 500–509.

Kovalchik, S. (2021), 'Why Tennis Is Still Not Ready to Play Moneyball', *Harvard Data Science Review* p. 2021.
URL: *https://hdsr.mitpress.mit.edu/pub/uy0zl4i1*

Ma, S. M., Liu, C. C., Tan, Y. & Ma, S. C. (2013), 'Winning matches in Grand Slam men's singles: An analysis of player performance-related variables from 1991 to 2008', *Journal of Sports Sciences* **31**(11), 1147–1155.

Makino, M., Odaka, T., Kuroiwa, J., Suwa, I. & Shirai, H. (2020), 'Feature Selection to Win the Point of ATP Tennis Players Using Rally Information', *International Journal of Computer Science in Sport* **19**(1), 37–50.

Nielsen, D. (2016), Tree boosting with xgboost-why does xgboost win" every" machine learning competition?, PhD thesis, NTNU.

O'Shannessy, C. (2019), 'Tennis Channel Feature: Moneyball In Tennis'.
URL: *https://www.braingametennis.com/tennis-channel-feature-moneyball-in-tennis/*

Oughali, M. S., Bahloul, M. & El Rahman, S. A. (2019), Analysis of NBA players and shot prediction using random forest and XGBoost Models, *in* '2019 International Conference on Computer and Information Sciences, ICCIS 2019', Institute of Electrical and Electronics Engineers Inc.

Perfect Tennis (2020), 'Tennis Court Surfaces and Court Speeds'.
URL: *https://www.perfect-tennis.com/tennis-court-surfaces-and-court-speeds/How$_D$o$_T$ournaments$_D$etermine$_C$ourt$_S$peed*

Sexena, S. (2020), 'Random Forest Hyperparameter Tuning in Python — Machine learning'.
URL: *https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/*

Shahul ES (2021), 'Hyperparameter Tuning in Python: a Complete Guide 2021 - neptune.ai'.
URL: *https://neptune.ai/blog/hyperparameter-tuning-in-python-a-complete-guide-2020*

sklearn.ensemble.AdaBoostClassifier (n.d.).
URL: *https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html*

Srivastava, T. (2019), 'Evaluation Metrics Machine Learning'.
URL: *https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/*

Tennis Quotes by Nadal (n.d.).
URL: *http://rajora.in/tennis/tennis-quotes-by-nadal/*

Top 10 Most Popular Sports in The World (2021).
URL: *https://sportsshow.net/top-10-most-popular-sports-in-the-world/*

XGBoost Parameters — (n.d.).
URL: *https://xgboost.readthedocs.io/en/latest/parameter.html*

11 Appendix

11.A Modelling feature list

P1GamesWon, P2GamesWon, GameNo, PointNumber, Tiebreak, P1Score, P2Score, P1PointsWon, P2PointsWon, P1_ServeWidth_B_A, P1_ServeWidth_BC_A, P1_ServeWidth_BW_A, P1_ServeWidth_C_A, P1_ServeWidth_W_A, P1_ServeDepth_CTL_A, P1_ServeDepth_NCTL_A, P1_ReturnDepth_D_A, P1_ReturnDepth_ND_A, P2_ServeWidth_B_A, P2_ServeWidth_BC_A, P2_ServeWidth_BW_A, P2_ServeWidth_C_A, P2_ServeWidth_W_A, P2_ServeDepth_CTL_A, P2_ServeDepth_NCTL_A, P2_ReturnDepth_D_A, P2_ReturnDepth_ND_A, P1AceA, P2AceA, P1WinnerA, P2WinnerA, P1DoubleFaultA, P2DoubleFaultA, P1UnfErrA, P2UnfErrA, P1NetPointA, P2NetPointA, P1NetPointWonA, P2NetPointWonA, P1BreakPointA, P2BreakPointA, P1BreakPointWonA, P2BreakPointWonA, P1BreakPointMissedA, P2BreakPointMissedA, P1DistanceRunA, P2DistanceRunA, RallyCountA, P1SetsWon, P2SetsWon, Game_timestamp, P1Rank, P2Rank, Total_time, Surface

11.B PointWinner models Hyperparameters

11.B.1 First serve PointWinner

subsamples	scale_position_weight	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
0.94	1.3	3	12	0.08	0.1	0.6

11.B.2 Second serve PointWinner

subsamples	scale_position_weight	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
0.72	1.4	5	4	0.01	0.4	0.3

11.B.3 Revised first serve PointWinner

subsamples	scale_position_weight	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
0.83	1.1	7	10	0.05	0.4	0.6

11.B.4 Revised second serve PointWinner

subsample	scale_position_weight	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
0.61	1	1	8	0.05	0.4	0.9

11.C Appendix: GameWinner Hyperparameters

Game_Timestamp	subsample	scale_pos_weight	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
1	0.83	1.1	5	5	0.01	0.4	0.9
2	0.5	1.2	7	6	0.01	0.2	0.6
3	0.83	1.2	5	5	0.05	0.2	0.3
4	0.5	1.4	9	4	0.05	0.2	0.3
5	0.61	1.1	5	12	0.01	0.4	0.9
6	0.83	1	9	4	0.05	0.2	0.6
7	0.61	1.1	7	6	0.01	0.3	0.6
8	0.94	1.2	7	6	0.01	0.0	0.9
9	0.61	1	7	4	0.01	0.0	0.9
10	0.94	1.2	7	6	0.01	0.0	0.9
11	0.61	1	7	4	0.01	0.0	0.9
12	0.5	1.3	3	12	0.01	0.2	0.6
13	0.94	1	9	5	0.01	0.4	0.9
14	0.72	1.4	3	12	0.05	0.4	0.9

11.D Appendix: SetWinner Hyperparameters

Set_Timestamp	subsample	scale_pos_weight	min_child_weight	max_depth	learning_rate	gamma	colsample_bytree
1	0.72	1.3	9	12	0.01	0.1	0.9
2	0.72	1.2	5	8	0.01	0.0	0.9
3	0.94	1.0	9	8	0.05	0.0	0.9
4	0.94	1.0	9	10	0.05	0.2	0.9
5	0.5	1.2	9	5	0.01	0.4	0.9
6	0.61	1.1	7	6	0.01	0.2	0.9
7	0.94	1.1	7	4	0.05	0.2	0.6
8	0.94	1.0	9	5	0.05	0.1	0.9
9	0.83	1.0	3	8	0.08	0.3	0.6
10	0.61	1.3	9	8	0.08	0.3	0.9
11	0.72	1.2	5	12	0.01	0.2	0.9
12	0.5	1.3	3	12	0.01	0.1	0.9
13	0.83	1.4	1	8	0.05	0.0	0.3