Machine Learning Engineer Nanodegree

Capstone Project

David Broadwater
February 2019

I. Definition

Project Overview

Tennis is the fourth most popular sport in the world [1], and it has also been growing in popularity in betting markets. For instance, in the UK, tennis was recently cited as second only to soccer (football) in terms of betting markets [2]. However, given its unique nature in comparison to other sports, there are a number of factors which make predicting the outcome of tennis matches difficult. First, tennis is a sport with a complex scoring system involving a series of games (made up of a minimum of 4 points each, and won by a margin of at least two points), which comprise sets (generally won by the first player to reach 6 games). Most matches are played as best of 3 sets, with the exception of certain higher level tournaments on the men's professional tour, known as Grand Slam tournaments, which are played as best of 5 sets. Based on this scoring system, it is possible for a player to win the match while actually winning fewer total points overall (somewhat similar to the way the Electoral College voting system works in the United States).

Another unique characteristic of tennis is that it is played on various court surfaces (clay, grass, hard court, and carpet), each of which can have wildly different playing conditions (i.e., height of bounce, speed of ball, etc.), in addition to variability in weather from tournament to tournament. For instance, clay courts are generally considered "slow" because the friction of the clay court slows tennis balls and causes them to bounce higher (making longer points more common), while grass courts are generally considered "fast" because the low friction of the grass causes balls to skid and bounce very low (making shorter points more common). Hot and dry weather can increase the "speed" (i.e., fast or slow) of a court, while cold or humid conditions can decrease the speed of a court. Altitude can also impact playing conditions, with higher altitude generally playing "faster."

There have been multiple previous efforts to predict the outcome of tennis matches for betting purposes, each with slightly different approaches and datasets used. Sipko and Knottenbelt [3] used historical men's professional singles match data to train Logistic Regression and Artificial Neural Network (ANN) models to predict the outcome of tennis matches, and used the predictions as the basis of simulated betting strategies to calculate simulated return on investment (ROI) based on each algorithm. Cornman et al. [4] followed a similar overall approach and combined historical men's professional singles match data with pre-match betting odds from various betting services in order to build a model to use as the basis for a simulated betting strategy; they tested Random Forest, Support Vector Machine, and ANN algorithms. In both cases, the historical men's professional singles match data is nearly identical as proposed in this effort, and each had the same general aim to predict the outcome of men's professional singles matches using pre-match information.

Personally, I am a huge tennis fan (and avid tennis player), so working with tennis data has long been a desire of mine since I discovered the field of data science. Until researching this project, I was unaware how much tennis data was available online or what efforts had previously been made to predict the outcomes of tennis matches. While looking at the previous research, I noticed most of them had been done many years ago, before the recent sudden burst of advancements and popularity in the field of data science, which has led to the development of much more powerful and efficient algorithms for performing machine learning tasks.

The dataset used for this effort is a set of professional tennis match results, as well as supporting tournament and country information. This dataset was obtained from Kaggle [5] and has common tennis statistics routinely captured during tour-level matches (e.g., aces, first serve points won, return points won, etc.). The scope of the data spans from 1968–2017 for top-level ATP (Association of Tennis Professionals) tournaments, although some data is missing prior to 1983. The playing statistics from these matches could be used to see if certain historical information can successfully predict the outcome of a match. These statistics could be used to generate information about each player to determine differences in skill levels, risk of fatigue, matchup problems/advantages, and if a player is having a particularly good streak of matches. Singles tennis is a totally individual sport, and as such matchups of playing styles (and court speed) can have a big impact on the outcome of a match. A certain player may play really well on clay because of their speed and defensive skills but may have poor results on a fast grass court against someone with an elite serve or a more offensive style. As a result, head-to-head records and court surfaces can be important in predicting the outcome of a match.

From an individual perspective, playing statistics can give insight into how a player compares to his peers; a high percentage of serve points won combined with a high average ace count could indicate a player has a very strong serve and would be very tough to "break" (a term used to describe when a player loses a game in which they are serving). Since servers are typically at an advantage (due to the speed of a serve shot in comparison to other tennis shots), getting "broken" in a game generally puts a player at a disadvantage in a set, since the opposing player would only need to "hold" (i.e., win) the rest of their service games in order to win the set, assuming there were no other previous breaks of serve. Similarly, a high percentage of return points won or break points won could indicate the strength of a player's service return or defensive skills. From what I've observed as a tennis fan, the combination of these skills and the matchups against the other player's skills can affect the style and

outcome of the match. A matchup of two very strong servers who only have average service return or defensive skills would likely result in a match with very few breaks of serve and a higher change of going to a tie-breaker (an extended game that occurs when a set score is 6–6 where the first player to reach 7 points by a margin of at least 2 points wins), where the entire match could come down to a few points.

Additionally, the tour rankings of players could be used to help predict the outcome of a match. Tour rankings are based on a points system, where a player gets a certain number of points based on how far a player proceeds in a tournament (i.e., first round, quarterfinals, winner, etc.) and the level of tournament (higher level tournaments where the playing field is larger and of a higher quality are worth more points). A running sum of the previous 12 months' worth of points is used to come up with a point total, which is used to rank the players; these rankings are updated weekly. So, a player ranked number two in the world would likely be heavily favored to win against someone ranked number forty, potentially even if their head-to-head record was even or unfavorable, since their ranking would indicate they had been playing at a very high level for a sustained period.

Additionally, factors such as age or the amount time spent on court in previous matches (or the combination of the two) could also have an impact. Age can play a factor in a player's ability to recover between matches, so a player who wins an extremely long match may be fatigued in his next matches and struggle more than he otherwise would in the beginning of a tournament when he is presumably more well rested. With the exception of the four Grand Slam tournaments, most players play matches on consecutive days (as long as they keep winning), so a string of consecutive 3 set matches (or 5 set matches in Grand Slams) can potentially take their toll on a player, since time to recover is limited within a tournament. However, fatigue can be an issue for players at any age; in fact, some of the youngest players sometimes have trouble with fatigue (especially in Grand Slam tournaments where men's singles matches are best of 5 sets instead of 3 or in extremely hot conditions), since they aren't used to the conditioning demands of the professional tour.

Problem Statement

The goal of this effort is to predict the winner of men's professional singles tennis matches, given historical playing statistics and information about the two tennis players (such as percentage of first serve points won, court surface, ranking, etc.). These predictions could be used as the basis of a betting strategy, or simply to better understand what factors potentially influence the outcome of tennis matches (for spectators or players themselves). Since there are no ties in tennis, there is a clear winner for every match, and the sample size for a given player throughout the course of a season (which lasts from January through November) can be quite large, spanning multiple court surfaces and potentially including matches against the same players. While previous efforts at predicting the outcomes of tennis matches used other classification algorithms such as Random Forests [4] and Logistic Regression [3], gradient boosted tree algorithms such as XGBoost (introduced by Chen and Guestrin [] [ChenGuestrin:2016]) have become very popular among machine learning competitions over the last few years [6]. These implementations represent significant speed improvements over previous gradient boosted tree algorithms which may have prevented their consideration, with each of the efforts previously mentioned listing computing resources as a constraint.

Metrics

The primary evaluation metric for this effort is logarithmic loss (log loss). The log loss metric is a common classification metric (especially in Kaggle competitions) [7] and directly evaluates the predicted probabilities of the model against the results. By using the predicted probabilities, log loss can give a more detailed view of how well the model is performing, since predicted Player 1 win probabilities of 0.9 and 0.6 would both yield the same overall predicted outcome, but 0.9 would be more correct in the event of a Player 1 win (the accuracy metric would treat both predictions the same). In our case, the formula for log loss is given by

$$Log\ Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log\ p_i + (1 - y_i) \log (1 - p_i)],$$

where N is the total number of data points or samples, y_i is a binary indicator for if Player 1 won the match, and p_i is the model predicted probability of Player 1 winning the match. If the model performs perfectly (predicting 100% probability with each Player 1 win and 0% probability with each Player 1 loss), it would have a log loss score of 0, though this is definitely not expected. Based on the formula for log loss (and in contrast to accuracy), predictions are penalized more if they are more incorrect, so for an actual Player 1 loss a 40% win probability would have a worse log loss score than a 10% win probability, even though from an accuracy perspective both would be treated as correct. If used for a betting strategy, it could also be especially helpful to understand how confident the prediction was in relation to the eventual outcome based on the predicted probability output by the model. For instance, a high confidence (i.e., probability) prediction with favorable odds may be a more attractive bet than a low confidence bet with unfavorable odds, depending on the risk strategy.

II. Analysis

Data Exploration

The initial dataset was provided as a CSV files and had 49 features and 163,966 matches, spanning men's professional singles matches from 1968 to 2017. A sample of this data is provided in Table 1. Upon importing this dataset, it was discovered there were some data issues which needed to be corrected. Namely, it appears a row of header column names erroneously appeared a second time in the middle of the data set, causing Pandas (a Python data analysis package) to assume all of the columns were `object` (e.g., string) types. After removing this row from the data, the columns were manually converted to either `object` (for categorical fields) or `float64` (for numerical fields) types. `float64` column types were chosen for all numerical fields for better compatibility with missing values in Pandas. This was required before any exploratory analysis could be performed.

Table 1. Data Sample

Field Name	Description	Median / Most Prevalent

best_of	Max number of sets played	3
draw_size	Size of tournament draw	32
w_1stIn | l_1stIn	Number of 1st serves in	44 | 44
w_1stWon | l_1stWon	Number of 1st serve points won	33 | 29
w_2ndWon | l_2ndWon	Number of 2nd serve points won	16 | 14
w_SvGms | l_SvGms	Number of serve games played	11 | 11
w_ace | l_ace	Number of aces	5 | 3
w_bpFaced | l_bpFaced	Number of break points faced	4 | 8
w_bpSaved | l_bpSaved	Number of break points saved	3 | 4
w_df | l_df	Number of double faults	2 | 3
w_svpt | l_svpt	Number of serve points played	72 | 75
winner_age | loser_age	Age of player in years	25.5 | 25.6
winner_entry | loser_entry	Method of entry into draw	Q | Q
winner_hand | loser_hand	Playing hand of player	R | R
winner_ht | loser_ht	Height of player in cm	185 | 183
winner_id | loser_id	Player identifier in dataset	102148
winner_ioc | loser_ioc	Country player is from	USA
winner_rank | loser_rank	Rank of player at time of match	47 | 77
winner_rank_points | loser_rank_points	Rank points of player at time of match	413 | 314
winner_seed | loser_seed	Tournament seed of player	5 | 6
match_num	Number of match played at tournament	1
minutes	Length of match in minutes	95
round	Round of tournament match	R32
score	Final score of match	6–4 6–4
surface	Court surface	Hard
tourney_date	Date of start of tournament	2017–02–03
tourney_id	ID of Tournament	1989–520
tourney_level	Level of ATP tournament	A
tourney_name	Tournament Name	Wimbledon

Next, a data profile was generated to provide an overview of each field, including the number of missing values for each, min/max values, and most common values, among other things. Looking at the overall data profile statistics, many of the match statistics fields are missing values (especially for between 1968–1991); most of these fields are missing exactly 52.8% of their values overall, suggesting there are many matches which don't have any statistics at all. Additionally, matches in 2015 had large amounts of missing data.

There were some abnormalities observed in the dataset as well. The `loser_age` field has values ranging from 14 to 63 years old, and the `winner_age` field ranges from 14 to 58 years old. While some players start in their teens, 14 seems a bit too young, and at the high end 63 is very improbable and highly likely to be incorrect. Most tennis players retire in their 30's, although some have played into their early 40's.

Some large outliers were observed in the match statistics fields, likely due to a well known marathon match which spanned three days and lasted 11 hours and 5 minutes [8]. It was the longest match ever played, so it's expected the match statistics would be much higher for this match, given that the data is provided as totals (instead of percentages). There also were small outliers as well; 50 matches had `minutes` values of 4 or less, which are too small to represent an entire completed match (the mean value was 101.75, and the 5th percentile was 55). These values were from matches which were started and weren't completed, generally due to injury (these are known as "retirements"). If a player forfeits and pulls out of a previously scheduled match before it starts, it is known as a "walkover." The `l_bpSaved` field had two invalid values (–6 and –4), which are not possible since they represent counts.

Exploratory Visualization

Figure 1 shows histograms of `winner_rank` and `loser_rank` next to each other, with values above 200 excluded for clarity. From this plot, a relationship between player rank and being the winning player is apparent. The distribution of `winner_rank` values is concentrated at the top rankings, while `loser_rank` values are distributed toward lower (i.e., higher in value) rankings. Below rankings around 50, the number of losing players is more prevalent than winning players; players at those rankings tend to lose at a higher rate than players with rankings above 50. The gap between the number of winning players and losing players in the top 3 rankings is even more apparent, with nearly 4.5 times more wins (7,994) than losses (1,800) at those levels. Since rankings are a direct reflection of previous results (and presumably a player's standing among their peers), this is expected; as players lose more, they are also more likely to move down in rankings, making a string of losses by players in those rankings relatively unlikely, since other players would likely overtake those rankings. This relationship between rank and match wins also plays out in comparing the overall statistics of these values, with `winner_rank` having a mean value of 79.7 and median value of 47, and `loser_rank` having a mean value of 119.6 and a median value of 77.

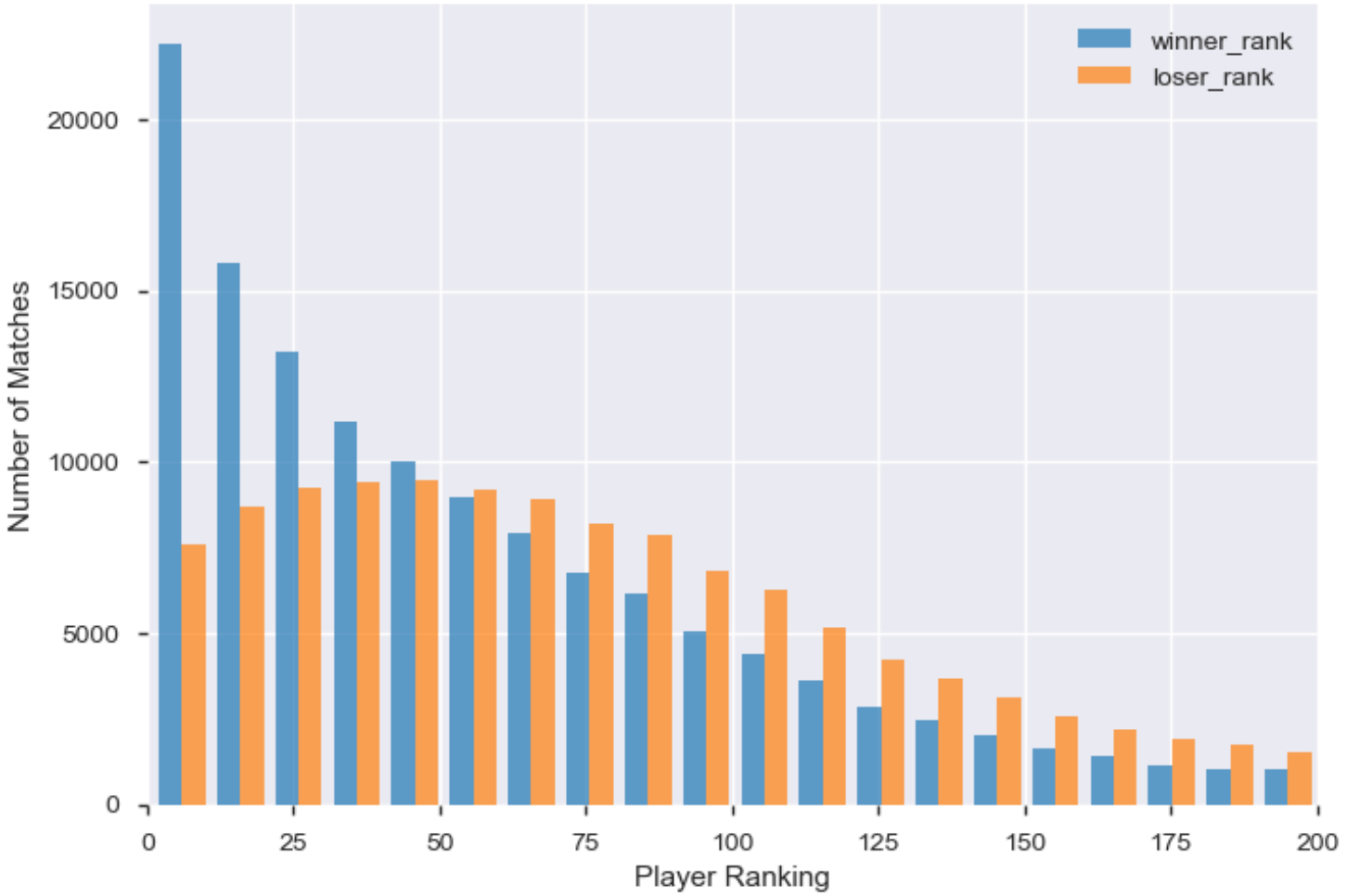

Figure 1. Comparison of `winner_rank` and `loser_rank` distributions

Algorithms and Techniques

The primary algorithm used to be used in this effort is XGBoost, which is a gradient boosted tree algorithm. Gradient boosted tree models are similar in some ways to Random Forest models in that they both are tree-based ensemble learning methods and use bagging to prevent overfitting. However, gradient boosted tree algorithms also use boosting techniques to iteratively improve results; they do this by fitting the trees sequentially in a way that tries to reduce the errors observed in the previous tree [9]. This combination of boosting and bagging helps improve both variance and bias. As a result, gradient boosted tree algorithms have produced state-of-the-art results in classification applications [10]. The XGBoost implementation of gradient boosted trees is unique due to its scalable architecture, which enables it to run as much as 10 times faster than other methods on a single computer [10]. XGBoost was selected based on its reputation for strong performance in both results and speed in classification applications, and due to the number of supporting resources available.

A cross-validation scheme will be applied during model tuning to try to provide an overall representation of the performance of the model. Cross-validation is a standard way to evaluate and compare machine learning models using multiple subsets of training/validation data to estimate how

well they would perform against unseen data. A time series splitting method will be used to split the data for cross-validation. This is due to the fact there is a time series element to the data, and to prevent any look ahead bias resulting from training on data from future matches. The time series splitting method works by splitting the training data into a `n` splits of data, with a constant validation set size across each iteration (given by $\frac{n_{samples}}{(n_{splits}+1)}$). However, the indices of each validation set increase so the test size can increase with each split. As a result, each split has different validation data points, while the training set increases in size with each split (somewhat similar to varying the sample size when evaluating learning rates). Additionally, cross validation can be combined with hyperparameter tuning efforts to more effectively test the effect of different hyperparameter values across many subsets of data.

A Bayesian hyperparameter optimization scheme will be used for model tuning. This was chosen instead of a more traditional grid search approach in order to more effectively find an “optimal” set of hyperparameters without an exhaustive grid search, which can be computationally expensive and can “miss” optimal parameter value if it wasn’t included in the grid. It is similar to grid search, but instead samples parameter values from defined ranges for each parameter to determine the optimal parameter values [11]. As outlined in Snoek et al [12], Bayesian hyperparameter optimization selects these values by assuming they were sampled from a Gaussian process, and then uses model results from each sample (and all previous samples) to determine the next value to try. Because of this, Bayesian hyperparameter optimization has been shown to be significantly faster and better than other hyperparameter optimization techniques [12].

Each of these techniques will be combined to refine the model by feeding the XGBoost model into a Bayesian hyperparameter optimization scheme, which will use time series split cross-validation to evalute each sampling step, and then ultimately provide the combination of hyperparameters that produced the best cross-validated results.

Benchmark

For this effort, there are two naive models and one benchmark model which can be used for comparison. A completely naive log loss benchmark would be to predict a probability of 50% for a Player 1 victory for every match, which would lead to a log loss value of 0.693. A naive log loss model based on ranking would be to predict a 100% probability of Player 1 winning if they had a higher ranking (i.e., lower in value), and a 0% probability if they had a lower ranking. This would lead to a log loss value of 12.1, surprisingly much worse than the completely naive “coin toss” prediction. A more applicable benchmark model is an Artificial Neural Network model based on historical match data (Sipko and Knottenbelt, [3]) which produced a log loss value of 0.61 on the test set. This logistic loss metric is the goal to beat in this effort. This benchmark is applicable because it used very similar source data to solve a similar problem: predicting the outcome of men’s professional singles matches.

III. Methodology

Data Preprocessing

First, the dataset was sorted by `tourney_date` and `match_num` to get the matches in chronological order. Then, some of the features in the original dataset were removed because they weren't expected to impact the model. These include `tourney_id`, `tourney_name`, `match_num`, `winner_name`, `loser_name`, `winner_ioc`, `lower_ioc`, and `score`. Additionally, some features such as `winner_entry`, `loser_entry`, `winner_seed`, `loser_seed`, `winner_ht`, and `loser_ht` were missing a significantly higher amount of values than the other fields and also weren't expect to influence the model, so they were removed.

Since historical match statistics are the primary input of the model and it's expected that any predictions on future matches by this model would have full match statistics available, any matches with missing values were removed from this reduced dataset, dropping the match dataset down to 72,650 matches. The previously mentioned `age` outliers were no longer present after this filtering step (the max value was 44, which is much more reasonable), but the negative `l_bpSaved` values remained, so the two matches associated with them were removed.

A `tourney_year` field was created based on the `tourney_date` field to make it easier to analyze the distribution of matches in this filtered dataset. An additional data profile was then created based using this filtered dataset in order to determine if any additional corrections previously identified in the data needed to be removed. Additionally, a `match_id` field was created based on indexing the filtered (and previously sorted) data, resulting in an updated chronological identifier of matches in the dataset.

Next, this filtered dataset of match results and statistics needed to be processed to remove any indicators of a win or loss in the field names in order to prevent any data leakage in our model. The filtered dataset was split into two datasets of player statistics: one for the winning players (using the fields with a `winner_` or `w_` prefix), and one for the losing players (using fields with a`loser_` or `l_` prefix). In both cases, the fields were renamed to remove the prefixes denoting a win or loss. In both datasets, a few additional features were created based on the supplied statistics, as shown in Table 2.

Table 2. New Features

Field Name	Description	Calculation
RtGms	Number of return games	opponent's SvGms
player_id	Player id	id
opponent_id	Opponent's player id	opponent's id
bpOpp	Number of break point opportunities	opponent's bp_faced
bpFailedOpp	Number of failed break point conversions	opponent's bp_saved
rtpt	Number of return points	opponent's svpt
rtpt1st	Number of 1st return points played	opponent's 1stIn
rtpt1stLost	Number of 1st return points lost	opponent's 1stWon
rtpt1stWon	Number of 1st return points won	rtpt1st - rtpt1stLost
rtpt2nd	Number of 2nd return points played	rtpt - rtpt1st
rtpt2ndLost	Number of 2nd return points lost	opponent's 2ndWon
rtpt2ndWon	Number of 2nd return points won	rtpt2nd - rtpt2ndLost
bpWon	Number of break points won	bpOpp - bpFailedOpp
svpt2nd	Number of 2nd serve points won	svpt - 1stIn

Special care was placed toward insuring there was no information leakage in the calculation of any of the historical statistics for each player, since the players are referenced as "winner" and "loser." For each match in the winning player dataset, a label of `1` or `2` was randomly applied, and a new field called `won_match` was created and assigned a value of `1`. Then, player labels for each match in the losing player dataset were assigned as `1` or `2` based on the label created in the winning player dataset, such that each match had a player 1 and player 2 label. A `won_match` field was also created in the losing player dataset and assigned a value of `0`. This resulted in a 50% split in the distribution of matches where player 1 won (36,329) and player 2 won (36,319), removing any association between player label and match outcome.

Next, the two datasets were combined via a union to create a dataset at the player-match level (i.e., with each row containing the playing statistics for a single player for a single match), doubling the size of the dataset (145,296 rows with 34 columns). This resulting player statistics dataset was then used to create features to be used by the model. Since the match statistics are unknown until the end of a match, the statistics for a given match to be predicted by the model must be unknown to it, only prior matches. So, all of the match statistics fed into the model must be based on historical data (prior to the match in question). A `hth_wins` feature was created based on the previous total number of wins by a player against a given opponent. A `match_count` feature was created and assigned a value of `1` to track the number of matches played in order create a `hth_matches` feature based on the total number of previous matches between two players.

With the exception of the head to head features, the rest of the player statistics features were calculated based on a subset of previous matches (rather than all previous matches). This was done as a way to rely on more recent matches, rather than including matches potentially from many years ago when a player could have been at a different relative skill level. For most of these features, a rolling window of 20 previous matches was used (when available). As a result, this introduced additional matches with no player statistics in the case where no previous data existed, either from playing their first professional match or first match against their opponent. These matches were not filtered out, since they represent valid scenarios we would expect the model to encounter; in these cases, the values were set to `0` after the rolling window calculations were applied, indicating no prior information. A value of 20 was chosen because most tournaments span 5–7 matches (if a player keeps winning and progressing), so it could encompass 3–4 tournaments for a player on a particularly strong streak, or many more tournaments for a more average player. However, it's small enough it should encompass less than a year's worth of matches, the results of which would be encoded to some degree in the rank features, which are based on a rolling 12 month window. The only exception to the 20 match window was `minutes`, which was calculated based on an average of the previous 3 matches, with the assumption that 3 matches is likely sufficient to recover from a particularly long match. Since some sum values were `0`, cases where both the numerator and denominator were `0` resolved to infinity; in these cases, they were replaced with `0`.

These rolling window calculations proved difficult to code, since it took significant trial and error to ensure only prior matches were considered and the calculations were being applied correctly; for example, at one point I discovered there was an error and they were incorrectly being calculated based on the 20 most recent matches amongst all players, not just for a given player. The pre-calculation sum window sizes and calculations for each of the features are shown in Table 3.

Table 3. Feature Transformation Calculations

Field Name	Description	Rolling Window	Feature Calculation
1stIn_pct	Percentage of first serves in play	20	$\frac{\sum 1stIn}{\sum svpt}$
svWon_pct	Percentage of serve points won	20	$\frac{\sum 1stWon + \sum 2ndWon}{\sum svpt}$
1stWon_pct	Percentage of first serve points won	20	$\frac{\sum 1stWon}{\sum 1stIn}$
2ndWon_pct	Percentage of second serve points won	20	$\frac{\sum 2ndWon}{\sum svpt2nd}$
ace_pct	Average number of aces per service game	20	$\frac{\sum ace}{\sum SvGms}$
df_pct	Average number of double faults per service game	20	$\frac{\sum df}{\sum SvGms}$
bpFaced_pct	Average number of break points faced per service game	20	$\frac{\sum bpFaced}{\sum SvGms}$
bpSaved_pct	Average number of break points saved per service game	20	$\frac{\sum bpSaved}{\sum SvGms}$
rtWon_pct	Percentage of return points won	20	$\frac{\sum rtpt1stWon + \sum rtpt2ndWon}{\sum rtpt}$
rt1stWon_pct	Percentage of first return points won	20	$\frac{\sum rtpt1stWon}{\sum rtpt1st}$
rt2ndWon_pct	Percentage of second return points won	20	$\frac{\sum rtpt2ndWon}{\sum rtpt1st}$
bpFaced_pct	Average number of break points opportunities per return game	20	$\frac{\sum bpOpp}{\sum RtGms}$
bpSaved_pct	Average number of break points won per return game	20	$\frac{\sum bpWon}{\sum RtGms}$
won_match_pct	Percentage of matches won	20	$\frac{\sum won\_match}{\sum match\_count}$
surface_won_pct	Percentage of matches won on current surface	20	$\frac{\sum won\_match_{surface}}{\sum match\_count_{surface}}$
minutes	Average number of minutes per match	3	$\frac{\sum minutes}{\sum match\_count}$
hth_pct	Percentage of head to head matches won	All previous	$\frac{hth\_wins}{hth\_matches}$

Finally, all of the player statistics features were combined with player attributes (`age`, `rank`, `rank_points`, and `hand`). Since XGBoost requires numerical features, the `hand` feature was transformed using one-hot encoding, resulting in three columns in its place (`hand_R`, `hand_L`, and `hand_U`). Next, the player 2 subset of these features was subtracted from the player 1 subset of these features, resulting in the difference between the statistics and attributes between each player based on pre-match information. This follows the same approach Sipko and Knottenbelt [3] and Cornman et al. [4] followed, and was done to reduce dimensionality (effectively cutting the number of potential features in half) and to capture the individual advantages or disadvantages in each area. Since the goal of this effort is to correctly predict whether player 1 will win the match, after these datasets were subtracted, any `won_match` values of `-1` (indicating player 2 won) were changed to `0` to correctly reflect the desired binary target `won_match` variable, where `1` corresponds to a player 1 win and `0` corresponds to a player 1 loss. Lastly, these player comparison features were combined with match attributes (`surface`, `tourney_year`, `tourney_level`, and `best_of`), and `tourney_level` and `surface` were transformed using one-hot encoding. The resulting final set of features is provided in Table 4.

Table 4. Final Features

Field Name	Description
`tourney_year`	Year match was played
`age`	Difference in age between players
`rank`	Difference in rank of player at time of match
`rank_points`	Difference in rank points of player at time of match
`won_match`	Target variable and indicator of whether player 1 won the match
`1stIn_pct`	Difference in percentage of first serves in play
`svWon_pct`	Difference in percentage of serve points won
`1stWon_pct`	Difference in percentage of first serve points won
`2ndWon_pct`	Difference in percentage of second serve points won
`ace_pct`	Difference in average number of aces per service game
`df_pct`	Difference in average number of double faults per service game
`bpFaced_pct`	Difference in average number of break points faced per service game
`bpSaved_pct`	Difference in average number of break points saved per service game
`rtWon_pct`	Difference in percentage of return points won
`rt1stWon_pct`	Difference in percentage of first return points won
`rt2ndWon_pct`	Difference in percentage of second return points won
`bpFaced_pct`	Difference in average number of break points opportunities per return game
`bpSaved_pct`	Difference in average number of break points won per return game
`won_match_pct`	Difference in percentage of matches won
`surface_won_pct`	Difference in percentage of matches won on current surface

minutes	Difference in average number of minutes per match
hth_pct	Difference in percentage of head to head matches won
hand_R	Difference in players using their right hand
hand_L	Difference in players using their left hand
hand_U	Difference in players having an undefined dominant hand
surface_Carpet	Binary indicator of carpet match surface
surface_Clay	Binary indicator of clay match surface
surface_Grass	Binary indicator of grass match surface
surface_hard	Binary indicator of hard court match surface
tourney_level_A	Binary indicator of ATP (500 & 250) level tournament match
tourney_level_C	Binary indicator of Challenger level tournament match
tourney_level_D	Binary indicator of Davis Cup level tournament match
tourney_level_G	Binary indicator of Grand Slam level tournament match
tourney_level_M	Binary indicator of Masters level tournament match
tourney_level_F	Binary indicator of Tour Finals level tournament match

Implementation Overview

For this effort, the XGBoost gradient boosted tree algorithm will be used. The data will be split into training, validation, and test sets. One of the benefits of the primary dataset is that it contains many years' worth of matches and tournaments. As such, there should be plenty of data available to split into three distinct training, validation, and test sets. Previously mentioned efforts split this data by year, and I expect to follow a similar approach. While it is true that certain factors affecting the outcome of tennis matches may change over time (due to changes in playing styles, court surfaces, racket/string technology, etc.), this approach is more representative of how such a model would be utilized in a real world setting (i.e., trained on previous years' data), and the game changes slowly enough on a year-to-year basis that any such factors should be minimal. Additionally, this helps preserve balance between sets in regard to court surface and tournament, since tournaments and their associated court surfaces rarely change year to year. Learning curves will also be examined to look for overfitting and to optimize the training set size (i.e., if it should be reduced so the models can generalize better).

Once the data is split, the training and validation sets will be used to train and refine the models. Time series split cross validation will also be employed to determine how robust the models are to changes in input. First, an XGBoost model with the following default parameters will be trained as a starting point.

```
XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100,
              silent=True, objective='binary:logistic',
              booster='gbtree', n_jobs=-1, nthread=-1, gamma=0,
              min_child_weight=1, max_delta_step=0,
```

```
                subsample=1, colsample_bytree=1, colsample_bylevel=1,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
                base_score=0.5, random_state=0, seed=None, missing=None)
```

Next, the XGBoost model will be fed into a Bayesian hyperparameter optimization tuning scheme to determine the combination of hyperparameters which produce the best results for each algorithm in the validation set. Then, feature importances will be examined to determine which features could be removed to reduce dimensionality. Finally, probability curves for the optimized model will be examined to determine if it needs to be fed into a probability calibration algorithm. The primary model output is the predicted probability of Player 1 winning, and calibrating output probabilities could help ensure predicted probabilities aren't skewed. Lastly, the optimized and calibrated XGBoost algorithms will be scored on the previously held-out test set. The results will be analyzed and compared to the benchmark models for log loss.

Implementation

First, the full dataset was split into a training/validation set and a test set. The split was made based on `tourney_year` to best approximate a 80/20 split. This was done to ensure as best possible that all surfaces and tournaments would roughly be represented equally between the two datasets, since in general tournaments and surfaces do not change year over year. The training/validation set had 57,588 tennis matches, and the test set had 15,060 matches. Then, the two sets were split further into a set of features and their associated `won_match` labels, for a total of four sets of data (training features, training labels, test features, and test labels). The `won_match` and `tourney_year` columns were dropped from both sets of features, and the label sets only included the `won_match` column. The resulting training/validation and test feature sets contained 34 features.

Next, a XGBoost classification model was fit on the training set using default hyperparameters as previously defined to obtain cross-validated log loss scores using the time series splitting scheme described previously. A split size of 4 was used for this effort, again approximating an 80/20 split. A visualization of the sizes of the training and validation sets are provided in Figure 2. Each validation split had 11,517 matches.

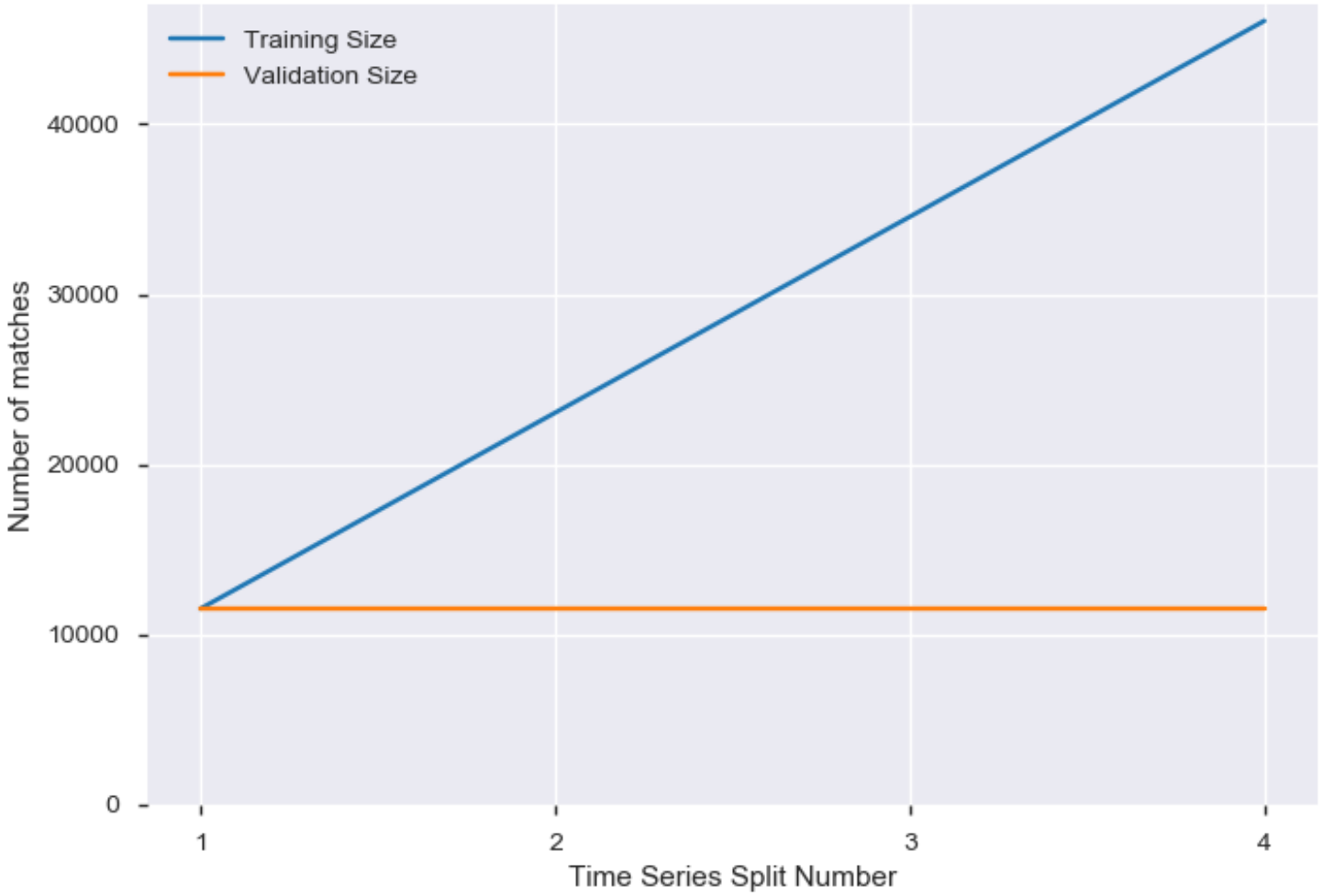

Figure 2. Time Series Split Sizes for Cross Validation

The default XGBoost parameters were used to provide an “out of the box” starting point for comparing improvement efforts. The best cross validation log loss score was 0.590, and the mean value across all splits was 0.614. A plot of the scores across each split (and with increasing test set size) is provided in Figure 3. Based on this graph, there appears to be room for improvement, since the model appears to be underfitting.

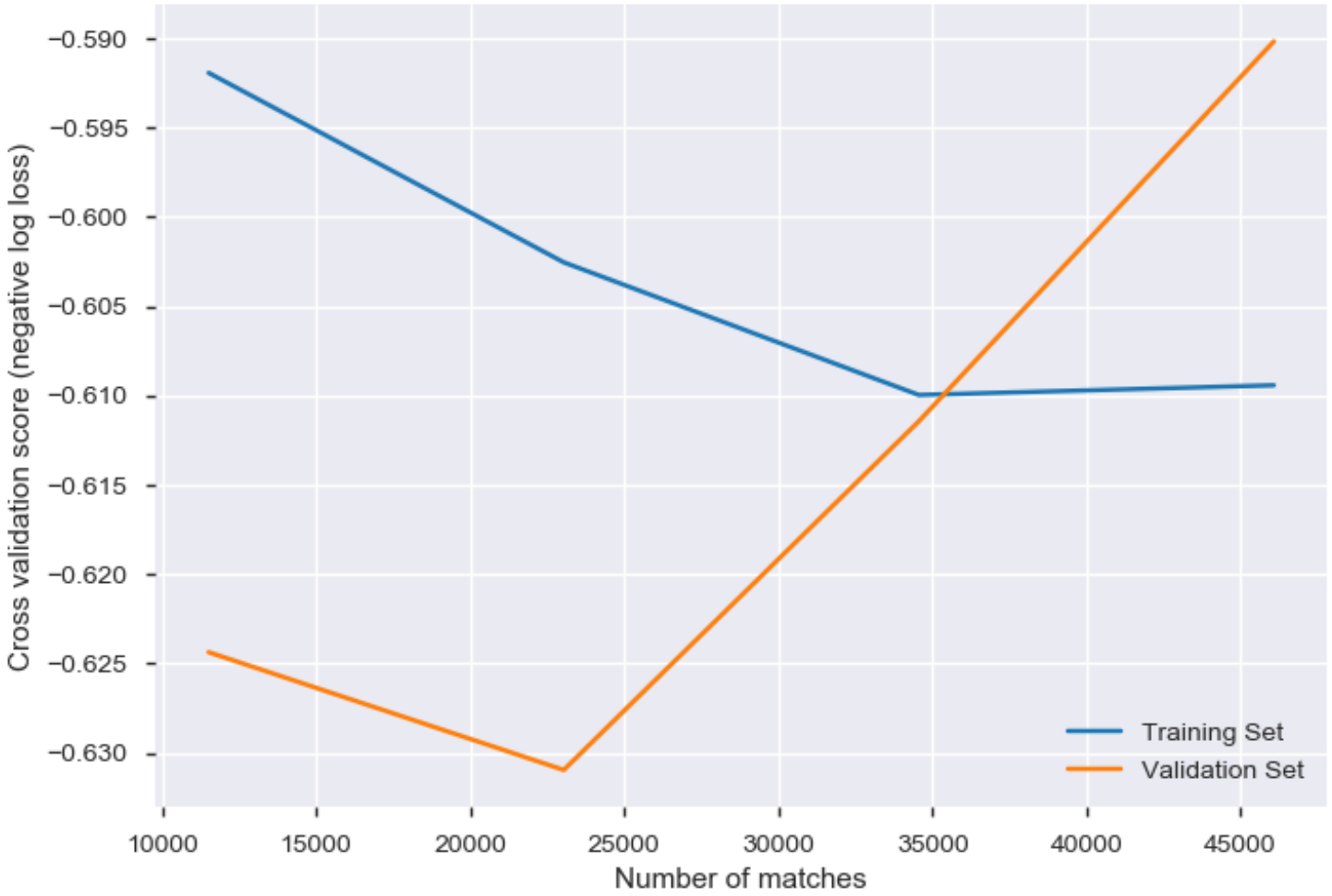

Figure 3. Default XGBoost Model Cross Validation Scores

Then, feature importances provided by the model were plotted and evaluated (Fig. 4). The two ranking features had the highest feature importances overall, followed by `age`, `surface_won_pct`, and `sv_won_percent`. None of the `hand` features appeared to influence the model significantly, along with the `best_of`, `minutes`, and `tourney_level` features. Surprisingly, `surface_clay` was the 13th most important feature and the only `surface` feature (other than `surface_won_pct`) to influence the model, likely due to the fact clay is a much slower surface than the others and plays much differently.

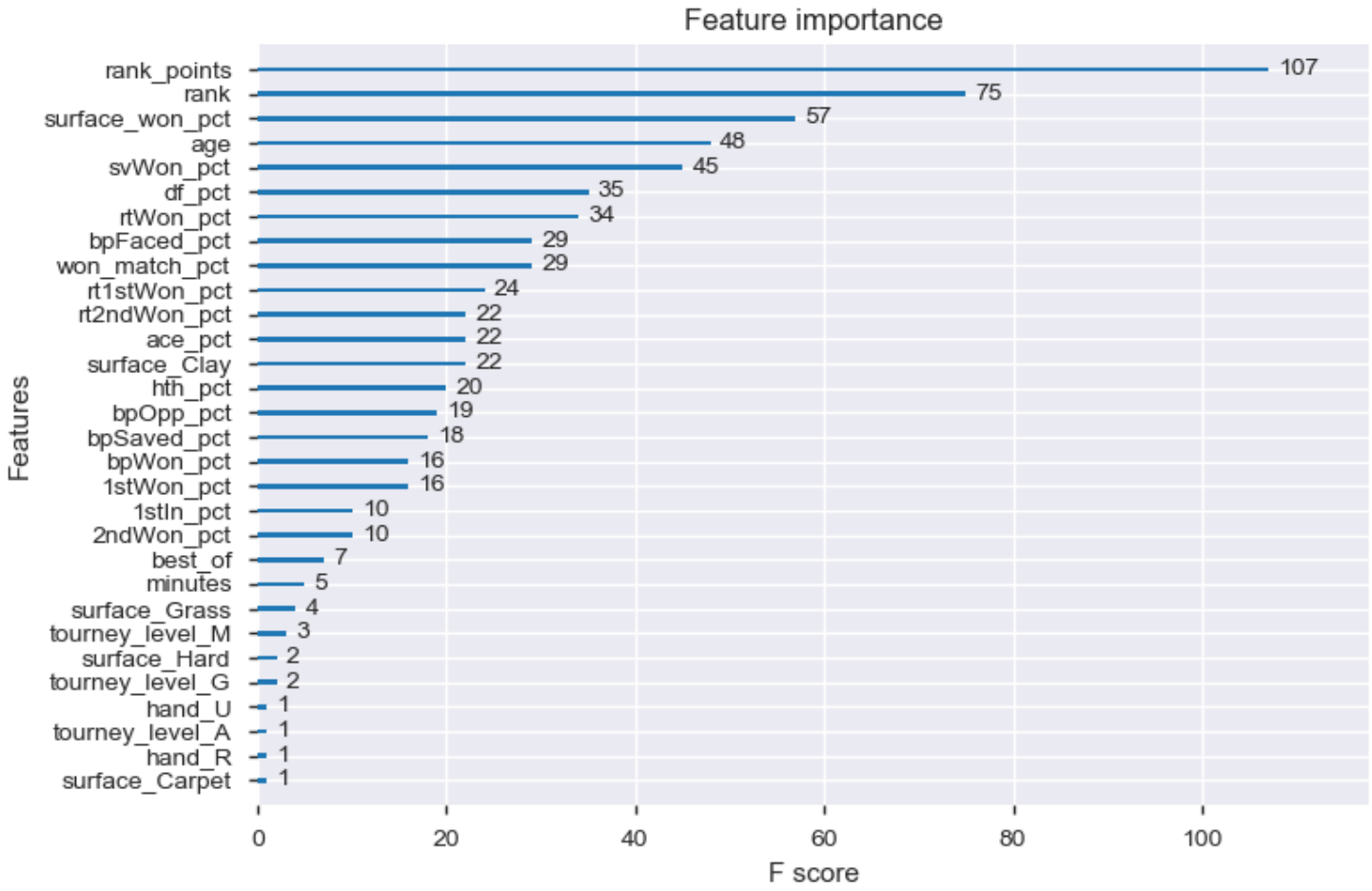

Figure 4. Default XGBoost Model Feature Importances

Finally, a probability curve was created based on the output probabilities of the model in the training/validation set (Fig. 5). This curve didn't indicate a need for probability calibration, with the resulting curves almost exactly matching a perfectly calibrated model, likely due to the way XGBoost calculates probabilities. As a result, the probabilities were not calibrated.

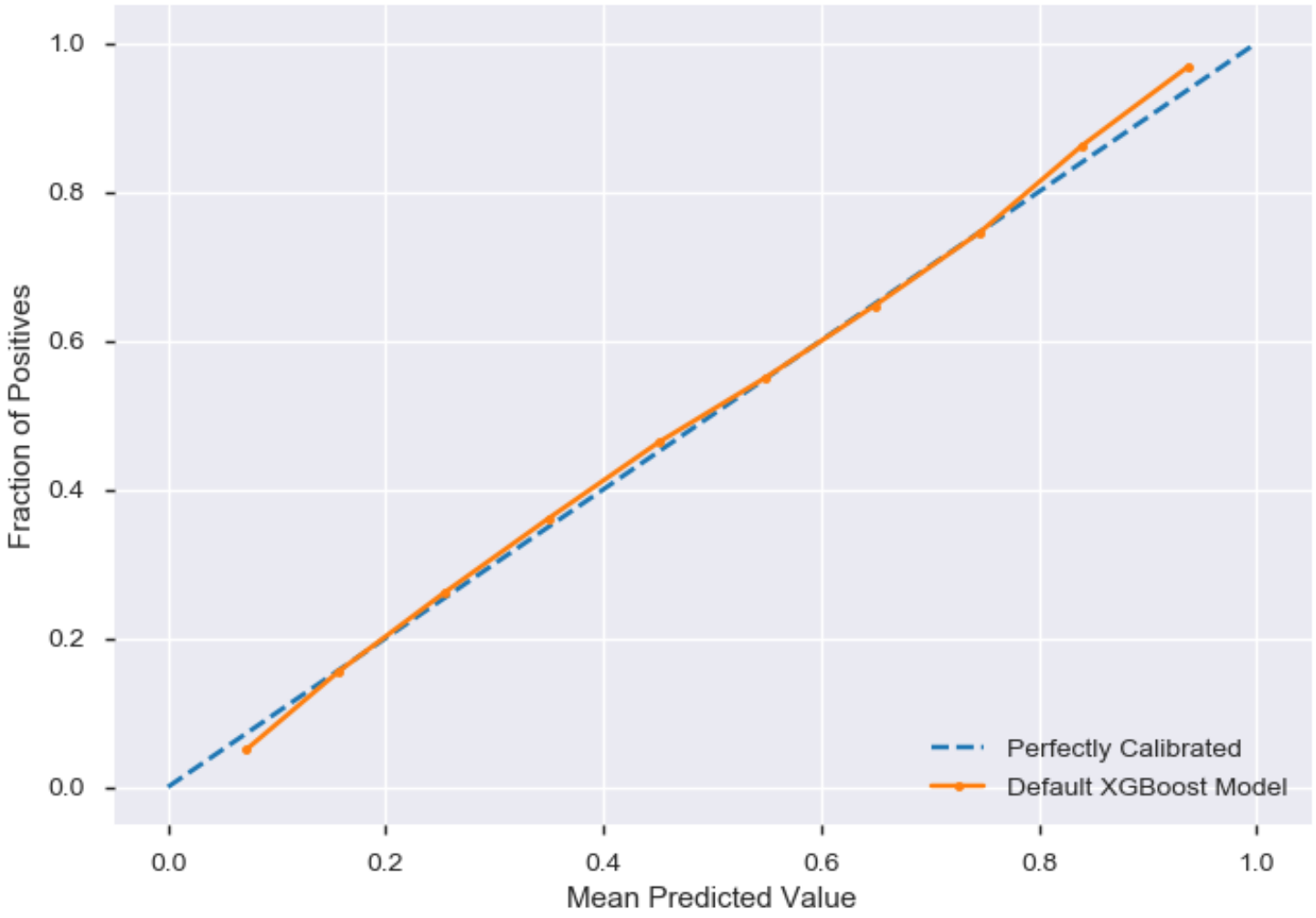

Figure 5. Default XGBoost Model Calibration Curve

Refinement

After the initial default model was trained and evaluated using a time series cross validation scheme, the hyperparameters were tuned using a Bayesian hyperparameter optimization scheme combined with the previous time series cross validation scheme; 25 optimization iterations (or samples) per parameter were used for this effort. The 8 features included in the hyperparameter optimization were: `learning_rate` (step size shrinkage at each step), `min_child_weight` (minimum sum of weights needed for a child), `n_estimators` (number of boosting rounds), `max_depth` (maximum depth of tree), `gamma` (minimum loss reduction for a split), `subsample` (subsampling ratio of training data), `colsample_bytree` (subsample ratio of columns used per tree), and `reg_alpha` (L1 regularization term). These parameters were chosen based on a XGBoost parameter tuning guide [13].

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bytree=0.5, gamma=10.0,
                  learning_rate=0.014668915557751905, max_delta_step=0, max_depth=5,
                  min_child_weight=20, missing=None, n_estimators=1500, n_jobs=-1,
                  nthread=-1, objective='binary:logistic', random_state=0,
```

```
                    reg_alpha=1e-05, reg_lambda=1, scale_pos_weight=1, seed=None,
                    silent=True, subsample=0.5)
```

This resulted in some improvement in the cross validated results, with the top validation set log loss score improving to 0.588 and a slightly better mean value of 0.612. Additionally, the learning curve wasn't quote as divergent as the previous curve, but the overall shape was the same. A probability calibration curve was generated as well, but also indicated that probability calibration was not required.

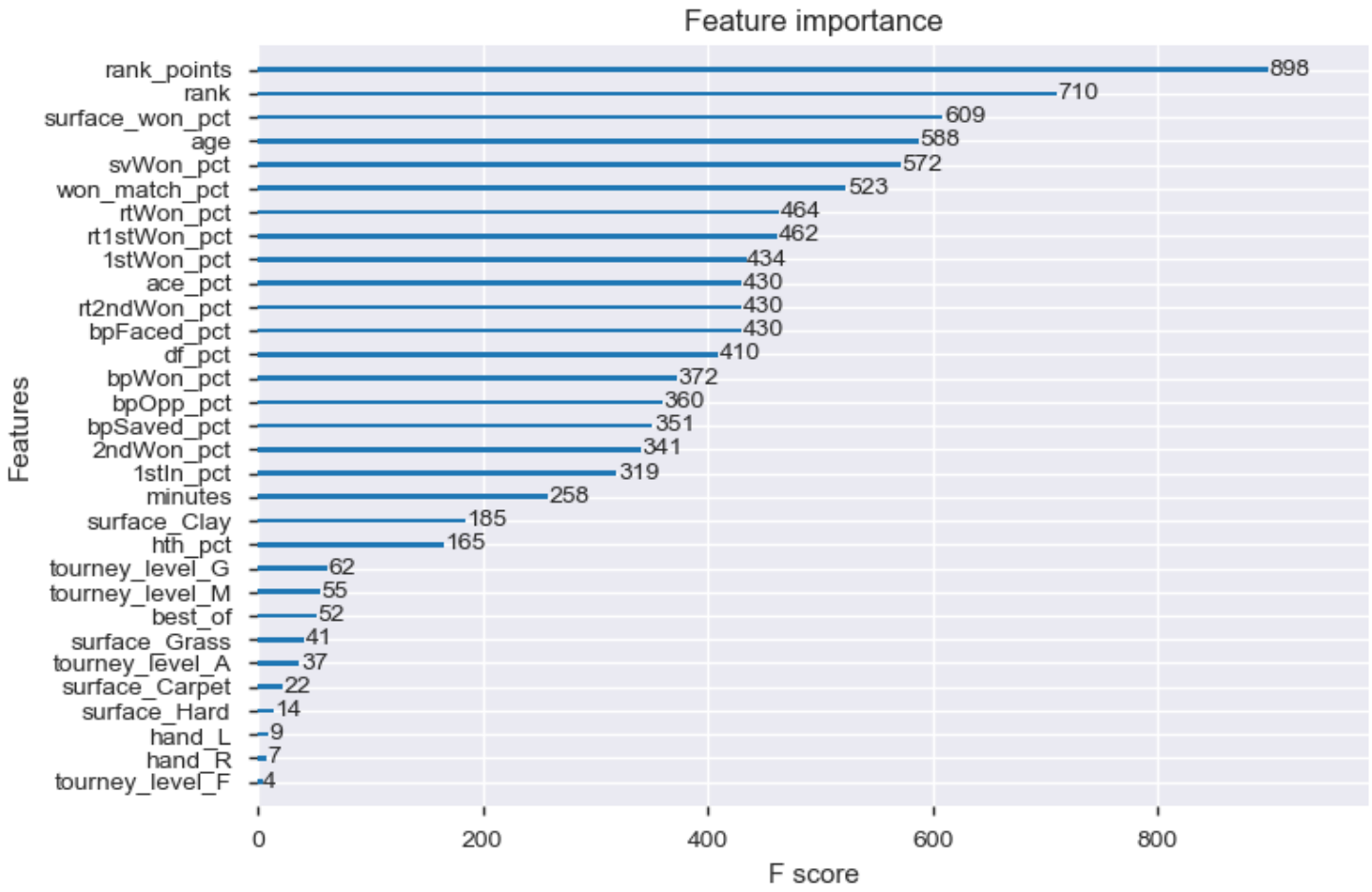

Figure 6. Optimized XGBoost Model Feature Importances

Next, based on the feature importances plots (Fig. 6), the 13 features with the lowest feature importances were removed from the training/validation and testing sets to see if their removal would lead to a model which generalized better. The removed features were `tourney_level_G`, `tourney_level_A`, `tourney_level_M`, `tourney_level_D`, `tourney_level_C`, `tourney_level_F`, `surface_Grass`, `surface_Carpet`, `surface_Hard`, `hand_L`, `hand_R`, and `hand_U`. Then, the reduced feature set was fed into the same Bayesian hyperparameter optimization scheme described above. The resulting model is described in detail in the next section, and had the following parameters:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                  colsample_bytree=0.5, gamma=10.0,
                  learning_rate=0.0034126039328430426, max_delta_step=0, max_depth=10,
               min_child_weight=1, missing=None, n_estimators=1500, n_jobs=-1,
                  nthread=-1, objective='binary:logistic', random_state=0,
                  reg_alpha=1e-05, reg_lambda=1, scale_pos_weight=1, seed=None,
                  silent=True, subsample=0.7396835114478809)
```

These values are slightly different, with the biggest changes in `subsample` (0.5 vs 0.73), `min_child_weight` (20 vs 1), `max_depth` (5 vs 10), and `learning_rate` (0.015 vs 0.003).

IV. Results

Model Evaluation and Validation

The optimized model trained on the reduced training/validation feature set was selected as the final model. While the max validation set log loss (0.589) was very slightly worse than the previous model (0.588), along with the mean log loss value (0.613 vs 0.612), the learning curves associated with this model indicated much better convergence toward a common value (Fig. 7). Combined with the reduction in features/dimensionality, I think this model will generalize better overall.

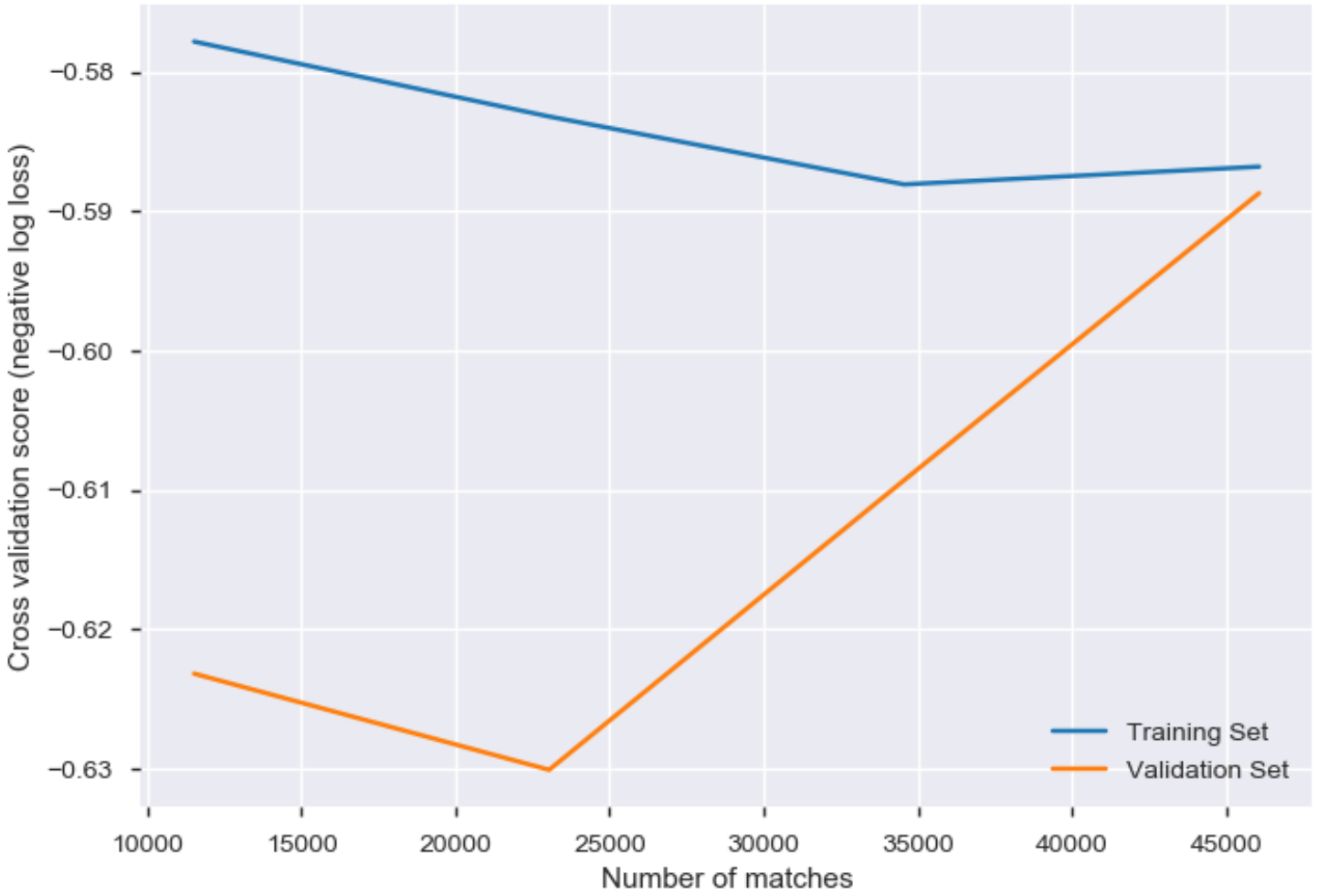

Figure 7. Optimized XGBoost Model with Reduced Features Cross Validation Scores

The resulting feature importances are different as well, and are provided in Figure 8. Interestingly, `age` had the second highest feature importance, and `surface_clay` moved down to have the second lowest feature importance.

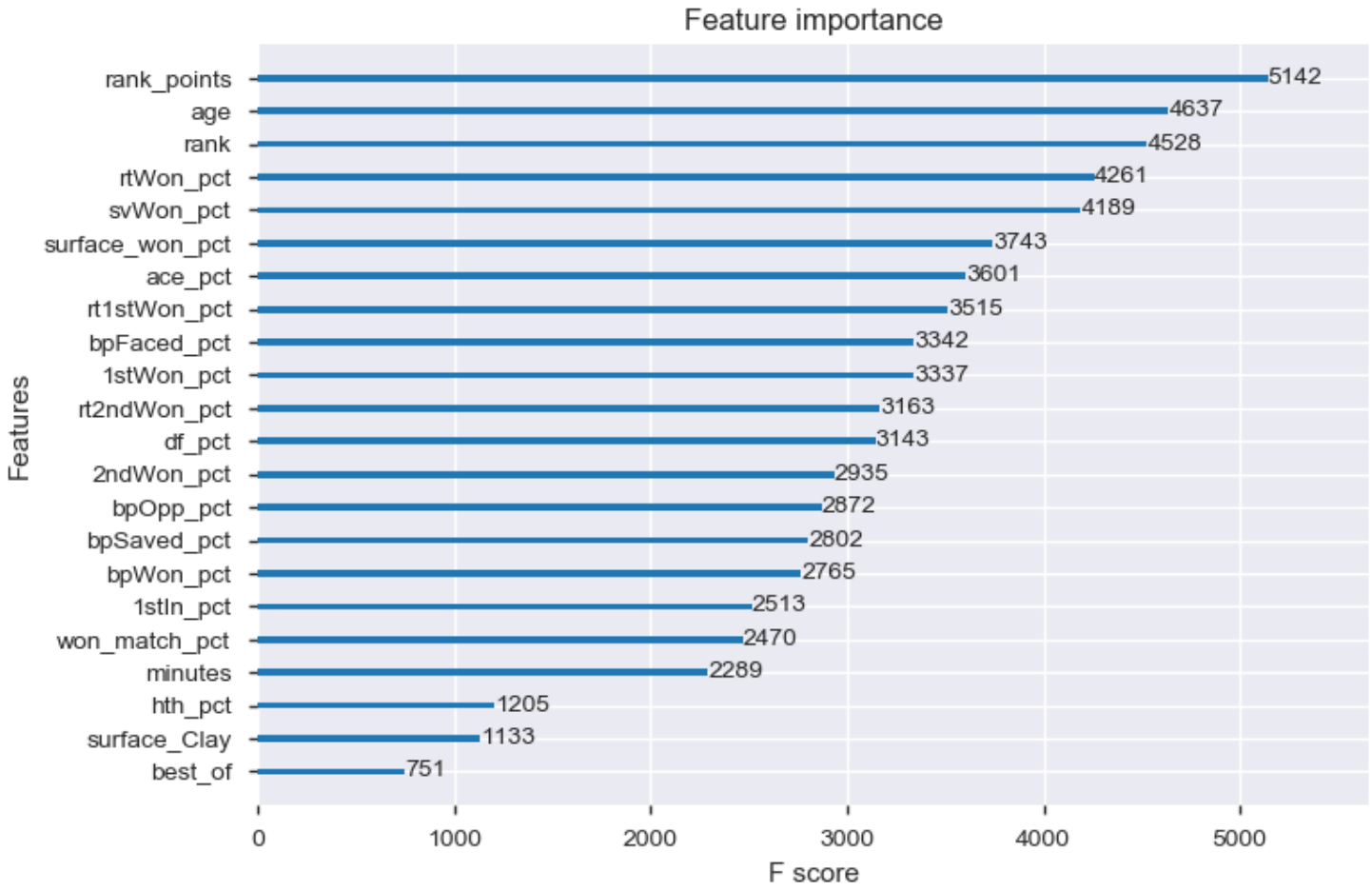

Figure 8. Optimized XGBoost Model with Reduced Features Feature Importances

Like the previous models, a probability calibration curve was evaluated and it was determined probability calibration was not required. Finally, each of the models were fit against the entire training/validation set and scored against the held out test set to see how well they would handle unseen data. The results are provided in Table 5.

Table 5. Test Set Scores

Model	Best Test Set Log Loss Score
Default XGBoost	0.5832
Optimized XGBoost	0.5815
Optimized XGBoost with Reduced Features	**0.5820**

The final model performed well on the test set, with a log loss score of 0.582. This score was better than its validation scores, and is an indicator this model generalizes well to unseen data. Surprisingly, the Optimized XGBoost model had a slightly better score on the test set. However, since the difference was small and for the reasons mentioned previously, Optimized Model with Reduced Features is the selected model. In order to test the robustness of the model, individual years were removed from the training data, then the model was fit to the perturbed data and scored against the test set. These

results showed very little variation in the resulting test scores, indicating the model should be robust to changes in the training data. Based on these results, I believe the results from this model can be trusted.

Justification

The final results for the chosen model are stronger than the benchmark models reported earlier. The chosen model represents a 4.4% improvement in log loss score over Sipko and Knottenbelt's benchmark model, while using a similar dataset with fewer features (their dataset had additional features like average serve speed). A full comparison between the benchmark models and the final model test set results is provided in Table 6.

Table 6. Benchmark Comparison

Benchmark	Score	Improvement
Naive Rank Prediction	12.1	11.52 (95.2%)
Naive Coin Toss	0.693	0.111 (16%)
Sipko and Knottenbelt	0.61	0.028 (4.4%)

The goal of this effort was to see if current state of the art algorithms like XGBoost would provide better results using a similar approach to Sipko and Knottenbelt, and while the difference is relatively small, I think this model achieved that goal. However, I don't think it could be considered to have necessarily solved the problem once and for all of predicting the outcome of tennis matches. I think there is definitely potential for improvement overall.

V. Conclusion

Free-Form Visualization

Figure 9 shows the impact of removing certain years' worth of data from the training set on the final model. It was created by iterating and removing data from individual years, training the final model on the resulting "perturbed" training set, and scoring the resulting model on the test set. As shown, the model appears to be very robust to these perturbations, with very little overall impact on the log loss scores. The mean value across all perturbations was 0.582, with a standard deviation of 0.0002. Based on these results, it appears the model has learned enough from the remaining training data that no individual year's worth of data is important to the predictions, as we would hope.

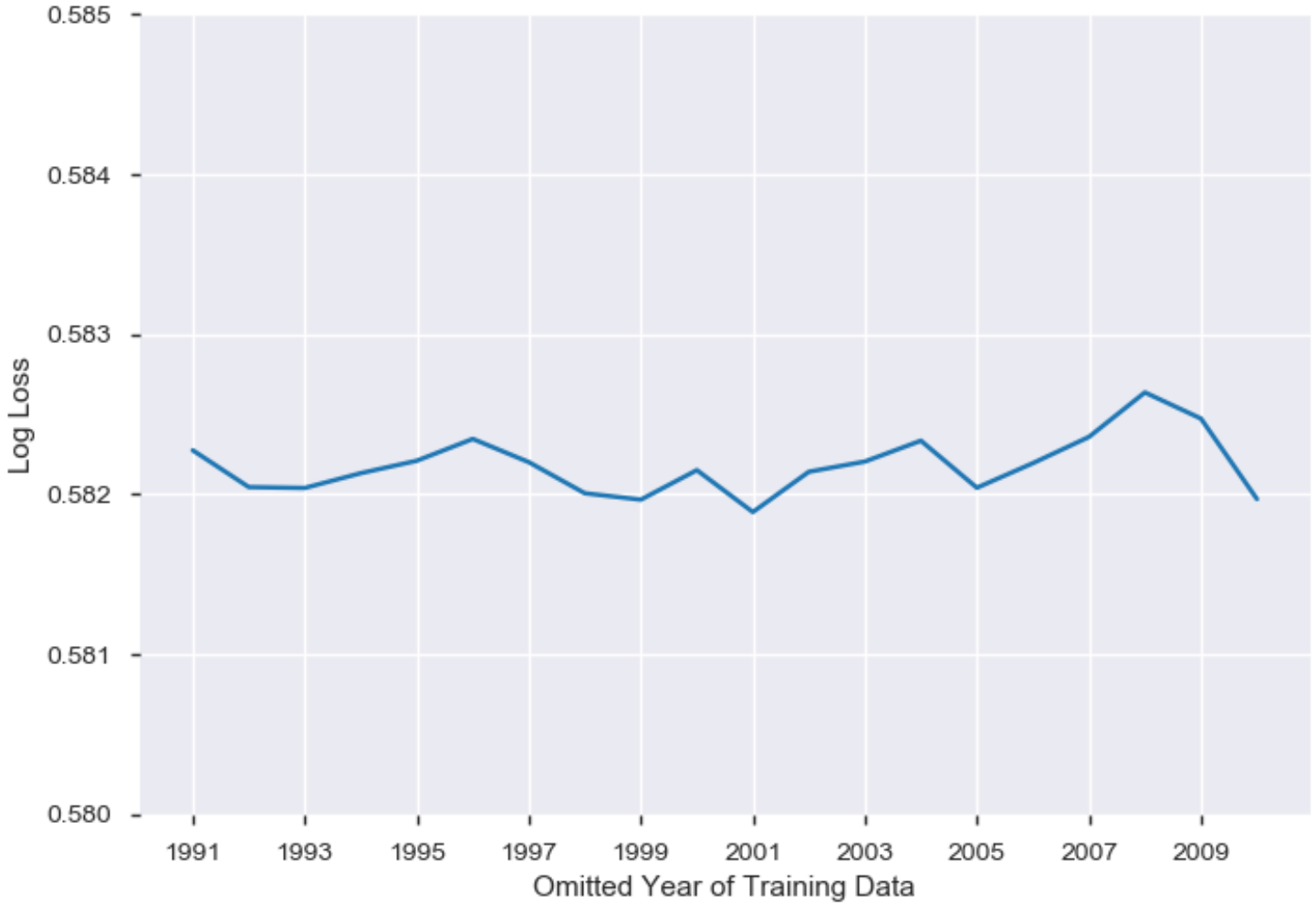

Figure 9. Perturbed Test Scores by Omitted Year

Reflection

In summary, ATP match data was used to predict the outcome of tennis matches using a gradient boosted tree machine learning model. The ATP match data was cleaned to remove missing values and processed to create a series of model features based on recent and head to head pre-match playing statistics for each player. Then, these features were used to train models with different settings in order to determine the best model. Additionally, learning curves, feature importances, and probability curves were analyzed for each model. Finally, the top performing model was selected based on training data performance and evaluated against test data. The results showed the final model performed better than both naive models and the benchmark model, and should generalize well to future matches.

The most interesting part of this project was getting a chance to analyze tennis data and see which features turned out to be most important for the model. While I expected player ranking to play a role, I was surprised at how important they were to the model. It was really helpful to have a “head start” in-terms of knowing the data and domain so well.

The most difficult part of the project was trying to determine the best way to process and split the data into a form which could be used by a model, especially in terms of processing all of the statistics. Once I started creating the models, I also found I was easily distracted by chasing down different approaches or hyperparameters to tune. Lastly, the length of time needed to optimize and train the models (2+ hours total) made refinements burdensome toward the latter stages of the project; this could have potentially been mitigated by utilizing cloud based computing resources instead of my personal computer.

The final model was in line with my expectations. While I'm certainly glad it improved upon the benchmark model, I wasn't expecting a drastic improvement since it was using similar data and similar processing techniques and features overall. While betting return on investment wasn't evaluated here, since this model performed better than Sipko and Knottenbelt's [3] model (which was shown to produce a positive ROI), I think it could be used as the basis of a betting strategy.

Improvement

There are a number of potential ideas for improvement for this effort. From a feature engineering perspective, it would be interesting to try different rolling window values (i.e., 10, 15, 30, all prior matches, etc.) or even combinations of them to determine if they could improve the results. Additionally, rather than taking the difference in player values, using the individual values themselves could prove useful if there are certain relationships that are dependent on those specific values. For example, maybe a match up of two players with similar strong serving statistics would have different relevant features than two players with similar poor serving statistics, each of which could potentially show up as very similar in the match data based on the difference calculation. Another approach would be to use women's professional tennis data, which is also available in some locations online. It would be interesting to see if any features are more/less important in women's tennis, and what the results would be of predicting matches based on the opposite gender's data. Lastly, more recent data (through 2019 to date) could be obtained and scored to see how well the model generalizes on even more unseen data.

From an algorithm perspective, model stacking or ensembling with different hyper parameters or algorithms could be investigated. Additionally, the LightGBM implementation of gradient boosted trees could be applied to see if that algorithm is potentially better suited to this problem (or it could be combined with XGBoost in an ensemble model). It would also be interesting to compare the final model prediction score to a log loss score generated from implied probabilities based on betting odds from various services as another benchmark.

Lastly, while I was able to improve on the benchmark model, I do think there's potential for a better solution (based on the improvement ideas above). I think there is likely a "ceiling" though, mainly due to the unpredictability of all sports. That unpredictability is part of what makes sports enjoyable, but it will also be fun to see where that tennis match predictability ceiling turns out to be!

References

1. B. Sawe. The Most Popular Sports In The World. *Worldatlas*, https://www.worldatlas.com/articles/what-are-the-most-popular-sports-in-the-world.html, 2018.
2. M. Townend. Tennis gambling market second only to football for bookmakers after boom in online and in-play betting. *The Daily Mail*, https://www.dailymail.co.uk/sport/tennis/article–3405544/Tennis-gambling-market-second-football-bookmakers-boom-online-play-betting.html, 2016.
3. M. Sipko and W. Knottenbelt. Machine Learning for the Prediction of Professional Tennis Matches. *Imperial College London, MEng Computing – Final year project*, https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/m.sipko.pdf, 2015.
4. A. Cornman, G. Spellman, D. Wright. Machine Learning for Professional Tennis Match Prediction and Betting, http://cs229.stanford.edu/proj2017/final-reports/5242116.pdf, 2017.
5. `Sijo VM`, ATP Matches, 1968 to 2017. *Kaggle*, https://www.kaggle.com/sijovm/atpdata, 2017.
6. I. Reinstein. XGBoost, a Top Machine Learning Method on Kaggle, Explained. *KDnuggets*, https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html, 2017.
7. A. Collier. Making Sense of Logarithmic Loss. *datawookie*, https://datawookie.netlify.com/blog/2015/12/making-sense-of-logarithmic-loss/, 2015.
8. Isner–Mahut match at the 2010 Wimbledon Championships. *Wikipedia*, https://en.wikipedia.org/wiki/Isner–Mahut_match_at_the_2010_Wimbledon_Championships, 2018.
9. R. Sundaram. An End-to-End Guide to Understand the Math behind XGBoost. https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/, 2018.
0. T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System, https://arxiv.org/abs/1603.02754, 2016.
1. https://scikit-optimize.github.io
2. J. Snoek, H. Larochelle, R. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. https://arxiv.org/abs/1206.2944, 2012.
3. A. Jain. Complete Guide to Parameter Tuning in XGBoost (with codes in Python). *Analytics Vidhya*, hhttps://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/, 2016.