DataScientest • com

Sports Betting

Project Report 2 - Modeling

Cohort
- MAR23CONTDA

Project Team
- Karim Medlej
- Thomas Schlepphorst
- Eleanor Ageni-Yusuf

Tutor
- Lucas Varela

Introduction

Context

Tennis has been an ever growing sport since the early 19th century. In today's world and the global entertainment industry, it has a steady following. Tennis has been played at the olympic games since 1988. Furthermore, since 1968, prize money of Wimbledon, one of the biggest tennis tournaments in the world, has grown by 1.174%, with attendance of up to half a million people in recent years. This not only shows the interest the sport generates, but also the potential the sport has for money making as the reach tennis generates makes it an interesting sport for betting purposes. Public interest is high and tennis attracts lots of viewership from all over the world. Additionally, sports betting in general has an estimated market value of over 80 billion $. In Great Britain alone, betting on tennis matches has a 5% market share on the total british betting market.

With the size and impact the sport of tennis has and the potential of the global betting market, the main question of this project comes to mind: can we use past official match data of tennis matches and machine learning to create a prediction model that will tell us who is most likely to win in a match up between a set of given tennis players?
In this project, we will try to answer exactly that question.

As a disclaimer on the topic of betting, it has to be mentioned that betting involves risk of addiction and the loss of money. The possible solution presented at the end of this project is only a crude prediction which is based on a certain set of variables and should not be, under no circumstances, used for actual betting purposes.

From a technical point of view, this project involves the processing and analysis of real life data, the application of machine learning algorithms for prediction, user intuitive visualization and at best an interactive application for presentation purposes.

Objectives

The aim of this project is as following:

1. Analyze existing data of ATP tennis matches and display possible points of interest in the data.

2. Analyze the variables given in the data and single out key factors that can be used for predicting the winner of a match.

3. Develop a model that can accurately predict match outcomes given a certain set of input variables, among others two player names.
   This involves feature and model selection, training, validation and evaluation of predictive models that will be used to generate the expected result.

Initial Analysis and Preprocessing Report

Framework

The data used for this project consists of all ATP tennis matches between the year 2000 and March 2018. Initially, the dataset includes 44708 rows and 23 columns, where each row represents data about a single tennis match. The dataset can be freely accessed through kaggle.com, which is one of the biggest freely available databases in existence.

The intial 23 columns represent the variables we will use for prediction. In the dataset, following variables are present:

- **ATP:** Ranging from 1-69 and representing the order of ATP tournaments by year.
- **Location:** The location of the ATP tournament.
- **Tournament:** The name of the tournament.
- **Date:** Starting date of the tournament.
- **Series:** Type of tournament according to the ATP category.
- **Court:** Outdoor or Indoor.
- **Surface:** Hard, clay, grass or carpet. Describes the surface of the court.
- **Round:** Indicator of tournament round, 1st to final.
- **Best of:** Best of 3 or 5, indicator of how many sets a match has.
- **Winner:** Name of the winning player.
- **Loser:** Name of the losing player.
- **WRank:** ATP rank of the winning player before the match.
- **LRank:** ATP rank of the losing player before the match.
- **Wsets:** Sets won by the winning player.
- **Lsets:** Sets won by the losing player.
- **Comment:** Comment on the completion of the match.
- **PSW:** Betting quota for the winning player of betting provider PS.
- **PSL:** Betting quota for the losing player of betting provider PS.
- **B365W:** Betting quota for the winning player of betting provider B365.
- **B365L:** Betting quota for the losing player of betting provider B365.
- **elo_winner:** Elo score of the winning player before the match.
- **elo_loser:** Elo score of the losing player before the match.
- **proba_elo:** Probability of the winner winning the match, based on their elo score.

Relevance

Most Relevant Variables

Of the initial variables, we deemed series, surface, player and the elo ratings as the most relevant variables. This is, because in theory the result of the tennis match should be mostly dependent on the involved players. Therefore, the elo should play a big role, as it gives the ‘strength’ of the player. Series and surface are important, as players have surface preferences and are more likely to play more serious in more difficult tournaments, i.e. grand slams vs ATP250.
In addition, we deemed winner, loser, Wrank, Lrank, B365W and B365L as relevant.
All other data was dropped for further processing, as we did not think it would impact our machine learning, i.e. the year the tennis match was played shouldn't impact the outcome, the ATP rank (Wrank and Lrank) of the player however should.

Target Variable

The initial data did not present our target variable, as we want to predict the outcome of a match between two players. However, the data only gave the name of the winning and the name of the losing player. Therefore, we created our target variable manually, which will be further explained in the Pre-processing and feature engineering chapter.

Highlights in the Dataset

One feature that we can highlight is that our initial dataset is fairly complete. Only six out of 23 columns contain missing values. Four of these six columns are columns containing betting data. As we are dropping the columns PSW and PSL, as well as the columns of Wsets and Lsets, only two relevant columns contain missing values. The highest percent of missing values can be found in the column B365W with 12,7%, with B365L slightly lower at 12,6%.
Overall, the most important data, namely tournament, player names, winner and loser, elo and ATP rank were all fully present in all the data lines, which we hope will give us a rather reliable database for our machine learning as no distortion from filling methods should occur.

Limitations of Data

The only limitation we have, is that the data only consists of matches up to march 2018. This will most likely impact the validity of predictions, as we are missing match data of about 5 years and 6 months with valuable insights into player ratings and power dynamics in the ATP rankings. For example, Roger Feder still had an elo rating of over 2.000 in 2018, however by now he is retired and predictions including Federer will not take that into account.

Pre-processing and feature engineering

The initial dataset contained match entries per each dataline, however as we want to predict the winning player of a match, some pre-processing steps were necessary.
First, we created additional data columns that consisted of the difference in ATP-rank and in the elo rating from each player's perspective, called diff_rank and diff_elo respectively. Furthermore, we added our target variable, which will show 1 for the winning player and 0 for the losing player.
Next, we split our dataset into two parts, each consisting only of winning or losing player data to be able to transform our match based database into a player based database. In this step included, we changed column names to more data analysis friendly column names, i.e. decapitalizing letters and exchanging space bars with underscores.
To complete the transformation, we concatenated the two parts into one final dataset, as well as filling missing values in the B365 columns with 0, as we just assumed no bets were available for those matches and therefore no betting quote should be filled.

With that, we arrive at our final dataset consisting of nine columns and 89.416 data lines and no missing values.

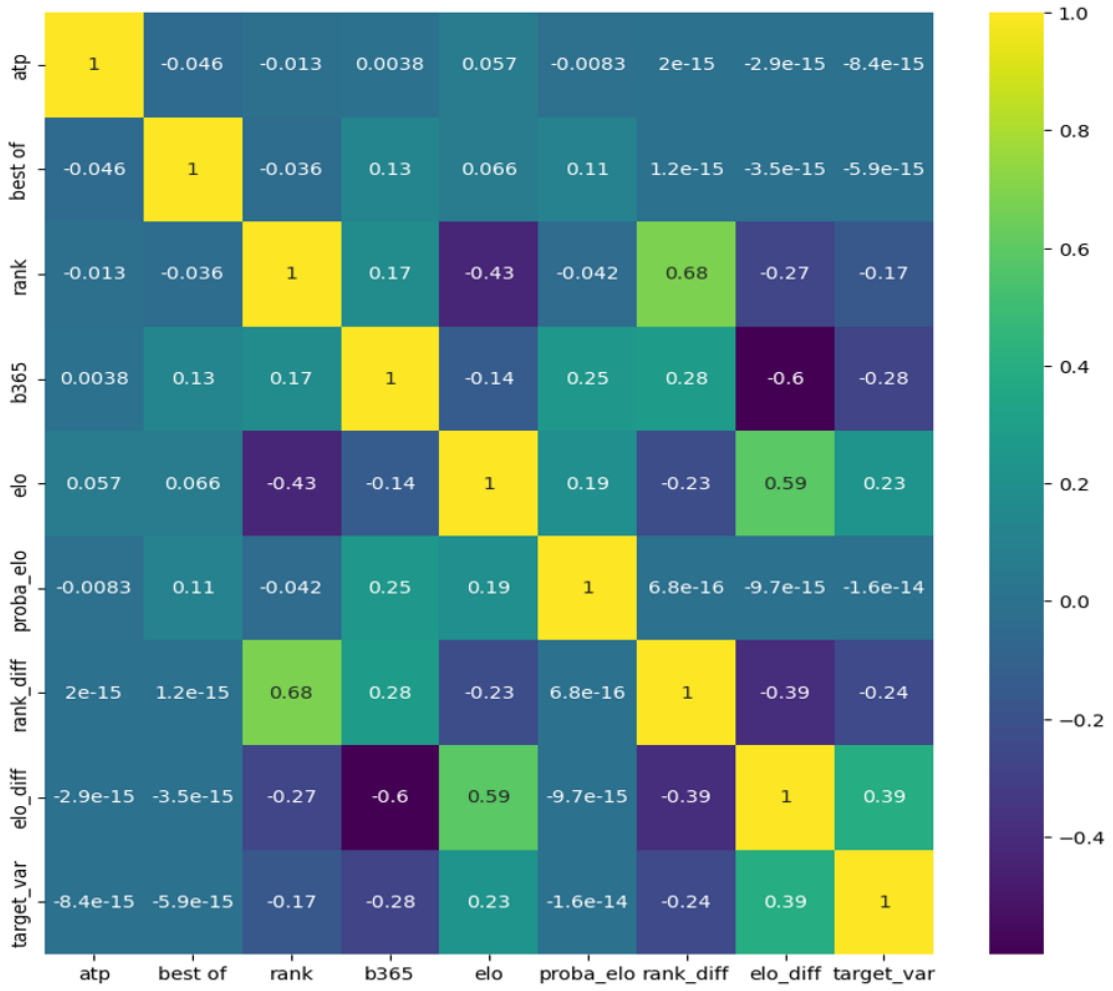
As betting quotas, elo rankings and ATP rankings are already standardized values no additional standardization or normalization transformations were necessary.

Visualization and Statistics

We used graphs to show the connections among the variables in our dataset. This helped us to explore their patterns and trends more easily.

Correlation Matrix

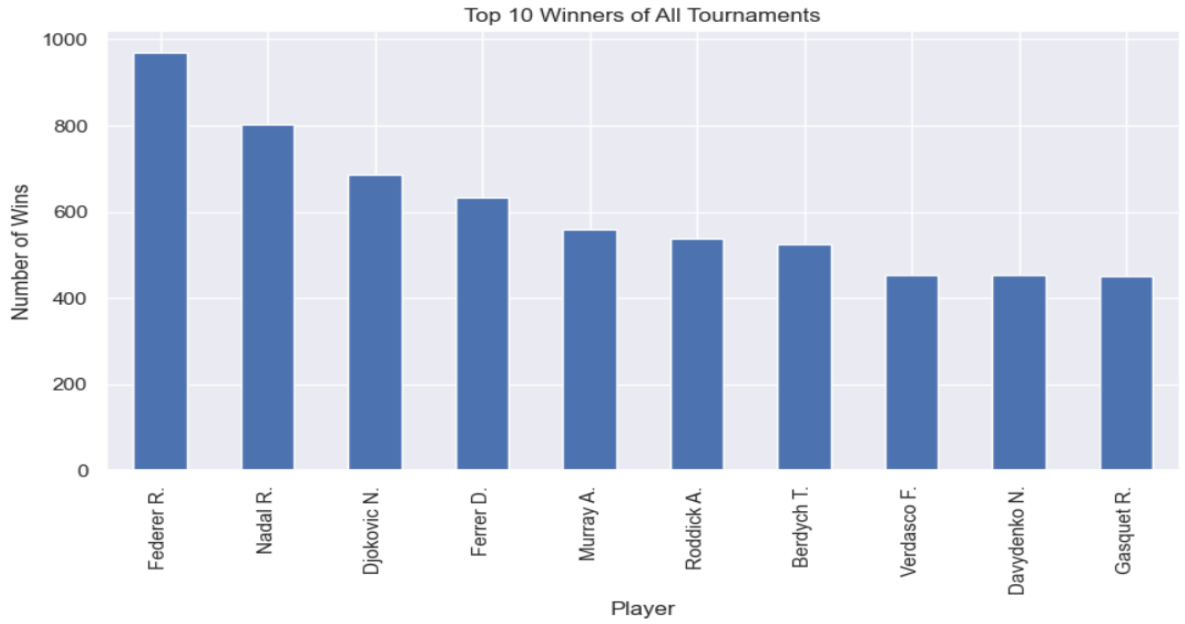
The correlation matrix is a visual and numerical summary of the relationships among the variables in our data set. It shows the direction and strength of the association between each pair of variables using colors and numbers. The numbers vary from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation. The colors also match the sign and magnitude of the correlation coefficients, with darker colors indicating stronger correlations and lighter colors indicating weaker correlations. The correlation matrix helps us to understand how the variables in our data set relate to each other and to spot possible patterns or outliers.



Grand Slam Visualization

The bar plot below shows the ranking of the top 10 players based on their number of wins. The x-axis lists the names of the players and the y-axis shows the number of tournaments they have won. The bars are proportional to the number of wins, so the higher the bar, the more tournaments the player has won.

The following plot aims to show the top players' performance in a graphical way. By doing this, we can analyze the different aspects that made them excel and the factors that contributed to their success in the ATP tour. This will help us to encode the right categorical variables for our data analysis.



We discovered that a key element in the success of the best ATP tennis players is their frequent participation in Grand Slams.

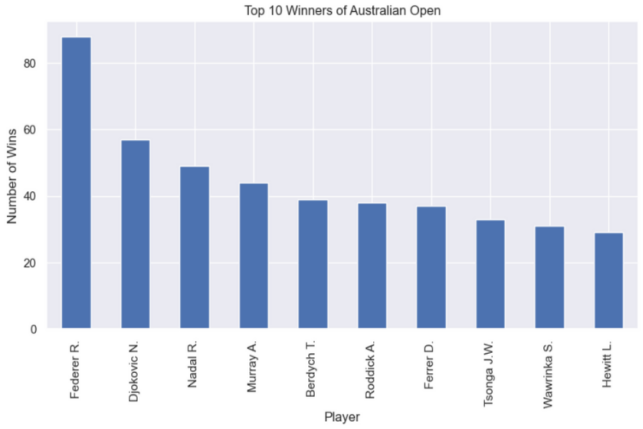
The Grand Slam tournaments, also known as majors, are the four most prestigious annual professional tennis tournaments in the world. They offer the most ranking points, prize money, public and media attention, the highest level and number of competitors.

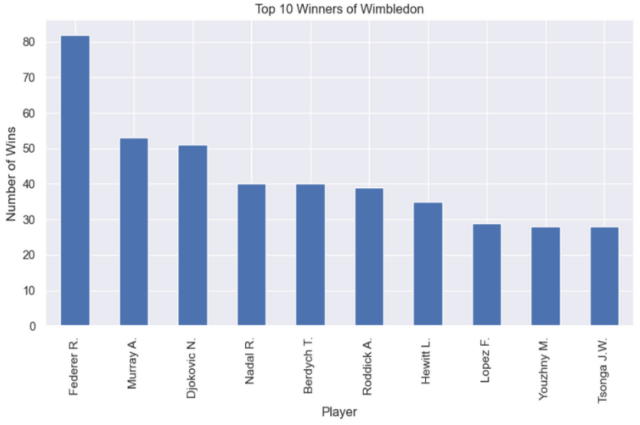
These Grand Slams are:

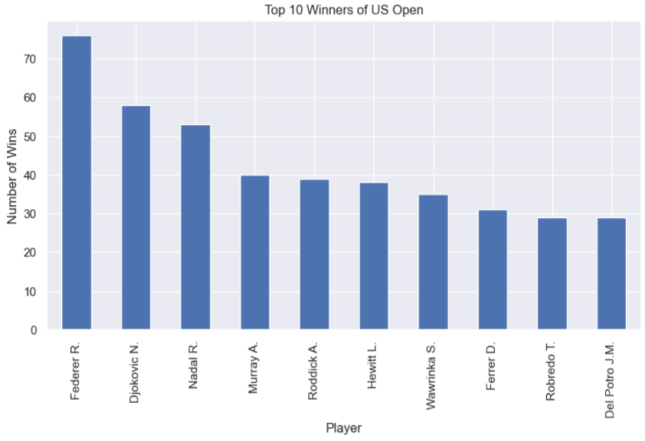
- Australian Open: hard surface.
- Roland Garros (French open): clay surface.
- Wimbledon: grass surface.
- US Open: hard surface.

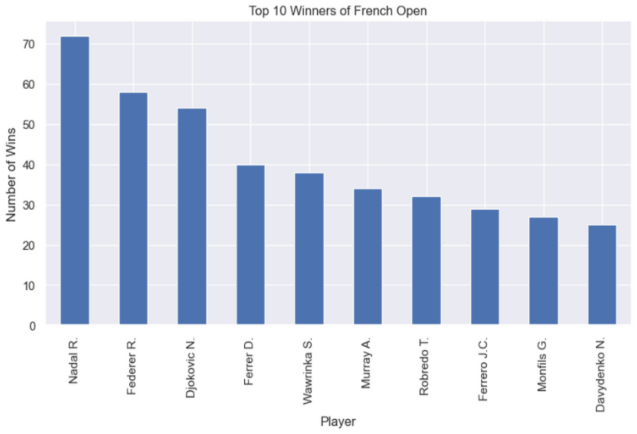
We can infer that winning a grand slam is a sign of high-level performance, since only the best players compete in these tournaments. Therefore, players with one or more grand slam titles have an advantage over those who have none, as they have proven their skills against the top opponents. This information is relevant for our prediction algorithm, as it can help us distinguish between different categories of players.

The bar plot below illustrates the previous analysis by showing the top 10 players for each grand slam along the x-axis and the number of wins along the y-axis. Each player is represented by a bar whose height indicates how many tournaments they have won.





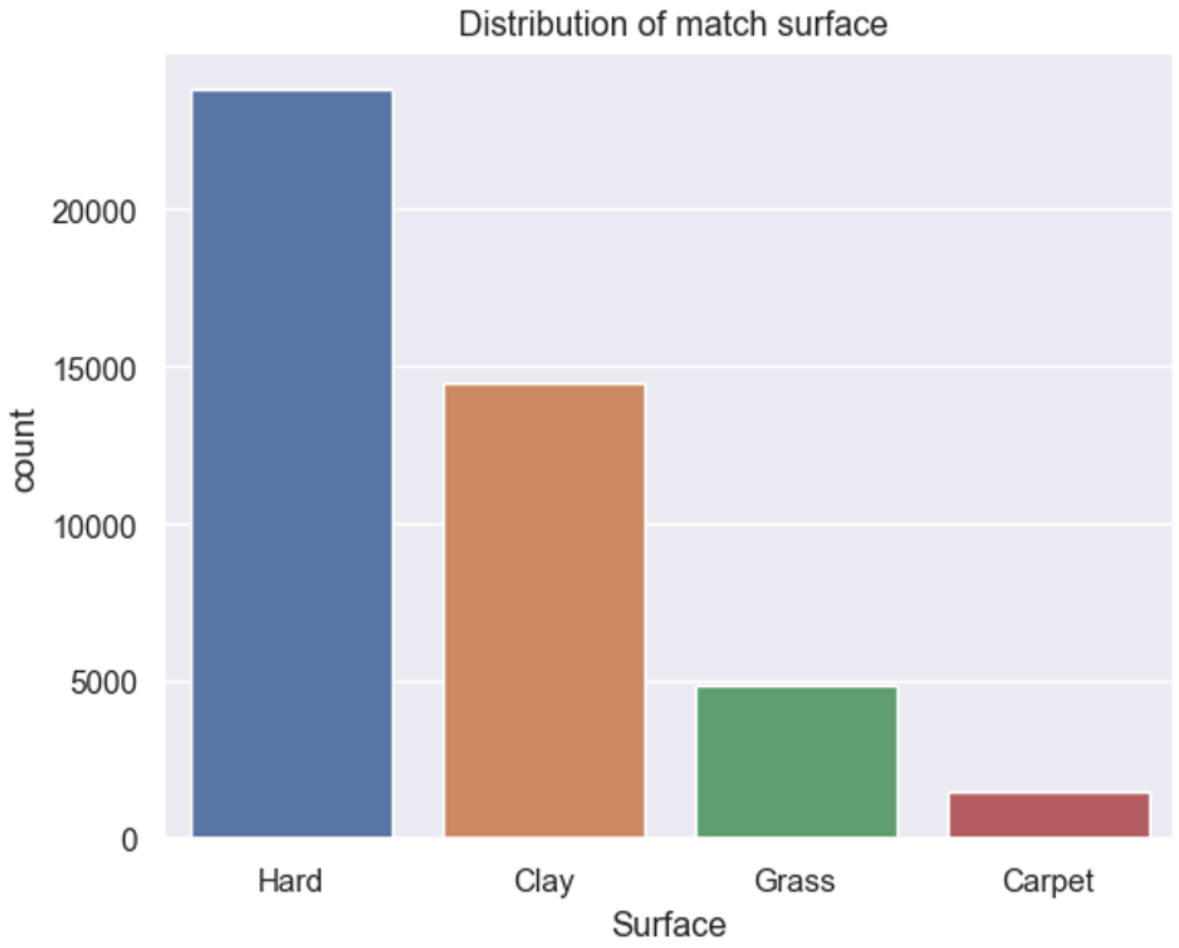




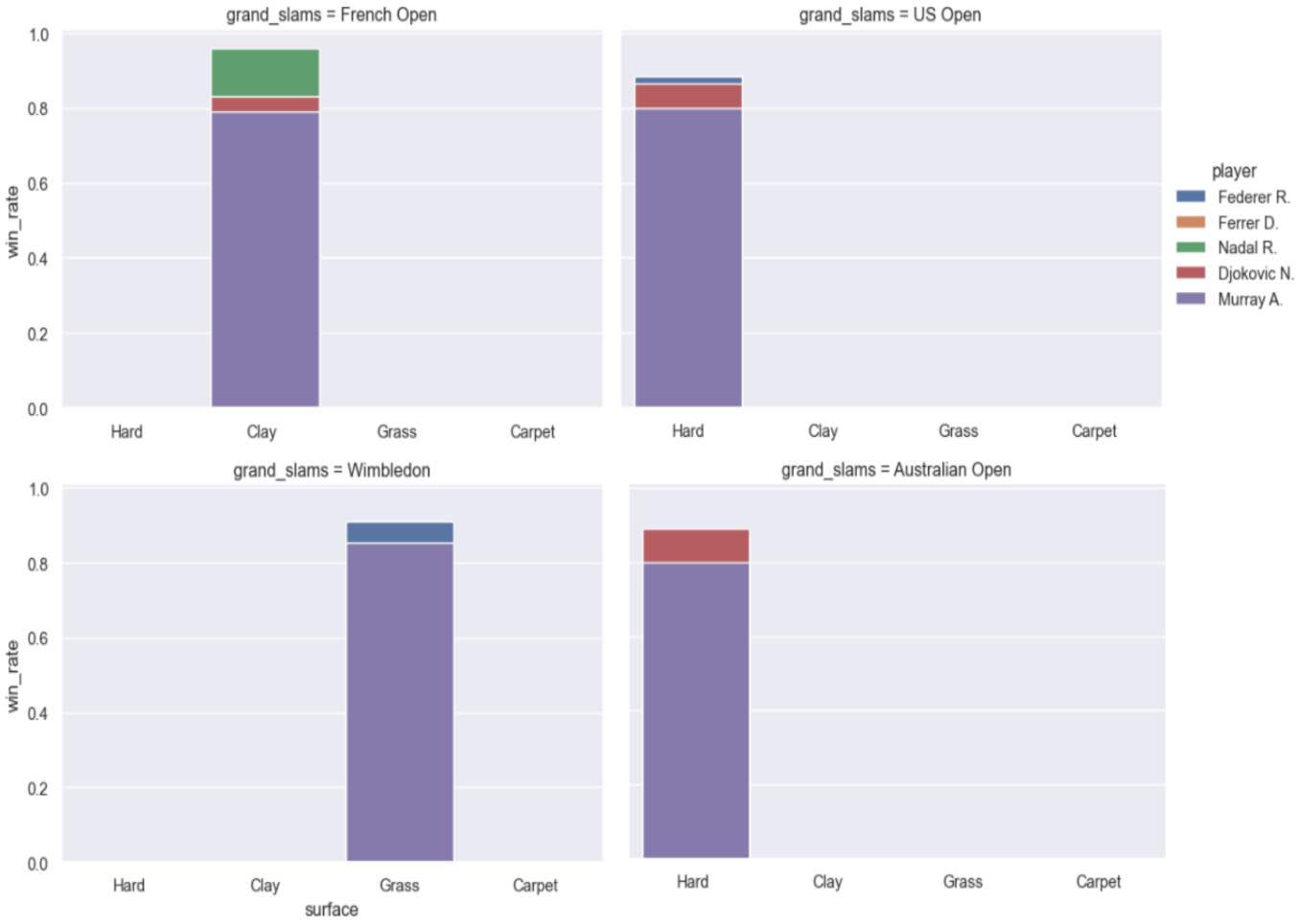
Distribution of Surface

We analyzed the 'surface' variable, which represents the court type where a player competes.

The bar chart compares the win/loss ratio of players on different court surfaces. The y-axis shows the number of counts. The chart shows that hard courts are the most favorable for most players, followed by clay, grass and carpet courts. This suggests that the 'surface' variable has some influence on the 'elo ranking', which is another variable that measures the skill level of a player over time before a match in a tournament. The value of this variable changes depending on the game outcomes that are assessed. This confirms our initial hypothesis about the significance of these variables in our dataset.



In the following graph, we first filtered the top 5 players in each Grand Slam event based on the number of matches they played. Then we calculated the win rate for each player on each surface at each Grand Slam event. The win rate is calculated as the number of matches won divided by the total number of matches played. Finally, we created a bar plot for each Grand Slam event showing the win rate for each player on each surface. The players are differentiated by different colors in the plot. The x-axis represents the court surfaces and the y-axis represents the win rate. Each facet represents a Grand Slam event. The legend shows the color corresponding to each player.



After visualizing the graph, we conclude that court surface is a significant factor that affects the performance of tennis players. For example, Nadal has a dominant record on clay courts, while Djokovic and Federer compete for the top spot on hard courts, especially in the US open. Djokovic also has a clear advantage on the Australian open hard courts, while Federer excels on Wimbeldon's grass courts.

This shows that the court surface can complement or challenge the players' skills and strategies. Therefore, the court surface can have a decisive impact on the outcome of a match.

Visualization Conclusion

Our prediction algorithm relies on two key variables: the type of tournament (Grand Slams) and the court surface. These variables affect the performance of the players and the outcome of the matches. We used visualizations to explore and confirm this relationship. Therefore, we encoded these two categorical variables in our train_test_split code to make our prediction model work.

Predictive Modeling and Forecasting

Classification

The goal of this machine learning project is to perform a regression analysis on a dataset of ATP tennis matches from 2000 to March 2018. The dataset contains various features such as player names, rankings, scores, and statistics and lacks the outcome of the match as a feature, which is what we want to predict. To create this feature, we divide the dataset into two subsets, one with the information of the winner of each match, and the other with the information of the loser. Then, we assign a binary label to each subset, indicating whether the player won or lost the match.

When evaluating our regression models we use quantitative performance metrics such as accuracy, precision, recall, and F1-score:

Accuracy: The proportion of correct predictions among all predictions. It is defined as (TP + TN) / (TP + TN + FP + FN), where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

Precision: The proportion of true positives among all positive predictions. It is defined as TP / (TP + FP).

Recall: The proportion of true positives among all actual positives. It is defined as TP / (TP + FN).

F1-score: The harmonic mean of precision and recall. It is defined as 2 * (precision * recall) / (precision + recall).

These metrics provide different insights into the performance of the model. Accuracy measures overall correctness, precision measures the ability to avoid false positives, recall measures the ability to find all positives, and F1-score provides a balance between precision and recall.

In the case of the three models we focused on, the XGBoost model achieved the highest accuracy and F1-score, indicating that it performs better overall.

Model Development

We started our modeling process by applying three models:

Logistic regression
Logistic regression is a type of statistical analysis that models the relationship between one or more explanatory variables and a binary or categorical outcome variable. The outcome variable is coded as 0 or 1 for binary logistic regression, or as a set of discrete values for multinomial logistic regression. The explanatory variables can be either binary or continuous. Logistic regression uses the logit function to transform the probability of the outcome variable into a linear function of the explanatory variables. The logit function is the inverse of the logistic function, which maps a real number to a probability between 0 and 1.

Logistic regression is not a classification method by itself, but it can be used to create a classifier by setting a threshold value and assigning inputs to different classes based on their predicted probabilities. Logistic regression is widely used in various fields of statistics, such as epidemiology, social sciences, machine learning, and bioinformatics. It is especially useful for modeling binary outcomes that depend on multiple factors, such as whether a patient has a disease, whether a customer buys a product, or whether an email is spam or not. In our case, if a player will win or lose.

Decision Tree Classifier
A decision tree classifier is a supervised machine learning algorithm that allows you to classify data with high degrees of accuracy. It is a non-parametric method used for classification and regression.

Random Forest Classifier
A random forest classifier is a supervised machine learning algorithm that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve predictive accuracy and control overfitting. It is a meta estimator that can be used for both classification and regression tasks.

We then used Machine Learning and statistical techniques to further analyze and improve our model performance. We experimented with some more advanced models such as:

KNeighborsClassifier
One way to perform both classification and regression tasks with supervised machine learning is to use the KNeighborsClassifier algorithm. This algorithm

belongs to the category of instance-based learning or lazy learning, where the function is only approximated locally and all computation is deferred until classification.

The algorithm works by finding the k-nearest neighbors of a given data point in the feature space, where k is a user-defined parameter. The class of the data point is then determined by a majority vote of its k-nearest neighbors. The distance metric used to determine the nearest neighbors can be customized, with the most common distance metric being Euclidean distance.

SVM
The Support Vector Machines (SVM) algorithm is a supervised machine learning algorithm that is often used for classification problems, though it can also be applied to regression problems. It is commonly used for supervised machine learning models and offers high degrees of accuracy in classification problems. SVMs separate data into different classes by using a hyperplane, which is supported by the use of support vectors. These vectors are used to ensure that the margin of the hyperplane is as large as possible.

XGBClassifier
One way to apply the XGBoost algorithm for classification problems is to use the XGBClassifier class from the scikit-learn library. XGBoost stands for eXtreme Gradient Boosting, and it is a fast and scalable implementation of gradient boosting machines. The XGBClassifier class allows you to fit, predict, and evaluate the model using the same methods as other scikit-learn estimators. To use the XGBClassifier, the `xgboost` module in scikit-learn must be imported.

Model Training

The initial results of the three models we used for classification are as follows:

Model	Accuracy score
Logistic Regression	0.6712704093044062
Decision Tree	0.6144039364795347
Random Forest	0.6735070454037129

From these results, we can see that the Logistic Regression and Random Forest Models have similar and higher accuracy scores than the Decision Tree Model. This suggests that these two models are better at capturing the patterns and relationships

in the data than the Decision Tree Model, which may be prone to overfitting or underfitting.

We applied KNeighborsClassifier, SVM and XGBClassifier as our more complex models and the results are as follows:

Model	Accuracy score
K Neighbors Classifier	0.6251397897562067
Support Vector Machine	0.6699284276448222
XGB Classifier	0.6930776112726459

As we can see, the XGBClassifier outperformed the other two models, achieving an accuracy score of 0.693. This means that it correctly classified about 69.3% of the instances in the test set. Therefore, we concluded that the XGBClassifier model is the most suitable model for our problem.

We applied Grid Search to optimize the hyperparameters of Logistic Regression, Random Forest Classifier and XGBoost Classifier. This technique helped us to enhance the performance of our model, as we observed a slight increase in the accuracy. The details and results of the Grid Search procedure are as follows:

Random Forest

Best parameters - The hyperparameters that yielded the best performance for the random forest classifier are:

max_depth	10
min_samples_leaf	2
min_samples_split	5
n_estimators	50

Random forest score - The score achieved by the random forest classifier is approximately 0.6684.

Classification report: The classification report provides a summary of the classifier’s performance on each class. Here are the key metrics for each class:

	Precision	Recall	F1-score	Support
Class 0	0.71	0.57	0.63	8999
Class 1	0.64	0.77	0.70	8885

accuracy			0.67	17884
macro avg	0.68	0.67	0.67	17884
weighted avg	0.68	0.67	0.67	17884

Confusion matrix - The confusion matrix shows the number of samples that were correctly or incorrectly classified by the random forest classifier. Here's a breakdown of the predictions:

Predicted by RF	0	1
True Class 0	5145	3854
True Class 1	2076	6809

Logistic Regression

Best parameters - The hyperparameters that yielded the best performance for the logistic regression classifier are:

C	0.01

Logistic regression score - The score achieved by the logistic regression classifier is approximately 0.6687.

Classification report - The classification report provides a summary of the classifier's performance on each class. Here are the key metrics for each class:

	Precision	Recall	F1-score	Support
Class 0	0.70	0.59	0.64	8999
Class 1	0.64	0.75	0.69	8885

accuracy			0.67	17884
macro avg	0.68	0.67	0.67	17884
weighted avg	0.68	0.67	0.67	17884

Confusion matrix - The confusion matrix shows the number of samples that were correctly or incorrectly classified by the logistic regression classifier. Here's a breakdown of the predictions:

Predicted by LR	0	1
True Class 0	5312	3687
True Class 1	2238	6647

XGB Classifier

Best parameters - The hyperparameters that yielded the best performance for the XGB classifier are:

learning_rate	0.01
max_depth	7
subsample	0.5

XGB Classifier score - The score achieved by the XGB classifier is approximately 0.6937.

Classification report - The classification report provides a summary of the classifier's performance on each class. Here are the key metrics for each class:

	Precision	Recall	F1-score	Support
Class 0	0.70	0.68	0.69	8999
Class 1	0.69	0.70	0.69	8885

accuracy			0.69	17884
macro avg	0.69	0.69	0.69	17884
weighted avg	0.69	0.69	0.69	17884

Confusion matrix - The confusion matrix shows the number of samples that were correctly or incorrectly classified by the XGBoost classifier. Here's a breakdown of the predictions:

Predicted by XGB	0	1
True Class 0	6155	2844
True Class 1	2646	6239

Comparison of Models

After comparing the grid search optimization results for the Random Forest, Logistic Regression, and XGBoost. The best hyperparameters for each model are shown along with their corresponding scores.

- **XGBoost**: The model achieved the highest accuracy of 0.69 on the test set, with a precision, recall, and f1-score of 0.69 for both classes (0 and 1). This indicates that the model is balanced and effective in predicting both classes. The model also has the lowest number of misclassifications in the confusion matrix, with 3053 false positives and 2649 false negatives.

- **Logistic Regression**: The model achieved a similar accuracy of 0.67 as the Random Forest, but with a slightly higher precision of 0.70 for class 0 and a slightly lower precision of 0.64 for class 1. The model also has a higher recall of 0.75 for class 1 and a lower recall of 0.59 for class 0. This suggests that the model is more sensitive to class 1 and less specific to class 0. The model has 3687 false positives and 2238 false negatives in the confusion matrix.

- **Random Forest**: The model achieved the lowest accuracy of 0.67 on the test set, with a lower precision of 0.71 for class 0 and a higher precision of 0.64 for class 1. The model also has a lower recall of 0.57 for class 0 and a higher

recall of 0.77 for class 1. This indicates that the model is more likely to predict class 1 and less likely to predict class 0. The model has 3854 false positives and 2076 false negatives in the confusion matrix.

In summary, the XGBoost model outperforms the other two models in terms of accuracy, precision, recall, and f1-score. The Logistic Regression and Random Forest models have similar accuracy, but different trade-offs between precision and recall. The Logistic Regression model is more sensitive to class 1, while the Random Forest model is more specific to class 0.

To evaluate the performance of our models, we need to look at different metrics such as precision and recall. These metrics measure how well our models can correctly identify the relevant cases and avoid the irrelevant ones. However, different problems may have different preferences for these metrics. For example, in our problem, we may want to focus more on recall than precision, because we want to capture as many true positives as possible, even if that means having some false positives. In this regard, the Random Forest model has a higher recall than the XGBoost Classifier, but also a higher false positive rate. The XGBoost Classifier, on the other hand, has a more balanced performance and a lower risk of misclassifying cases. Therefore, we need to weigh the pros and cons of each model and choose the one that best suits our problem.

Conclusion

In this project, we aimed to explore and visualize the data of ATP tennis matches from 2000 to 2018. We wanted to find out what kind of insights and patterns can be derived from the data, such as the most successful players, the most common match outcomes, the influence of different surfaces, etc. We also wanted to identify the main features that affect the probability of winning a match, such as serve statistics, ranking, head-to-head record, etc. Based on these features, we built a machine learning model that can predict the winner of a match given two player names and some other input variables. We evaluated our model using various metrics and compared it with some baseline models.

The data was preprocessed and feature engineered to build a predictive model. The variables were explored using visualizations and statistics, such as correlation matrices, to understand how they relate to each other. The visualization analysis showed that the tournament type and court surface are important predictors of match outcomes. These categorical variables were encoded in the train_test_split code to make the prediction model work well.

We compared three different machine learning algorithms for our classification problem: Random Forest, Logistic Regression, and XGB Classifier. The results show

that the XGBoost algorithm has the best performance across all the evaluation metrics, such as accuracy, precision, recall, and f1-score. The Logistic Regression and Random Forest algorithms have comparable accuracy scores, but they differ in how they handle the two classes. The Logistic Regression algorithm has a higher recall for class 1, meaning it can identify more positive cases correctly. The Random Forest algorithm has a higher precision for class 0, meaning it can avoid more false positives.

This is how we can apply risk analysis to our betting strategy. Depending on our preference, we might want to prioritize recall over precision, meaning that we are willing to accept some false positives in order to catch more true positives. For this purpose, the Random Forest model has a higher recall than the XGBoost Classifier, but also a higher false positive rate. The XGBoost Classifier, however, has a more balanced performance and a lower chance of making wrong predictions.

This project concludes with a tentative solution that relies on specific variables and does not account for all possible outcomes. Therefore, it is strongly advised not to use this solution for any real betting scenarios.

To summarize, we have gained useful knowledge from the analysis of ATP tennis matches data about the factors that affect match outcomes. The predictive model that we built, based on the type of tournament and court surface, can reliably predict match outcomes given a certain set of input variables. This model can be applied to generate expected results and assist in decision-making processes related to ATP tennis matches.

Reference List

https://en.wikipedia.org/wiki/Tennis

https://de.statista.com/statistik/daten/studie/308099/umfrage/wimbledon-preis-herren-einzel/

https://www.statista.com/statistics/957232/wimbledon-championships-aggregate-attendance/

https://www.grandviewresearch.com/industry-analysis/sports-betting-market-report/

https://industrywired.com/how-big-is-the-uk-online-gambling-industry/

https://www.kaggle.com/datasets/edouardthomas/atp-matches-dataset/

Grand Slams:
Grand Slam (tennis) - Wikipedia

Logistic regression - Wikipedia. https://en.wikipedia.org/wiki/Logistic_regression.

What is Logistic regression? | IBM. https://www.ibm.com/es-es/topics/logistic-regression.

sklearn.linear_model.LogisticRegression - scikit-learn.
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

sklearn.tree.DecisionTreeClassifier — scikit-learn 1.3.1 documentation.
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

Decision Tree Classifier with Sklearn in Python • datagy. https://datagy.io/sklearn-decision-tree-classifier/.

1.10. Decision Trees — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/tree.html.

How to create a decision tree classification model using scikit-learn.
https://practicaldatascience.co.uk/machine-learning/how-to-create-a-decision-tree-model-using-scikit-learn.

Decision Tree Classifier in Python Sklearn with Example.
https://machinelearningknowledge.ai/decision-tree-classifier-in-python-sklearn-with-example/.

Decision Tree Classifier with Scikit-Learn from Python.
https://medium.com/@chyun55555/decision-tree-classifier-with-scikit-learn-from-python-e83f38079fe.

sklearn.tree.DecisionTreeClassifier — scikit-learn 1.3.1 documentation.
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

1.10. Decision Trees — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/tree.html.

sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.1 documentation.
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

How to create a random forest classification model using scikit-learn.
https://practicaldatascience.co.uk/machine-learning/how-to-create-a-random-forest-model-using-scikit-learn.

Random Forest Classifier in Python Sklearn with Example.
https://machinelearningknowledge.ai/python-sklearn-random-forest-classifier-tutorial-with-example/.

Multiclass Classification using Random Forest on Scikit ... - Codementor.
https://www.codementor.io/@agarrahul01/multiclass-classification-using-random-forest-on-scikit-learn-library-hkk4lwawu.

sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

What is Random Forest? | IBM. https://www.ibm.com/topics/random-forest.

Random Forest - Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Random_forest.

Specific Cross Validation with Random Forest - Stack Overflow. https://stackoverflow.com/questions/38151615/specific-cross-validation-with-random-forest.

sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

K-Nearest Neighbors (KNN) Classification with scikit-learn. https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn.

Algoritmo k-Nearest Neighbor | Aprende Machine Learning. https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/.

How does sklearn KNeighborsClassifier score method work?. https://stackoverflow.com/questions/51154607/how-does-sklearn-kneighborsclassifier-score-method-work.

Scikit Learn - KNeighborsClassifier - Online Tutorials Library. https://www.tutorialspoint.com/scikit_learn/scikit_learn_kneighbors_classifier.htm.

sklearn.neighbors.KNeighborsClassifier — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

Support Vector Machines (SVM) in Python with Sklearn • datagy. https://datagy.io/python-support-vector-machines/.

Support Vector Machines (SVM) in Python with Sklearn • datagy. https://datagy.io/python-support-vector-machines/.

How to Implement Support Vector Machines in Python (2023) - Dataquest. https://www.dataquest.io/blog/support-vector-machines-in-python/.

Scikit-learn SVM Tutorial with Python (Support Vector Machines). https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python.

1.4. Support Vector Machines — scikit-learn 1.3.1 documentation. https://scikit-learn.org/stable/modules/svm.html.

sklearn.ensemble - scikit-learn 1.2.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

How to Develop Your First XGBoost Model in Python. https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/.

Get Started with XGBoost — xgboost 2.0.0 documentation - Read the Docs. https://xgboost.readthedocs.io/en/stable/get_started.html.

sklearn.ensemble - scikit-learn 1.2.2 documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html.

Get Started with XGBoost — xgboost 2.0.0 documentation - Read the Docs. https://xgboost.readthedocs.io/en/stable/get_started.html.