Machine Learning for Professional Tennis Match Prediction and Betting

Andre Cornman, Grant Spellman, Daniel Wright

Abstract

Our project had two main objectives. First, we wanted to use historical tennis match data to predict the outcomes of future tennis matches. Next, we wanted to use the predictions from our resulting model to beat the current betting odds. After setting up our prediction and betting models, we were able to accurately predict the outcome of 69.6% of the 2016 and 2017 tennis season, and turn a 3.3% profit per match.

1. Introduction

1.1. Motivation

Tennis is an international sport, enjoyed by fans in countries all over the world. Unsurprisingly, professional tennis players come from an equally diverse background, drawn from countries throughout North America, South America, Europe and Asia. From each of these regions players come equipped with different playing styles and specialties. In addition to this, tennis fans know that the sport is played on three unique surfaces (clay, grass, and hard courts), each lending itself to different play strategies. There are a huge number of variables that define each and every tennis match, making the sport both exciting and unpredictable. Being tennis fans ourselves, we decided to move away from our gut instincts and take a new approach to predicting the outcomes of our favorite matches.

1.2. Project Outline

The data from our project came primarily from author and sports data aggregator Jeff Sackmann [1], as well as betting data from Tennis-Data.co.uk [2]. See the dataset section for more details. Once we combined and processed this data, we tried fitting it to different models to see see which prediction model yielded the best performance for predicting match outcomes. The classification models we tried include logistic regression, SVM, random forests and neural networks. Finally, we incorporated our prediction model into a single shot decision problem to decide, for a given match, who to bet on, or whether to bet all.

1.3. Related Work

There are a number of papers related specifically to modeling and machine learning techniques for tennis betting. Barnett uses past match data to predict the probability of a player winning a single point [4]. This prediction is then extended to predict the probability of winning a match. Their approach claims a 6.8% return on investment for the 2011 WTA Grand Slams. Clarke and Dyte used a logistic regression to predict match outcomes by using the difference in the ATP rankings of players [5]. Both of these models used a single feature to predict outcome. However, Somboonphokkaphan used artificial neural network (ANN) using a number of features (including previous match outcomes, first serve percentage, etc.) [6]. This model had a 75% accuracy for predicting matches in the 2007 and 2008 Grand Slam tournaments.

2. Datasets

Tennis match data was retrieved from an open source data set available on GitHub [1]. It includes all match results from the Open Era (1968) to September of this year. More recent matches (after the year 2000) include match statistics such as the number of aces hit, break points faced, number of double faults, and more. Betting data was retrieved from [2], which has odds from various betting services from 2001 on. The data here also includes match scores. Ultimately, these two datasets had to be merged so that we could incorporate the betting odds into our prediction model along with the match results and statistics. We were able to merge about 93% of the data. The merged dataset has 46,114 matches. This was split into a training set of size 41,324 and a test set of 4,790 (roughly a 90-10 split).

In the data, the statistics, odds, etc., were labeled only for the winner and loser, so for each match we randomly assigned "Player 1" to be either the winner or loser, and "Player 2" to be the other person. We also added a label for each match as to whether Player 1 won the match. This would be the label that our model would try to predict. These random labellings were done once as part of our data pre-processing and then were held constant through the rest of the project.

2.1. Feature Engineering and Selection

The merged data set offered a large number of potential features that we could use to train our model, including player rankings, ranking points, age, height, as well as in-match statistics including aces, break points and double faults.

In addition to these features, we also computed a number of our own features to capture a player's recent performance. For each match, we calculated the average of each match statistic over the most recent 5, 10, and 20 matches. Finally, we calculated a players head to head record against their opponent, both overall and with respect to the given play surface. Altogether, we hoped that these additional features could help quantify the current state of a player's game.

All features were of the following form:

$$\text{FEATURE}_i = \text{STAT}_{i,\text{player1}} - \text{STAT}_{i,\text{player2}}$$

This means the features we trained on were all differences between certain statistics about the players (such as ranking, ranking points, head-to-head wins, etc.). This was done to achieve symmetry. We wanted a model where the labeling of the players as Player 1 or Player 2 doesn't matter. This would help us to avoid any inherent bias to/for the player randomly labeled as "Player 1" [3].

3. Methods

We decided to try a number of machine learning algorithms. We briefly summarize how each works below.

3.1. Logistic Regression

In logistic regression, we have a hypothesis of the form:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

where g is the logistic function. We assume that the binary classification labels are drawn from a distribution such that:

$$P(y = 1 \mid x; \theta) = h_\theta(x)$$
$$P(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

Given a set of labeled set of training examples, we choose θ to maximize the log-likelihood:

$$\ell(\theta) = \sum_i y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

We can maximize the log-likelihood by stochastic gradient ascent under which the update rule is the following (where α is the learning rate) and we run until the update is smaller than a certain threshold:

$$\theta \leftarrow \theta + \alpha(y^{(i)} - h_\theta(x^{(i)})x^{(i)}$$

3.2. SVM

While fitting our data to an SVM model, we tried a number of kernels with varying results. In general, a SVM model solves the problem for a data set $(x^{(i)}, y^{(i)})$ where $i = 1, 2, ...m$, given by

$$\min_{\gamma,w,b} \frac{||w||^2}{2} \text{ s.t. } \gamma^{(i)} \geq 1 \text{ for all } i = 1, 2, ...m$$

where the geometric margin $\gamma^{(i)}$ is equal to

$$\gamma^{(i)} = y^{(i)}((\frac{w}{||w||})^T x^{(i)} + \frac{b}{||w||})$$

The specifications for the SVM problem shown here are given by Stanford CS229 lecture notes, and further details and intuition for the SVM problem can be found there [7].

3.3. Neural Network

A neural network is composed of "layers." The input to each layer is either the data itself or the output from a previous layer. Each layer applies a linear transformation to the data and then an activation function, which is typically nonlinear. Mathematically, this is represented as:

$$z^{[i]} = W^{[i]} a^{[i-1]} + b^{[i]}$$
$$a^{[i]} = g(z^{[i]})$$

Here $a^{[i]}$ represents the output vector for each layer and g is the activation function. The output of the last layer is the output of the network. The input to the first layer $a^{[0]}$ is the original data, x.

The neural network needs to learn the weight arrays, $W^{[i]}$ and biases, $b^{[i]}$. This can be done by "back-propagation" which uses gradient descent (typically batch gradient descent) to update the parameters until convergence. In batch gradient descent, rather than using just a single example (stochastic gradient descent) or the entire dataset to calculate the gradient of the parameters, the gradient is computed for "batches" of data. The batch size of the gradient descent is a hyperparameter of the algorithm.

3.4. Random Forest

A random forest model is a nonparametric supervised learning model. It is a generalization of a random tree model in which several decision trees are trained.

A tree is grown by continually "splitting" the data (at each node of the tree) according to a randomly chosen subset of the features. The split is chosen as the best split for those features (meaning it does the best at separating positive from negative examples).

A random forest is made up of many trees (how many is a hyperparameter of the model) trained like this. Given a data point, the output of a random forest is the average of the outputs of each tree for that data point.

4. Experimental Results

We first discuss the results for the machine learning classification algorithms we tried.

Model	Train	5-Fold CV
Random Forest	73.5	69.7
Neural Network	81.8	65.2
(1 HL, 300 nodes, logit.)		
SVM		
→ Linear Kernel	69.8	69.9
→ RBF Kernel		51.0
→ Polynomial Degree 3		54.0*
Logistic Regression w/L1 Reg.		69.9
Logistic Regression w/L2 Reg		69.7

Table 1. Training and 5-Fold Cross Validation Accuracies for Models
**SVM with polynomial kernel was too slow to validate*

4.1. Logistic Regression

Logistic regression had good accuracy in 5 fold cross validation, however training the model was very slow to run. We were able to perform tuning of the regularization strength for L1 and L2 regularization, which both resulted in low variance. We think that adding polynomial terms might help, but we did not have the computational resources to be able to carry out these experiments in a reasonable timeframe.

4.2. SVM

We tried three different SVM kernels for our model: RBF, polynomial, and linear. A general discussion of each follows below. For the respective training model accuracies, see Table 1.

1. **RBF:** Also known as the Gaussian kernel, the feature mapping is of the form

$$K(x, z) = \exp(-\frac{||x-z||_2^2}{2\sigma^2})$$

   From the form of the kernel, we can see that it gives an estimate of roughly how far apart x and z are. When trying this kernel, the model was slow to train and prone to over fitting, even as we tried various hyperparameters like the regularization strength. Because of the size of our training set, and the non-linear nature of the kernel, this model proved to be impractical.

2. **Polynomial:** The polynomial kernel has the form of

$$K(x, z) = (x^T z)^d$$

   for a d degree expansion of the features. We tried a second degree polynomial fit. Similar to the RBF kernel, the polynomial kernel was very slow to train and prone to over fitting. This is likely due to our large feature space, where the polynomial kernel would create a larger number of higher order term and cross term features.

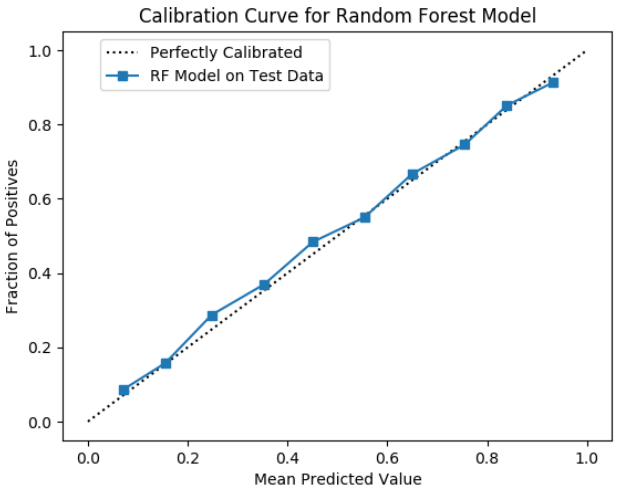
3. **Linear:** The linear term only uses a linear weighting of the first order of the feature set. The linear kernel proved to be the most appropriate for our data. It was faster to train and allowed us to more finely tune the hyper-parameters, further increasing prediction accuracy. In addition to this, we were able to experiment with the ensemble library in sklearn to better estimate the linear parameters. Ultimately, this kernel gave us the best performance under the SVM model.

4.3. Random Forests

The random forest model had the advantage of being much faster to train. This allowed us to more easily iterate with the model. For example, it was easier to tune the hyperparameters because training the model several times for cross validation was not too computationally expensive.

One of the hyperparamters we tuned was the minimum samples per leaf. When this was set to one, i.e., the leafs could be as small as a single match, the training accuracy was 100% while the cross-validation accuracy was only around 69%. This represents poor generalization and extreme overfitting to the training set. We tuned this parameter to be 50, which reduced the training accuracy to 73% and increased the cross-validation accuracy modestly, improving generalization.

Below is the calibration curve for the random forest model. For 10 "buckets" it plots the mean predicted value against the fraction of "positives," which in this case means player 1 winning. The curve shows that the random forest model is extremely well calibrated. This is important for our betting strategy.



One can extract "feature importances" from a random forest model. These are computed as the mean decrease in node "impurity" for that feature across all trees, where node impurity is a measure of how separated the data is (by positive/negative label). Below are the estimated feature importances from our trained random forest model.

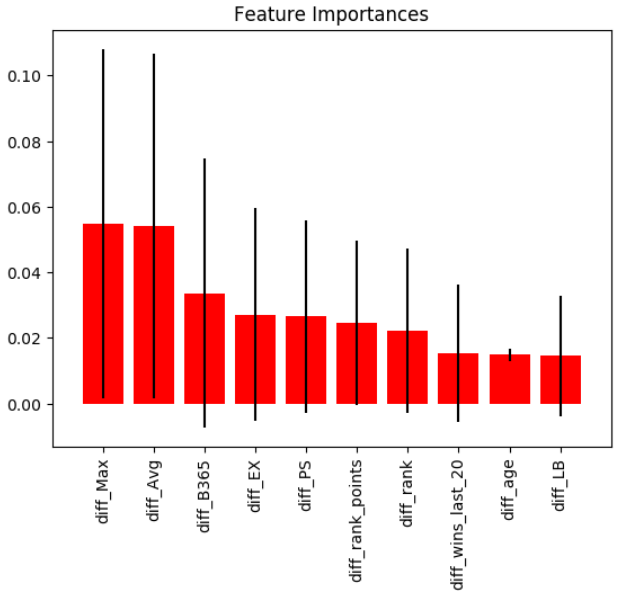


Figure 1. 10 most important features in the random forest model. The black lines indicate the standard deviation of the importance estimate (estimated as the standard deviation of the feature importance across trees). The feature importances are normalized to sum to one over all features.

The first five of these features refer to betting data: "Max" refers to the best betting odds, "Avg" refers to the average betting odds across all quoted odds, and "B365", "EX," and "PS" refer to the odds of specific betting services. The most important non-betting related features are the difference in rank, ranking points, wins in the last 20 matches, and, interestingly age.

4.4. Neural Network

We thought that a neural network might be useful in discovering non-linearities. However, the accuracy of the neural network was not as high as the other models. It is possible that this would have improved if we continued to try to tune the hyperparameters of the model (such as the number of hidden layers, the nodes per hidden layers, the activation function, etc.) but the time it took to train was prohibitive to do on our own computers.

5. Betting Model Results

We developed a simple betting model that uses the output of our random forest model and the odds data to bet on the player that maximizes the expected returns. The odds data is represented as two numbers per match (greater than 1). For example, if you bet correctly on player 1, who's odds are 1.5, then for each dollar you bet, you win 0.5 dollars. However, if you bet incorrectly, you lose the initial bet amount.

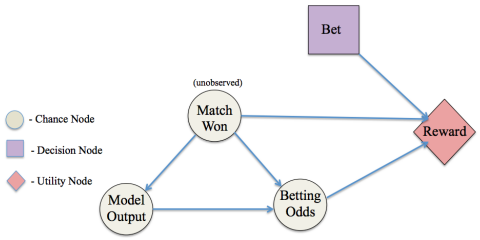


Figure 2. Decision network for single-shot betting strategy.

We modeled our betting strategy as a single-shot decision problem for each match, where we aim to maximize the expected earnings. This was a better approach than, say, a reinforcement learning approach because there is no sense of underlying system dynamics in this setting: that is, there is no real reason for one match to "transition" to another. Instead, each match should be thought of as an independent decision problem.

Possible actions include betting 1 dollar on player 1, betting 1 dollar on player 2, and not betting. The strategy chooses the action with the highest expected reward:

$$b^* = \underset{b \in \{0,+1,-1\}}{\arg\max} \; E[U(win, odds, b)]$$

On test set data, the betting strategy earns an average of 3.3% per match. Interestingly, the strategy takes the no bet action 29% of the time, where the expected utility of betting on either player is negative. We also find it interesting that the betting strategy has streaks of winning, and streaks of

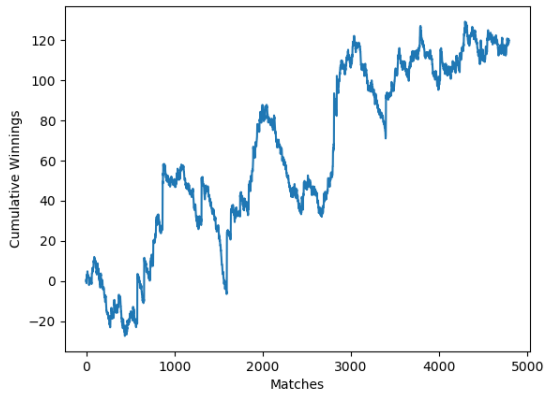


Figure 3. Cumulative winnings on the test set for simple betting strategy.

losing, however we do not have a good explanation for this result.

We computed the Sharpe ratio for our strategy over this period. The Sharpe ratio is a commonly used evaluation metric for strategies in financial markets and is defined as the mean return over the standard deviation of the returns. The Sharpe ratio for our strategy was 0.02, which would be considered very poor in the financial industry. We think that this can be explained by the substantial risk of losing all of one's money in a given bet, which causes the standard deviation for the return across all bets to be quite high. People who gamble on sports, one might hypothesize, are attracted to risk, rather than averse to it.

6. Error Analysis

Below are tables, similar to confusion matrices, for our random forest model on the test set. We see that most of the time, the model is predicting the higher ranked player and the favored player. (Here favored means the player has more favorable odds.) In fact, on the test set, the model is predicting worse than 50% when it predicts the disfavored player.

Predicted	Total #	Correct	Pct.
Lower Ranked Player	814	486	59.7
Higher Ranked Player	3976	2850	71.7

Predicted	Total #	Correct	Pct.
Favored Player	4669	3279	70.2
Disfavored Player	121	57	47.1

This indicates that our model is not capturing insights about when/why a lower-ranked or disfavored player would win a match (and instead mostly relies on these features for its predictiveness).

This aspect of the model could potentially be improved by a dataset with richer features, such as injury information about each player, more fine-grained statistics, weather, coaching, strategy, etc.

7. Future Work

First and foremost, we are excited to try out our prediction model on upcoming tournaments in the 2018 season. Perhaps we can give ourselves an edge on the betting websites.

As for our prediction model, it would be very interesting to further explore and properly validate the models that demanded more computational power than we had available to do our project. This would be relevant primarily for the non-linear SVM models, logistic regression model, and neural network model. Because of the size of our dataset (approximately 45,000 rows with upwards of 80 features), it was difficult to train and tune these models locally on our machines. In particular, we were able to train these models on our dataset, however, the difficulty came in when iterating over the hyper-parameters of these models. Given more computational resources, we could optimize these hyper-parameters. With regards to our betting model, one goal of ours is to add more flexibility to our betting decision model. For instance, our current model only makes $1 bets, or no bets at all. It would be interesting to scale our bets to larger or smaller amounts given the confidence reported by our model. For example, we could adjust the bet amount based on the difference between the model output probability and the odds data probability of a player winning.

8. Contributions

8.1. Andre

I worked on setting up and implementing the betting strategy.

8.2. Danny

I worked on merging the two datasets and on feature engineering and building/testing models (logistic regression, neural network, random forest).

8.3. Grant

I worked on setting up and testing all of the SVM models we tried to fit to our merged data set. I cross validated these models as well.

References

[1] https://github.com/JeffSackmann/tennis_atp , *Jeff Sackmann*

[2] http://www.tennis-data.co.uk/alldata.php

[3] https://www.doc.ic.ac.uk/teaching/distinguished-projects/2015/m.sipko.pdf , Section 3.1.2 *Michal Sipko and Dr. William Knottenbelt of the Imperial College London*

[4] T. Barnett and S. R. Clarke. Combining player statistics to predict outcomes of tennis matches. *IMA Journal of Management Mathematics*, 16:113120, 2005.

[5] S. R. Clarke and D. Dyte. Using official ratings to simulate major tennis tournaments. *International Transactions in Operational Research*, 7(6):585594, 2000.

[6] A. Somboonphokkaphan, S. Phimoltares, and C. Lursinsap. Tennis Winner Prediction based on Time-Series History with Neural Modeling.*IMECS 2009: International Multi-Conference of Engineers* and Computer Scientists, Vols I and II, I:127132, 2009.

[7] http://cs229.stanford.edu/notes/cs229-notes3.pdf , *Stanford University, CS229 Lecture Notes, Andrew Ng*