Investigating the Use of Gaussian Process Classification in Cricket Run Prediction

John H. R. Steward

MComp Computer Science with Placement
2022-2023

Investigating the Use of Gaussian Process Classification in Cricket Run Prediction

Submitted by John H R Steward

Copyright

Declaration

This Dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this Dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

The Duckworth/Lewis method (D/L) for target-resetting in limited-over cricket is an inherently flawed system that is burdened by its necessary simplicity. It was devised as a method that can be executed exclusively on pen and paper and therefore misses key data that is required to effectively predict the correct target score for the batting team. We propose a method that uses Machine Learning (specifically Gaussian Process Classification) to predict the outcomes of interrupted overs. The data we will use takes specific player statistics into account, instead of using averages as is done with D/L. This means, in theory, it will give a more accurate measure of what would take place during the match had it not been interrupted, and therefore will give a fairer outcome to the match than if D/L is used to reset the target. The overall goal of our model is to simulate overs that have been interrupted by rain, so play can continue as it normally would had there not been an interruption.

With our model, we have achieved a similar accuracy in our predictions as other Cricket prediction algorithms, despite attempting to predict a more precise outcome than them, which typically are predicting win or loss. We believe there can be a lot to improve with our model to take it above the previous work in Cricket prediction, by improving our data to therefore be able to simulate an over effectively. There is a lot of room for Cricket prediction to be improved and this dissertation is a large step towards such improvements.

Contents

Acknowledgements

I would like to thank my supervisor Emelie Barman for her continued support over the course of this dissertation.
I would also like to thank my family, for calming me in my stressful spiral the week before submission, and to my friend Alex for sending me various funny videos to keep me sane.
I would finally like to thank my housemates for supporting me and getting me to work when I didn't want to.

1 Introduction

1.1 Rules

In a game of T20 cricket, the batting team has 20 'overs' of 6 balls each to score as many runs as they possibly can, while the bowling team is attempting to minimise the number of runs scored and get the batting team out as quickly as possible. Runs can either be scored by the batsmen running between the wickets (See figure 1) after the ball is played or, if the batsman hits the ball to the boundary (the outside edge of the pitch), they will receive either 4 or 6 runs[15], depending on if the ball hits the ground before making it past the boundary. If the ball hits the ground before passing the boundary, the batting team will receive 4 runs while, if it passes the boundary first, they will receive 6 runs.

Figure 1: Batsmen running between the wickets

The game must be played on a dry pitch so, in the case of rain, the game needs to be stopped until the rain clears, after which the teams will carry on as they were. In some circumstances, however, rain causes an interruption that means that the full game cannot be played, so one team will have a lower maximum number of overs than the other. Without a system in place to make things more fair, this would give a large advantage to the team who's batting was not interrupted.

1.2 Current Solution

In such a case, the Duckworth/Lewis method (D/L) is currently employed. This is a target-resetting method first utilised in 1997 to alter the target score for the team with the lower maximum overs, in an attempt to make it as fair as possible[6]. This method makes use of the notion of "resources" (how many wickets and balls the batting team has left) and average batting rates in certain situations in order to calculate a new target for the batting team. This encounters problems when the game in question is particularly high-scoring as, in

these games, the run rate (number of runs scored per over) shows a pattern of straightening towards average run rate that is not uniform across the innings[17]. D/L does not take this lack of uniformity into account, and therefore can make incorrect predictions for these kinds of games. The Duckworth/Lewis/Stern method (DLS) came to be used from 2017 for these situations, however it still is not perfect.
This method still only uses data for the average run rates across all players in T20 or One Day International (ODI), which is another form of Cricket where each side has a maximum of 50 overs, depending on the form of the game in which it is used, so it cannot take the specific situation into account in terms of who is playing during the interrupted overs.
DLS is only used in circumstances where there is a higher-than-average scoring rate in the match. Otherwise, the standard D/L method is used, meaning the issues talked about with D/L are still present within the game.
A large issue with D/L is the amount of weight it places on the number of wickets lost. For example, in the 2019 ODI World Cup qualifiers, Scotland needed 74 runs off 88 balls when rain interrupted the game, a very achievable score. Scotland lost their 5th wicket before the rain interrupted and Scotland lost the match by D/L. Had they not lost this wicket and been on the same score when the rain interrupted the match, Scotland would have won the match and qualified for the World Cup [4].

1.3 Our Solution

In this dissertation, we propose an approach that uses Machine Learning to predict the outcomes of each ball during interrupted overs in T20 Cricket, in this case specifically, the Indian Premier League (IPL), given that our data only comes from the IPL. This method will make use of Gaussian Process Classification in order to predict the number of runs that will be scored for each ball, and add that result onto the batting team's total. This means that, instead of resetting the target for the batting team to pursue, the target will remain the same, but runs will be added to the batting team's total based on how many runs they would score had the game not been interrupted.

We will be using ball-by-ball data from the 2022 season of the IPL, using this data to create feature vectors for all of the players involved. This means we can make use of the matchups between players within the Machine Learning model, instead of having to use the average data across all players in every circumstance. Within this data, instead of only taking each batsman's average run rates over the course of the season, we also use their overall strike rates ((number of runs scored/balls faced) as a percentage) for certain situations during a match and against certain types of bowlers, making it easier to predict the outcome with a batsman up against a specific bowler.

1.4 Dissertation Overview

In section 2, we will review how Machine Learning is currently used in sport analytics and how it can be improved, as well as different ways of pre-processing our data in order to make it more manageable to work with, while also looking in-depth into D/L and its issues. In section 3, we detail the data we will be using for our models and how we have processed this data. In section 4 we detail our hypotheses. In section 5, we detail the metrics we will use to evaluate our models, and in sections 6 and 7 we show how our models perform on their

corresponding test datasets, as well as new data within a simulation of an over from the 2023 IPL season, and analyse the meaning behind these results.

2 Literature, Technology and Data Survey

2.1 Target-Resetting Principles

There are two main target-resetting principles: probability-based and resource-based.

2.1.1 Probability-Based

Probability-based methods, such as WASP (Brooker and Hogan, 2012)[5], rely on preserving the batting team's "probability of victory" across the interruption. This would mean that these methods do rely on the number of runs currently scored by the batting team, as the probability of victory relies on that. This type of method is flawed as, over the course of a match, win probability changes significantly every ball, as it is continuously calculated based on the number of runs scored, the number of balls remaining and the number of wickets lost[2]. As shown in [17], over the three scenarios given, with a team chasing the exact same target, being given the same resources and finishing on the exact same score, each scenario gives a different result in the match, showing that probability-based methods are logically inconsistent.

2.1.2 Resource-Based

Resource-based methods, which include D/L, DLS and the Jayadevan system (VJD), rely solely on the amount of "resources" available to either side. These rely on determining the proportion of these resources a team has compared to the other and scaling the targets accordingly, and such, these methods only require knowledge of the resources available[17], not any knowledge of the current run rate, or even total runs scored by the team that is currently batting.

2.2 Duckworth/Lewis Method (D/L)

The current accepted method for setting an adjusted target score for interrupted cricket matches is known as the Duckworth/Lewis method (D/L), devised by Frank Duckworth and Tony Lewis[6], now known as the Duckworth/Lewis/Stern method (DLS) when the method was adopted and adjusted by Professor Steven Stern in 2015.
The usefulness of D/L was reviewed in 2002. Within [6], it is claimed that D/L was a fair system since there was roughly a 50/50 split between the team that batted first winning, and the team that batted second winning. While on the surface this seems like the method works as intended and is completely fair, confirmation of the fairness of this system would require a much deeper analysis of the fixtures in which D/L was used, as it is very possible that it was more likely for the first team to win most matches, which would mean that D/L would be unfairly weighted in favour of the second team. It is generally seen as the most effective and fairest method of deciding interrupted matches that is currently used, with the International Cricket Council (ICC) and the Board of Control for Cricket in India (BCCI) rejecting the use of VJD in 2011 since it was considered to not perform better than D/L and

so D/L is still used to this day.
According to the graphs shown in [6], the D/L model of average runs scored when early wickets have been lost significantly underestimates the true amount of runs scored (See figure 2), whereas the model significantly overestimates the true amount of runs scored for late wickets (See figure 3). This overestimation is explained in [6] as due to the fact that there are significantly fewer data points for late overs compared to early overs[6].

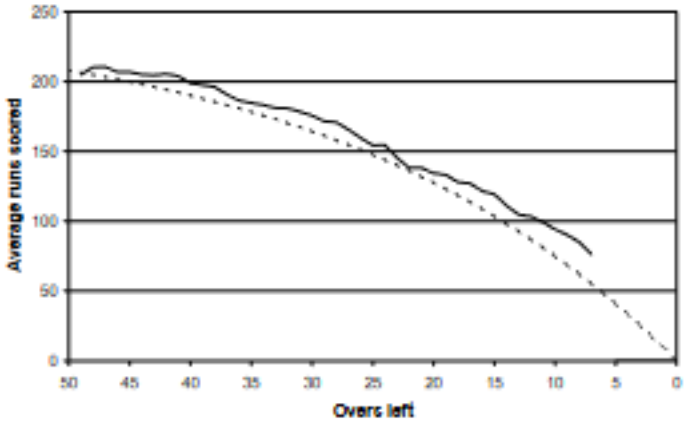

Figure 2: A graph showing the average runs scored after losing the first wicket(Solid line), and the D/L model for the same situation(dotted line) [6]

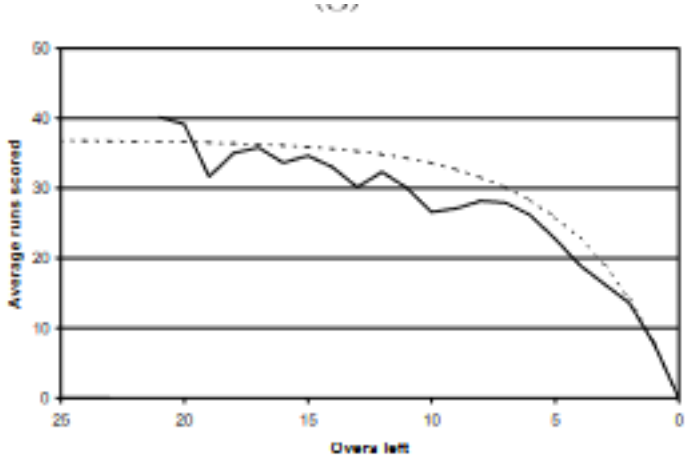

Figure 3: A graph showing the average runs scored after losing the eighth wicket(Solid line), and the D/L model for the same situation(dotted line) [6]

D/L can be done entirely on pen and paper, removing the reliance on computers, allowing there to be no fear of technical issues. This also means that it can be used at all levels of the game. Back when [6] was written in 2002, not everyone had access to computers that allowed the use of complex prediction methods, so having an effective way of deciding interrupted games that could be done by hand was very important, which is what D/L offered, however this causes the method to miss out on key details such as the specific player matchups. This accessibility is much less important in the modern day, as there is much

more advanced technology available even to the general public, which allows the use of more complex prediction methods for interrupted cricket matches. There were still issues with D/L, as it only takes into account the average run rates[6], meaning that, if team 1 scores well above average and team 2 is interrupted, team 2's par score that they would need to keep up with according to D/L would be very low compared to what they would actually need to score if the match were not interrupted, making it much more manageable for team 2 to win the match (Example shown in section 6.2 in [6]). Professor Steven Stern attempted to remedy this issue when he adopted D/L in 2015 and the method was updated to be DLS and was first implemented in the game in 2017.

2.3 Duckworth/Lewis/Stern Method (DLS)

D/L was updated to include the D/L professional edition, which added a "match factor" which attempts to 'straighten' the par curve towards the average run-rate line in a uniform manner. However, realistically, the 'straightening' of the run-scoring pattern does not actually occur uniformly throughout the innings, as run rates tend to be quicker at the start of an innings and slow down throughout[17]. This is because there is a "power play" at the beginning of an innings which restricts the number of fielders allowed outside the inner circle, causing teams to start more aggressively than how they finish an innings.
For this reason, DLS adds an additional damping factor to the D/L professional edition, which is a function of the number of overs remaining , so acts differentially throughout the innings. The DLS formula for the expected runs with u overs remaining and w wickets down in M-over matches, with a final total of $S_1 > Z_{std}(M, 0)$ is given by:

$$Z_{dls}(u, w, \lambda) = Z_0 F_w \lambda^{n_w+1}\{1 - e^{-ubg(u,\lambda)/F_w\lambda^{nw}}\}$$

where

$$g(u, \lambda) = (\frac{u}{50})^{-(1+\alpha_\lambda+\beta_\lambda u)}$$

with $\alpha_\lambda = -1/\{1 + c_1(\lambda - 1)e^{-c_2(\lambda-1)}\}$ and $\beta_\lambda = -c_3(\lambda - 1)e^{-c_4(\lambda-1)}$. The $\alpha, \beta, g(u, \lambda)$ and c constants are based on a detailed examination of run-scoring patterns in high scoring matches[17]. In particular, the $g(u, \lambda)$ function, as it depends on u, it allows for the decay constant to be differentially adjusted throughout an innings to more closely match observed patterns[17]. DLS is only used for high scoring games, while D/L standard edition is still used for games that score either below or in line with the average, as run-scoring patterns more closely match the average run rates, so D/L is still generally effective.
However, since DLS is only used for high scoring games, there still exists the issue that D/L has of underestimating runs scored when early wickets are lost, and overestimating runs scored when wickets are lost late on[6]. This still leaves the issue of differing strategies for when a team believes it is going to rain, where they try to preserve wickets until it rains, causing them to have an easier target than if they tried to bat normally and lost a wicket. For target resetting methods, we want to preserve the fundamental nature of the game, so we want to force the players to bat as they normally would, as if play were not interrupted. This would mean that there would be no obvious advantage for either team having the game interrupted.

2.4 Machine Learning Applications in Sport

Almost all sport prediction algorithms use some kind of feature extraction or feature selection on the data before processing it[8]. This may be the case with our data, since there are many features to begin with, and so may be necessary to reduce it to its most representative features in order to process it effectively. The prediction algorithms analysed in [8] (See figure 4) are also treated as classification problems, as they are usually predicting which team is going to win. The problem at hand will also be a classification problem, however we are not going to predict which team is going to win, but what will be the outcome of each ball, be it a wicket or some amount of runs (from 0 to 6). This means that, given information about the batsman, the bowler and how far into the game the ball is bowled, we will predict how many runs the batsman would score off that ball, if any. ours is seemingly a more complex problem than those analysed in [8], as we are attempting to predict something much more specific than the outcome of a game, and it is a multi-class classification problem rather than binary, which is the case when predicting the outcome of a match, with the 2 classes being win or loss.

In the modern day, it is simple to find a lot of data for any sport, be it results of matches or more in-depth statistics, such as a scorecard in cricket, or individual player statistics in any sport. Making an effective win prediction algorithm can have a significant effect on the betting community if made public, however, the prediction methods presented in [8] are not publicly available, so cannot be used by the general public for betting purposes. For example, one of the feature selection methods presented in [8] is based on expert experience. This means that the features used in the model are chosen by experts as being the most important features to be used in the prediction algorithm. This adjusted dataset is not made publicly available so cannot be exploited to win money from betting companies. Neural Networks are the most used machine learning algorithm used in sport prediction (See figure 4).

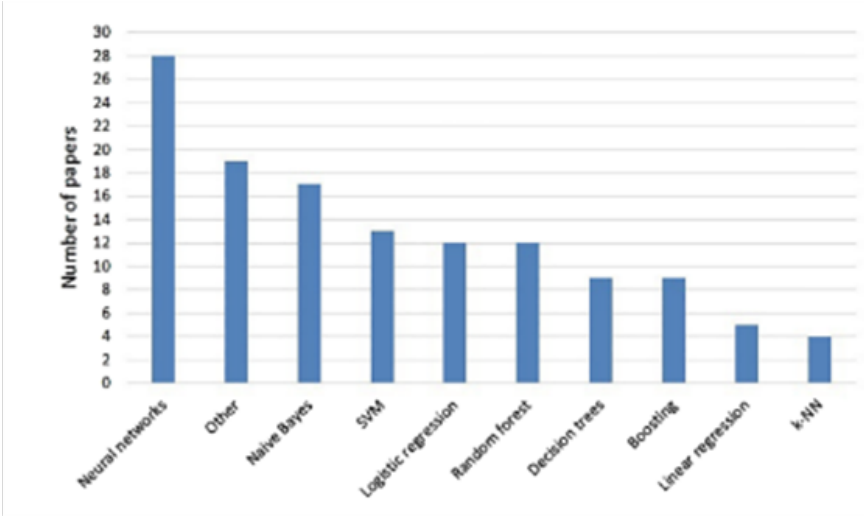

Figure 4: Number of papers using a particular ML group [8]

For predicting the outcome of Champions League group games in football, when using feature selection based on expert experience, Artificial Neural Networks (ANNs) achieved an accuracy

of 68.8%. This is very similar to the accuracy of methods that were used for English county T20 Cricket [9]. This would be far too low to be used in any official sense, such as being used to predict the outcome of a ball in Cricket for when the game is interrupted, therefore we would need to use a more sophisticated method of feature selection than are presented in [8]. All methods presented in [8] involve removing features from the initial dataset, so it is possible that, for higher dimensional data, a dimensionality reduction/feature extraction method such as Principal Component Analysis (PCA) could be more effective than these feature selection methods.
Given the volatile nature of Cricket, there are many factors that go into who is more likely to win a match, making it one of the more difficult sports of which to predict match outcomes (See figure 5).

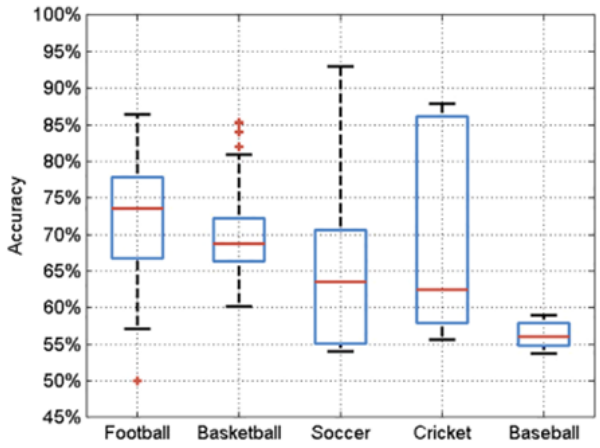

Figure 5: Box plots of achieved maximum accuracies by sport [8]

As the box plots show in figure 5, there is a very large Inter-Quartile Range of maximum accuracies achieved by different methods when predicting Cricket results, with the mean value being the second lowest of the sports presented, being approximately 62.5%.
Predicting the outcome of a single ball will be a very different prospect to predicting the outcome of a match, since there is a much smaller scope, and more context of the game as a whole. We also have more in-depth data for the IPL than general statistics from the scorecard of a match, as detailed in section 3.2. In [8], there is no clear advantage to using feature extraction over feature selection as there are models that achieve lower and higher accuracies in both categories. If we were to require reducing the number of features in our data, we will likely use a feature extraction method such as PCA as it reduces the dimensionality further than feature selection, while retaining a very similar amount of information.

2.5 Data Pre-processing

Our data requires some pre-processing, such as cleaning it up by removing unnecessary features, and adding features that will be useful to our predictions. This is elaborated on further in section 3.2. Also, given our large number of features and datapoints, it may be be necessary to perform feature selection or feature extraction in order to speed up the runtime

of training our model, while also not removing so much information that it would become detrimental to our predictions.

2.5.1 Chi-Square Test

The Chi-Square test is a numerical test that measures deviation from the expected distribution considering the feature event is independent of the class value[18]. The Chi-square metric is calculated as:

$$t(t_p, (t_p + f_p)P_{pos}) + t(f_n, (f_n + t_n)P_{pos}) + t(f_p, (t_p + f_p)P_{neg}) + t(t_n, (f_n + t_n)P_{neg})$$

Where $t(count, expect) = (count - expect)^2/expect$, P_{pos} is the probability of number of positive cases, P_{neg} is the probability of number of negative cases, t_p is the number of true positives, f_p is the number of false positives, t_n is the number of true negatives and f_n is the number of false negatives.
Once the hypothesis has been specified (meaning we have defined what we are trying to predict), we devise an analysis plan which specifies the significance rank, which tends to be 0.01, 0.05 or 0.10[18], but can be any value $x : 0 \leq x \leq 1$. The plan must also specify the test method, being the chi-square test.
We then analyse the sample data, calculating the degrees of freedom(DF), predictable frequencies, test value and P-value associated with the test[18].

$$DF = (r - 1) * (c - 1)$$

Where r is the number of levels of one categorical variable[3], and c is the number of levels of the other categorical variable. The test statistic is defined as:

$$\chi^2(f, c) = \frac{N * (AD - CB)^2}{(A + C)(B + D)(A + B)(C + D)}$$

Where A is the number of times feature f and class label c co-occur, B is the number of times f appears without c, C is the number of times c appears without f, D is the number of times that neither occur, and N is the total number of records[18].
At each time instance, one input feature is removed from the data. This new data is then trained and tested using this chi-square approach and the input features are then ranked based on the performance metrics of the chi-square test[18].
The chi-square test is a simple method for feature selection which ranks the importance of the input features, allowing us to remove the least important features while keeping the ones that are most representative of the data. The complexity of this method is $O(n^2)$ where n is the number of features, meaning it is quite inefficient for high-dimensional data. Given that this is such a simple and inefficient method, that only allowed a maximum accuracy of prediction from the data in [9] of around 65%, this method may be unfeasible for the results that we are attempting to achieve. With high-dimensional data, we may also need to completely remove many input features to justify the use of a feature selection method such as this to significantly reduce the time taken to predict unknown inputs. However, doing so may lose too much information to be able to accurately predict new data.

2.5.2 Principal Component Analysis (PCA)

PCA is a well-established method for dimensionality reduction, often used for image processing and pattern recognition[19]. PCA computes new features, called "principal components", as linear combinations of the features from the original dataset [1]. The first principal component is the one that has the largest variance, and extracts the components that separate the data the best.
The covariance matrix of every pair of features is computed, where the covariance of a feature with itself is equal to the variance of that feature so, along the diagonal, we have the variance of each feature. We then compute the eigenvectors of the covariance matrix and their corresponding eigenvalues. The eigenvalues are used to rank the principal components in order of importance, with the largest eigenvalue corresponding to the most important principal component. PCA also has a time complexity of $O(n^2)$ where n is the number of features, however, for high-dimensional data, it tends to be more effective in reducing the number of features than the Chi-Square test, as it has the same complexity, but can reduce the number of features further while keeping a similar amount of information. This means that a model will train quicker and will, in theory, be similarly accurate.

2.5.3 Probabilistic PCA

We can obtain a probabilistic formulation of PCA from a Gaussian latent variable model[19]. Within this framework, instead of only ranking the principal components by variance, we get maximum likelihood parameter estimates which are computed in the same way[19]. A benefit of this model is that, as well as being used for dimensionality reduction, we can use probabilistic PCA to efficiently calculate the maximum likelihood estimates from the principal components. This can be applied to classification problems such as ours.
PCA defines a linear projection, which is quite simplistic, so to help with more complex projections, we can combine multiple PCA models. This is readily implemented as a mixture of probabilistic PCA models[19]. This method of dimensionality reduction could be useful for a more complex problem such as ours.

2.6 Gaussian Processes

Gaussian Processes define the prior probability distributions over non-linear functions implied by the Bayesian Neural Network[11]. A non-linear function $y(x)$ parameterised by parameters $\mathbf{w}$ is assumed to underlie the data $\{x^{(n)}, t_n\}_{n=1}^N$, and the adaptation of the model to the data corresponds to an inference of the function given the data. If the input vector is defined as: $\mathbf{X}_N = \{x^{(n)}\}_{n=1}^N$ and the corresponding labels as $\mathbf{t}_N = t_{n}{}_{n=1}^N$, then the inference of $y(x)$ is described by the posterior probability distribution:

$$P(y(x)|\mathbf{t}_N, \mathbf{X}_N) = \frac{P(\mathbf{t}_N|y(x), \mathbf{X}_N)P(y(x))}{P(\mathbf{t}_N|\mathbf{X}_N)}$$

[11]

Where $P(\mathbf{t}_N|y(x), \mathbf{X}_N)$ is the probability of the labels given the function $y(x)$, and $P(y(x))$ is the prior distribution on functions assumed by the model. The prior typically specifies that $y(x)$ is expected to be smooth and continuous. The simplest type of prior is a Gaussian Process, which can be thought of as a generalisation of a Gaussian Distribution over a finite

vector space, to an infinite function space. A Gaussian Process is defined by a mean and a covariance function, where the mean is a function of x, and the covariance is a function $C(x, x')$ which is the expected covariance between the points x and x' on the function y.

Gaussian Processes are already well established models for various temporal problems[11], which can be well related to our domain of Cricket, as we can assume that each datapoint in our dataset, being a specific ball that is bowled, is part of a time series, the context for which can be found within the data from the delivery information.
Given that our model will need to be non-parametric (See section 3.2), predictions will be obtained without giving $y(x)$ an explicit parameterisation. One well known approach to estimating $y(x)$ is spline smoothing. We define the estimator of $y(x)$ to be the function $\hat{y}(x)$ that minimises the functional:

$$M(y(x)) = -\frac{1}{2}\beta \sum_{n=1}^{N} (y(x^{(n)}) - t_n)^2 - \frac{1}{2}\alpha \int dx [y^{(p)}(x)]^2$$

where $y^{(p)}$ is the pth derivative of y and $p \in \mathbf{N}$, therefore, if $p = 2$, then $\hat{y}(x)$ is a cubic spline. This estimation method can then be turned into a Bayesian method by identifying the prior function for $y(x)$ as :

$$\log P(y(x)|\alpha) = -\frac{1}{2}\alpha \int dx [y^{(p)}(x)]^2 + const$$

[11]
and the probability of the data measurements $\mathbf{t}_N = t_n{}_{n=1}^{N}$ assuming independent Gaussian noise as:

$$logP(\mathbf{t}_N|y(x), \beta) = -\frac{1}{2}\beta \sum_{n=1}^{N} (y(x^{(n)}) - t_n)^2 + const$$

[11]
where the constants are functions of α and β respectively. We elaborate further on how we use Gaussian Processes in section 3.3.

3 Our Data

3.1 New Approach

Our new approach, instead of aiming to reset the target score for the batting team, aims to use the data that is available (section 3.2) to find the probability of each outcome of a ball (be that a dot ball (0 runs), 1, 2, 3, 4 or 6 runs, or a wicket) and add to the batting team's run total while chasing the same target as they had initially, being one more than the run total set by the first team. This, in turn, would help to solve the issue that D/L has of underestimating runs when early wickets are lost[6], since it will take into account batsman/bowler matchups, instead of just looking at the average run rates. It also will not take into account the win probability, as it is very volatile, so assuming it will be preserved across the interruption is logically flawed as this is extremely unlikely[17].

3.2 Available Data

Our main dataset is an in-depth ball-by-ball analysis of every ball from the IPL in the 2022 season. This includes the batting team, the bowler, the positions of every fielder, the number of runs scored from that ball, the delivery type(how the ball was bowled) and whether the batsman played the ball. The dataset also included the trajectory data for each ball as they were bowled, however, these would need to be removed for our model as we cannot use trajectory data for the ball as input data for predictions, as the ball has not been bowled yet, so trajectory data does not exist. There were also other pieces of data that we removed, since they were either consistent across all of the dataset, or were unnecessary to predicting the outcome of a ball. For example, we removed the names of the league and the country they play in as they were all played in India, in the IPL. We also removed the names of the teams playing as they have no bearing on how many runs are scored.
Using this data, a feature vector for every batsman and bowler was made, storing their attributes such as, for batsman, their overall strike rate, the distribution of runs from early-game overs (the first 5 overs), mid-game overs (overs 6-15) and late-game overs (the last 5 overs), and how aggressive each player is against each delivery type (ratio of attacked balls : defended balls). For bowlers, the attributes calculated were: economy (number of runs conceded/number of overs bowled), and the ratios of dot balls, balls that conceded 1 run, 2 runs, 3 runs, 4 runs, 6 runs and those which yielded wickets. Once all of these attributes were calculated for every player, they were added to the dataset to give more information about the players involved. These feature vectors are also individually stored, with bowler vectors and batsman vectors separated, so that we can easily access each players' attributes for the purposes of predicting new datapoints.

The data here is not normally distributed (See figure 6), so a non-parametric model will be required, such as Gaussian Process Classification.

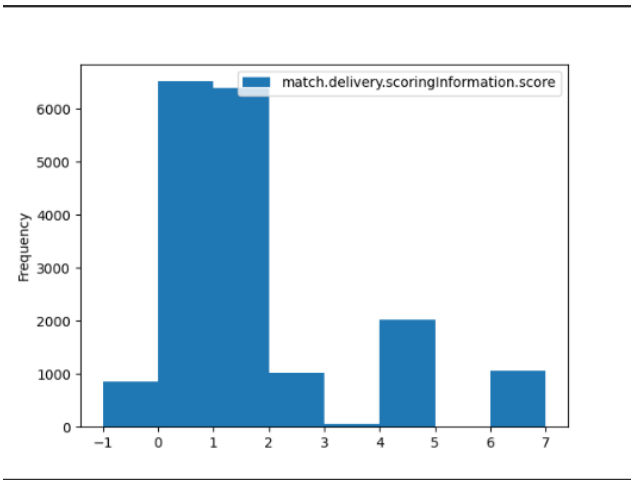

Figure 6: Frequency of each score, -1 to 6 (where -1 means a wicket was taken), yielded by the balls over the 2022 IPL season

3.3 Gaussian Process Classification

Gaussian Process Classification is a non-parametric, probabilistic model that can be used for both binary and multi-class classification. We have already discussed in section 2.6 that Gaussian Processes are used to specify prior distributions for Bayesian inference. However, with a classification problem, the posterior is non-Gaussian, so exact inference is analytically intractable. For this reason, we need to be able to approximate the non-Gaussian posterior in order to deal with classification problems. Since we will be using Gaussian Process Classification in SciKit Learn[7], we will be approximating this using the Laplace approximation. Laplace approximation uses a second order Taylor expansion around the posterior mode $\mathbf{m}$ to naturally construct a Gaussian approximation to the log-posterior $\Psi(f) = \ln \mathbf{P}(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta)$ where θ is the hyperparameters of the model[13]. The mode $\mathbf{m}$ is taken as the mean of the approximate Gaussian. Linear terms of Ψ disappear because the gradient at the mode is 0. The quadratic term of Ψ is given by the negative Hessian $\mathbf{W}$ which, due to the likelihood's factorial structure, is diagonal. $\mathbf{m}$ is found by Newton's method[12]. The posterior is calculated as:

$$\mathbf{P}(\mathbf{f}|\mathbf{y}, \mathbf{X}, \theta) \approx \mathcal{N}(\mathbf{f}|\mathbf{m}, (\mathbf{K}^{-1} + \mathbf{W})^{-1})$$

where

$$\mathbf{m} = \arg\max_{\mathbf{f}\in\mathbf{R}^n} \mathbf{P}(\mathbf{y}|\mathbf{f})\mathbf{P}(\mathbf{f}|\mathbf{X}, \theta)$$

and

$$\mathbf{W} = -\frac{\partial^2 \ln \mathbf{P}(\mathbf{y}|\mathbf{f})}{\partial \mathbf{f} \partial \mathbf{f}^\top}\bigg|_{\mathbf{f}=\mathbf{m}} = -\left[\frac{\partial^2 \ln \mathbf{P}(y_i|f_i)}{\partial f_i^2}\bigg|_{f_i=m_i}\right]_{ii}$$

[13]
The un-normalised posterior $\mathbf{P}(\mathbf{y}|\mathbf{f})\mathbf{P}(\mathbf{f}|\mathbf{X}, \theta)$ has a maximum $h = \exp(\Psi(\mathbf{m}))$ at $\mathbf{m}$, where the gradient is 0. A Taylor expansion of Ψ is then given by $\Psi(\mathbf{f}) \approx h - \frac{1}{2}(\mathbf{f}-\mathbf{m})^\top(\mathbf{K}^{-1}+\mathbf{W})(\mathbf{f}-\mathbf{m})$. The log marginal likelihood of the model can then be approximated by substituting the approximation for $\Psi(\mathbf{f})$:

$$\ln Z = \ln \mathbf{P}(\mathbf{y}|\mathbf{X}, \theta) = \ln \int \mathbf{P}(\mathbf{y}|\mathbf{f})\mathbf{P}(\mathbf{f}|\mathbf{X}, \theta) d\mathbf{f} = \ln \int \exp(\Psi(\mathbf{f})) d\mathbf{f}$$

$$\approx \ln h + \ln \int \exp(-\frac{1}{2}(\mathbf{f}-\mathbf{m})^\top(\mathbf{K}^{-1}+\mathbf{W})(\mathbf{f}-\mathbf{m})) d\mathbf{f}$$

$$= \ln \mathbf{P}(\mathbf{y}|\mathbf{m}) - \frac{1}{2}\mathbf{m}^\top\mathbf{K}^{-1}\mathbf{m} + \frac{1}{2}\ln|\mathbf{I} + \mathbf{K}\mathbf{W}|$$

[13]
The model that we are implementing from SciKit Learn[7] does not implement a true multi-class Laplace approximation, but will instead fit multiple binary "one-versus-rest" Gaussian Process Classifiers using Laplace approximation. This means that it will create a prediction model for each of our classes against the rest and, using the probabilities from these predictions, we can choose the model with the largest probability to of the "one" to be the class that is predicted. For example, if we had 3 classes: A, B and C, if "A versus (B or C)" has a probability of 0.2 for A, "B versus (A or C)" has a probability of 0.3 for B, and "C versus (A or B)" has a probability of 0.5 for C, our multi-class model would predict C, as it has the largest probability. Note that the probabilities of the separate classes individually will always sum to 1.

4 Experimental Hypothesis

We have not encountered any previous work that looks to predict the number of runs scored from a given ball, only those that look to predict the outcomes of an overall match, therefore we cannot directly compare our work to any existing models that attempt to achieve the same outcome. Previously we saw that Machine Learning models that are used in sport prediction are often deterministic rather than probabilistic, such as logistic regression[8]. This is logically flawed as, given that human error will always be an element when playing a sport, there is an inherent probabilistic aspect within predicting the outcome of a game. Running a deterministic model to predict an outcome in sport removes a crucial aspect of any game. Given the in-depth nature of our data, and the choice of a more complex probabilistic model, we expect our model to perform more accurately than the models used currently within sport to predict outcomes of games. Even though we cannot directly compare our approach to those used already as they are attempting to predict something different, we are still able to determine the accuracy of our model in the same way as we would with any model. We can give it some test data and allow it to predict the number of runs scored, and compare the prediction to the true outcome, to calculate the accuracy. Unlike models that predict the outcomes of matches, our model will allow us to simulate entire overs by predicting the number of runs scored on a given ball and adjusting who the batsman is based on the number of runs predicted for that ball, since we have stored a feature vector for each batsman and bowler, which we can use as input data for the next ball (See section 5).

Our hypothesis is that our model will perform more accurately than other models that are currently used in Cricket[8], and that it will accurately predict the number of runs that will be scored from a given ball. We expect our model to predict this well enough to be considered a more accurate target resetting technique than the D/L or DLS methods that are currently used within limited-over Cricket, given that those methods only use data from the averages across T20 and ODI, while our method will use more specific data from the IPL and its players to give more context to the game.

5 Design of Experiments

Since our model is being proposed as a replacement to the current target resetting system, we want to be able to simulate the outcomes of full overs, not just specific balls in isolation. Since our dataset is ordered by earliest balls in the matches to latest balls (i.e first ball of every match, then second ball etc.), we will need to shuffle it before training so that our training set can be representative of the entire dataset. Given that this shuffle is random, we will train multiple models to find the one that consistently performs the best. Given that the time complexity for training our model is $O(n^3)$ where n is the number of datapoints, we will train our model on different amounts of the data and test the difference in accuracy to see whether it is worth using extra time to train our model on a greater proportion of the dataset, or whether we can train it on less, greatly reducing the amount of training time, without losing much accuracy.
When training a new model, we will need to store the corresponding training and test sets to ensure that we do not use a datapoint from its training set when testing. The reason we need to store the datasets is that each time the code is run, there is a new shuffle for the training

and test sets. This means that, if we try to run a model multiple times without reloading the same dataset, there is a very high chance that there will be datapoints from the original training set, now within the test set. If this is the case, our evaluation of the model will be inaccurate due to this data contamination. We will train 2 models with each proportion of the data used for training (being 30%, 50% and 70%), to see which is more consistent and more accurate. We will then need to test how well they can simulate full overs of an IPL match. To do this, we will hand-pick the first ball from an over in the 2023 IPL season, from 3 different points during 2 matches and simulate the full over. We will then see how many runs were actually scored during that over, and check against the amount of runs our model predicts would be scored given the first ball of the over. This means that, after the first ball of the over, it is possible that the next ball will be a completely new datapoint that has not been seen by any model.

In order to simulate an over, we will predict how many runs would be scored from the first ball of an over and, depending on what it predicts, we will adjust who would be batting for the next ball, retrieving the feature vector for that batsman and using it for input data for the next ball. For example, if our model predicts that 1 run would be scored off the first ball, then the batsmen would switch, but if it predicts that 2 runs were scored, then the batsman would stay the same. We will do this until the end of the over, adding up the number of runs scored as we go, then compare the result with the true number of runs scored from that over. We will not update the feature vectors for the players based on these predicted outcomes as these may not be exactly what happened during any match, meaning the addition to their statistics would be false.

We will use our models to simulate overs from 2 matches in the 2023 IPL season: one that was played fully with no interruptions, and one that was decided by D/L. We will simulate 3 overs from from an innings in each match: one from an early-game over, one from a mid-game over, and one from a late-game over (see section 3.2). We will use each model to simulate the same overs from the same matches and determine which model is the most accurate in predicting the outcome of an over.

We will not be able to directly measure the efficacy of our model against the current target resetting methods using code as they are very different methods, and D/L uses a lookup table to change the target that the batting team needs to aim for. This means that our main metric for the efficacy of our model is how close it is to simulating each over accurately. We can look intuitively at how well it compares to the usage of D/L in the IPL, but we cannot show any concrete accuracy in this dissertation.

We will be training a total of 6 models, all of which will be tested on 30% of the original dataset. 2 of these will be trained on 30% of the original dataset, 2 will be trained on 50% of the original dataset, and 2 will be trained on 70% of the original dataset. These will all be tested on a separate 30% of the dataset than they were trained on to gauge a general accuracy for the different models before testing them them on a simulation of an over.

We are training 2 of each model in order to check if they are consistent in their accuracy and whether they are robust, or if the data it was trained on just happened to coincide well with the corresponding test set. The increase in training time is large as we add more datapoints, given the model's $O(n^3)$ time complexity, where n is the number of datapoints, so we want to see how significantly training a model on a greater subset of the data increases its consistency, to see whether it is worth taking longer to train, or if it achieves a similar accuracy and consistency when trained on a smaller subset of data. If a model performs just

as well when it is trained on a smaller subset, we may want to use this model in the future as it will take much less time to train on a new dataset, such as ones from ODI or different T20 leagues. The model will also take up less storage if it is trained on a smaller subset of data, and so will take less time to load in than models that are trained on a greater proportion of the data.
The only data that can change during an over in our dataset will be the delivery number of the ball, and which batsman is batting for the next ball. The bowler will always stay the same throughout a single over. If an odd number of runs is scored on a ball, then the batsmen will switch for the next ball. If an even number of runs is scored, then the batsman will stay the same. If we were to simulate multiple overs, then the batsmen would switch at the end of the over, and the bowler may be replaced by another.

6 Experimental Results

	Model30%1	Model30%2	Model50%1	Model50%2	Model70%1	Model70%2
ACC	63.6%	41.3%	49.1%	59.1%	58.8%	61.7%

Table 1: Accuracies of each model on their test set

Table 1 shows the accuracy of each model on their corresponding test set, where, for example, "Model30%1" means that this was the first model that we trained on 30% of the dataset.
To ensure that there was no data contamination within the datasets when not using the entire dataset, we split the data into two sets, using one to train the model, and splitting the other again to form the final test set. This ensured that none of the training datapoints would be used within the test set, and our evaluation of the model will be accurate, as we could guarantee that all of the testing datapoints are new to the model. We only needed to split the data twice when training on 30% and 50% of the dataset as, when we train on 70%, we use the rest of the dataset for testing, still guaranteeing that there is no overlap. We shuffle our dataset before training so that we can train it on datapoints all throughout a match, instead of just training on early-game overs. This will make our model more generalisable to new datapoints, however late in an innings it may be. If we were to not shuffle the dataset, when we input a testing datapoint from late on during a match, the model would be less likely to be able to predict the number of runs scored accurately, as it has not encountered a datapoint from a late-game over on at all during training.
When training the model on 30% of the dataset, it takes an average of 37 minutes to train. This short training time has quite a large effect on testing as it gives very inconsistent results, with one of the models yielding a 63.6% accuracy, with the other yielding a 41.3% accuracy on their respective test sets. This shows that, when training on such a small subset of the data, results are very dependent on the test set, meaning that the model is not very generalisable to new data. When we train on 50% of the dataset, it took an average of 3 hours to train, with the first model yielding an accuracy of 49.1%, and the second yielding an accuracy of 59.1%. These, along with the models trained on 30%, are quite inconsistent, highlighting the importance of utilising our entire dataset for training and testing. When training on 70% of the dataset and testing on the rest, the models take an average of 4.2 hours to train, with the first model yielding an accuracy of 58.8%, and the second yielding

an accuracy of 61.7%. This shows that, when we train the model on a larger subset of data, it is more generalisable to new data, given that it is relatively consistent with its accuracies compared to the other sets of models.

After training and testing these models, we simulated 3 overs from 2 matches from the 2023 IPL season to see how closely they match to the real outcomes of the overs.
In order to run these simulations, we had to retrieve the feature vectors from the batsmen and the bowlers that were involved for each over and create the input data for the first ball of the over, with the feature vectors for the batsman who will be first to bat, and the bowler. We then predicted the number of runs scored from that first ball and, depending on that prediction, we created the input data for the second ball to be predicted, and so on until the end of the over. The only features that we would change between balls would be the delivery number of the ball (increasing it by one for each ball bowled) and, if an odd number of runs is scored on a ball, we change the input data to include the feature vector of the other batsman that is in, instead of the one that batted the ball that was predicted. This meant we could automate the predictions of a full over instead of having to re-run our code every time we wanted to predict a new ball. If this method were to be used in Cricket in the future, we would still need to predict each over discretely, as the bowling team decides which bowler will bowl each over as it comes, so they would need to decide who to send in before we could predict a new over.
We ran simulations on 3 overs in the match between the Punjab Kings and the Kolkata Knight Riders, being overs 2, 12 and 16 of the Kolkata Knight Riders' innings (simulations 1, 2 and 3 respectively), which was decided by D/L. We also ran simulations on 3 overs in the match between the Kolkata Knight Riders and the Gujarat Titans, being overs 2, 8 and 16 of the Gujarat Titans' innings (simulations 4, 5 and 6 respectively), which was not interrupted by rain. A common trend in the predictions of an over in our models is that, if the model predicts that an even number of runs will be scored (and therefore the batsman will not change), then it will predict that same score for the rest of the over. For example: when we run simulation 2 using our models that were trained on 30% of the dataset, they predicted that the first ball of that over would yield 6 runs. For this reason, very little data changes for the next ball, with the only feature that changes being which ball of the over is being bowled. This means that our model then predicts that the next ball will also yield 6 runs, leading to our model predicting that 36 runs were scored in total for that over when, in reality, only 8 runs were scored. The true amount of runs scored in the overs used for the simulations 1, 2 and 3 were 4, 8, and 10 respectively[14]. The true amount of runs scored in the overs used for simulations 4, 5 and 6 were 10, 7, and 13 respectively[10].

	Sim 1	Sim 2	Sim 3	Sim 4	Sim 5	Sim 6
Model30%1	1	36	6	6	6	36
Model30%2	0	36	0	0	0	36
Model50%1	24	24	24	24	6	36
Model50%2	24	36	31	24	6	36
Model70%1	1	6	6	6	6	6
Model70%2	1	36	36	36	6	36

Table 2: Number of runs predicted by our models over the course of our simulated overs

Table 2 details the number of runs yielded from our simulations for each model, with the

naming conventions remaining the same as in table 1. There were only 4 instances where a model predicted a different number of runs within an over, being the first simulation with our first model trained on 30% of the dataset, which predicted that the first ball would yield 1 run, and the rest would yield 0. Another instance was in our third simulation using our second model trained on 50% of the dataset, which predicted the first ball would yield 1 run, and the rest would yield 6 runs. Both of our models that were trained on 70% of the dataset varied on the first simulation, predicting that the first ball would yield 1 run and the rest would yield 0 runs.

7 Analysis of Results

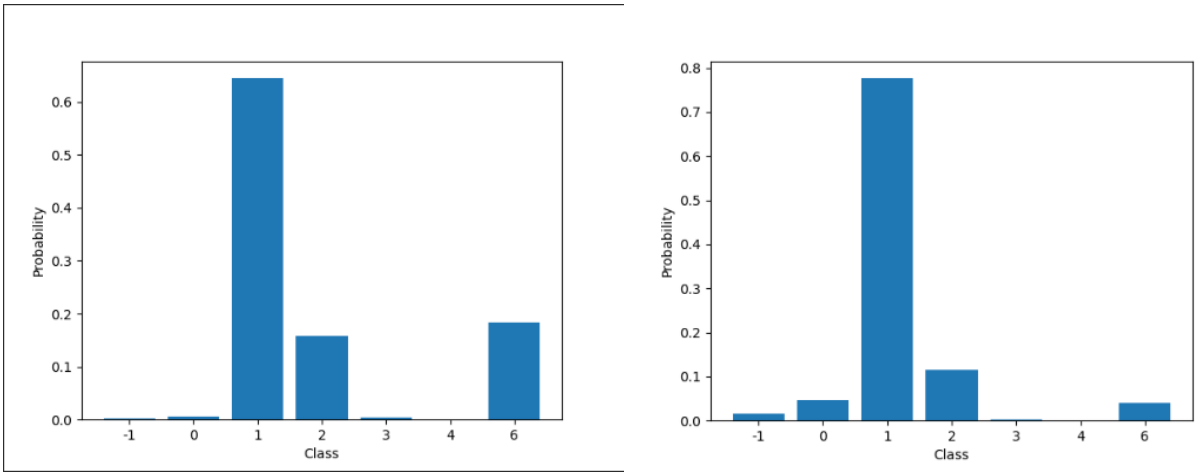

Figure 7: Probabilities of the different classes predicted by the first model trained on 70% of the dataset, on the first 2 balls of our 6th simulation

Figure 7 shows the prediction probabilities of each individual class as predicted by our first model predicted on 70% of the dataset. Judging by these probabilities, the model is confident on which class is the most likely, with a much larger probability than the rest. These probabilities change slightly per ball, as shown in figure 7 but, most of the time, not significantly enough for the model to change its prediction for the next ball, which means that the model gets stuck predicting the same outcome for each ball. The small change in probabilities per ball is due to the fact that there is a very small change in the data when an even amount of runs predicted, given that the batsman and the bowler will stay the same. In our dataset, if the batsman stays the same, the only thing that will change per ball is the number of the delivery, so the model predicts that the same outcome will occur as before. We believed that including overall player statistics within the data for each ball would be enough to give the models context to predict the outcome effectively, but the variation in the data was not enough to make the datapoints very distinct from one another, causing our models to predict the same outcome for each ball of an over the majority of the time.

In its current state, there is no significant evidence from our models for us to be able to reject the null hypothesis that states that D/L is the fairest and most accurate system that can be used within Cricket at this time [6]. The main issue with the data as it is now, is that it does not include any context for each individual ball. This context could include

the number of runs that the batting team has at that moment in the game, the number of wickets they have lost, or the number of runs scored from the last ball, for example. In order for us to be able to keep a track of the context in which the ball is, we would need to fully restructure the dataset to be pre-processed game-by-game instead of just processing the overall batsman and bowler statistics to be placed within the data for each ball. If the data were pre-processed per game, we would be able to keep track of how many runs had been scored up to each ball in the innings and store it in the data for that ball. We would also be able to do the same with the total number of wickets at that moment in the match. Also, if we store the balls for the games in chronological order, we would be able to keep track of running batsman and bowler statistics and store them in the data for the ball with their current statistics at that time, instead of storing their overall statistics for the entire dataset in every ball that they are involved in. If data is processed chronologically, we would also be able to store partnership statistics between two batsman, such as the number of balls faced as a partnership at that point during the match, and the number of runs scored between them at that point. Using these values, we could then calculate a strike rate for the partnership at that time as well.

If we are able to include these metrics in our dataset in order to give our models more context for each ball, then there will be more variations between each datapoint, giving the models more of a chance to not become stuck predicting the same number of runs for every ball. Given the probabilistic nature of sport, we could also use the probabilities returned by the "predict_proba" function[7] as weights and randomly choose the class using these probabilities. For example, in the first simulation with the first model trained on 30% of the data, the first ball would have a 14.7% chance to yield a wicket, a 23.1% chance to yield 0 runs, a 41.1% chance of yielding 1 run, and so on. This would be another way of preventing the models from becoming stuck predicting the same outcomes. If we are able to restructure the data in order to process it with more context for each ball, and run it using the probabilities that are returned, it could be a much more accurate representation of the game than the models we have achieved with our current data and prediction technique.

Our most accurate model that was trained on the majority of the dataset, achieved an accuracy of 61.7% on its test set. While this is not an ideal result and cannot be used to effectively simulate an over in its current state, it still achieves a similar accuracy to models presented in [8], being only 0.8% lower than the mean (See figure 5), despite attempting to predict a much more precise outcome with many more classes. While we are attempting to predict the outcome of every ball, the papers presented in [8] are attempting to predict the outcome of a match, which is a binary metric in limited-over Cricket (win or loss). We believe that, with our choice of model, and if the data were restructured in the ways we outlined in this section, we could, in future, create a model that performs better than the models showcased in [8], despite attempting to predict a more difficult metric, and be able to simulate overs much more effectively.

8 Conclusions

This dissertation set out to create a Machine Learning model, using ball-by-ball data from the 2022 IPL season, to be able to predict the number of runs scored off of a specific ball and, in turn, be able to simulate an entire over. The purpose of this was to potentially create a replacement to the D/L and DLS methods that are currently in place within limited-over

Cricket with a more accurate representation of what would occur during interrupted overs, rather than reducing the batting team's target score based solely on the amount of "resources" they lost.
We proposed that using player stats from the season in our data would provide more context to each ball, and therefore be a fairer metric in resetting the target for the batting team than D/L. In its current state, our model is unsuccessful in providing a fairer metric than D/L, given that our best model, that is trained on the majority of our dataset, yields an accuracy of 61.7% on its test set and its simulations of the overs we chose (see section 6) were inaccurate.

Through our literature review, we discovered that Cricket is one of the most inconsistent of the popular sports to predict match outcomes using Machine Learning, with the highest accuracy seen being approximately 87%, and the lowest being approximately 56% (See figure 5). This shows that our model, which is attempting to predict a much more precise outcome compared to those reviewed in [8] and is predicting a multi-class model as opposed to a binary model, can still achieve very similar accuracies to these simpler models, we believe, due to our choice of model being more complex and specialised to our data than many of those seen in [8]. We chose to utilise Gaussian Process Classification in SciKit Learn[7] to train our model as, since our data was not normally distributed, it required a non-parametric model. Also, given that our domain is Cricket, which always has some kind of inherent probability involved, we needed a probabilistic model to train on our data, and Gaussian Process Classification allowed for both of these criteria, along with Gaussian Processes already being an established model for priors in time series data, which ours is (see section 2.6).
As noted in section 7, our model did not perform accurately when attempting to simulate the outcome of a full over, because there was not enough variation in the data from ball to ball when the model predicts that the first ball will yield an even number of runs, causing it to continually predict the same number of runs for each ball. The only instances where our model predicted a different number of runs between 2 balls in an over were the 4 instances outlined in section 7 where the model predicted the first ball to yield 1 run. This was because, when the model predicts 1 run to be scored, the batsmen will switch and the batsman's partner will become the batsman, so the statistics for the batsman will change, giving the new datapoint enough variation from the previous one to allow the model to change its prediction.
Given the large number of datapoints in our dataset, it may have been useful to use Sparse Gaussian Process Classification[16], in order to speed up the training time of our models, instead of just training them on a smaller subset of the data, potentially causing the training data to not be truly representative of the dataset as a whole, as seen by the inconsistency of our models that were trained on 30% and 50% of the dataset, compared to our model that was trained on 70% (table 1). Another method we could have used to reduce the training time of our models, which would reduce it less significantly than using Sparse methods, would be to perform some kind of feature selection or extraction (See section 2.5). These would also lead to more accurate models than if we were to train on less of the dataset, but would reduce the training time less significantly. However, with the data in its current state, these methods would not have helped to increase the accuracy of our models, only to reduce the training time slightly. Hence, the first thing we would do to improve our results would be to restructure our data in order to accommodate chronological data, as outlined in section 7, to allow more context to be added for each ball to ensure datapoints can be more distinct from each other.

In conclusion, our model performs similarly to other Machine Learning models used to predict outcomes in Cricket, despite attempting to predict an outcome that is more precise than the outcome of a match. We did not achieve our aim of predicting the outcome of an over accurately enough to be able to create a potential replacement for D/L by predicting what would have occured had the game not been interrupted. However, with improvements to our data, including adding chronological context to each datapoint and adding more of a probabilistic aspect to prediction, we could retrain our models and potentially create a model that will predict the number of runs scored much more effectively. This would require a complete restructure of our data and more pre-processing. Implementing these improvements would likely require a significant amount of testing and research, that could be done in future work. This dissertation has offered valuable insight into how Cricket data should be viewed from a Machine Learning perspective, rather than just from a sport perspective and, we hope, will be the first step in creating an accurate run prediction algorithm in the future using the framework discussed here.
(Word count: 9539)

References

[1] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[2] M. Asif and I. G. McHale, "In-play forecasting of win probability in one-day international cricket: A dynamic logistic regression model," *International Journal of Forecasting*, vol. 32, no. 1, pp. 34–43, 2016.

[3] R. Bortolotti, "Tutorial k - data prep 2-2: Dummy coding category variables," in *Handbook of Statistical Analysis and Data Mining Applications (Second Edition)*, R. Nisbet, G. Miner, and K. Yale, Eds., Second Edition, Boston: Academic Press, 2018, pp. 497–514, ISBN: 978-0-12-416632-5. DOI: https://doi.org/10.1016/B978-0-12-416632-5.00033-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124166325000335.

[4] "Controversial cricket matches that were affected by d/l method," [Online]. Available: https://sportstar.thehindu.com/cricket/top-controversial-cricket-match-endings-duckworth-lewis-method-2003-world-cup-west-indies-john-dyson/article31240580.ece.

[5] *Cricket and the wasp: Shameless self promotion (wonkish)*. [Online]. Available: http://offsettingbehaviour.blogspot.com/2012/11/cricket-and-wasp-shameless-self.html.

[6] F. Duckworth, "Review of the application of the duckworth/lewis method of target resetting in one-day cricket," in *Proceedings of the Sixth Australian Conference on Mathematics and Computers in Sport*, 2002, pp. 127–140.

[7] *Gaussian process classification in scikit learn*. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessClassifi
html.

[8] T. Horvat and J. Job, "The use of machine learning in sport outcome prediction: A review," *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 5, e1380, 2020. DOI: https://doi.org/10.1002/widm.1380. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1380. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1380.

[9] S. Kampakis and W. Thomas, *Using machine learning to predict the outcome of english county twenty over cricket matches*, 2015. DOI: 10.48550/ARXIV.1511.05837. [Online]. Available: https://arxiv.org/abs/1511.05837.

[10] *Kolkata knight riders vs gujarat titans*. [Online]. Available: https://www.iplt20.com/match/2023/894.

[11] D. J. MacKay *et al.*, "Introduction to gaussian processes," *NATO ASI series F computer and systems sciences*, vol. 168, pp. 133–166, 1998.

[12] J. J. Moré and D. C. Sorensen, "Newton's method," Argonne National Lab., IL (USA), Tech. Rep., 1982.

[13] H. Nickisch and C. E. Rasmussen, "Approximations for binary gaussian process classification," *Journal of Machine Learning Research*, vol. 9, no. Oct, pp. 2035–2078, 2008.

[14] *Punjab kings vs kolkata knight riders*. [Online]. Available: https://www.iplt20.com/match/2023/857.

[15] *Rules of cricket*. [Online]. Available: https://www.cricket-rules.com/.

[16] M. Seeger and M. Jordan, "Sparse gaussian process classification with multiple classes," Department of Statistics, University of Berkeley, CA, Tech. Rep., 2004.

[17] S. E. Stern, "The duckworth-lewis-stern method: Extending the duckworth-lewis methodology to deal with modern scoring rates," *Journal of the Operational Research Society*, vol. 67, no. 12, pp. 1469–1480, 2016.

[18] I. Sumaiya Thaseen and C. Aswani Kumar, "Intrusion detection model using fusion of chi-square feature selection and multi class svm," *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 4, pp. 462–472, 2017, ISSN: 1319-1578. DOI: https://doi.org/10.1016/j.jksuci.2015.12.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157816300076.

[19] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 61, no. 3, pp. 611–622, 1999. DOI: https://doi.org/10.1111/1467-9868.00196. eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00196. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00196.