CS230

DeepTennis: Mid-Match Tennis Predictions CS230-Fall 2019

Sven Lerner
ICME
Stanford University
svenl@stanford.edu

Dipika Badri
Department of Electrical Engineering
Stanford University
dipika98@stanford.edu

Kevin Monogue
ICME
Stanford University
kmonogue@stanford.edu

Abstract

The field of sports analytics has experienced major growth in the past decade, including the area of match prediction. In comparison to pre-match predictions, the task of producing mid-match live predictions remains an area open to emergent study. We investigate the use of a recurrent neural network utilizing the Long Short-Term Memory (LSTM) model infrastructure to compute 'live' win probabilities for tennis matches from mid-game data. Using a detailed point-by-point dataset [1] for the four Grand Slam tennis tournaments, our model is intended to learn deep relationships amongst the sequential data and to classify the probability of winning the match for each player after any given point. In combination with traditional pre-match prediction priors, our model obtains an average accuracy of 79.5% across all given points, roughly 3 percentage points higher than traditional statistical point-by-point models [2].

1 Introduction

The past decade has seen an incredible emergence of analytical methods being applied to the world of sports. Simultaneously, the world of sports betting has expanded rapidly. The sports betting market in the US is estimated to bring in hundreds of billions of dollars in revenue [3]. Most bookmakers now provide the opportunity for live betting, in which odds are updated throughout the game, allowing users to place a bet based on the progression of the match. One match between Roger Federer and Rafael Nadal saw 50 million Euro exchanged mid-match on the betting exchange Betfair [4].

A natural consequence of the emergence of richer sports data and increased betting opportunities is the production of match prediction models. Tennis provides a great microcosm for this analysis due to its discrete scoring nature. Most published research on tennis prediction has been focused on using pre-match data. Furthermore, existing studies on mid-match prediction models largely act by updating input parameters, rather than establishing temporal relationships amongst point-by-point data. We propose that an RNN should provide benefits beyond traditional mid-match prediction methods by better incorporating the temporal nature of point-by-point data and extracting more complex features from the data, such as fatigue or momentum.

The input to our algorithm is a series of feature vectors representing each point in a tennis match. Each point contains data such as the current game score, the speed of the shot, etc., as well as pre-match

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

win probabilities to act as an anchor. We then train an LSTM on the data, feeding the points of a given match as the sequential data. The model outputs the probability of "player 2" winning winning the match at each point during the match.

2 Related work

As previously mentioned, tennis prediction has been host to a variety of methodologies. A paper by Klaasen and Magnus [6] provides a seminal approach to this problem using a hierarchical Markov model, often replicated or modified in future work. The model takes as an input the probability that each player would win a single point if serving, and then constructs a tree of match outcomes using this probability to arrive at the probability each player would win the match. Other efforts have expanded upon this method, such as updating the single-point probabilities over the course of a match [9] [13].

Kovalchik (2016) [8] provides an analysis of published results using this method, along with two others: regression and paired comparison. Regression models attempt to predict a win probability using some set of input features, such as the model used by Clark and Dyte (2000) [10]. Comparison based models use previous match results to update the "rating" of a player, and use these ratings to predict winners - a notable example is the ELO ranking method. Prediction results ranged from 59% to 72%, with comparison based models fairing the best. Gollub (2017) provides another strong review of these models, as well as several improved implementations. The paper tests modified points-based models, a logistic regression model, and modified ELO models on pre-match as well as mid-match data. Accuracy ranging from 63% to 69% were found for pre-match models, and from 71% to 76.5% for mid-match models.

Approaches using modern machine learning methods have been explored for pre-match prediction. A former CS 229 project by Cornman, Spellman, and Wright (2017) [11] tested a variety of these models, including support vector machine, random forest, and even a neural network, finding accuracy from 65% to 70%. Sipko (2015) explores basic logistic regression and neural network modeling for pre-match prediction, finding improvements over betting market predictions.

3 Dataset and Features

We used the point-by-point dataset for the tennis Grand Slams (Wimbledon, US Open, French Open, and Australian Open) from 2011 - 2019 provided by Jeff Sackman [1]. The data contains feature vectors for each point in a match, including player distance run, whether a fault occurred, type of point won, and more. We were able to extract roughly 41 of these features, either from existing for the entire data set or by replacing blank values by the global mean of the field. We also created new fields communicating how many additional games or sets were needed to be won by each player in order to win the match from that point. A few sample features can be seen in Figure 1.

Figure 1: Sample of Single Point Feature Vector

match_id	2019-usopen-1101
player1	Novak Djokovic
player2	Roberto Carballes Baena
winner	1
SetNo	2
P2Winner	0
P1DoubleFault	0
P2DoubleFault	0
P1UnfErr	0
P2UnfErr	0
P1NetPoint	0
P2NetPoint	0
P1NetPointWon	0
P2NetPointWon	0
P1BreakPoint	1
P2BreakPoint	0
P1BreakPointWon	1
P2BreakPointWon	0
Speed_MPH	116
RallyCount	4
P1DistanceRun	14.894
P2DistanceRun	17.513
p1_sets_to_win	2
p2_sets_to_win	3
p1_games_to_win	7
p2_games_to_win	17

We also used the pre-match predictions produced by the 13 models used in Gollub [15]. We matched this data with the Sackman data and appended the pre-match predictions to each point vector. Our intention was to capitalize on the variety of methods Gollub provides to include a robust pre-match prediction as an anchor point for our mid-match predictions.

We split our data by year with 3373 training examples (matches), 693 dev set examples, and 799 test examples. The dev set is all 2017 matches and the test set is all 2014 matches - the same year tested on by the prediction models in Gollub, the best comparison metric available.

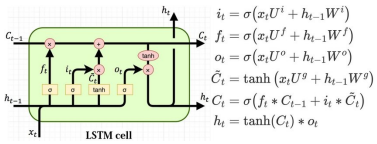
4 Methods

As previously mentioned, the core of our approach was to apply a sequence model to tennis match point-by-point data to predict the win probability of each player after every point in the match. We experimented with both GRU and LSTM models, however the LSTM models consistently outperformed the GRU based models, so we present their results in this report.

Long short-term memory (LSTM) networks networks are a special class of Recurrent Neural Networks (RNN). RNNs, unlike regular neural networks, process temporally linked data points within training examples. Long Short-Term Memory(LSTM) networks are utilize a structure of gates, or filters, to control the flow of information between points in time (data points).

The common architecture of an LSTM is comprised of building blocks called units (Figure 2). Each unit has two pipelines of inputs/outputs (C_t and h_t in the above graph). C_{t-1} is responsible for propagating previous units output forward, while h_{t-1} propagates previous units hidden states. $ht-1$ is passed through an input gate (i_t) and a forget gate (f_t) to control which information is passed on and combined with the current input data x_t to produce a new proposed output $\tilde{C}_t$. C_t is then produced by using the forget gate and input gate to combine the previous output C_{t-1} with the new proposed output $\tilde{C}_t$. Finally, an output gate o_t is used to manage the flow of information in the hidden state to the next unit h_t, which is produced by a tanh activation of C_t. C_t and h_t is passed horizontally to the next temporal unit, while h_t is also passed to the next layer of a multi-layer LSTM (Figure 3). Alternatively, if this is the final layer, an activation output is used at each point in time t to produce $\hat{y_t}$ - in our case the probability of player 2 winning. Note, U and W each represent sets of trainable parameters for each component of the model.

Figure 2: Single LSTM Unit [16]



We optimized our model by minimizing a loss function that compares $\hat{y_t}$ with the true value y_t. We tried several approaches to our loss function. An interesting aspect of our problem is that we are trying to predict the win probability at a given point in time, however there is no real 'ground truth' for that value. As a proxy, we used the eventual winner of the match, inducing our model to optimize towards predicting the winner with as much certainty as possible. We experimented with treating our problem as a regression problem (predicting the win probability of a player) and as a classification problem (predicting which player will win), concluding that the former fit our problem better and implementing a binary cross entropy loss function. We also considered weighting the loss for each point based on its temporal location in the data example. After experimenting with different weighting mechanisms, we found a linear weighting scheme to produce the strongest results. This scheme reduces punishment for early data points while placing a heavy emphasis on predicting the match correctly by the late stages. Our loss function following these guidelines, with k as the number of points in the match, was:

$$L = -\sum_{i=1}^{k} (\frac{i}{k}) * (y_i * log(\hat{y_i}) + (1 - y_i) * log(1 - \hat{y_i})) \tag{1}$$

Another interesting approach to the weighting we attempted was allowing the model to predict it's confidence in each point, and base the weighting on this. Now the output was 2 dimensional - a win probability and a confidence value - and the loss was formulated as such with c_i as the confidence value:

$$L = \lambda||1 - c||_2 - \sum_{i=1}^{k} c_i * (y_i * log(\hat{y_i}) + (1 - y_i) * log(1 - \hat{y_i})) \tag{2}$$

We found that as the model started training, the c output started off as close to a vector of zeros, however as the model trained, it converged to a vector of ones. This formulation of the loss achieved within 1 percent of the accuracy of the simpler loss function above, but proved less effective in the end.

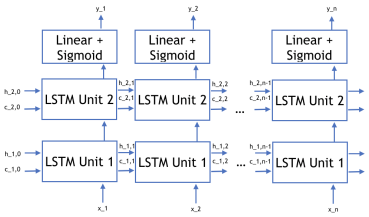
5 Experiments/Results/Discussion

The primary evaluation metric used was total accuracy across all points. Precision and recall do not make intuitive sense for our problem (as our true goal was to predict probabilities), thus instead we also monitored calibration. The optimization procedure is described below, followed by results.

We explored several hyperparameter choices for our model. We performed a brief search for our learning rate, resulting in $\alpha = 0.003$ using an ADAM optimizer and involving a 10% cut at epoch 20 and 30. We also used a batch-size of 1, as training was efficient and this simplified computing loss over variable size sequences.

Other hyperparameters we explored were the model depth (number of stacked units) and width (dimension of hidden state per unit). We attained significant gains on the order of 5 percent by stacking two modules together over one, however if we increased the depth beyond two we actually noticed a slight degradation in evaluation performance despite a further reduction in training performance (introduced variance). We were able to mitigate this increased variance somewhat by using dropout between our LSTM modules, however the depth of two remained superior. In terms of width, we achieved best results with a hidden state of size 50, which roughly follows the rule of thumb from Hagan (2014) [18]:

Figure 3: Multi-Layer LSTM Network [17]



$$N_h = \frac{N_s}{\alpha * (N_i + N_o)} \tag{3}$$

t Where N_h is the number of hidden neurons, N_s is the number of samples, N_i is the dimension of the input, N_o is the dimension of the output, andα is a scaling factor. For our models with increased depth, a decreasing the width per layer in the stack did not show performance improvements.

We utilized dropout and early stopping to prevent overfitting with larger models. We are confident that we have not overfit to our training dataset as our accuracy matches tightly across our train, eval, and test sets (79%).

We present our test results and comparisons in Table 1. "K-M Logit Elo" is a point-based model [6] used by Gollub [2] to predict results using point-by-point data. "Logistic Regression" is a simple classifier using the 13 Gollub models as an input feature vector. Our model produces an accuracy of 79.5% on the test set, a 3% improvement over the the best mid-match Gollub model and a 6% improvement over the basic pre-match predictor and the best pre-match results reviewed by Kovalchik [8]. Results for our model are also provided for data points after each set.

Table 1: Model Results (%)

Model	Net Accuracy	Set 1	Set 2	Set 3	Set 4	Set 5
DeepTennis	79.5	84	85	93	85	90
K-M Logit Elo (Gollub)	76.5	N/a	N/a	N/a	N/a	N/a
Logistic Regression	73.4	N/a	N/a	N/a	N/a	N/a

Table 2: Accuracy by Match Progression (%)

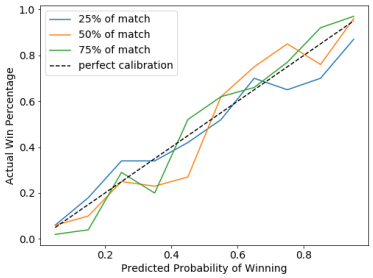
Percentage of Match Played	25	50	75	100
Accuracy	73.2	84.9	89.2	99

Notably, our models performance is significantly higher after Set 4 at 85% than at the beginning of the match or overall. By definition, the score of the match is tied after Set 4, thus indicating our model has learned important features of the match beyond simply the score. It is not surprising our

accuracy peaks after Set 3, as this includes many matches that end 3-0. Table 2 shows our model's predictive power increases as the match progresses.

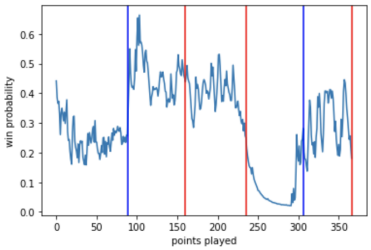
Finally, we investigated the calibration of our model as a replacement for precision or recall numbers. Figure 4 plots our models predicted probability (in buckets of size 10%) of winning against the actual win-rate of matches in each bucket. The model proved relatively well calibrated.

Figure 4: Calibration

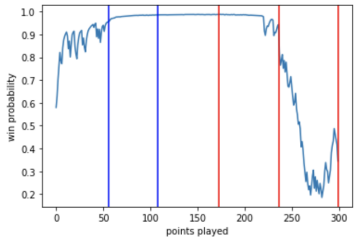


The individual match results from the network are interesting to examine qualitatively. Referencing Figure 5 (blue vertical lines are set wins for player 2, red are for player 1):

(a) Djokovic vs. Federer, Winbledon 2014



(b) Thiem vs. Gulbis, US Open 2014



(a) Novak Djokovic defeated Roger Federer in 5 sets in the men's final of the 2014 Wimbledon tournament. Novak Djokovic entered the match as the favorite, and we can see that the model has learned to incorporate the prematch priors at the beginning of the match. After Federer wins the first set, his win probability rises sharply. Following two set wins by Djokovic, the match seems all but decided, until Federer wins 5 games in a row to push the match to a back and forth 5th set. Finally Djokovic pulls of the win.

(b) This match from the 2014 US Open is an example where our model performed very poorly in terms of training loss. We see that the winner of the match was Dominic Thiem, (player 1), however for nearly the entire match, our model predicted that his opponent Ernests Gulbis would prevail. In fact, this was a highly publicized upset by Thiem, a 20 year old playing in his first US Open coming back from an early two set deficit [14].

Regardless of the incredibly surprising storylines from these matches, our model appears to well capture notions of heavy favorites (Gulbis' consistently high win probability) or momentum (Djokovic winning two sets in a row and nearly winning in set 4).

6 Conclusion/Future Work

The implementation of a two-layer, 50 hidden node LSTM model appears to have improved performance over existing methodologies for mid-match tennis prediction. Our model exhibits characteristics that demonstrate learning connections between tennis points (momentum, consistency of pre-match information, accuracy after Set 4) and better incorporates the relationships of point-by-point data than traditional statistical models.

We believe our model could be further improved with the incorporation of more data or more detailed features. For example, more abstract pre-match features (beyond just win probability) may provide information that interacts uniquely with future point data, or visual data for each point could be useful. Coupled with more detailed data, greater complexities in the model architecture, particularly with regards to width or depth, may improve performance more than they did during our experimentation.

Our project would also be improved with greater connections to real world applications. Comparisons or feature augmentation with live betting market data would be useful for analysis, and study regarding the practical collection of live data is needed for proper implementation of our model.

7 Contributions

All team members contributed equally to the project. Specifically Sven focused on model design and training, Kevin focused on data processing/feature engineering and research, and Dipika focused on model architecture.

8 Acknowledgements

Special thanks to Hao Sheng for help and guidance along the way. We also extend our gratitude to Jeff Sackman for providing the dataset and to Jacob Gollub for providing a comprehensive resource on tennis prediction which we leveraged greatly.

9 Codebase

https://github.com/sven-lerner/DeepTennis

References

[1] Courtesy Jeff Sackman: https://github.com/JeffSackmann/tennis_slam_pointbypoint

[2] Gollub, Jacob. Producing Win Probabilities for Professional Tennis Matches from any Score. Diss. 2019.

[3] https://www.legalsportsbetting.com/how-much-money-do-americans-bet-on-sports/

[4] Huang, Xinzhuo, William Knottenbelt, and Jeremy Bradley. "Inferring tennis match progress from in-play betting odds." Final year project), Imperial College London, South Kensington Campus, London, SW7 2AZ (2011).

[5] https://ftw.usatoday.com/2017/02/super-bowl-espn-win-probability-atlanta-falcons-new-england-patriots-stats-tom-brady

[6] Klaassen, Franc JGM, and Jan R. Magnus. "Forecasting the winner of a tennis match." European Journal of Operational Research 148.2 (2003): 257-267.

[7] Easton, Stephen, and Katherine Uylangco. "Forecasting outcomes in tennis matches using within-match betting markets." International Journal of Forecasting 26.3 (2010): 564-575.

[8] Kovalchik, Stephanie Ann. "Searching for the GOAT of tennis win prediction." Journal of Quantitative Analysis in Sports 12.3 (2016): 127-138.

[9] Madurska, Agnieszka M. "A set-by-set analysis method for predicting the outcome of professional singles tennis matches." Imperial College London, Department of Computing, Tech. Rep. (2012).

[10] Clarke, Stephen R., and David Dyte. "Using official ratings to simulate major tennis tournaments." International transactions in operational research 7.6 (2000): 585-594.

[11] Cornman, Andre, Grant Spellman, and Daniel Wright. "Machine Learning for Professional Tennis Match Prediction and Betting." (2017).

[12] Sipko, Michal, and William Knottenbelt. "Machine learning for the prediction of professional tennis matches." MEng computing-final year project, Imperial College London (2015).

[13] Bevc, Martin. "Predicting the Outcome of Tennis Matches From Point-by-Point Data." (2015).

[14] https://www.nytimes.com/2014/08/30/sports/tennis/dominic-thiem-tops-ernests-gulbis-to-reach-us-opens-third-round.html

[15] https://github.com/jgollub1/tennis_match_prediction

[16] Varsamopoulos, Savvas, Koen Bertels, and Carmen G. Almudever. "Designing neural network based decoders for surface codes." arXiv preprint arXiv:1811.12456 (2018).

[17] Nicholas, Lee, et al. "Study of long short-term memory in flow-based network intrusion detection system." Journal of Intelligent & Fuzzy Systems Preprint (2018): 1-11.

[18] Demuth, Howard B., et al. Neural network design. Martin Hagan, 2014.

[19] Imported python modules pytorch, pandas, numpy, ipython, jupyter, torchvision, matplotlib.