Predicting T20 Cricket Matches With a Ball Simulation Model

towardsdatascience.com • Andrew Kuo

'Tis the season to be watching lots of cricket and once again I have obliged. Yes, every summer I plonk myself down on the lounge and observe approximately seven million hours of my favourite sport, but last year I decided to spend some of the time between overs working on a predictive model. Modelling sports is a passion of mine and was actually the gateway drug that initially got me interested in data science. In the past, I have trained models to predict the outcomes of rugby matches, tennis matches and horse races and even won a lucrative first prize in a data science competition to predict the Australian Open.

My primary goal with this project was not to build the most accurate algorithm ever (nice as that would be) but rather to experiment with a probabilistic, bottom-up approach to modelling. Instead of trying to predict match results based on historical data, I trained a neural-network model to predict the outcome of individual balls and then built a custom Monte Carlo simulation engine to generate tens of thousands of possible games.

Image by Nathan Saad.

Data

For this project, I used data on 677291 balls bowled in 3651 T20 matches involving 2255 players. The games took place between 2003 and 2020 and come from the following 7 leagues:

- Indian Premier League
- Big Bash League (Australia)
- Caribbean Premier League
- T20 Blast (England)
- Mzansi Super League (South Africa)
- Pakistan Super League
- Bangladesh Premier League

Unfortunately, I can't share the data source for this project since I leveraged my personal sports database but this kind of data is quite easy to find publicly. Below are some samples from my starting tables to give an idea of the kind of data I was working with (not all columns are shown).

Match

date	venue_name	hm_team_name	aw_team_name	is_reduced	hm_order	hm_winner	hm_score	aw_score
2020-08-30	Headingley, Leeds	Yorkshire	Derbyshire	0	1	true	220/5	121/9 (20 ov, target 221)
2016-11-21	Zahur Ahmed Chowdhury Stadium, Chattogram	Dhaka Dynamites	Rajshahi Kings	0	1	false	182/4	184/7 (19.5/20 ov, target 183)
2018-01-12	Docklands Stadium, Melbourne	Melbourne Renegades	Melbourne Stars	0	2	false	144/9 (20 ov, target 168)	167/4
2018-09-09	Providence Stadium, Guyana	Guyana Amazon Warriors	Trinbago Knight Riders	0	2	true	158/4 (14.1/20 ov, target 155)	154/7
2020-09-04	The Cooper Associates County Ground, Taunton	Somerset	Birmingham Bears	1	2	false	120/7 (12/12 ov, target 125)	107/4 (12/12 ov)

Image by Andrew Kuo.

Player

name	full_name	date_of_birth	bat_style	bwl_style
Alviro Petersen	Alviro Nathan Petersen	1980-11-25	Right-hand bat	Right-arm offbreak
Saif Badar	Saif Badar	1998-07-03	Right-hand bat	Legbreak googly
Jack Davies	Jack Leo Benjamin Davies	2000-03-30	Left-hand bat	None
Steven Finn	Steven Thomas Finn	1989-04-04	Right-hand bat	Right-arm fast-medium
Imran Khalid	Imran Khalid	1982-12-29	Left-hand bat	Slow left-arm orthodox

Image by Andrew Kuo.

Player Match Statistics

team_name	bat_position	player_name	balls_faced	runs	fours	sixes	minutes	outs	balls_bowled	bowled	catches	runs_conceded	no_balls	wides
Sunrisers Hyderabad	1	David Warner	25	50	10	0	29	1	0	0	0	0	0	0
Sunrisers Hyderabad	2	Jonny Bairstow	44	61	3	3	74	0	0	0	1	0	0	0
Sunrisers Hyderabad	3	Kane Williamson	5	3	0	0	7	1	0	0	0	0	0	0
Sunrisers Hyderabad	4	Vijay Shankar	11	7	0	0	21	1	12	1	1	11	0	0
Sunrisers Hyderabad	5	Deepak Hooda	16	13	1	0	20	1	0	0	0	0	0	0

Image by Andrew Kuo.

Commentary

innings	ball	over	runs	text	wide	noball
1	1	9	1	Nadeem to du Plessis, 1 run	0	0
1	2	9	1	Nadeem to Watson, 1 run	0	0
1	3	9	6	Nadeem to du Plessis, SIX runs	0	0
1	4	9	1	Nadeem to du Plessis, 1 run	0	0
1	5	9	0	Nadeem to Watson, OUT	0	0

Image by Andrew Kuo.

Approach

Rationale

For this project, I decided to use a bottom-up approach by training a model to predict the outcome of each ball bowled (rather than the overall match result). My reasons were as follows:

- Head to head battles are a key part of cricket and this methodology allowed me to capture the matchups between individual batsmen and bowlers.
- Cricket matches are often decided by a few pivotal moments and the outcome of a single ball can have a huge impact on a result. The difference between a good or bad ball can be a matter of centimetres (e.g. a catch being taken on the boundary vs clearing the rope for six) but by simulating a match thousands of times we can approach these events probabilistically.
- It allows us to answer more complex questions (e.g. who is likely to take the most wickets in a match). A traditional top-down model can only make predictions about the specific task it was trained on (i.e. who will win a given match).
- We can get conditional probabilities of results e.g. the probability of the Chennai Super Kings winning, given they batted first and scored 167 runs OR the probability of the Brisbane Heat winning if Chris Lynn scores less than 20 runs.

Ball Prediction Model

At the heart of this project is my ball prediction model which was trained on the ~700k balls in my dataset. This was a multi-class classification problem and for simplicity, I limited the classes (possible outcomes of a ball) to 8 options (0, 1, 2, 3, 4, 6, Wicket, Wide). The inputs to the model were:

- Match State (innings, over, number of runs, number of wickets, first innings score when relevant)
- Bowler Statistics (historical distribution of ball results across all balls that bowler has bowled)
- Batter Statistics (historical distribution of ball results across all balls that batsman has faced)

The output from the model was the predicted probability of each of the 8 possible ball results. In the simulation stage, we run the inputs for every ball through the model and then sample from this predicted probability distribution.

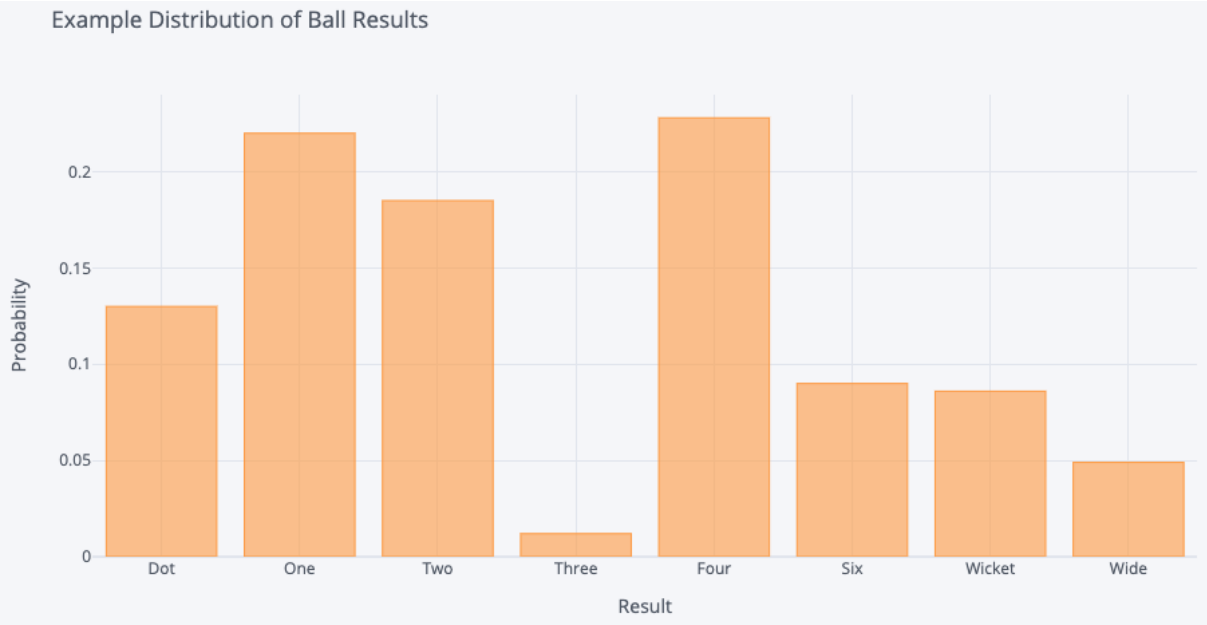

Image by Andrew Kuo.

I won't go into too much detail about model choice and hyperparameter optimisation here since it is not the focus of this article but I have included the Keras summary of my final model below. In short, I trained a feed-forward neural network model with 2 dense layers of 50 nodes with a ReLU activation function on each. I also applied Batch Normalisation and Dropout.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 50)                1150
_________________________________________________________________
batch_normalization_8 (Batch (None, 50)                200
_________________________________________________________________
re_lu_8 (ReLU)               (None, 50)                0
_________________________________________________________________
dropout_8 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_13 (Dense)             (None, 50)                2550
_________________________________________________________________
batch_normalization_9 (Batch (None, 50)                200
_________________________________________________________________
re_lu_9 (ReLU)               (None, 50)                0
```

```
_________________________________________________________________
dropout_9 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_14 (Dense)             (None, 8)                 408
=================================================================
Total params: 4,508
Trainable params: 4,308
Non-trainable params: 200
_________________________________________________________________
```

Simulation Engine

The next step was to build a simulation engine in Python that could be used to generate full matches using the ball prediction model. At a high level the steps to generate a simulation are:

1. Create two lists of player IDs (one for each team).
2. Instantiate two Team objects (defined below) using the team lists from Step 1. Also, pass relevant historical batting and bowling statistics for each team to this class.
3. Instantiate a MatchSimulator object (defined below) using the two Team objects, the Ball Prediction model and venue information.
4. Run the sim_match module of the MatchSimulator object created in Step 3. The simulation steps are given below.

5. Return the simulated match and player statistics.

Simulation Steps

1. Each match begins with a simulated coin toss and the winning team chooses to bat or bowl.
2. Simulate each ball by running the Ball Prediction model with the current match state and relevant bowler and batter stats as input. Take a random sample from the distribution output by the model and update the MatchSimulator and Team states as required (e.g. if the ball result is "3" then add 3 runs to the on-strike batter's score and his team's score, add 3 runs to the bowler's stats, rotate the strike to the other batter and increment the ball counter).
3. Repeat Step 2 until the innings is complete.
4. Repeat for the second innings.

This is not a tutorial but I have included some of my code below to illustrate the mechanics of this simulation engine.

Team Class

Match Simulation Class

I also included functionality to run the simulation engine in "verbose" mode which outputs a running commentary of a match's progress. An example of running a single simulation in this mode is shown below.

Running A Simulation

Example Output

```
The Brisbane Heat have won the toss!
Sydney Sixers will bat first. Brisbane Heat to bowl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
1ST INNINGS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
MJ Swepson to bowl. 0/0 after 0.
    4 to JL Denly!
JK Lalor to bowl. 0/4 after 1.
    4 to J Avendano!
    4 to J Avendano!
Mujeeb Ur Rahman to bowl. 0/12 after 2.
    4 to JL Denly!
    6 to JL Denly!
JL Pattinson to bowl. 0/23 after 3.
    WICKET! JL Denly out, bowled JL Pattinson.
    WICKET! MC Henriques out, bowled JL Pattinson.
    6 to DP Hughes!
BCJ Cutting to bowl. 2/31 after 4.
    WICKET! J Avendano out, bowled BCJ Cutting.
MJ Swepson to bowl. 3/34 after 5.
JL Pattinson to bowl. 3/36 after 6.
Mujeeb Ur Rahman to bowl. 3/38 after 7.
    4 to JC Silk!
BCJ Cutting to bowl. 3/46 after 8.
Mujeeb Ur Rahman to bowl. 3/52 after 9.
    4 to JC Silk!
BCJ Cutting to bowl. 3/60 after 10.
    WICKET! JC Silk out, bowled BCJ Cutting.
JK Lalor to bowl. 4/63 after 11.
MJ Swepson to bowl. 4/68 after 12.
JK Lalor to bowl. 4/74 after 13.
    4 to DP Hughes!
JL Pattinson to bowl. 4/82 after 14.
    6 to JR Philippe!
    6 to JR Philippe!
MJ Swepson to bowl. 4/96 after 15.
    4 to JR Philippe!
JK Lalor to bowl. 4/106 after 16.
    4 to JR Philippe!
MJ Swepson to bowl. 4/114 after 17.
    WICKET! JR Philippe out, bowled MJ Swepson.
BCJ Cutting to bowl. 5/118 after 18.
    6 to DP Hughes!
    4 to TK Curran!
Mujeeb Ur Rahman to bowl. 5/132 after 19.
    6 to TK Curran!

Final Score: 5/144 off 20.
The Brisbane Heat are going in to bat. Sydney Sixers to bowl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2ND INNINGS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
BJ Dwarshuis to bowl. 0/0 after 0.
    4 to M Bryant!
    4 to M Bryant!
TK Curran to bowl. 0/8 after 1.
BAD Manenti to bowl. 0/9 after 2.
    WICKET! M Bryant out, bowled BAD Manenti.
```

```
BJ Dwarshuis to bowl. 1/13 after 3.
    4 to SD Heazlett!
    4 to SD Heazlett!
MC Henriques to bowl. 1/25 after 4.
    WICKET! SD Heazlett out, bowled MC Henriques.
    4 to BB McCullum!
    4 to BB McCullum!
    WICKET! BB McCullum out, bowled MC Henriques.
SA Abbott to bowl. 3/34 after 5.
    4 to JA Burns!
BAD Manenti to bowl. 3/41 after 6.
    4 to JA Burns!
TK Curran to bowl. 3/46 after 7.
    6 to JA Burns!
BJ Dwarshuis to bowl. 3/57 after 8.
    4 to CA Lynn!
    4 to CA Lynn!
    6 to CA Lynn!
    WICKET! JA Burns out, bowled BJ Dwarshuis.
SNJ O'Keefe to bowl. 4/72 after 9.
    WICKET! CA Lynn out, bowled SNJ O'Keefe.
BJ Dwarshuis to bowl. 5/76 after 10.
    4 to BCJ Cutting!
SA Abbott to bowl. 5/83 after 11.
    WICKET! BCJ Cutting out, bowled SA Abbott.
TK Curran to bowl. 6/87 after 12.
    6 to JL Pattinson!
    4 to JJ Peirson!
    6 to JL Pattinson!
BJ Dwarshuis to bowl. 6/105 after 13.
    6 to JL Pattinson!
    6 to JL Pattinson!
BAD Manenti to bowl. 6/123 after 14.
    4 to JL Pattinson!
TK Curran to bowl. 6/129 after 15.
    6 to JL Pattinson!
    WICKET! JJ Peirson out, bowled TK Curran.
MC Henriques to bowl. 7/137 after 16.
    4 to JL Pattinson!

Final Score: 7/145 off 17.

~~~~~~~~~~~~~~~~~~~~
Brisbane Heat WIN!
~~~~~~~~~~~~~~~~~~~~
```

To make predictions, we use the simulation engine described above. Each simulation represents a single possible outcome but using Monte Carlo methods we are able to estimate the distribution of match results by simulating many times. To predict the probability of an outcome, simply count up all the simulations where that outcome occurred and divide by the total number of simulations. For example, if you simulated a match between the Sydney Sixers and the Melbourne Stars 1000 times:

- If the Sixers were the winner in 570 of the simulations, then the predicted probability of the Sixers winning is 0.57

- If Steve Smith scored the most runs in 237 of the simulations, then the predicted probability of Steve Smith being the highest run-scorer of the match is 0.237

Results

So after all that, is my model actually any good? To try to answer this question I took 1126 historical matches from my dataset and simulated them 500 times each. Ideally you would simulate each match many more times than this but it was computationally impractical for this project. Due to the bottom-up nature of my model, there are actually numerous modes of appraisal, some of which I outline below.

Simulation Realism

The first common-sense test was to see if the predicted matches looked realistic and fortunately they did. Remember that the matches are simulated using the Ball Prediction model and therefore we had no guarantees that the outputs would be sensible. For example, if the Ball Prediction model was performing poorly we might see batting teams getting bowled out too frequently or teams racking up astronomical scores.

Having watched so much cricket in real life, I was able to quickly see that my simulations were reminiscent of real games, which was reassuring if not overly scientific. To further support my intuition, I sampled some real matches and simulations and compared the distributions of numerous match outcomes. In the examples below you can clearly see that the distributions for these outcomes are very similar for the real and simulated matches.

First Innings Totals

Image by Andrew Kuo.

Match Wickets

Image by Andrew Kuo.

Highest Individual Scores

Image by Andrew Kuo.

Match Results

The next way I assessed my model was by predicting the winner of matches. As described earlier, to get the predicted probability of Team A winning Match X, simply add up all the simulations where Team A was the winner and divide by the total number of simulations. Across the simulated matches, my model was able to predict the winner with **55.6%** accuracy and with a log loss of **0.687**.

While I was initially quite underwhelmed by these results, I was buoyed by further analysis using historical betting odds from Bet365. When I compared my model predictions to the predictions implied by Bet365's odds I found that my model actually out-performed them. Bet365 picked the winner with lower accuracy (**54.2%**) and with a higher log loss (**0.695**). To put this in perspective, a benchmark model that predicts 0.5 probability for both teams for every match corresponds to a log loss of **0.693** meaning that the Bet365 odds are worse than a model with no information!

Now the take away here is not that my model is amazing at predicting T20 match results, but rather that T20 match results are inherently difficult to predict. I do suspect that I would get an uplift in model performance by simulating more than 500 times per match but I will need to test this hypothesis in future.

Other Predictions

Finally, I tried predicting some other features of matches. Again I ran into the intrinsic unpredictability of the sport but I have included a few results below.

First Innings Score

To predict the first innings score of each match I used the mean first innings score in all the simulations where the correct team batted first. In the scatter plot below we can see that there is a relationship between the predicted and actual scores but it is pretty noisy. The correlation here is **0.305** with a p-value of **1.18e-25** suggesting that the simulations are doing a reasonable job of modelling this feature but there are just too many unknowns to do it accurately. Further evidence for this is the average standard deviation for first innings score across all the simulated matches which was quite high at **28.6**.

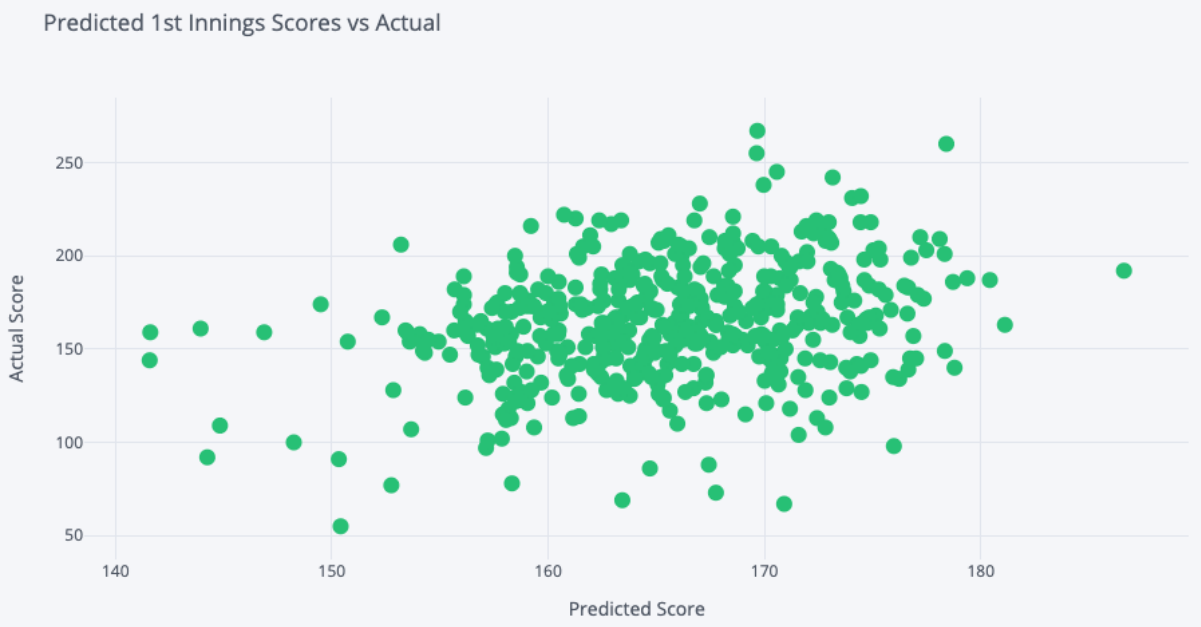

Image by Andrew Kuo.

Top Run Scorer

To predict the top run-scorer for a team in a given match, I chose the player who was the highest run-scorer in the most simulations. Using this method I was able to pick the highest run-scorer with ~25% accuracy. In the plot below we can see varying levels of accuracy when we choose the top N most likely highest run-scorers (rather than just the top 1). For example, in 80% of the matches I simulated, I had the true highest run-scorer in my top 4 predictions based on simulations.

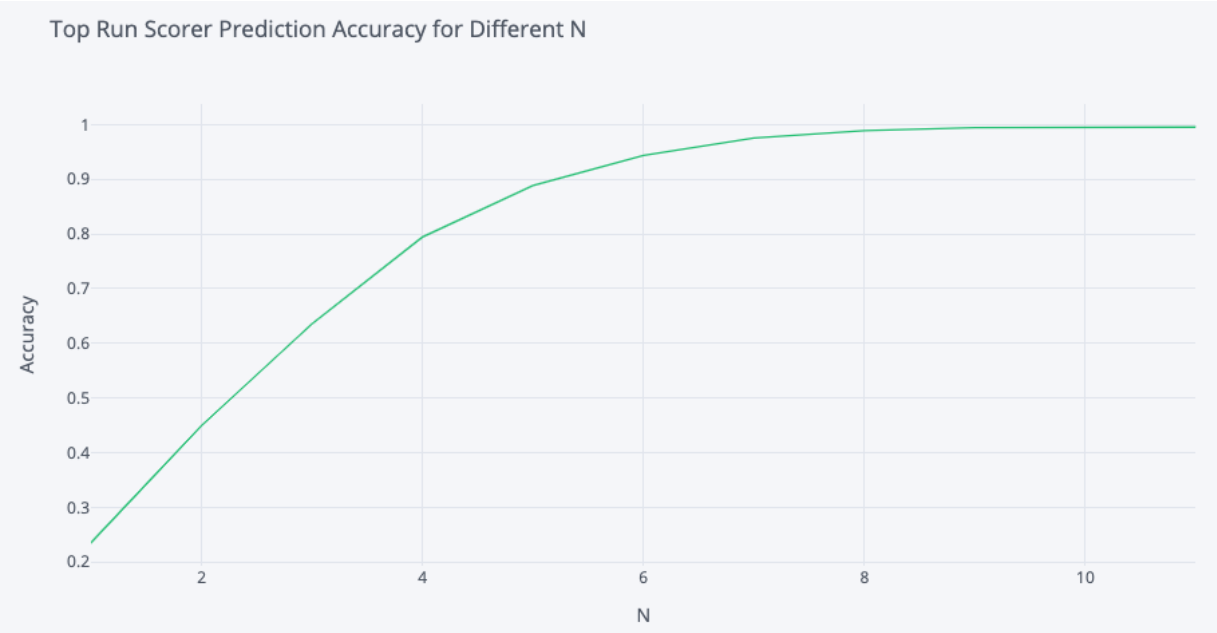

Image by Andrew Kuo.

Top Wicket Taker

I used the same methodology here as for the Top Run Scorer prediction above but this time I was trying to pick the top wicket-taker for each team. The plot below shows that my top pick was correct in ~35% of matches and that the true top wicket-taker was in my top 3 predictions ~75% of the time.

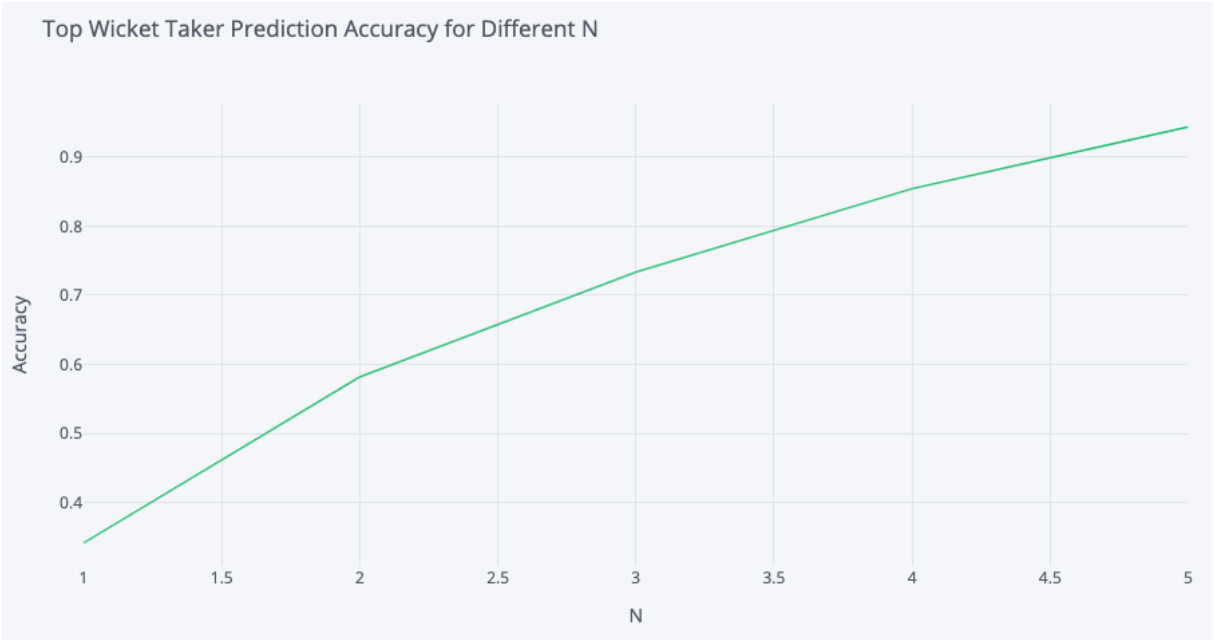

Image by Andrew Kuo.

Conclusion

From the outset, the aim of this project was not really world-class prediction accuracy but rather to see if I could model whole T20 matches with ball-by-ball granularity, which I did. The model is very good at generating realistic simulations of T20 matches but unfortunately, it struggles to predict outcomes with much confidence. My intuition told me that this could be due to the sport just being difficult to predict in general and this was supported by my analysis of betting odds. I found that not only were the bookmaker's odds worse predictors than my model, they

were even worse than a model with no information at all.

While this was a fun experiment, I discovered that there were some limitations to using it in practice. Firstly it takes a long time to run the simulations (approximately 10k simulations per hour on my machine). This wouldn't necessarily be a problem except for the fact that team lists are often not announced until 10–15 minutes before the game starts. Since one of the key advantages to the bottom-up style of modelling is the fact that it can better capture head to head matchups, simulating games with incorrect lineups is unlikely to yield good results.

In terms of next steps there are a few things I would like to try:

1. Add more complexity to the ball prediction model. Because each simulation takes a relatively long time to run, I decided to keep my ball prediction model fairly light-weight, but I would like to experiment with adding more features.
2. Try increasing the number of simulations per match. I did a small experiment with this and saw an immediate uplift in prediction accuracy so I would like to try simulating each match 10k+ times.
3. Try building a similar model for a more predictable sport. The flexibility of this kind of modelling is very appealing and so I would like to try this bottom-up approach with a sport that is a bit easier to model.