Dual Degree Project Report
Cricket Scorecard Problem

R Arun Prakash (MM17B026) and Raghunathan Rengaswamy (Guide)

Indian Institute of Technology, Madras

Abstract

Cricket is without a doubt one of the most popular sports worldwide, especially in the southern region of Asian subcontinent. Having originated in South-East England, this form of sports is now widely played in more than 125 countries and recognized by the International Cricket Council. With the increasing popularity of this sport, various aspects of the game are being automated with the advent of technology. In this project, we intend to predict the phase-wise averages and strike rates of batsmen for any phase specified by the user, when only a set of end-of-innings scorecards is given as input. To predict these phase-wise metrics, it is necessary to predict the various possible ball-by-ball outcomes of a cricket innings. The ball-by-ball data can also be extensively used to derive a suite of advanced metrics for context-informed evaluation of players' batting and bowling performances in cricket matches where the ball-by-ball database is unavailable. Our work will help the cricket teams in spotting the potential talents at the domestic level, by using the available scorecard data of domestic tournaments which only maintain scorecard data to generate phase-wise averages and strike rates. We approach this by predicting the different possible ball-by-ball outcomes using the end-of-innings scorecards, then further compute the phase-wise metrics of the batsman.

1 Introduction

Cricket has an estimated fan base of 2.5 billion, mostly concentrated in the Indian subcontinent. Cricket is the most popular sport in Bangladesh. It is the 2^{nd} most famous sports in Asia, 4^{th} in Europe and also 2^{nd} most famous sport in the whole world after soccer/football. Similar to the game of baseball, Cricket is a bat-and-ball game played between 2 teams of 11 players each. This sport can be easily quantified ball-by-ball. Every ball bowled is an event and the outcome of each ball for both batsman and bowler can be quantified in terms of runs scored/conceded, balls faced/delivered, wicket dismissals, etc. This game is primarily played in 3 formats - Test, One-Day International (ODI) and Twenty20 (T20).

A relatively new format is also being introduced by the England and Wales Cricket Board (ECB) called as *The Hundred*, wherein each team plays a total of 100 balls. *Test* cricket is a game of cricket with the longest match duration and is considered the game's highest standard. It can last up to 5 days and each team is allowed a maximum of two batting innings with no upper bound on the number of deliveries a team is allowed to bat in an innings. The other formats are together called as limited overs cricket as they have an upper bound on number of deliveries that can be bowled. The limit in *T20* cricket is 20 overs (each over consists of six deliveries) and *ODI* cricket is 50 overs.

In Test cricket, an innings is completed when either 10 batsmen are dismissed or the batting team voluntarily "declares" to not bat anymore and in limited overs cricket, an innings is completed when either 10 batsmen are dismissed or the limit on the number of deliveries is reached. The team that scores more runs in total than their opposition is declared the winner of the match.

Various franchise tournaments are held all over the globe due to the popularity of this sport. T20 cricket is the primary format of such tournaments. Indian Premier League (IPL), Big Bash League (BBL) are a few tournaments to name being held in India and Australia respectively. The Indian Premier League (IPL) was founded by the Board of Control for Cricket in India (BCCI) in 2007. There have been fifteen seasons of the IPL tournament, including the current year.

A cricket scorecard summarizes the details of a match in a precise manner for both the batting and bowling team in each innings of the cricket match. A typical batting scorecard contains the runs scored and balls faced by each batsman along with the name of the bowler who dismissed the batsman. A bowling scorecard displays the number of overs bowled by each bowler, number of wickets dismissed, economy rates etc. International matches and franchise cricket's also maintain ball-by-ball statistics of a cricket match whereas domestic tournaments like Syed Mushtaq Ali Trophy (SMAT), Tamil Nadu Premier League (TNPL) only maintain the end-of-innings scorecard.

In this report, we discuss the cricket scorecard problem of predicting the phase-wise averages of batsmen from a given set of end-of-innings scorecards for such domestic tournaments. To do so we predict the various possible ball-by-ball outcomes of a cricket match from the end-of-innings scorecard. The ball-by-ball data gives us a detailed description of the complete match which can be used to determine how players performed in different phases of the cricket match. This detailed statistic of each player can then be used by team management to build a well rounded squad during player auctions.

2 Literature Review

Quantifiable nature of the game makes it readily amenable to data mining. The increasing number of cricket matches played all over the world, owing to the increasing popularity of T20 leagues and the interest surrounding the shortest format has given rise to a deluge in data and the scope for advanced analytical applications in the game. Extensive work has been done in this space, primarily focusing on end-of-innings score predictor, win probability of batting team from current state of the match, advanced metrics to explain batsman and bowler statistics and simulation of cricket matches. The problem of predicting the phase-wise averages of batsmen from a set of end-of-innings scorecards is relatively new, with almost no prior work done in this space. This whole idea of only using the information available in a set of scorecards to predict the phase-wise averages of a batsman is first of its kind.

Winning and Score Predictor (WASP) for limited overs cricket has been used in live television since 2012. It was developed by Scott Brooker and Seamus Hogan using dynamic programming. It is used to predict the end-of-innings score during the first innings and the win probabilities of the teams during the second innings. It utilizes the historic data to find out how an average batting team would fare against an average bowling team from the current state of the match and factors in the innings run rate, weather forecast as well as the cricket ground dimensions into the predictions.

Bailey and Clarke used a more statistical route to predict the margin of victory of an ongoing match by building a multiple regression model that factors in the home ground advantage, playing

XI ratings along with the current form of players. Muhammad Asif and Ian G. McHale have done the most extensive work till now in modelling the win probabilities. They developed a logistic regression model to predict win probabilities in both the first and second innings of ODI matches. They concluded that the most influential variables were the innings run rate at each state of the match, whether the batting team won the toss or not, and whether it is a day game or day-night game in order to include the dew factor against the team bowling second.

A more probabilistic approach was proposed by Swartz et al. to simulate ODI matches. At every ball of the match, the probability of each possible outcome is predicted from the historical data using a Bayesian latent variable model which factors in the current batsman, bowler, number of wickets fallen, innings number, number of balls bowled, innings run rate, etc. This approach was further revisited by David et al. to develop a T20 match simulator using a combination of hierarchical empirical Bayes approach and standard classical estimation techniques.

For modeling and simulating an ODI innings, Sankaranarayanan et al. employed linear regression and nearest-neighbor clustering methods. Six performance measures concerning the team over the previous three years are used as inputs to their model, including runs scored by the batting team per match, runs conceded by the bowling team per match, and so on. Inputs include information on the current innings, such as the current score, batting statistics of the presently batting players, and so on. The model is set up in such a way that it predicts how many runs will be scored in the following five overs. The expected score after five overs is calculated by adding this estimate to the current score, which is then used to forecast the score after the next set of five overs. The method is repeated until an estimate of the end-of-innings score is obtained. For our problem, we have utilized stochastic simulation to predict the ball-by-ball outcome and by making sure that we satisfy all the cricketing constraints and the constraints extracted from the scorecard.

3 Problem Definition

Given a set of end-of-innings scorecards of cricket matches, develop an algorithmic framework to predict the phase-wise averages of batsmen for any phase specified by the user. The sub-problem is to first predict the ball-by-ball outcome of the matches. Predicting the outcome of each ball of an innings from only the information available in the scorecard poses a huge combinatorial problem.

3.1 Approach

Each innings of a cricket match contains a set of batting and bowling scorecards. Figure 1 displays the batting scorecard of an IPL 2020 final match between Delhi Capitals and Mumbai Indians held on November 20, 2020. An end-of-innings batting scorecard contains the batting order of the players, number of runs scored and balls faced by each player in the innings, player's strike rate and the number of boundaries (fours/sixes) hit by each player. It also displays the fall of wickets which informs the ball number at which a particular player got dismissed and score at the point of dismissal.

A bowling scorecard contains a similar set of information about the bowlers. It displays the number of overs bowled by the players, the order in which they bowled their first over, number of wickets scalped by each bowler and individual economy rates. As we are predicting the ball-by-ball outcome to compute phase-wise averages of the batsman, the bowling scorecard is not of

Delhi Capitals Innings						156-7 (20 Ov)
Batter		R	B	4s	6s	SR
Stoinis	c de Kock b Boult	0	1	0	0	0.00
Dhawan	b Jayant Yadav	15	13	3	0	115.38
Rahane	c de Kock b Boult	2	4	0	0	50.00
Shreyas Iyer (c)	not out	65	50	6	2	130.00
Pant (wk)	c Hardik Pandya b Coulter-Nile	56	38	4	2	147.37
Hetmyer	c Coulter-Nile b Boult	5	5	1	0	100.00
Axar	c (sub)Anukul Roy b Coulter-Nile	9	9	1	0	100.00
Rabada	run out (Suryakumar Yadav/Coulter-Nile)	0	0	0	0	0.00
Extras		4 (b 0, lb 1, w 3, nb 0, p 0)				
Total		156 (7 wkts, 20 Ov)				
Did not Bat	Ashwin , Praveen Dubey , Nortje					

Fall of Wickets

0-1 (Stoinis, 0.1), 16-2 (Rahane, 2.4), 22-3 (Dhawan, 3.3), 118-4 (Pant, 14.6), 137-5 (Hetmyer, 17.2), 149-6 (Axar, 19.2), 156-7 (Rabada, 20)

Figure 1: IPL 2020 Final Innings 1 Scorecard

much importance to us right now. Though we will discuss at the end how we can further utilize the bowling scorecard to improve our algorithm.

We extract all the information available in the batting scorecard and fall of wickets section. Using this information, we divide the entire innings into smaller segments. These segments are basically the partnerships that happened during the course of the innings. In each partnership, we compute the runs scored and balls faced by each batsman as well as the number of boundaries they might have possibly hit. Then we go through each segment, come up with a random ball-by-ball outcome for each ball in that segment making sure that the total number of runs scored, balls faced for each player is equal to what we computed earlier. We perform this on each segment to arrive at the ball-by-ball outcome of the innings. We include randomization in the algorithm under the constraints that all the cricketing rules are satisfied. So each time we run this algorithm, we get a different outcome satisfying the same constraints. Ball-by-ball outcome from each run is termed as a *realization*.

So we run the algorithm multiple times to get different sets of realizations. While computing phase-wise averages, we obtain phase-wise runs and number of dismissals for that player in all the innings in which he/she has batted. While computing phase-wise runs, we take an average of multiple realizations of each innings. The number of such realizations of each innings to include in the computation is specified by the user. The algorithm is explained in further details in future sections, but this is the high level overview of the algorithm.

3.2 Motivation

Figure 2: Teams featuring in Syed Mushtaq Ali Trophy

The primary motivation for this problem is to identify the cricketing talent pool available at the domestic level. Syed Mushtaq Ali Trophy (SMAT), Tamil Nadu Premier League (TNPL), Karnataka Premier League (KPL) are a few T20 tournaments to name. The Syed Mushtaq Ali Trophy is a domestic Twenty20 cricket championship in India, organized by the Board of Control for Cricket in India (BCCI), among the teams from the Ranji Trophy. The 38 teams (Fig 2) have been divided into five Elite groups, namely A, B, C, D, E, comprising six teams each, and one Plate group with eight teams. Each team will play a total of five league matches, regardless of the number of teams in the group, with all 38 teams playing the same number of matches. A total of 6 teams from the top ranked teams in each group (5 elites and 1 plate) will make it to the knockout stage, and the 2nd ranked teams from the 5 elite groups will join to make a total of 8 teams.

We observe that the number of players participating in this one tournament is itself very huge. Many teams do not make it to the knockout stages. And only knockout stages maintain a record of ball-by-ball outcomes of the match. Excluding this, we only support/watch top teams in these tournaments like Tamil Nadu, Maharashtra, Karnataka etc. There are players in some low performing teams who give match winning performances, but go unnoticed just because we only look at the end-of-innings scorecard. With the help of ball-by-ball outcomes, we can compute how a player has performed in different phases of the match and come up with his detailed analytics.

With the introduction of two new IPL teams, the demand for uncapped Indian players has risen. Each IPL team is scouting for Indian talents to fill up their squad. We believe that our algorithm will help them in spotting the potential talents, using the available scorecard data of such domestic tournaments. These potential talents spotted through this means will be given the opportunity to play at the highest level, learn from some of the best in the field and in future represent their country with good performances.

4 Algorithm Explanation

We intend to develop an algorithmic tool, which takes in as input a set of end-of-innings scorecards, player/batsman name and phase definition (range of ball numbers), and gives as output the phase-wise averages of the given batsman in the user-defined phases computed from the given set of end-of-innings scorecards only. We do not use any historical information of the batsman, as within the scope of our problem and the given input, we know exactly how the batsman performed.

4.1 Introduction

To arrive at the phase-wise averages from the given set of end-of-innings scorecards, we first need to predict the ball-by-ball outcomes of each innings using these scorecards only. In reality, there is only one actual realization for each innings and a scorecard is a summary of that. Recreating the ball-by-ball outcome from the limited information available in the scorecard is the primary purpose of this algorithm. While recreating the ball-by-ball outcome from the scorecard, it is necessary to adhere to each and every detail given in the scorecard and also the cricketing rules. Now there are various possible ball-by-ball outcomes possible for each innings. We formulate an algorithm in such a way that the user can also specify the number of possible ball-by-ball outcomes (realizations) to consider for each innings while computing phase-wise averages. This is done to reduce the bias.

4.2 Constraints

The following cricketing constraints and constraints from scorecard should be satisfied while predicting ball-by-ball outcome of an innings:

1. For each player, total runs scored predicted by ball-by-ball outcome should be equal to the actual value in the scorecard.
2. Similarly, number of balls faced, fours/sixes hit should be equal to the actual values in the scorecard.
3. The players must change strike when single is taken and at the end of over.

Table 1: Random batsman XYZ data from scorecard

Player	**Runs Scored**	**Balls Faced**	**Fours**	**Sixes**
XYZ	40	18	4	3

Table 2: Predicted realization for the random batsman XYZ

Player	**Realization**
XYZ	[1, 0, 2, 6, 4, 4, 1, 6, 4, 1, 4, 0, 1, 0, 6, 0, 0, 0]

For example, table 1 shows the actual batting data of a random player XYZ from a match scorecard and table 2 show a predicted realization for the same player. Therefore the following set of constraints should be satisfied for each player:

$$\mathsf{sum}(\mathsf{realization}) = \mathsf{data}[\text{'runs scored'}] \tag{1}$$

$$\mathsf{len}(\mathsf{realization}) = \mathsf{data}[\text{'balls faced'}] \tag{2}$$

$$\mathsf{realization.count}(4) = \mathsf{data}[\text{'fours'}] \tag{3}$$

$$\mathsf{realization.count}(6) = \mathsf{data}[\text{'sixes'}] \tag{4}$$

4.3 Database

For the purpose of this project, we have been using RCB's ball-by-ball database. It contains data of all the T20 matches held in the time period 2018-2020. It has entries of total 2650 matches including 180 matches of IPL 2018/19/20.

4.4 Algorithm Breakdown

The algorithm follows the following process steps to compute phase-wise averages from the scorecard.

S1: Extract required information from scorecard

S2: Use the available data to divide the innings into smaller segments

S3: Identify the pair of batsman at each segment

S4: Compute runs scored, balls faced, fours and sixes hit by each pair in each segment

S5: Go through each segment ball-by-ball and find possible outcome at each ball

S6: Compute phase-wise average for the given player using the obtained ball-by-ball data

At each step of the process, it is necessary to satisfy the cricketing constraints and the constraints as mentioned in section 4.2. Now we will have a look at each of these steps one-by-one and elaborate on them.

4.4.1 S1 : Extract required information from scorecard

Extract batting order, each batsman statistics, fall of wickets statistics from the scorecard. These are the primary data that will be required to compute the ball-by-ball outcome. We will have Match 60 of IPL 2020 final between Mumbai Indians and Delhi Capitals as reference for the illustration of the algorithm.

Table 3: Match 60, IPL 2020 : Delhi Capitals : Scorecard

Player	Player Key	Runs Scored	Balls Faced	Fours	Sixes
Stoinis	29	0	1	0	0
Dhawan	110	15	13	3	0
Rahane	187	2	4	0	0
Shreya Iyer	142	65	50	6	2
Pant	136	56	38	4	2
Hetmeyer	44	5	5	1	0
Axar	109	9	9	1	0
Rabada	126	0	0	0	0

Table 3 is the first set of data extracted from the scorecard. They are primarily the number of runs scored, balls faced, fours and sixes hit by each player in the batting team. Each player is associated with a player key in the database which is used as his reference as shown in table 3.

Table 4: Match 60, IPL 2020 : Delhi Capitals : Fall of Wickets

Player	Player Key	Balls at dismissal	Score at dismissal	Dismissal type
Stoinis	29	1	0	caught
Rahane	187	16	16	caught
Dhawan	110	21	22	bowled
Pant	136	90	118	caught
Hetmeyer	44	104	137	caught
Axar	109	116	149	caught
Rabade	126	120	155	runout

Table 4 is the second and final set of data extracted from the fall of wickets section of the scorecard. They are primarily the player name who got dismissed, at what ball did he get dismissed, what was the score at the point of dismissal and the type of dismissal. These sets of information will now be used in the following sections.

4.4.2 S2 : Partitioning into segments

We utilize the balls at the dismissal column in table 4 to partition an innings (120 balls) into smaller segments. Each segment is represented by a tuple (start, end) which signifies the ball number of the start and end of the segment. Between one segment and the next, one ball is unaccounted for which is a wicket ball.

Table 5: Match 60, IPL 2020 : Delhi Capitals : Segment Partition

Segment Number	Segment Start	Segment End	Num Balls
1	1	0	0
2	2	15	14
3	17	20	4
4	22	89	68
5	91	103	13
6	105	115	11
7	117	119	3

In table 5, segment start is the starting ball number of that segment and start end is the ending ball number. There are end-start+1 number of balls in each segment. As the first ball itself is a wicket, the first segment starts at 1 and ends there itself with a total 0 (0-1+1) ball in that segment. In other segments we can observe that the next segment starts only after a wicket ball. Table 4 and table 5 can be compared simultaneously to validate their coherence. If we sum the number of balls we get 113, which when added to 7 wicket balls gives us a total 120 balls.

4.4.3 S3 : Identifying the pair of batsman at each segment

In this step, we identify the pair of batsmen who are batting on crease in each of the segments determined earlier. For this purpose we utilize the segments information from the previous section, the batting order and balls at dismissal for each player from table 4. Pair of batsmen in the

first segment is determined straightforwardly from the batting order. Then each time a player is dismissed, he is removed and the next player in the batting order replaces him in the next segment.

Table 6: Match 60, IPL 2020 : Delhi Capitals : Pair of batsman at each segment

Segment Number	**Segment**	**Pair of batsman**	**Batsman Keys**
1	(1, 0)	Stoinis and Dhawan	(29, 110)
2	(2, 15)	Dhawan and Rahane	(110, 187)
3	(17, 20)	Dhawan and Shreyas	(110, 142)
4	(22, 89)	Shreyas and Pant	(142, 136)
5	(91, 103)	Shreyas and Hetmeyer	(142, 44)
6	(105, 115)	Shreyas and Axar	(142,109)
7	(117, 119)	Shreyas and Rabada	(142, 126)

Table 6 illustrates the pair of batsmen playing at each segment in the reference match. We can observe that Shreyas Iyer remained in the crease for a long time, was a part of 5 partnerships with the longest being with Rishabh Pant and remained not out at the end.

4.4.4 S4 : Compute runs scored, balls faced, fours and sixes hit by each pair in each segment

In this step we compute all the important data of each of the pairs in each segment. These are the number of runs scored by each batsman in each segment, number of balls faced and fours/sixes hit. For this purpose, we utilize the segments information from section 4.4.2, batting pair information from section 4.4.3, information of each batsman from scorecard (table 3), balls at dismissal and score at dismissal information from table 4.

We first compute the total runs scored in each segment using the score at dismissal information. This computation is straightforward, basically for a segment, it is the difference of the score at dismissal at the ball number corresponding to the end of that segment and the previous score at dismissal. Now we need to compute how much of the runs and balls in each segment does each of the players contribute to.

The idea is to always first assign runs and balls to the batsman who is going to get dismissed at the end of the segment. Then assign the remaining runs and balls based on total segment runs and balls to the other batsman. Always keep track of the sum total of the batsman's runs and balls from the scorecard and keep reducing it when we assign them runs and balls in each segment. This ensures that both individual segment constraints are met as well as the scorecard constraints of each batsman. This assignment of runs and balls to each batsman in each segment is deterministic and gives the same result each time we run the algorithm.

Distribution of sixes and fours for each batsman across the segments in which they have batted is randomized under the constraints that the total runs and balls incorporating these sixes and fours do not exceed their runs and balls in each segment. We first distribute sixes and then fours, but the procedure remains the same.

We first identify the segments in which a batsman has batted and has sufficient runs and balls to accommodate a six. We then randomly select a segment from these, and assign a six to that segment. We modify the runs and balls and continue doing the same for n number of times, where

n is the total number of sixes hit by that batsman obtained from the scorecard. Once we assign all the sixes, we assign fours using the same procedure. We do this distribution for all the batsman who batted in the innings.

Table 7: Match 60, IPL 2020 : Delhi Capitals : Partnership Data

S N	Seg Num	Segment	Batsman	Type	Runs	Balls	Fours	Sixes
1	1	(1, 0)	Stoinis	B1	0	0	0	0
2	1	(1, 0)	Dhawan	B2	0	0	0	0
3	2	(2, 15)	Rahane	B1	2	3	0	0
4	2	(2, 15)	Dhawan	B2	14	11	3	0
5	3	(17, 20)	Dhawan	B1	1	1	0	0
6	3	(17, 20)	Shreyas Iyer	B2	5	3	1	0
7	4	(22, 89)	Shreyas Iyer	B2	40	31	3	2
8	4	(22, 89)	Pant	B1	56	37	4	2
9	5	(91, 103)	Shreyas Iyer	B2	14	9	2	0
10	5	(91, 103)	Hetmeyer	B1	5	4	1	0
11	6	(105, 115)	Shreyas Iyer	B2	3	3	0	0
12	6	(105, 115)	Axar	B1	9	8	1	0
13	7	(117, 119)	Shreyas Iyer	B1	3	4	0	0
14	7	(117, 119)	Rabada	B2	0	0	0	0

Table 7 gives the illustration of all the algorithm sections until this point. Segment gives the starting and ending ball number of that segment. Each segment has two rows corresponding to two batsmen at the crease. They are being designated as B1 and B2. Except for the last segment, the B1 player corresponds to the player who gets dismissed at the end of that segment and player B2 continues batting in the next segment. Basically player B1 is used as reference to compute runs and balls and then runs and balls of player B2 is computed based on total runs and balls in that segment. As explained earlier all the columns in table 7 are deterministic except column *fours* and *sixes*. The allocation of these two columns are randomized as explained above and can be different each time we run the algorithm, but the rules and constraints are always satisfied except in some rare edge cases.

4.4.5 S5 : Find individual batsman predicted realizations in each segment and combine them to obtain entire innings simulation

We begin with predicting the ball-by-ball outcome of each batsman in each segment using his runs, balls, fours and sixes ensuring that constraints 1-4 are satisfied. We will call this individual batsman realizations. This part of algorithm takes the following as input :

1. Number of runs scored by batsman in that segment
2. Number of balls faced by batsman in that segment
3. Number of fours hit by batsman in that segment
4. Number of sixes hit by batsman in that segment

	rem_runs	rem_balls	boundary_balls	non_boundary_balls	runs_bb	runs_non_bb	prob_bb	prob_non_bb	rand_num	check_bb	random_outcome
0	40.0	31.0	5.0	26.0	24.0	16.0	0.16	0.84	-1.000000	-1.0	-1.0
1	40.0	31.0	5.0	26.0	24.0	16.0	0.16	0.84	0.593800	0.0	0.0
2	40.0	30.0	5.0	25.0	24.0	16.0	0.17	0.83	0.869793	0.0	1.0
3	39.0	29.0	5.0	24.0	24.0	15.0	0.17	0.83	0.288648	0.0	2.0
4	37.0	28.0	5.0	23.0	24.0	13.0	0.18	0.82	0.200351	0.0	1.0
5	36.0	27.0	5.0	22.0	24.0	12.0	0.19	0.81	0.496275	0.0	1.0
6	35.0	26.0	5.0	21.0	24.0	11.0	0.19	0.81	0.346998	0.0	1.0
7	34.0	25.0	5.0	20.0	24.0	10.0	0.20	0.80	0.105490	1.0	4.0
8	30.0	24.0	4.0	20.0	20.0	10.0	0.17	0.83	0.939543	0.0	0.0
9	30.0	23.0	4.0	19.0	20.0	10.0	0.17	0.83	0.642935	0.0	1.0
10	29.0	22.0	4.0	18.0	20.0	9.0	0.18	0.82	0.141696	1.0	6.0
11	23.0	21.0	3.0	18.0	14.0	9.0	0.14	0.86	0.782798	0.0	0.0
12	23.0	20.0	3.0	17.0	14.0	9.0	0.15	0.85	0.038712	1.0	4.0
13	19.0	19.0	2.0	17.0	10.0	9.0	0.11	0.89	0.935740	0.0	0.0
14	19.0	18.0	2.0	16.0	10.0	9.0	0.11	0.89	0.630341	0.0	0.0
15	19.0	17.0	2.0	15.0	10.0	9.0	0.12	0.88	0.999201	0.0	1.0
16	18.0	16.0	2.0	14.0	10.0	8.0	0.12	0.88	0.658587	0.0	0.0
17	18.0	15.0	2.0	13.0	10.0	8.0	0.13	0.87	0.557425	0.0	1.0
18	17.0	14.0	2.0	12.0	10.0	7.0	0.14	0.86	0.765820	0.0	2.0
19	15.0	13.0	2.0	11.0	10.0	5.0	0.15	0.85	0.520278	0.0	0.0
20	15.0	12.0	2.0	10.0	10.0	5.0	0.17	0.83	0.633926	0.0	0.0
21	15.0	11.0	2.0	9.0	10.0	5.0	0.18	0.82	0.167865	1.0	6.0
22	9.0	10.0	1.0	9.0	4.0	5.0	0.10	0.90	0.470323	0.0	0.0
23	9.0	9.0	1.0	8.0	4.0	5.0	0.11	0.89	0.960040	0.0	0.0
24	9.0	8.0	1.0	7.0	4.0	5.0	0.12	0.88	0.304389	0.0	0.0
25	9.0	7.0	1.0	6.0	4.0	5.0	0.14	0.86	0.564410	0.0	1.0
26	8.0	6.0	1.0	5.0	4.0	4.0	0.17	0.83	0.273096	0.0	0.0
27	8.0	5.0	1.0	4.0	4.0	4.0	0.20	0.80	0.248441	0.0	0.0
28	8.0	4.0	1.0	3.0	4.0	4.0	0.25	0.75	0.887369	0.0	1.0
29	7.0	3.0	1.0	2.0	4.0	3.0	0.33	0.67	0.479793	0.0	2.0
30	5.0	2.0	1.0	1.0	4.0	1.0	0.50	0.50	0.822903	0.0	1.0
31	4.0	1.0	1.0	0.0	4.0	0.0	1.00	0.00	0.128270	1.0	4.0

Figure 3: History of ball-by-ball prediction of Shreyas Iyer in Segment 4

For illustration, let us take Shreyas Iyer in segment number 4 in which he scored 40 of 31 balls and hit 3 fours and 2 sixes. Figure 3 illustrates the complete breakdown of the ball-by-ball outcome prediction for Shreyas Iyer in segment 4. The column *random outcome* gives the predicted outcome for each of the 31 balls. Row 0 is just for reference and is the initial state of the batsman.

Column *rem runs* is the remaining number of runs left, *rem balls* is the remaining number of balls left, *boundary balls* is the number of boundary balls left to be allocated, *non boundary balls* is the number of non boundary balls left to be allocated, *runs bb* is the runs scored on boundaries that are left to be allocated, *runs non bb* is the runs scored without boundaries that are left to be allocated, *prob bb* is the probability of the boundary ball, *prob non bb* is the probability of the non boundary ball, *rand num* is the random number generated to determine whether current ball will be a boundary ball or not, *check bb* gives the result of whether the current ball results in a boundary or not.

The procedure for simulating a realization given runs, balls, fours and sixes is given next. Before starting the simulation, we randomly permute the boundaries (fours and sixes). In our case, we have 3 fours and 2 sixes hit by Shreyas Iyer in segment 4. In the simulation results which we have presented in Figure 3, the randomly permuted boundaries are [4, 6, 4, 6, 4]. We will call this list as *permuted boundary*, and assign variable $idx = 0$ as a pointer to iterate through this list. The simulation goes as follows:

1. Compute P(BB) and P(NBB), where BB = Boundary Ball, NBB = Non-Boundary Ball and P(X) = Probability of X
2. Generate a random number *rand num* from the range [0, 1)
3. If *rand num* < P(BB), then Outcome = 'Boundary' ; else Outcome = 'No Boundary'
4. If Outcome = 'Boundary', then *ball outcome* = *permuted boundary*[idx]; idx += 1
5. If Outcome = 'No Boundary', then *ball outcome* = *random choice*([0, 1, 2]) with prob = [0.466, 0.452, 0.082]
6. Modify remaining number of boundary and non-boundary balls left.
7. Continue from step 1 until total balls left are zero.

We follow these sets of steps for each batsman in each segment. The history of one complete simulation to get a ball-by-ball outcome for Shreyas Iyer in segment 4 is given in Figure 3. In step 5, probabilities associated with 0, 1 and 2 runs are obtained from the RCB ball-by-ball database. We basically determine whether the current ball is going to be a boundary ball or not and based on the outcome, we further determine the outcome of that ball. If it's a boundary ball, we iterate through the *permuted boundary*, else we choose a random number between 0/1/2 based on a pre-determined probability distribution. For simplicity, we have ignored outcome 3 as it is a very rare outcome in T20 cricket.

We can observe that the above algorithm uses randomization at three different stages. Firstly randomization is used to permute the boundaries. This ensures that the order of boundaries is different each time. Secondly while determining whether the next ball is going to be a boundary or not, we have used a random number generator. This ensures that there is no prior way to determine whether the next ball is going to result in a boundary or not. Thirdly, when we determine that a ball is a non boundary ball, then again we randomly choose between 0, 1 and 2. This introduces a further level of randomization. By incorporating three stages of randomization, we ensure that each time the algorithm is run, it gives a different set of ball-by-ball outcome predictions following the constraints mentioned earlier each time.

Now for the simulation of the entire innings, we go through each segment and use the above algorithm to obtain one possible realization of each batsman in each segment. Then in each segment we need to combine two realizations of two batsmen to get one individual realization for that part of that segment. We follow the cricket strike rules to determine which player is going to be on the strike on each ball (i.e. play that particular ball). We also use the batting order from the scorecard for this purpose. At the start of the innings, the batsman at first position will be on strike on the first ball. A batsman remains on strike if he/she scores a 0/2/4/6 and rotates the strike if he/she scores a 1. If a batsman is dismissed, the next batsman as per the batting order will be on strike. Likewise, at the end of an over (after 6 balls each), batsmen rotate their strike. Using these sets

of rules, we combine the realizations of each pair of batsmen one by one across each segment. We finally end up with one realization of the entire innings, which gives the ball number, the batsman who faced that ball and the outcome of that ball.

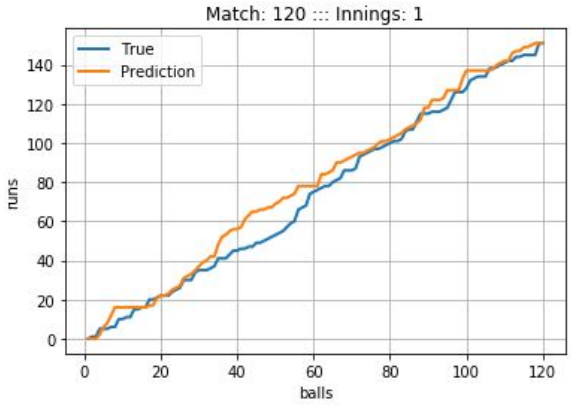

Figure 4: IPL 2020 Final : Delhi Capitals : Entire Innings Simulation Plot

For the illustration of the algorithm and its approximation with actual match events, figure 4 shows the plot of runs vs balls for the entire innings. The predicted line is the ball-by-ball outcome prediction from our algorithm and the true line is the actual match scenario. We can observe from the figure that the predicted line has a high overlap with the true line.

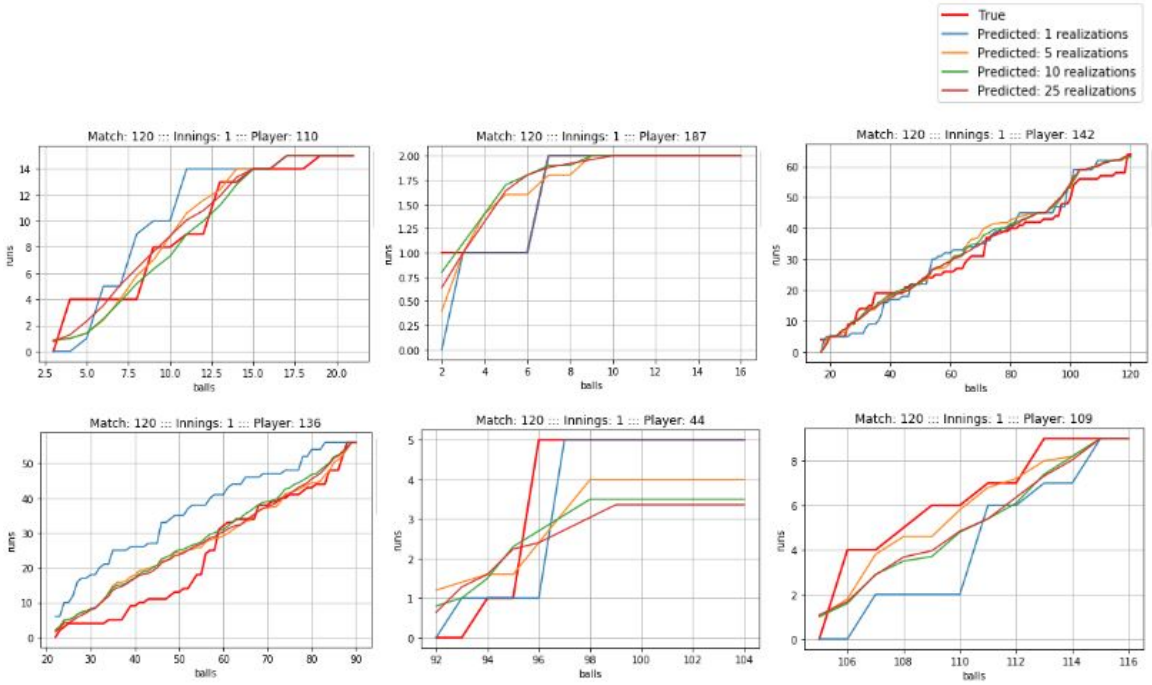

Figure 5: IPL 2020 Final : Delhi Capitals : Individual Batsman Simulation Plots

Figure 5 shows the plots of runs vs balls for each batsman separately. The y-axis represents the runs scored by them and the x-axis is the entire duration of balls during which they were on crease and is not equal to the total number of balls faced by them. In this case, we also have averaged the predictions for 5, 10 and 25 realizations. Table 3 can be used as reference to obtain individual batsman details.

We can observe from the plots in figure 5 that the averaged predictions tend to overlap with the true line. This is clearly true when the batsman has faced a sufficient number of balls, but the plots do not match much when the player has only faced a very few balls. Overall, as we keep on taking average of more and more realizations, they start to strictly overlap with each other. This explains that taking the average of a threshold number of realizations is sufficient for our purpose and that we don't have to run the algorithm for a large number of realizations as it will only increase the time complexity without having any improvement on the results.

4.4.6 S6 : Compute phase-wise averages of batsman using the ball-by-ball data

Now we have an algorithm to generate multiple different realizations (ball-by-ball outcomes) of each innings in the database. This algorithm has stochastic simulation incorporated into it which ensures that all the predicted realizations follow the constraints but are randomized each time. Next step is to predict the phase-wise averages of any batsman given a set of scorecards, phase definition (ball ranges) and number of realizations to be considered of each innings in which that batsman has batted. For any given batsman, his/her phase-wise average is computed as follows:

1. For each innings in which that batsman has batted, compute the predicted ball-by-ball outcome of the entire innings using our algorithm.
2. Then go through each phase and compute the number of runs scored by the batsman in that phase and sum up the runs scored in each phase separately.
3. Repeat 1 and 2 for *num rlz* times, where *num rlz* is the total number of realizations of each innings to be averaged, it is specified by the user.
4. Then divide the total runs scored across different realizations by *num rlz* (separately for each phase)
5. After running through all the matches in the database (or the given set of scorecards), phase-wise average in each phase is given by :

$$\text{Batting Average} = \frac{\text{Runs scored}}{\text{Number of times out}} \tag{5}$$

6. Note: In case the batsman is dismissed 0 times in any particular phase, then the batting average is not defined in such cases and we assign Batting Average = 0 for such phases.
7. Similarly, we can compute phase-wise strike rates using the following equation :

$$\text{Batting Strike Rate} = \frac{\text{Runs scored}}{\text{Balls Faced}} \tag{6}$$

4.5 Validation

We perform validation of our algorithm by observing the predicted ball-by-ball outcome and comparing it with the actual scorecard. We try to see if the predicted realizations are consistent with the scorecard. It is not possible to be accurate to the point, but looking at the batsman statistics from the predicted realizations and comparing it with the actual scorecard gives a good idea. The phase-wise average results and validation is done in the next section under *Results*.

Table 8: Match 60, IPL 2020 : Delhi Capitals : Validating Innings Simulation

Player	**Player Key**	**Runs Scored**	**Balls Faced**	**Fours**	**Sixes**
Stoinis	29	0	1	0	0
Dhawan	110	15	13	3	0
Rahane	187	2	4	0	0
Shreya Iyer	142	64	49	6	2
Pant	136	56	38	4	2
Hetmeyer	44	5	5	1	0
Axar	109	9	9	1	0
Rabada	126	0	1	0	0

Table 8 gives the validation of innings simulation for one random run. It can be compared with the actual scorecard in table 3. We can observe that our predictions are consistent with the actual scorecard except some minor differences like Shreyas Iyer has only faced 49 balls as per our simulation but in the actual match he faced 50 balls and Rabada has faced 1 ball as per our simulation but in the actual match he faced none.

These minor errors occur because we are simulating an entire innings from a very limited set of information. Moreover, these minor errors do not have much effect on the end result computation of phase-wise averages when we consider multiple realizations for each innings. Thus it is safe to proceed further with this algorithm.

5 Results and Discussion

We have illustrated the intermediate outcomes of the subsections of the algorithm in the respective sections above. In this section we present various other results which showcase the effectiveness of the algorithm and compare it with actual match results. For the purpose of this illustration, we will be using scorecards of all the IPL matches belonging to season 2018-2020. In this period, a total of 180 matches were played between 8 IPL teams. We have defined the phases as given in table 9.

Table 9: Phases Definition

Phase Number	**Ball Range**	**Num of Balls**	**Remarks**
1	1 - 36	36	Powerplay
2	37 - 66	30	Middle Over 1
3	67 - 96	30	Middle Over 2
4	97 - 120	24	Death

5.1 Phase-Wise Averages

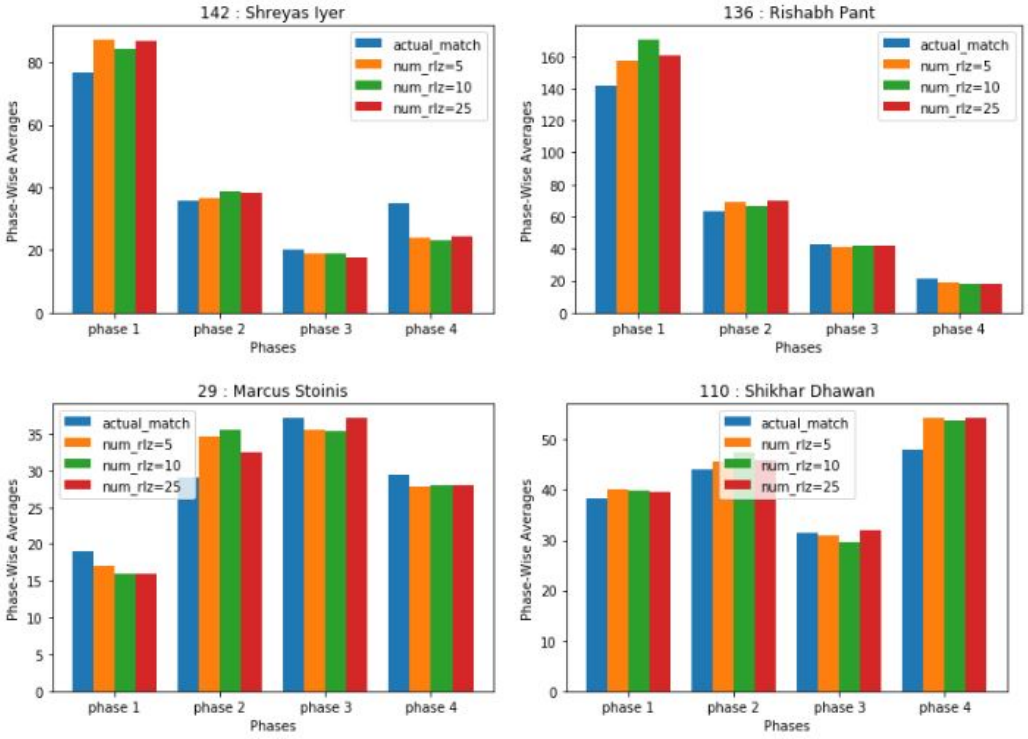

Figure 6: Phase-Wise Averages of DC batsmen during IPL 2018-20

Figure 6 shows the bar plot of phase-wise averages of batsmen belonging to Delhi Capitals. In the plot, the blue bar represents the phase-wise averages computed from the actual match. The orange, green and red bars represent the phase-wise averages computed from our algorithm by setting the number of realization as 5, 10 and 25. Basically, we have taken the average by generating these number of realizations of each innings. Similarly figure 7 shows the bar plot of phase-wise averages of batsmen belonging to Mumbai Indians.

Considering that we only have scorecard information available to start with, it is clearly visible from the plots in figures 6 and 7 that the phase-wise average predictions from our approach and algorithm has approximated the actual phase-wise averages to a very good extent. In most cases, the predicted phase-wise averages are in very close range to the actual phase-wise averages of the players. For example, actual and predicted phase-wise averages of Shreyas Iyer, Rishabh Pant, Rohit Sharma, Ishan Kishan are very similar.

If we compare across the number of realizations, we can observe that there is not much change when we set the number of realizations as 5, 10 and 25. From this we can infer that as we take an average of more realizations, the phase-wise average saturates and does not change much. Thus, we can set the number of realizations as 5 per innings to obtain the same results and be computationally less intensive.

In some cases, especially in phase 4, the actual phase-wise average is pretty high compared to

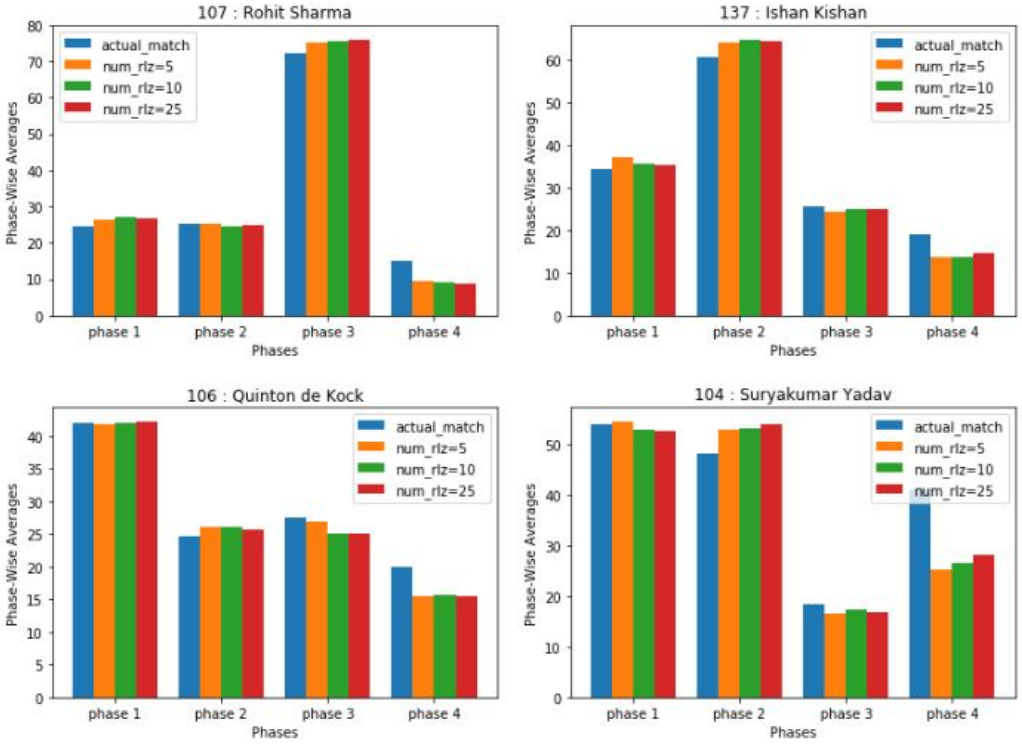

Figure 7: Phase-Wise Averages of MI batsmen during IPL 2018-20

the predicted phase-wise averages. This problem is clearly visible in the case of Suryakumar Yadav of Mumbai Indians. Figure 8 shows MAE for all the four phases and different settings of number of realizations. The MAE is relatively higher for phase 4 but overall MAE is low across phases.

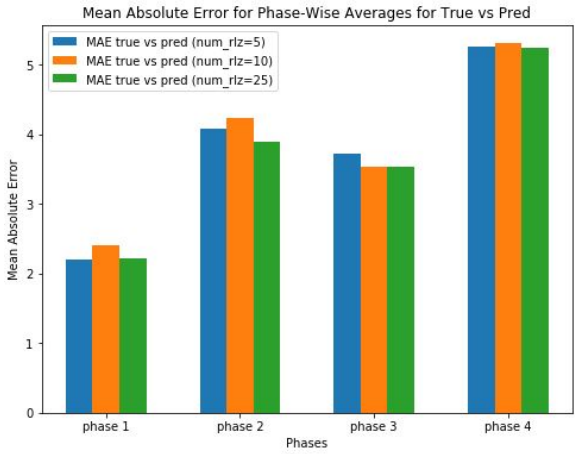

Figure 8: Mean Absolute Error between True and Predicted Phase-Wise Averages

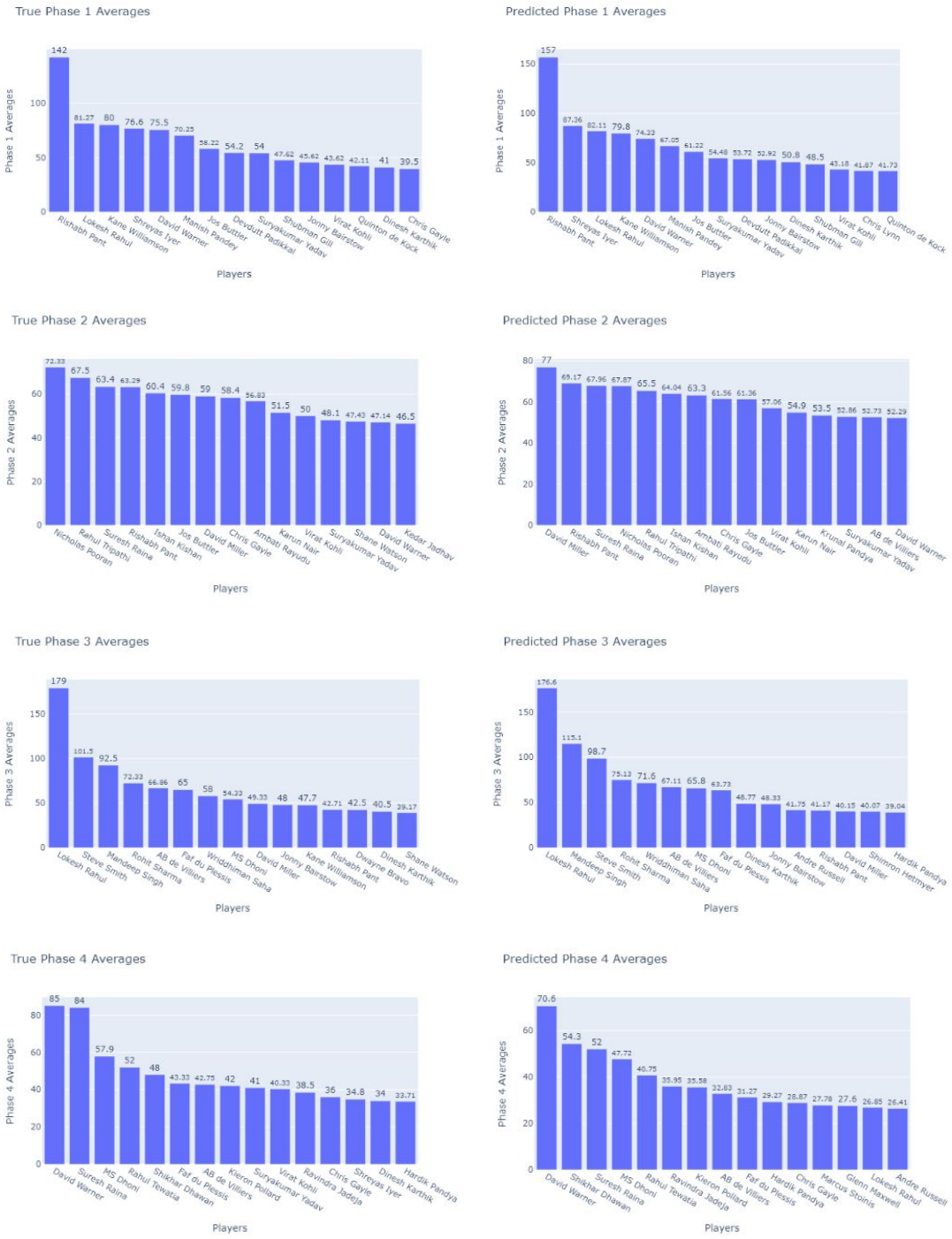

Figure 9: Comparison of top 15 players with highest average in various phases (true and predicted)

From figure 9, we can observe that the same set of players are in the top 15 range albeit in a slightly different order of true and predicted averages. This is a good achievement from our algorithm, as in cricket, we generally look at the averages and strike rates in relative terms i.e who has a better average than who and so on. For instance, in phase 1, Rishabh Pant is clear of all with a huge average of around 142 (true) and 152 (predicted). Lokesh Rahul has a better average than Quinton de Kock in both true and predicted values. This trend is similar across different phases.

5.2 Phase-Wise Strike Rates

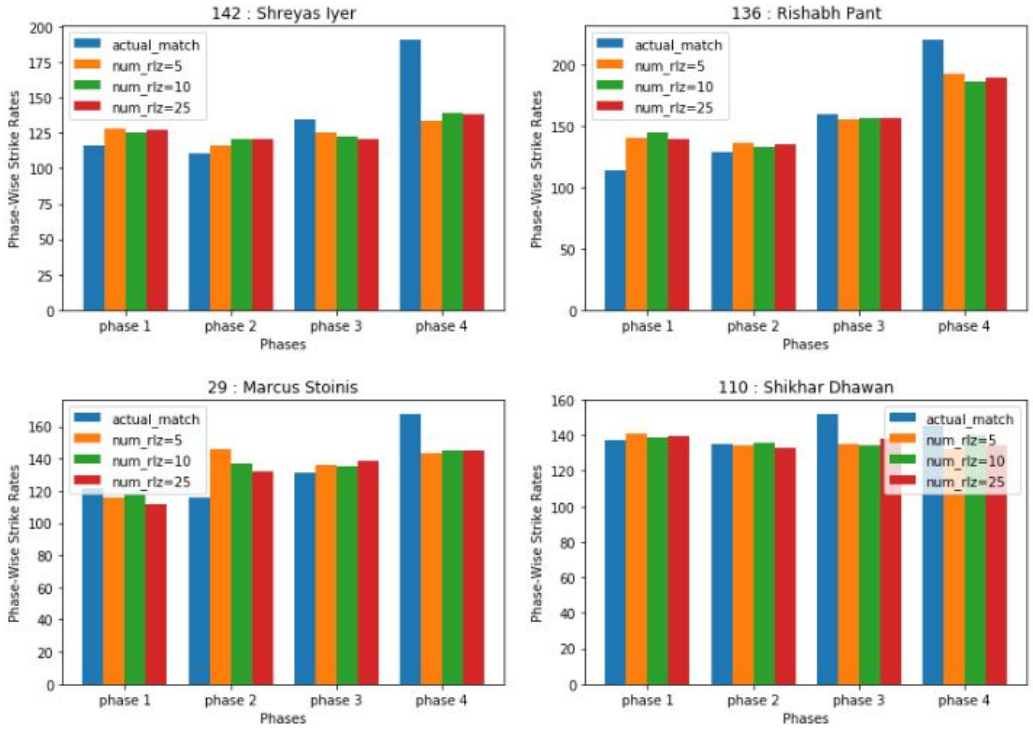

Figure 10: Phase-Wise Strike Rates of DC batsmen during IPL 2018-20

Figure 10 and 11 display the bar plot of phase-wise strike rates of batsmen from DC and MI. Similar to the plots in the previous section, the blue bars indicate the true values obtained from actual match ball-by-ball data whereas the other color bars indicate the predicted values for different number of realizations.

Compared to phase-wise average results in previous sections, in the case of phase-wise strike rates the true and predicted values are relatively more off in phase 4. We can also observe from figure 12 that the error in phase 4 is around 35.

We experimented to figure out the reason behind the anomaly in phase 4 by further dividing that phase into smaller phases with 6 balls in each phase. We observed from the results that the anomaly predominantly generates from the final 6 balls, that particular phase had an MAE of 70, which is almost double of the error when we considered the final 24 balls as single phase (figure 12).

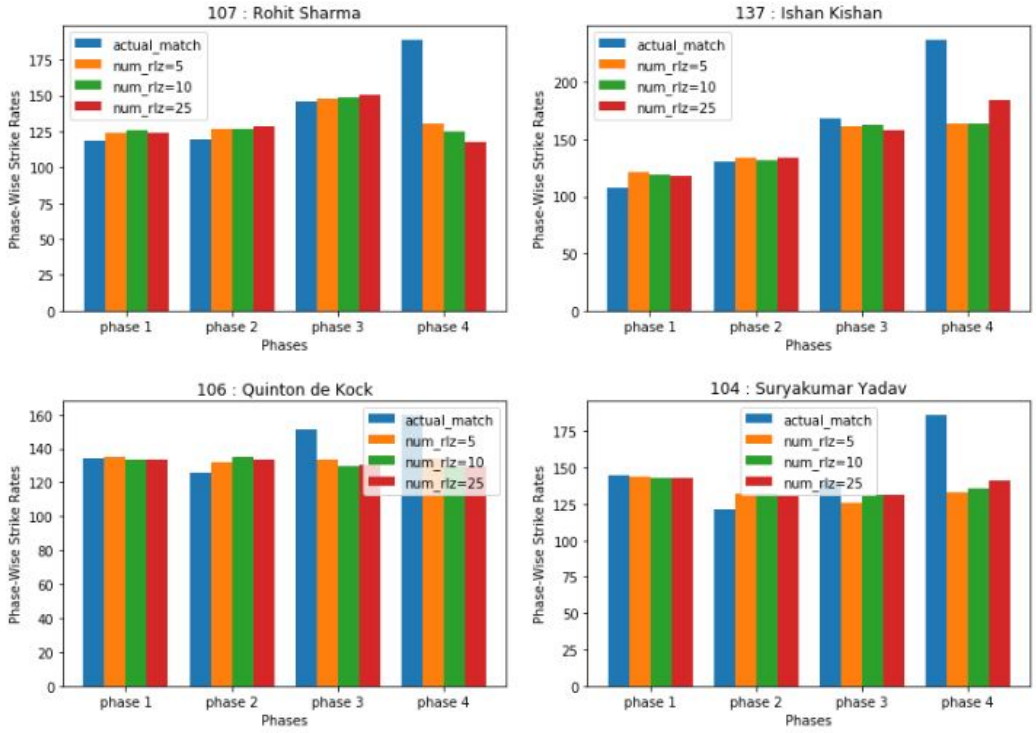

Figure 11: Phase-Wise Strike Rates of MI batsmen during IPL 2018-20

The reason behind this high error during the final few balls is not entirely known. The algorithm can be slightly modified to take care of this anomaly, but we leave it as future work. Overall, our algorithm does a decent work in predicting phase-wise strike rates from a given set of end-innings scorecards.

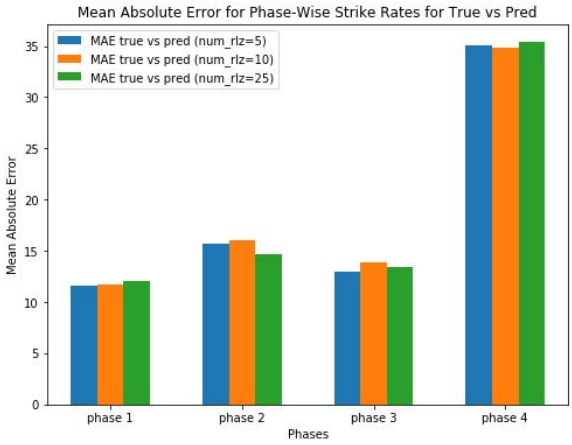

Figure 12: Mean Absolute Error between True and Predicted Phase-Wise Strike Rates

Figure 13: Comparison of top 15 players with highest strike rates in various phases

Figure 13 gives us the top 15 players with the highest strike rate in each of the defined phases. Similar to phase-wise average results, the same set of players are in the top 15 with a slight difference between true and predicted. Unlike phase-wise averages, many players have a very close strike-rates, so a combination of averages and strike rates in each phase will be a good indicator of player performance.

5.3 Phase-Wise Strike Rates Vs Averages

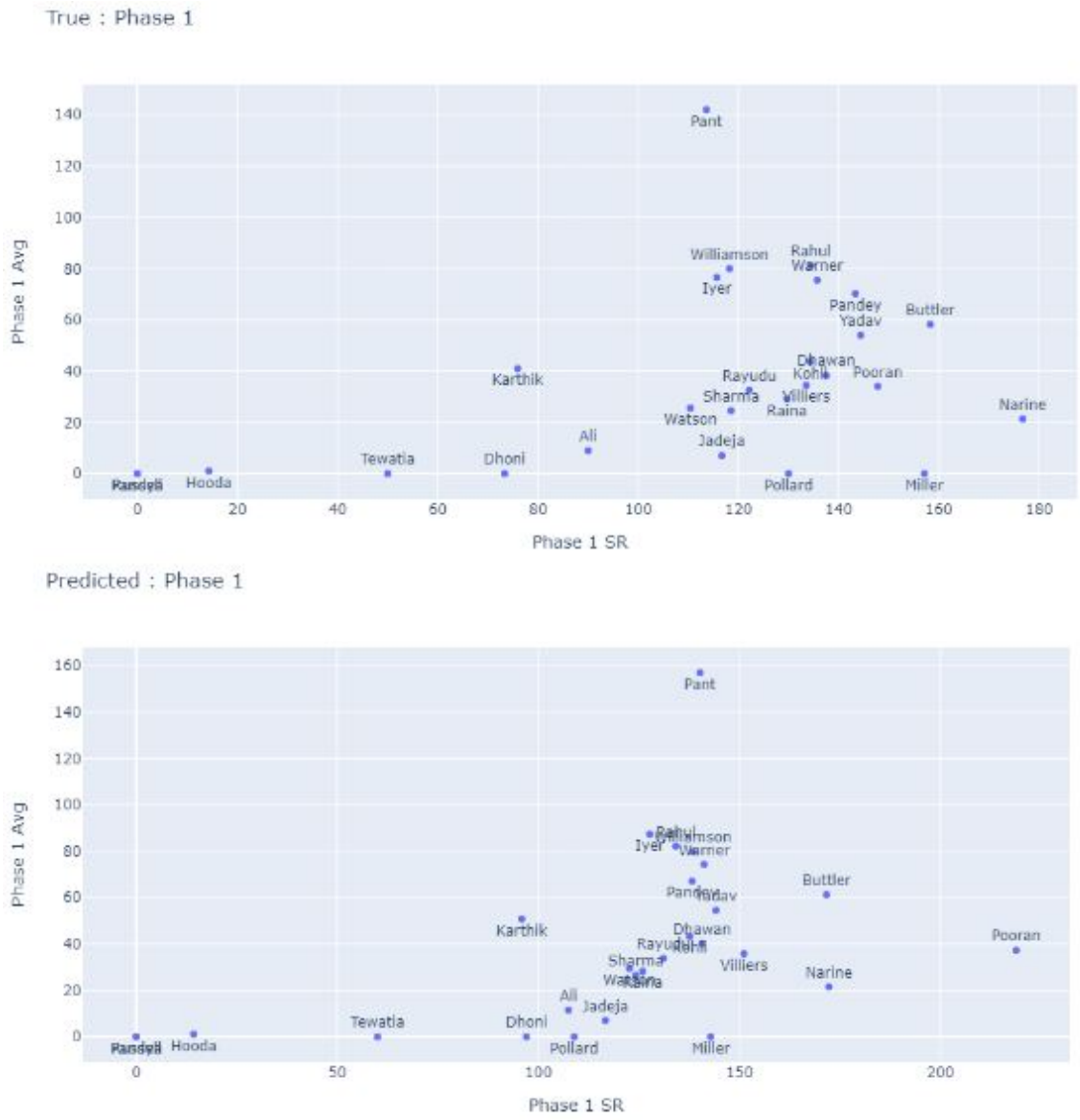

Figure 14: Phase-Wise Strike Rate Vs Average Plots for Phase 1

Figure 15: Phase-Wise Strike Rate Vs Average Plots for Phase 2

Figure 14-17 gives a combined plot of players average and strike rate with strike rate in x-axis and average in y-axis. These figures have plots generated from true and predicted ball-by-ball data for all the 4 phases. These plots give a clear indication of players performance in different phases relative to other players. Team management can use these kinds of results to form a well-rounded squad as per their requirements and team plan.

We can observe from these plots that the relative performance of most of the players is similar in the true and predicted plots. This is true across all the phases. As expected, some of the players positioning does not match in the true and predicted plots, but overall our algorithm has performed well in predicting players performances from the ball-by-ball data using only end-of-innings scorecards as inputs.

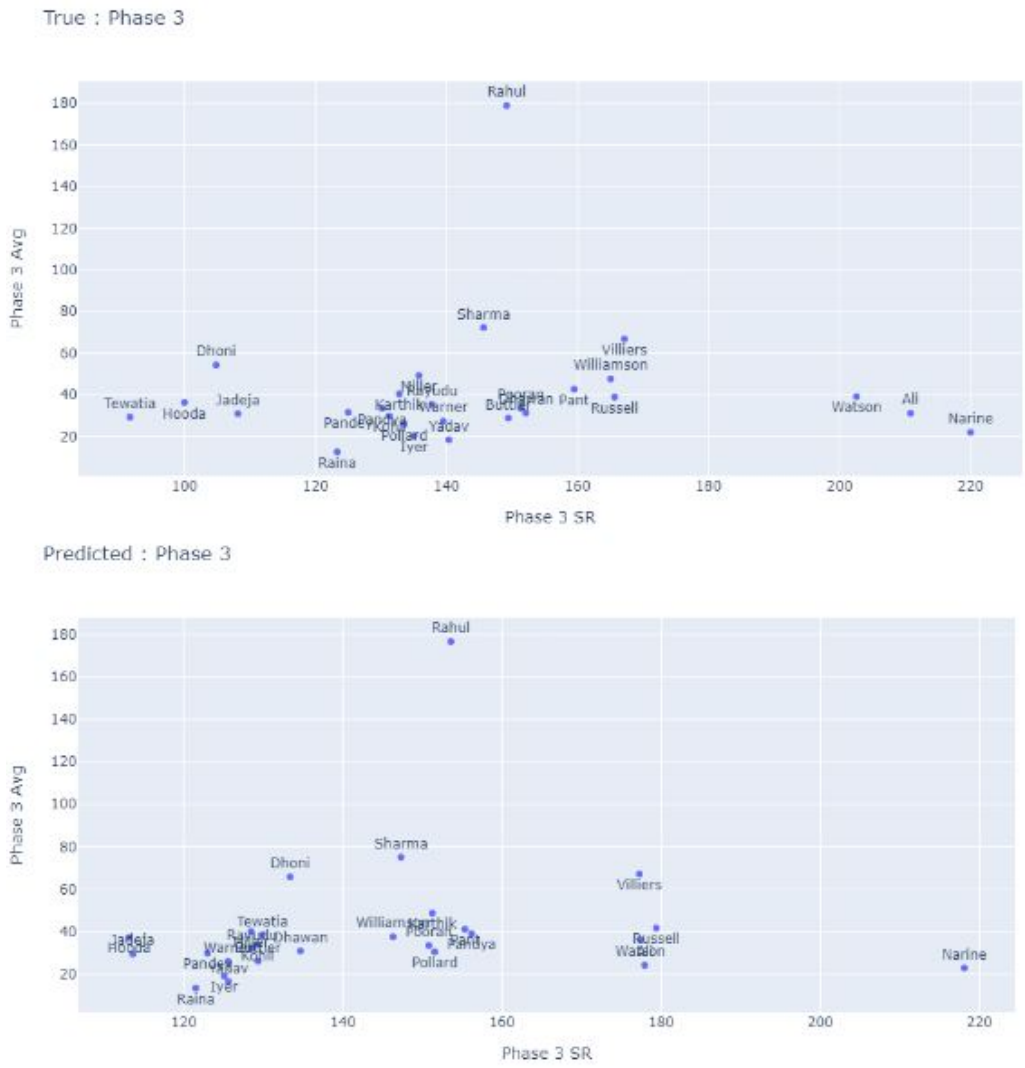

Figure 16: Phase-Wise Strike Rate Vs Average Plots for Phase 3

6 Conclusion

The ball-by-ball data gives the complete summary of a cricket innings and can be used to derive different sets of batting and bowling metrics of players. These metrics are used in various capacities by the team management to form a well-rounded squad as well as an ideal playing XI. In one of my previous collaborative works, we developed an algorithmic framework to predict end-of-innings score and win probabilities in a live match. This framework is further used in the development of Smart Stats, a suite of advanced metrics for context-informed evaluation of players' batting and bowling performances.

Thus, the use case of our work is multi-fold. We first generate ball-by-ball data using a set of

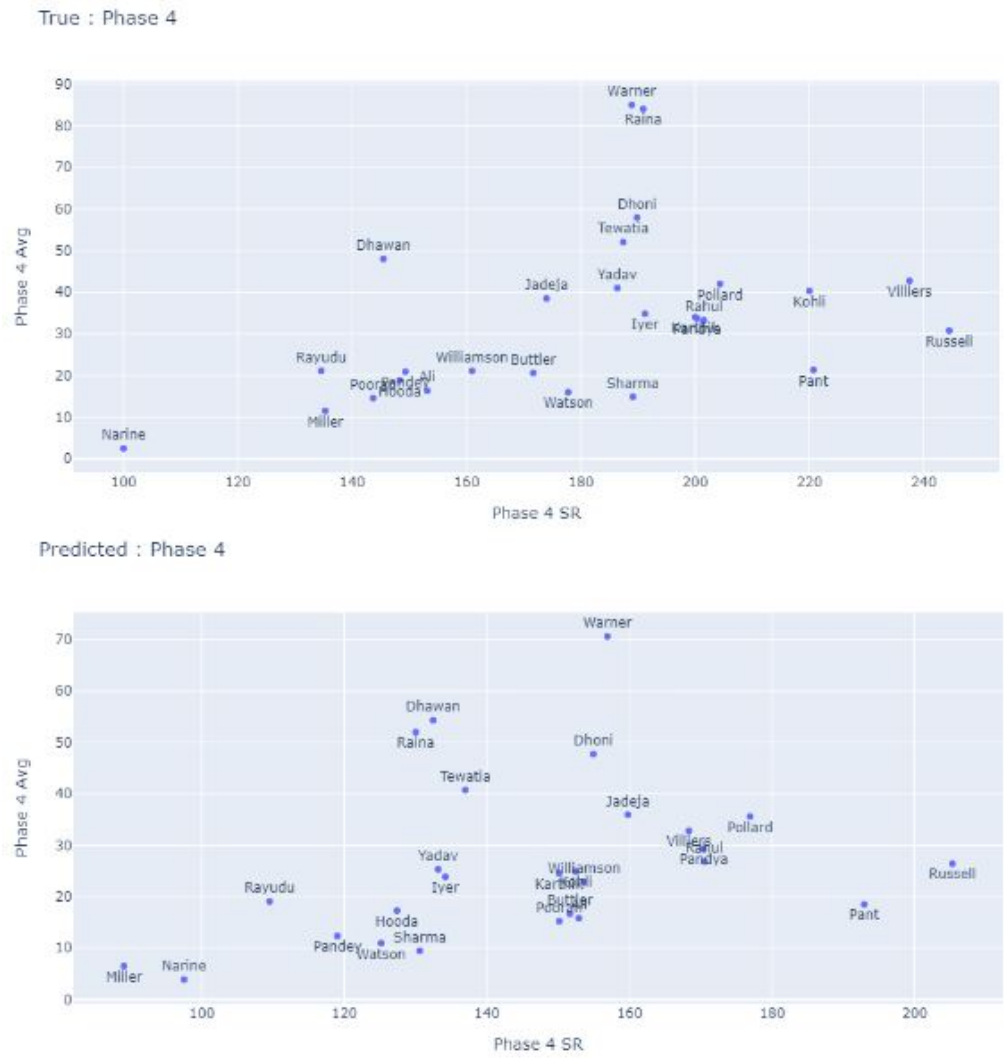

Figure 17: Phase-Wise Strike Rate Vs Average Plots for Phase 4

end-of-innings scorecards. As far as we know, this is the first of its kind work done in the field of cricket analytics. We further validated our work by generating and comparing phase-wise average and strike rates. We also observed that our algorithm gives satisfactory performance in different kinds of results generated through our algorithm. The focal point of our work is the generation of ball-by-ball data from an end-of-innings cricket scorecard.

Such ball-by-ball data is only available for international matches and some marquee T20 leagues around the world. Domestic tournaments like Syed Mushtaq Ali Trophy (SMAT) and Tamil Nadu Premier League (TNPL) only maintain the end-of-innings scorecard. Our scorecard problem attempts to solve this issue by predicting the ball-by-ball outcome from the end-of-innings scorecard. This database can then be used by team management to build a well rounded squad in player auctions.

7 Future Work

The overall problem of predicting the phase-wise averages and strike rates by predicting the ball-by-ball outcome from the end-of-innings scorecard is a complex one. We have taken a step-by-step approach to solve this complex problem. The fundamental idea is derived from stochastic simulation where we try to generate multiple possible ball-by-ball outcomes, wherein each prediction is randomized and different from each other. Then we compute phase-wise averages and strike rates from these predicted ball-by-ball outcomes. The main use case for this is in tournaments/matches where only scorecard information is available, SMAT and TNPL tournaments to name a few.

We observed that there is a slight anomaly in phase 4 predictions, predominantly generated from the final 6 balls of the innings. This is a possible future work to identify and rectify this problem with our algorithm. Another possible future work is incorporating the bowling scorecard into the algorithm. It will introduce additional constraints and will increase the complexity of the algorithm, but finding a suitable method to incorporate the details in the scorecard might give some interesting results.

Further, we have used a particular approach for the generation of ball-by-ball outcome from an end-of innings scorecard. Another possible future work is to identify other possible approaches for the same problem statement. Other approaches might lead to some interesting results and observations.

8 References

[1] https://en.wikipedia.org/wiki/Cricket

[2] https://en.wikipedia.org/wiki/Cricket statistics

[3] https://www.cricbuzz.com/live-cricket-scorecard/31622/mi-vs-dc-final-indian-premier-league-2020

[4] Wikipedia contributors. "Syed Mushtaq Ali Trophy." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia

[5] WASP (cricket calculation tool). Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 10 Nov. 2021.

[6] Michael Bailey. Stephen R Clarke. Predicting the Match Outcome in One Day International Cricket Matches, while the Game is in Progress. December 2006.

[7] Tim B. Swartz, Paramjit S. Gill and Saman Muthukumarana. Modelling and simulation for one-day cricket.

[8] Jack Davis, Harsha Perera, Tim B.Swartz. A Simulator for Twenty20 Cricket.

[9] Vignesh Veppur Sankaranarayanan, Junaed Sattar,Laks Lakshmanan. Auto-play: A Data Mining Approach to ODI Cricket Simulation and Prediction.

[10] White Ball Analytics. T20 Match Simulator : under the hood

[11] Davis, J., Perera, H. and Swartz, T.B. (2015b). Player evaluation in Twenty20 cricket. Journal of Sports Analytics.

[12] Explained: From Talent Scouting, Selection Trials and Mock Auctions – How IPL Franchises Identify and Spend Big Bucks on Unknown Players. News18. CricketNext.