Cricket Delivery Outcome Prediction Using XGBoost and Mixtures of Gaussian Processes: An Exploration of Model Performance

Louie Middle

Bachelor of Science in Computer Science
The University of Bath
2023

Cricket Delivery Outcome Prediction Using XGBoost and Mixtures of Gaussian Processes: An Exploration of Model Performance

Submitted by: Louie Middle

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

Cricket is a popular sport with a large following globally. In recent decades, sports analytics has become an established industry, utilising statistical and machine learning algorithms to extract meaningful insights from sports data. However, current research on machine learning in cricket and sport in general has overlooked the potential use of Gaussian processes. This project aims to fill these gaps by introducing Gaussian processes for modelling the outcome of cricket deliveries and provides the necessary background and method. This is done through exploring a state-of-the-art Gaussian process model: Data Association for Gaussian Processes. Models using XGBoost were also developed. This work provides a platform for future studies into the use of Gaussian processes in cricket analytics.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to acknowledge my supervisor Mr. Adam Hartshorne for his support and advice throughout the development of this project. His countless hours spent meeting with all his students, in addition to his availability on slack during the evenings and weekends is a testament to his dedication and passion towards his work. In addition, my partner Hoi has provided much personal support for me throughout this challenging final year. Lastly, I would like to thank my parents, for who without I would not have had the amazing oppurtunities in life that I have had, including University and the chance to work on this project.

Chapter 1

Introduction

Over the last few decades the amount of data driven techniques to improve the outcomes of sports games has increased greatly. The multi-billion pound market of cricket is no exception.

Cricket is a game of fine margins; matches can be decided by a small number of runs. Therefore, every insight that can be made as to what is good or bad cricket is extremely valuable. For this reason, there is a strong incentive to improve the machine learning techniques used in current sports analytics to improve a teams performance.

This study aims to investigate the possibility of predicting the outcome of a cricket delivery [1] using modern machine learning techniques and how this could then aid cricket bowling choices and team selection. The hope is that using knowledge of bowler pitch trajectories and batter shot trajectories gathered from modern ball tracking will add another layer of granularity in addition to simply considering the resultant runs scored off each delivery. Furthermore, building a model which can incorporate pitch, atmospheric and ground conditions could further improve any models.

1.1 Sports Analytics

Sports analytics is the use of data analysis techniques to gain insights and make decisions related to various aspects of sports. It involves the use of statistical and machine learning techniques to extract meaningful insights from large and complex datasets.

Sports analytics has become an integral part of the decision making process for many professional sports teams and organisations. By analysing data, owners, coaches and managers can make more informed decisions on talent recruitment, tactics, training regimes and more.

1.2 Drawbacks of Current Methods in Sports Analytics

Sport predictors are most commonly modelled using traditional statistical methods or supervised machine learning (ML) methods. ML approaches are typicallly neural networks, classification methods, regression methods or gradient boosting [Horvat and Job, 2020] (see sections 2.3

[1] Whilst knowledge of cricket is useful to understand this project, it is not necessary to appreciate the project and its applications. Despite this, I present a brief introduction to the game and its laws for those unfamiliar with the sport in chapter A. A delivery is a single action of bowling a ball toward a batter.

and 2.4). Whilst existing predictors can achieve impressive results, unpredictable outcomes in sport still happen. For example, bookmaker's odds of Leicester City Football Club winning the Premier League at the start of the 2015/2016 season were 5000-1. Despite this, Leicester won the Premier League that season, in one of the biggest sport surprises in recent memory. Further analysis of their performances using a data segmentation evaluation method in Ruiz et al. [2017] show their title was absolutely deserved. Examples like Leicester City winning the Premier League illustrate the limitations of current predictions in sport.

Currently in sports analytics and machine learning in general there is a popular set of methods that are frequently adopted. In particular, neural networks and gradient boosting algorithms are currently popular and have a large research output. Whilst they have gained popularity for their ability to solve complex problems and deliver accurate predictions, all methods have drawbacks.

1.2.1 Overfitting

One significant drawback of neural networks and gradient boosting models is that they can easily overfit the training data. This means the model has extracted some of the variation in the training data into the model structure, as if that variation was part of the actual underlying model structure. This can mean they fail to generalise well to new data. Processes exist that can alleviate this issue, for example regularisation [Girosi et al., 1995] techniques such as L1 or L2 regularisation, dropout or early stopping. This can help reduce overfitting, but these methods may not always be sufficient.

1.2.2 Discretisation of Continuous Features

Gradient boosting, decision trees and random forests all in some way discretise their input space by binning continuous features into intervals or categories. This allows these models to handle continuous variables as categorical variables or decisions, however this is another potential drawback. This discretisation can lead to a loss of information, as the fine-grained details of the continuous variables are lost. This can be particularly problematic for variables with high granularity or subtle variations, where the loss of information can result in reduced accuracy.

In the case of cricket, this could be a problem. For example, a perfect yorker [2] is a very difficult ball to play against. However, pitching fuller or shorter can turn the ball into a full toss [3] or half volley [4] respectively, which are both considered easier deliveries to score high runs off of. So in the case of discretising a feature such as the location of pitch bounces, it wouldn't appropriately represent the realities of this continuous feature.

In addition to discretisation, another way to handle continuous variables is by bucketing data into tabular format. This approach involves partitioning continuous variables into distinct ranges or "buckets" and assigning them categorical labels. This method can preserve more of the original information compared to discretisation, but bucketing can also be prone to certain issues such as overfitting and bias, particularly when the number of buckets is too large or

[2] In cricket, a yorker is a delivery which hits the cricket pitch around the batsman's feet.

[3] In cricket, a full toss is a delivery which does not pitch before it reaches the batsman.

[4] In cricket, a half volley is a delivery which hits the cricket pitch ahead of a good length, but before the batsman's feet.

too small. Furthermore, the choice of bucketing method and the number of buckets can be subjective, and may require experimentation and tuning.

1.2.3 Limited Data

Limited data is another challenge faced by machine learning models. In some cases, obtaining enough data to train a model can be expensive, time-consuming, or simply impossible. For example, if attempting to model player performance when a player may have only played a handful of professional matches. It is impossible to model such players with more data, until they have played more matches.

1.3 Potential Solutions

A potential solution is to use an entirely different approach to the methods that are currently popular in sports analytics, namely Gaussian processes (GPs). GPs are particularly effective in handling small datasets and can model complex distributions. One of the main benefits of Gaussian processes is their ability to incorporate uncertainty into the model. This means that instead of providing a single prediction, Gaussian processes can provide a range of possible outcomes along with the associated probabilities.

Although neural networks and boosting algorithms can be extended to attempt to predict with uncertainty, they often rely on heuristic methods, which may not be as accurate as a fully Bayesian approach. In contrast, Gaussian processes are inherently Bayesian, which means they can provide more accurate and reliable estimates of uncertainty.

1.3.1 Benefits of Gaussian Processes for Cricket Modelling

Cricket is a sport that involves many variables, including pitch conditions, weather, player skills, tactics, team selection and more. These variables can all impact the outcome of a game and as a result it can be challenging to model cricket accurately. Additionally, there may be inherent noise in the training data, such as errors in data collection or variations in player performance. Furthermore, different outputs can constantly overlap the same regions of input space.

In a case such as this, it would be extremely benefitial to develop a model that can make a prediction and provide some gauge as to how uncertain it is in that prediction, either based on large noise, variance, or lack of training data. For these reasons, Gaussian processes are an underutilised model choice for cricket. They have much to offer in their handling of limited and noisy training data and can provide probabilistic predictions.

1.4 Privacy in Sports Analytics

Whilst there is publically available research in the field of sports analytics, many professional sports teams and sports technology companies are secretive about their methods and data. This is for various reasons, but mostly to maintain an advantage over their competitors. For this reason, it is important to note that large amounts of current sports analytics work is not publically visible and often has to be gathered from various sources.

Chapter 2

Literature and Technology Survey

2.1 Moneyball

The release of the book Moneyball [Lewis, 2004] and its subsequent film adaptation, was a big driver in the increased use of statistical driven techniques in Baseball player selection and scouting techniques. This empirical analysis of baseball is known as sabermetrics and was pioneered by the likes of Bill James. The impact of Moneyball has meant that moneyball has entered baseball's lexicon; teams that value sabermetrics are often said to be playing "moneyball". One of the notable benefits of a moneyball approach is in reducing player salaries, whilst maintaining high performance. Notable recent successes of baseball teams that have adopted a moneyball approach include the Tampa Bay Rays, whose entire 2019 roster was around 63% of the $40 million their competitor the Houston Astros were spending soley on Gerrit Cole's contract. It was reported that the Tampa Bay Rays spending totalled $648,000 per win, compared to the Astros $1.54 million per win. Despite this large difference in pay, the Tampa Bay Rays still had a successful season and finished second in the American League East [Sports, 2019].

Similar successes can be found in other sports, such as association football (or soccer). In 2010 Liverpool F.C. were purchased by Boston Red Sox owner John W. Henry. During this time with the Red Sox, Henry hired sabermetrics pioneer Bill James and their moneyball approach saw the team win the World Series in 2004, 2007, 2013 and 2018. Attempting to replicate these results with Liverpool, Henry hired University of Cambridge PhD Ian Graham in 2012 as head of analysis and Jürgen Klopp as Manager in 2015 [News, 2022]. Graham influenced the signings of key players, such as Mohammed Salah, Philippe Coutinho and Naby Keïta. Graham's data suggested Salah would pair especially well with Roberto Firmino, who creates more expected goals than nearly anyone else in his position [Echo, 2019]. Expected goals turned to real goals in the 2017-2018 season, with Salah scoring 32 goals and Firmino scoring 15. The combination of Klopp and his intuitive knowledge, paired with the likes of Grahams data-driven knowledge, has led to Liverpool having fantastic recent success winning the 2018-2019 UEFA Champions League and the 2019-2020 English Premier League.

2.2 T20 Cricket

Whilst there are varying formats of cricket, the focus in this project is the T20 format (20 over cricket) that is played in the IPL. T20 cricket gives the first team 20 overs to score as many runs as possible, or until they have used all 10 of their wickets. The teams then swap roles and the opposing team attempt to "chase" the total runs plus one scored by the first team. If they manage to do so they win the match, but if all their batters get out before the target total, or use all 20 overs before reaching the target, they lose.

Longer formats of the game such as 40/50 over one day matches and four/five day matches have different tactics compared to T20 cricket. Due to T20s shorter nature, a bowling team's goal is generally to minimise the runs the batting team can score, in addition to the traditional goal of taking wickets. Typically, once a batter settles into [1] their innings they attempt to score more boundaries [2] to maximise the runs scored in the short playing period. Furthermore, some "power hitters" are selected for their specialism in scoring boundaries at the end of an innings, with no settling in period that is afforded to batters in longer formats or at the start of a T20 innings. Whilst this style of play is more risky as the chances of misplaying increase, it is generally more beneficial to score as many runs as possible in the short time T20 cricket allots. In general T20 batters protect their wicket less than those in other formats and play a riskier, high run scoring style of play. Irvine and Kennedy [2017] and the general consensus of the cricket community is that this high boundary scoring carries a greater importance than scoring singles [3]. Irvine and Kennedy [2017] found that the higher the number of singles scored was detrimental to a batting team's chances of success.

Due to the batters desire to score boundaries in T20, bowlers might adopt different strategies to reduce the batters ability to score boundaries. This is often done through bowling a wider variety of deliveries and aiming for dot balls [4] or singles. Consequently, this should reduce the runs the batters score and increase the chance of taking a wicket due to the batter needing to play a wider variety of shots to score runs, which increases the chance of them making a mistake. In general, T20 bowling attacks often assess the ways to reduce the number of boundaries the opposing team can score. Irvine and Kennedy [2017] found that increased number of dot balls led to a higher chance of winning for the bowling team.

2.2.1 Indian Premier League

The Indian Premier League (IPL) is a professional cricket league based on the T20 format. As reported by ESPNcricinfo [2018], Star Sports invested $2.55 billion for exclusive broadcasting rights for the 2018 IPL season. This season saw a 29% increment in the number of viewers, through both digital streaming and television. The interest in the IPL is clear. This increases the desire to use modern machine learning techniques to improve results.

[1] "Settling in" refers to the process where a batter takes time to adjust to the conditions of the pitch and the opposition bowling. Whilst settling in a batter plays more cautiously, playing more defensive shots whilst they get their eye in, before then playing more aggressively.

[2] A boundary is a delivery that has been hit all the way to the boundary rope for 4 or 6 runs (see section A.3.2).

[3] A single is when the batters run one time between the wickets, scoring one run (see section A.3.2).

[4] A dotball is a delivery that went for 0 runs (see section A.3.2).

2.3 Existing Machine Learning Research on Sport

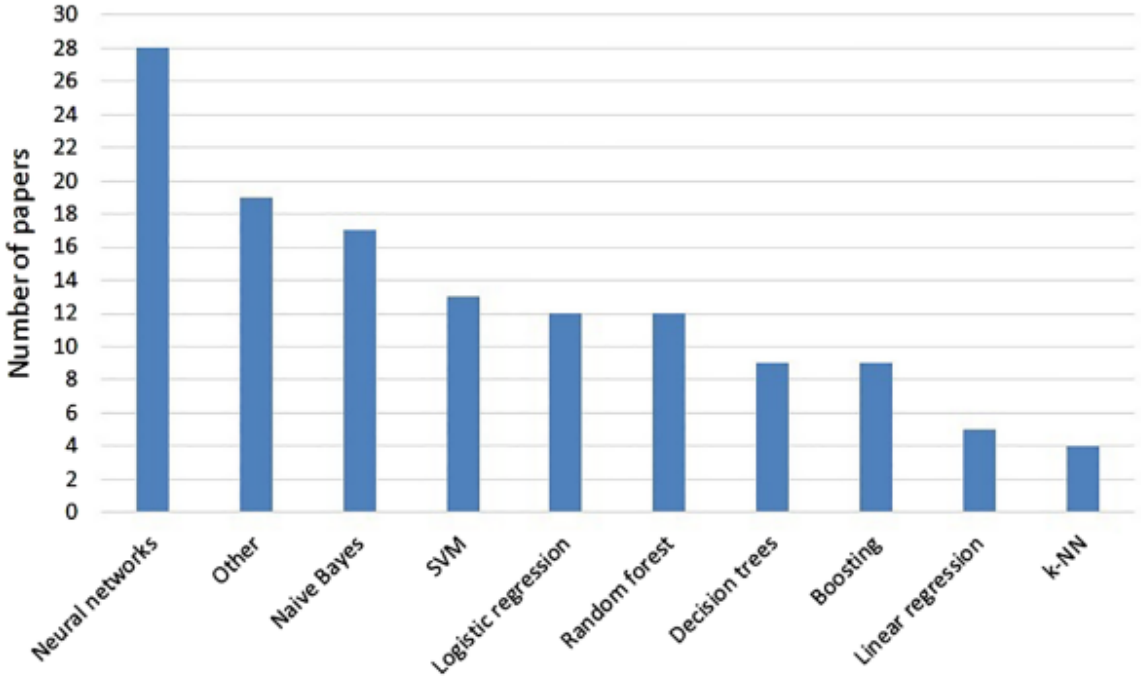

Figure 2.1: Total number of papers using a particular ML algorithm group [Horvat and Job, 2020]

Figure 2.1 from Horvat and Job [2020]'s literature review of machine learning in sport for score prediction show the methods that are currently used in the field as of May 2020. As shown in the figure, neural networks have a large research output. This is followed by other popular methods such as regression, support vector machines (SVMs), regression, decision trees, and gradient boosting, with k-nearest-neighbours (k-NN) being the least used. It is important to note that whilst a larger body of research has been carried out with certain methods, it does not necessarily mean they are the optimal method.

Horvat and Job [2020] also showed that including excessive seasons worth of data for training models reduces the quality of results. To those with a basic knowledge of cricket and sport this is not surprising given that in just a few years, many factors relating to team composition and tactics can change. The most accurate results were achieved by researchers who used data from a single season and a data segmentation evaluation method. When using data from a single season, the majority of the data was used for training and a small portion for testing. Some researchers used the same data set for training and testing, yielding unrealistically accurate results.

2.3.1 Expected Goals and Football

Expected goals (xG) is a statistical metric used in football to quantify the quality of scoring chances created by a team or player in a match. Each shot is assigned a probability of being scored, which is calculated by analysing a range of variables such as shot location, shot angle,

distance from goal, type of shot and more. This can be done with traditional statistical methods, or machine learning approaches. Expected goals provides a more accurate assessment of a team's performance than simply looking at the number of goals scored as it takes into account the quality of the chances made.

Large sports analysis company StatsBomb have referenced in their blog posts that their xG model uses gradient boosting models [Vatvani, 2022]. Blumberg [2020], a Master's graduate of The University of British Columbia in data science, implemented his own xG model. He made use of XGBoost [Chen and Guestrin, 2016] and achieved comparable results to StatsBomb. One of the valuable things about XGBoost models is that they can provide a feature interpretation of what features are most important. This is valuable for sanity checking models. For example, Blumberg's model found the closer the shot is to the goal, the higher the probability of a goal, which matchs our intuitive knowledge of the game.

2.4 Existing Machine Learning Research on Cricket

Table 2.1: Research areas in cricket and their percentage of the total research output in cricket [Wickramasinghe, 2022]

Research Area	% of Studies
Game Outcome Prediction	35
Player's Performance Classification	16
Batting Style/Stroke Classification	9
Other	9
Umpire's Decision/Gestures	6
Score Prediction	5
Cricket Commentary/Media	4
Pitch Behaviour Predicition	4
Team Selection/Performance	3

Wickramasinghe's [2022] review of machine learning in cricket from December 2022 classified the current areas of research in cricket. He found that over half of studies aim to predict the outcome of a game or classify different players performances. The full table of results can be seen in table 2.4. From this it is clear to see that most of the focus is on predicting game outcomes or overall player performance. There is very little research in predicting batter bowler matchups or the outcome of a single delivery. However, there would be value in predicting the outcome of a single delivery or over. For example, knowing the best type of ball to a certain batter given the game state. This is a potential avenue of new research.

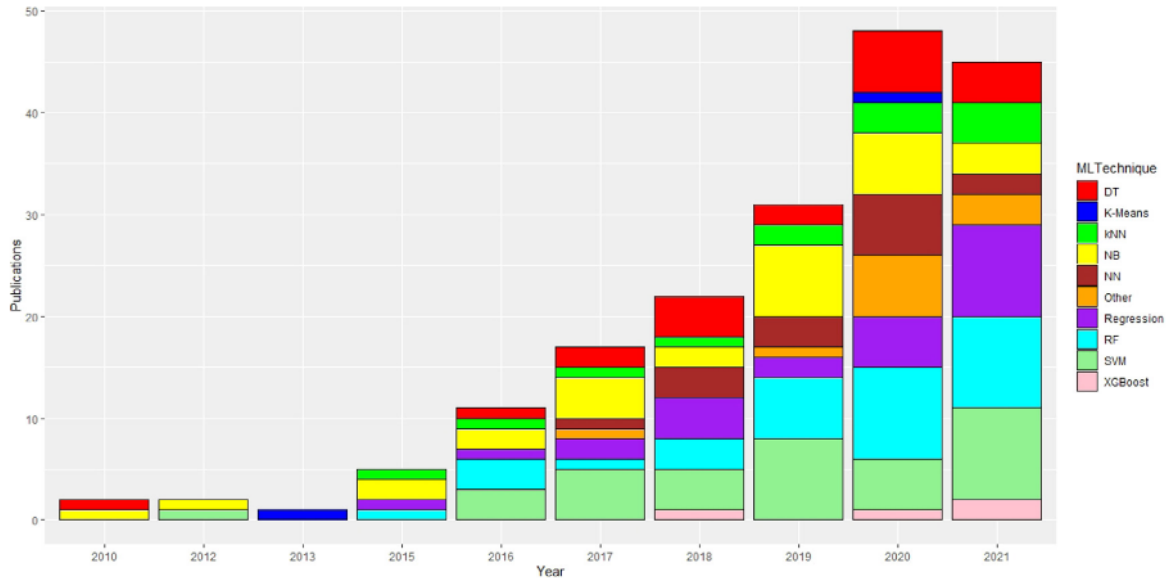

Figure 2.2: Total number of publications for each ML technique each year in cricket research [Wickramasinghe, 2022]

Wickramasinghe [2022] also evaluated the frequently used ML techniques in cricket, as shown in figure 2.2. There is a noticeable lack of deep learning, reinforcement learning or Gaussian process techniques. Furthermore, better boosting algorithms such as XGBoost have only recently begun to be adopted. There is clearly gaps in current research in sports analytics that can be explored.

Lastly, a clear uptick in the number of publications can be seen over the years, which is also corroborated in sport as whole in Horvat and Job [2020]. It once again highlights the growing interest in ML in sport.

2.5 Research With Similar Goals

Research in other sports has attempted to predict the outcome of a batter bowler matchup. Namely, the Singlearity plate appearance model (Singlearity-PA) [Silver and Huffman, 2021] attempted to solve one of the most fundamental questions in baseball: How can we predict the outcome of a batter vs. pitcher plate appearance (PA) [5]? This is similar to my goal with cricket: To predict the outcome of batter vs bowler.

The Singlearity-PA model [Silver and Huffman, 2021] was able to accurately predict the results of a batter versus pitcher plate appearance using a neural-network based AI model. The details of the model used are vague, however the network was able to take in 87 inputs and then output probabilities for each of the 21 possible outcomes of a plate appearance (PA) in baseball. Comparisons can be made between this and cricket. A plate appearance can be compared to an over in cricket, where there is a single bowler against one or more batters for 6 deliveries. The outcome of the over could be considered to be the runs scored.

[5] A plate appearance in baseball is credited to a batter each time they complete a turn batting. A player completes a turn batting when he is out or becomes a runner.

Silver and Huffman [2021] also split their player base up by how many PAs they had for each player. The best players had greater than 500 PAs worth of data each, but the vast majority had less than 100. Singlearity-PA was able to accurately predict the result of match-ups for these players with fewer PAs better than existing solutions. Parallels can be drawn between this and cricket, as there are often players who have little data, yet team selectors would want to know who the best player is to match-up against them.

Extending Singlearity-PA with Markov chains improved more complicated strategies, such as optimal player lineups or to decide on pinch hitters and relief pitchers. Similarly, in cricket the batting lineup and choice of bowler at different points in a game have a large impact on the score. In the example provided, Singlearity-PA's predicted runs scored for an optimal lineup was 6.7% better than the actual lineup in the 2019 National League All-Star game. It is important not to compare baseball and cricket too closely, but the techniques used by Silver and Huffman could potentially work well in Cricket.

Chapter 3

Background

This chapter will firstly introduce some background on Hawk-Eye and current computer vision in sports. Next, the requisite background for Gaussian processes and gradient boosting is provided, the two key methods used in this project. Additionaly, the idea of a data association problem is introduced.

3.1 Hawk-Eye and Ball Tracking

The dataset used in this project contains tracking data from Hawk-Eye systems. Hawk-Eye Innovations is a sports technology company, that provides many products, including computer visison services. In particular there is a focus on ball and player tracking in sports, including cricket. In this project the emphasis is placed on the ball tracking. Hawk-Eye have a number of cameras at IPL grounds, whose video is then triangulated and combined to create a three-dimensional representation of the ball's trajectory. It has been adopted as an ajudication tool by many governing bodies in cricket and is generally accepted to be accurate.

Hawk-Eye tracking data is accumulated and stored. It can then be used for various data science and machine learning purposes. Various features tracked by Hawk-Eye are included in the dataset used in this project. Further explanation of the dataset and features used in this project are explained in section 4.2.

3.2 Gaussian Processes

This section introduces Gaussian processes and the following section then explains how to extend their application to solve data association problems. A background on using Gaussian processes for regression is provided first as a basis before their extension to classification tasks. As Gaussian processes are not computationally cheap, sparse approximations are also reviewed in this section.

Gaussian processes are a *non-parametric* model. Non-parametric models are not based on insights about the concrete structure of the function to be modelled, but instead make assumptions about the function itself, such as its smoothness or differentiability. Instead of modelling a distribution of parameter values, a non-parametric model tries to find a distribution $p(f*)$ of probable functions that represents the function f to be estimated [Kaiser, 2017].

In addition, Gaussian processes are a supervised learning technique. It starts with a training data set $\mathcal{D}$ of n observations, $\mathcal{D} = (\mathbf{x}_i, y_i | i = 1, ..., n)$. $\mathbf{x}$ denotes an input vector (covariates) of dimension D and y denotes an output or target. The column vector inputs for all n cases are aggregated in the Dxn design matrix X and the targets collected in the vector $\mathbf{y}$, such that $\mathcal{D} = (X, \mathbf{y})$ [Rasmussen and Williams, 2006].

3.2.1 Gaussian Processes for Regression

Function-Space View

Definition 3.2.1 (Gaussian Process)**.** A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [Rasmussen and Williams, 2006].

Gaussian processes are a generalisation of the Gaussian distribution to function spaces. A multivariate Gaussian $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ describes a distribution over the finite elements in the vector $\mathbf{x}$. Every element of $\mathbf{x}$ is normally distributed. For two points in $\mathbf{x}$, x_i and x_j, their covariance is given by $cov[x_i, x_j] = \Sigma_{ij}$.

A Gaussian process is completely specified by its *mean function*, $m(\mathbf{x})$, and its co-variance function $k(\mathbf{x}, \mathbf{x}')$. Usually, the mean function is assumed to be constant zero, but this need not be the case. A mean function and covariance function can be defined for a real process $f(x)$ as

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})], \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))], \end{aligned} \tag{3.1}$$

where the Gaussian process can be written as

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{3.2}$$

The covariance function, or *kernel*, specifies the covariance between pairs of random variables. It is the covariance functions which encode the assumptions about the underlying function.

3.2.2 Kernels

Kernels are crucial in encoding the assumptions about the function a Gaussian process should estimate. It is a measure of similarity of different points in the observed data and of new points to be predicted. For example, a natural assumption is that the closer two points lie, the more similar their function values should be. Furthermore, when prediciting test points, training points close to it are probably more informative than those further away. However, it should be noted this need not be the case. For example, consider a sinusoidal wave where two points which are multiple wavelengths apart should have similar function values [Kaiser, 2017]. A kernel that only depends on the distance between two points is called *stationary*. Conversely, kernels that do depend on two points' positions in the input space are called *non-stationary*.

A common kernel and one used throughout this project is the squared exponential (SE) covariance function (also known as the radial basis function, RBF). For a finite dimensional input space $\mathbb{R}^d$, the SE kernel is defined by

$$cov(f(\mathbf{x}), f(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T \Lambda^{-1}(\mathbf{x} - \mathbf{x}')). \tag{3.3}$$

Kernels have characteristic *length scales* which informally can be thought of as roughly the distance you have to move in input space before the function value will change significantly. $\Lambda^{-1} = diag(l_1^2, ..., l_d^2)$ is a diagonal matrix of the squared length scales $l_i \in \mathbb{R}_{>0}$ in each axis. The variable $\sigma_f^2 \in \mathbb{R}_{>0}$ is the *signal variance*. The signal variance specifies the average distance of function values from the mean value. The vector of all hyperparameters in a model is called $\boldsymbol{\theta}$ [Rasmussen and Williams, 2006].

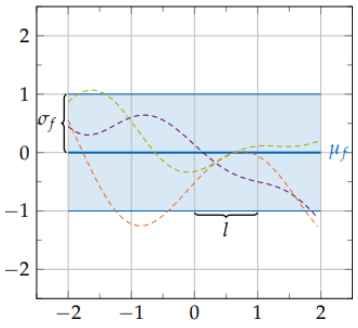

Figure 3.1: SE kernel with $\sigma = 1$ and $l = 1$ [Kaiser, 2017]

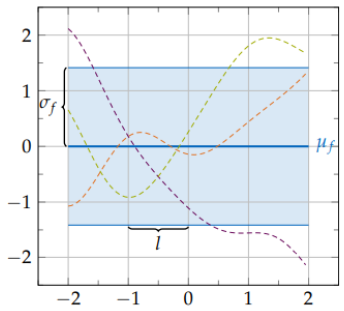

Figure 3.2: SE kernel with $\sigma = \sqrt{2}$ and $l = 1$ [Kaiser, 2017]

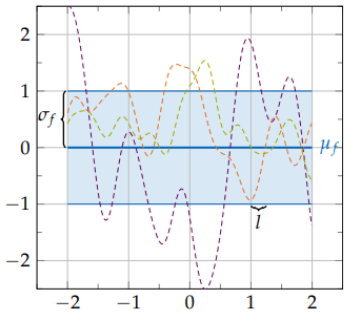

Figure 3.3: SE kernel with $\sigma = 1$ and $l = 0.25$ [Kaiser, 2017]

Figure 3.1, figure 3.2 and figure 3.3 compare sample functions drawn from Gaussian processes with SE kernels with different hyperparameters. Since the mean function $m(x)$ is assumed to be constant zero, the kernel specifies the prior assumptions about the function. The SE kernel describes arbitrary smooth functions. The hyperparameters l and σ of the kernel describe the dynamic range in the x and y directions respectively. As can be seen in figure 3.3, reducing the length scale increases the curvature of drawn sample functions from the kernel. Increasing the signal variance as shown in figure 3.2, increases the range of values the sample function will take.

3.2.3 Predictions and Posterior

In order to use Gaussian processes for regression, it is necessary to combine observations with a Gaussian process prior $f \sim GP(0, K)$. The distribution is obtained by integrating over all possible latent function values f and therefore taking all possible functions into account. This is called the *marginilisation of f*.

The joint distribution of the training outputs, $\mathbf{y}$, and the test outputs $\mathbf{f}_*$ according to the prior is

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{3.4}$$

If there are n training points and n_* test points then $K(X, X_*)$ denotes the nxn_* matrix of covariances evaluated at all pairs of training and test points. This is the same for $K(X, X)$, $K(X_*, X)$ and $K(X_*, X_*)$.

This results in the key predicitve equations for Gaussian process regression

$$\mathbf{f}_*|X, \mathbf{y}, X_* \sim \mathcal{N}(\bar{\mathbf{f}}_*, cov(\mathbf{f}_*)) \tag{3.5}$$

$$\bar{\mathbf{f}}_* \triangleq \mathbb{E}[\mathbf{f}_*|X, \mathbf{y}, X_*] = K(X, X_*)[K(X, X) + \sigma_n^2 I]^{-}1\mathbf{y} \tag{3.6}$$

$$cov(\mathbf{f}_*) = K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-}1K(X, X_*). \tag{3.7}$$

Eq. 3.6 and eq. 3.7 represent the mean function and the covariance function of the Gaussian posterior process respectively [Rasmussen and Williams, 2006].

Accordingly a compact form of $K(X, X)$ and $K(X_*, X_*)$ etc. will be introduced where $K = K(X, X)$ and $K_* = K(X, X_*)$.

Marginal Likelihood

The *marginal likelihood* is the marginalisation over the function values $\mathbf{f}$. Observing that $\mathbf{y} \sim \mathcal{N}(0, K + \sigma_n^2 I)$ yields the log marginal likelihood

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(K + \sigma_n^2 I)^{-}1\mathbf{y} - -\frac{1}{2}log|K + \sigma_n^2 I| - \frac{n}{2}log2\pi. \tag{3.8}$$

With this, an algorithm for Gaussian process regression can be formed, which is given in alg. 1. The matrix inversion required by eq. 3.6 and 3.7 is addressed with Cholesky factorisation, explained in section B.1 [Rasmussen and Williams, 2006]. The cholesky decomposition costs $O(N^3/6)$.

Algorithm 1 Algorithm for Gaussian process regression [Rasmussen and Williams, 2006]

Require: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), X_* (test input)

1: $L \leftarrow cholesky(K + \sigma_n^2 I)$
2: $\boldsymbol{\alpha} \leftarrow L^T \setminus (L \setminus \mathbf{y})$ ▷ eq. 3.6
3: $\bar{f}_* \leftarrow \mathbf{k}_*^T \boldsymbol{\alpha}$ ▷ eq. 3.6
4: $\mathbf{v} \leftarrow L \setminus \mathbf{k}_*$ ▷ eq. 3.7
5: $\mathbb{V}[f_*] \leftarrow k(X_*, X_*) - \mathbf{v}^T\mathbf{v}$ ▷ eq. 3.7
6: $\log p(\mathbf{y}|X) \leftarrow -\frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha} - \Sigma_i \log L_{ii} - \frac{n}{2}\log 2\pi$ ▷ eq. 3.8
7: **return**: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

Loss Functions

For practical applications, there must be a decision on how to act - a point prediction is necessary. To achieve this, a *loss function* $\mathcal{L}(y_{true}, y_{guess})$ is needed which specifies the loss incurred by guessing the value y_{guess} when the true value is y_{true}. The goal is to then make the point predicition y_{guess} that incurs the smallest loss. This can be done by minimising the expected loss or risk by averaging w.r.t. our model what the truth might be. Thus the best guess is

$$y_{optimal}|\mathbf{x}_* = \underset{y_{guess}}{\operatorname{argmin}} \widetilde{R_{\mathcal{L}}}(y_{guess}|\mathbf{x}_*) \tag{3.9}$$

where $\widetilde{R_{\mathcal{L}}}$ represents the risk and $\widetilde{R_{\mathcal{L}}}(y_{guess}|\mathbf{x}_*) = \int \mathcal{L}(y_*, y_{guess}) p(y_*|x_*, \mathcal{D}) dy_*$ [Rasmussen and Williams, 2006].

Choosing Hyperparameters

Up until now, the hyperparameters $\boldsymbol{\theta}$ have been assumed constant. If this were the case, GPs would have no training stage. However, knowing the correct hyperparameters is not clear a priori, and so a method of determing the hyperparameters must be used [Kaiser, 2017].

The correct way to model uncertainty about the hyperparameters is to give them a prior probability $p(\boldsymbol{\theta})$. The prior chosen should be broad to reflect the vagueness of the parameters before training. To derive the dependent distributions marginalise the prior to get

$$\begin{aligned} p(f) &= \int p(f|\theta) p(\theta) d\theta \\ p(y|X) &= \int p(y|X, \theta) p(\theta) d\theta. \end{aligned} \tag{3.10}$$

A new distribution can be obtained by combining the prior with the likelihood of the training data observed using Baye's theorem:

$$\begin{aligned} p(\boldsymbol{\theta}|X, y) &= \frac{p(y|X, \theta) p(\theta)}{p(y|X)} \\ &= \frac{p(y|X, \theta) p(\theta)}{\int p(y|X, \theta) p(\theta) d\theta} \end{aligned} \tag{3.11}$$

The integration required in the denominator of eq. 3.11 is very hard in practise as y is a complicated function of $\boldsymbol{\theta}$. Instead, $p(\boldsymbol{\theta}|X, y)$ is maximised to provide an estimate and does not require the calculation of the integral in the denonimator because it is a constant. The solution of the numerator $p(y|X, \theta)$ you might recognise as the log marginal likelihood in eq. 3.8. For practical reasons minimising the negative of the log marginal likelihood is easier than maximising it directly. Therefore, the estimation of hyperparameters is the solution of the following optimisation problem [Kaiser, 2017]:

$$\boldsymbol{\theta}^* \in \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \; - p(y|X, \theta). \tag{3.12}$$

3.2.4 Gaussian Processes for Classification

A Gaussian process is a generalisation of the Gaussian probability distribution. Both classification and regression can be seen as function approximation problems. Unfortunately, the solution of classification problems using Gaussian processes is tougher than regression problems. For regression problems, the likelihood is often assumed to be Gaussian. A Gaussian process prior combined with a Gaussian likelihood gives a posterior Gaussian process over functions, where everything remains analytically tractable [Rasmussen and Williams, 2006]. For classification models, the Gaussian likelihood is inappropriate; a different likelihood such as a Bernoulli

or multiclass likelihood must be used. When the likelihood is non-Gaussian, the bayes rule results in an intractable posterior $p(f|y)$ [Yi, 2020b]. However, a variational approach can approximate the true posterior distribution.

3.2.5 Variational Gaussian Processes

Variational Gaussian processes (VGP) models use a *variational approximation* to the true posterior distribution over functions. The variational approach in VGP models involves approximating the true posterior distribution with a simpler, tractable distribution. This distribution is the variational distribution and $q(f)$ is used to denote its probability density function. This forces some key assumptions, namely that there exists some concrete variational distribution that is similar to the true posterior distribution. Next is that the parameters for this approximate distribution can be found in some way [Yi, 2020b].

Ultimately, the parameters of this approximate distribution are optimised to minimise the Kullback-Leibler (KL) divergence (see section B.2 for an overview) between the approximate and true posterior distributions, where the KL divergence signifies the closeness between the two distributions. Gradient descent maximises the expression for the evidence lower bound (ELBO) to find concrete model parameter values.

3.2.6 Sparse Variational Gaussian Processes

A major drawback of Gaussian processes is their $O(N^3)$ complexity when computing the cholesky factorisation of $[K + \sigma_n^2 I]$ from alg. 1, but it can be done as a preprocessing step since it is independent of the test points. After this, each single test point costs $O(N)$. To predict its variance it is still necessary to perform matrix multiplication which costs $O(N^2)$ [Kaiser et al., 2020]. Consequently, as the size of the dataset grows the more infeasible it is to evaluate a GP efficiently.

An approach to reduce this complexity is *Sparse Variational Gaussian Processes* (SVGP) [Hensman et al., 2015]. Spare approximations of GPs are a method of approximating a GP using a $M < N$ set of *inducing points* $\mathbf{Z}$ and *inducing variables* $\mathbf{u}$ that can represent the entire dataset, rather than X itself. This approach suits datasets with high levels of redundancy, but also introduces the problem of choosing the subset.

Using inducing points, we arrive at the Nyström approximation

$$K_{NN} \approx K_{NM} K_{MM}^{-} 1 K_{MN}, \tag{3.13}$$

where $K_{NM} = K(\mathbf{X}, \mathbf{Z})$ and $K_{MM} = K(\mathbf{Z}, \mathbf{Z})$. The SVGP complexity now costs $O(NM^2)$ instead of $O(N^3)$ [Lui et al., 2021].

When the inducing points are the real data points, such that $\mathbf{Z} = \mathbf{X}$ then $K_{NM} = K_{MM} = K_{NN}$ and the bound on the marginal likelihood becomes tight. Therefore, the approximate marginal likelihood equals the real marginal likelihood [Hensman et al., 2015].

The SVGP approach has been shown to provide accurate predictions with significantly reduced computational cost. Additionally, it also allows for more efficient training of GP models using stochastic gradient descent. Lastly, it allows for variational inference, accepting non-Gaussian likelihoods. For these reasons, it is a popular choice in Gaussian process models [Yi, 2020a].

Choosing the Inducing Points

When choosing your inducing points you want to use a method that will select M inducing points, that can represent the N real points. This can be done in many ways; an obvious example is to select inducing points linearly across the input range. However if your input data is not linearly spaced, then this can misrepresent your data. A better method used in this project and used in Hensman et al. [2015] is k-means clustering, which can appropriately find all major regions of the input space so long as there are enough inducing points.

3.3 Data Association

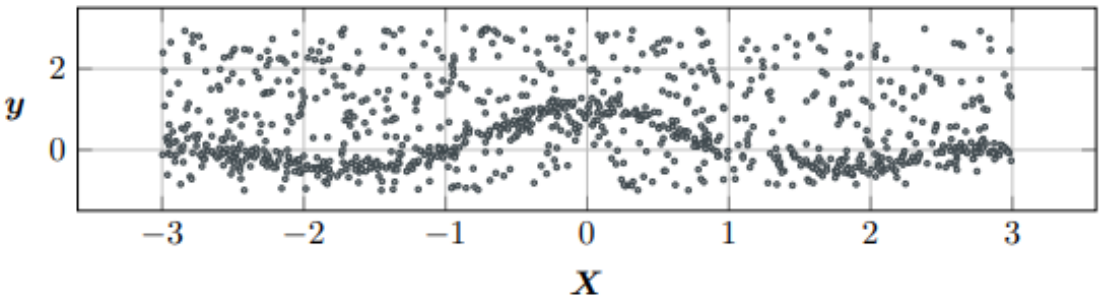

Figure 3.4: A data association problem consisting of two generating processes, one of which is a signal to recover and one is an uncorrelated noise process [Kaiser et al., 2020]

A *data association problem* is one where we consider the data to have been generated by a mixture of processes and we are interested in factorising the data into these components. For example, as described by Kaiser et al. [2020], figure 3.4 could represent faulty sensor data, where sensor readings are disturbed by uncorrelated and asymettric noise. Standard machine learning approaches can pollute any models, where the model starts to explain the noise instead of the underlying signal.

3.3.1 Data Association and Cricket

The cricket dataset could be deemed a data association problem, as each outcome (runs scored) of a cricket delivery could be generated by a different process. Each of these processes are overlapping as identical cricket deliveries can produce each of the possible outcomes. Despite this, there is a belief that there is a possibility of modelling each outcome separately, given that we intuitively know that there are good and bad delivery.

In addition, the relationship from a good delivery to a bad delivery is not a simple one. For example, the bowlers length of pitch in IPL cricket (the "pitchY" feature in the dataset) is not a simple case of fuller or shorter is better. A good short ball from a quick bowler is one that pitches and then reaches the batsmen at shoulder or above height. A good yorker is a ball pitching right at the batsmens feet. A "good length" delivery is one that pitches and then will pass the stumps at around the top of off stump. Yet, pitch lengths in between these values are generally considered bad. Similar examples can be made with the width of a ball, its swing, its spin, and many other features.

A data association solution that can uncover the underlying processes in this noise would be very beneficial for cricket delivery prediction. Data association techniques could provide a framework to attempt to seperate some of the differences in how runs are scored in cricket, and model the overlap in the varying outcomes.

3.4 XGBoost

XGBoost is a scalable machine learning gradient boosting library. It is an extremely popular, versatile and state-of-the-art gradient boosting technique. For example, among the 29 challenge winning solutions published at Kaggle's blog during 2015, seventeen of the solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. The solutions demonstrated that XGBoost gives impressive results on a wide range of problems, including: store sales prediction, high energy physics event classification, web text classification, customer behavior prediction, motion detection, ad click through rate prediction, malware classification, product categorisation, hazard risk prediction and massive online course dropout rate prediction [Chen and Guestrin, 2016].

The versatility and performance of XGBoost make it a popular choice of boosting algorithm, yet it hasn't been widely adopted in cricket research (see section 2.4). For this reason, I wanted to experiment with XGBoost in this project and use it as a baseline performance compared to solutions with GPs.

Chapter 4

Method

4.1 Objectives

Following on from the literature review and background, the primary objectives of this project can be listed as follows:

1. From 2.2 it is clear that it is key for bowling attacks to reduce the number of boundaries scored and increase the number of dot balls in T20 cricket. Therefore, a key objective of this project is to accurately predict which delivery would be most optimal to stop a boundary and cause a dot ball or a single.
2. From sections 2.3 and 2.4, it is clear that there is a lack of current research in delivery outcome prediction and use of a wider breadth of ML techniques. Another objective of this project, is to attempt to explore new prediciton types in cricket, as well as experiment with potentially under-utilised ML techniques, notably:
   (a) Data association with Gaussian processes
   (b) XGBoost

4.2 Dataset

The dataset used is provided by Hawk-Eye Innovations. At Hawk-Eye's request, the full data-set will not be uploaded due to its commercial sensitivity. Furthermore, the dataset contains identifying information such as batter names, bowler names, match IDs and so on. For the purposes of this project and executability of the code submitted, a small subset of the data has been uploaded that I have called the "John Doe dataset". This is all the data for one batter, who will be called John Doe from here, that has had any identifying information removed. See C.11 for code to produce the John Doe dataset.

The John Doe dataset is a CSV where each row is a different cricket delivery. The data includes features such as the bowling style, ball speed, the runs scored (including how many were from the batter or extras), where the ball pitches, where the ball passed the stumps and more. The purpose is to use John Doe to explore what would be the best and worst deliveries to bowl against this person.

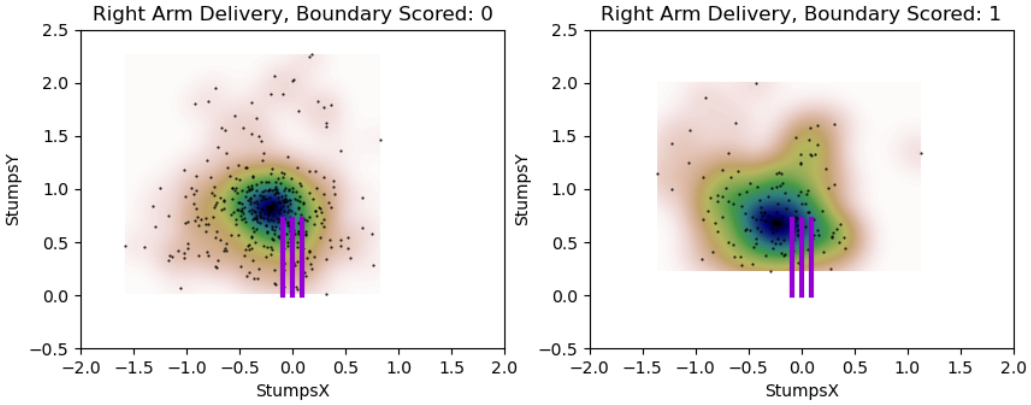

Figure 4.1: The "StumpsX" and "StumpsY" features of the dataset. This is all the right arm seam deliveries faced by John Doe. The left is deliveries that went for 0/1 runs and the right is deliveries that went for 4/6 runs. The purple lines are the stumps if you are looking at the stumps face on.

4.2.1 Features of the dataset

Ball Trajectory Features

Figures 4.1, 4.2 and 4.3 visualise the "stumpsX", "stumpsY", "pitchX" and "pitchY" features of the dataset. "StumpsX" and "stumpsY" represent where the ball passed the stumps. "PitchX" and "pitchY" represent where the ball bounced on the pitch from a top-down view. Each figure contains black dots to represent a delivery and a heatmap to highlight areas where a high concentration of deliveries have been bowled. It is visible that for the deliveries that don't go for a boundary, the heatmap concentrates towards a single location compared to those that go for boundaries. This location is at around the top of off stump in the case of figure 4.1, which is classically considered a good target location for a seam bowler.

Figure 4.4 [1] shows the strike rate [2] for the different areas of the pitch. It highlights that too short, too wide, or too full deliveries have a higher strike rate than the areas at a classically good pitch length and width.

Whilst there are potential signs of a good or bad pitch location, one noticeable thing in figures 4.1, 4.2 and 4.3 is how much the data overlaps for different outcomes i.e. locations of no

[1] This figure was generated by modified code from `https://github.com/opengoalapp/cricket_tracking`

[2] Strike rate is a common metric in T20 cricket to evaluate a batter's performance. The batters strike rate is their runs scored divided by the balls faced times 100. A high strike rate is good for a batter. Note that there is a metric for bowlers also called strike rate that this is not to be confused with.

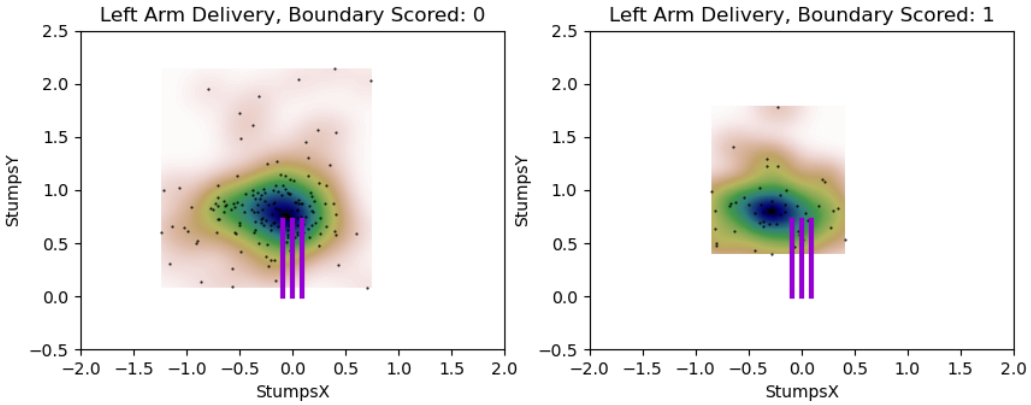

Figure 4.2: The "StumpsX" and "StumpsY" features of the dataset. This is all the left arm seam deliveries faced by John Doe. The left is deliveries that went for 0/1 runs and the right is deliveries that went for 4/6 runs. The purple lines are the stumps if you are looking at the stumps face on.

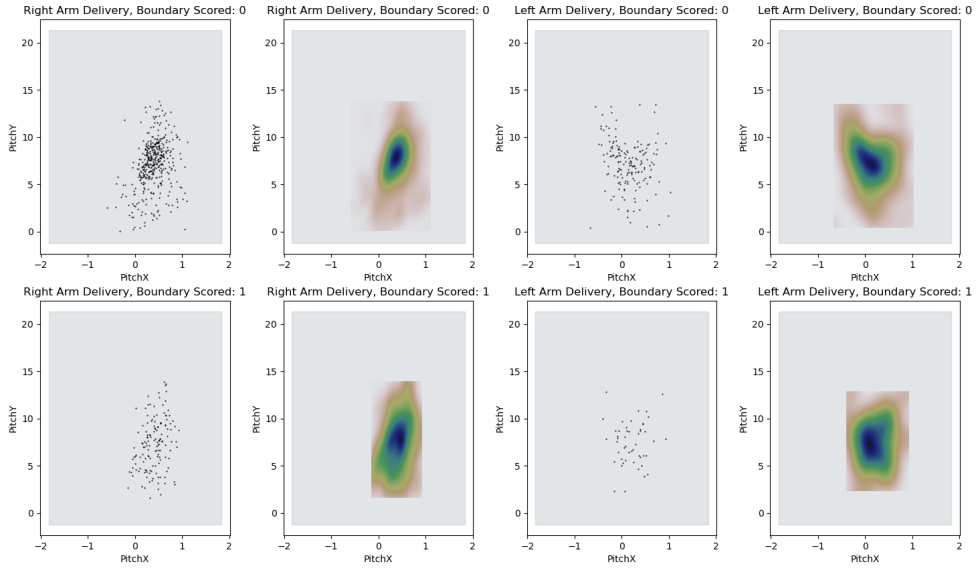

Figure 4.3: The "PitchX" and "PitchY" features of the dataset. This is all the seam deliveries faced by John Doe. The view is birds-eye (from above the stumps). The grey area is the pitch, with the x-axis stretched, such that 0.2 units in the y direction equals 1 unit in the x direction. This is to make each delivery clearer.

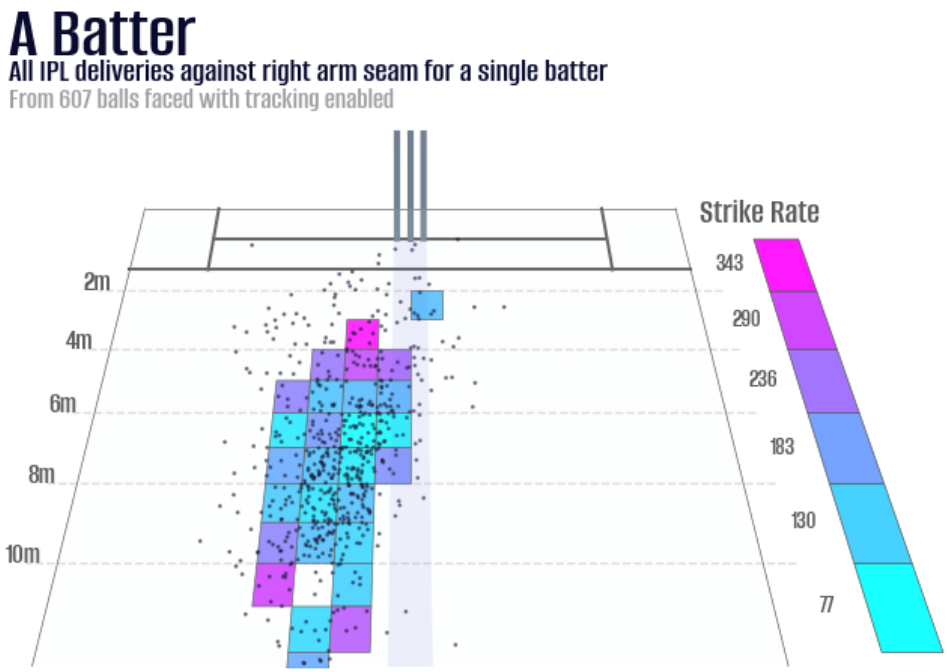

Figure 4.4: The pitches of right arm seam deliveries for John Doe. For exemplary purposes, the areas on the pitch have been bucketed. Each bucket has an associated strike rate, shown by the scale on the right.

boundary scored are also locations of a boundary being scored.

Delivery Outcome Features

To know if a ball was delivered right arm or left arm a "rightArmedBowl" column could take either true or false values. The following column contained the "bowlingStyle", which included various values such as "FAST_SEAM", "MEDIUM_SEAM", "OFF_SPIN" and more. Features for the runs scored also existed, that included "runs", "batterRuns" and "bowlerRuns". "BatterRuns" represent runs scored off the batsmen, such as running between the wickets or scoring a boundary. "BowlerRuns" represent runs scored off the bowler. This includes any "batterRuns", but also any additional wides and no balls. Therefore the column "runs" was a summation of the "batterRuns" and "bowlerRuns", plus any additional cases where the runs will not be attributed to either the batter or bowler, but simply to their respecting teams (for example byes and leg-byes). The type of extra is indicated in the "extras" column, for example "Wd" (wide) or "Nb" (no ball).

4.2.2 Pre-processing

The first step of the pre-processing was to load all the data into a pandas dataframe and then filter it for one batter. After that all columns with identifying information were removed.

Next, deliveries were removed where the "pitchX" and "pitchY" were impossible values. This could of happened for a number of reasons, such as rare extremely poor deliveries or bad tracking. Regardless, they added unneccessary extra noise and were removed.

Lastly a new feature "boundary" was added. Boundary was a binary feature, where a value of 0 indicated 0 or 1 "batterRuns" were scored. A value of 1 indicated 4 or 6 "batterRuns" were scored. I decided to only consider the "batterRuns" value, as this felt more indicative of a good or a bad delivery, versus a good delivery that the batsmen missed and went for 4 byes.

4.3 Mixture of Gaussian Processes

4.3.1 Data Association with Gaussian Processes

The data association with Gaussian processes model introduced by Kaiser et al. [2020] assumes there exists K independent functions. A data point is generated by evaluating one of the K functions and adding Gaussian noise from a corresponding likelihood. The assignment of the n^{th} data point to a function is specificed by the indicator vector $\mathbf{a}_n \in 0, 1^K$ which has exactly one non-zero entry. The k^{th} latent function and its value with the n^{th} data point are denoted as $f_n^k = f^k(\mathbf{x}_n)$ and collected as $\mathbf{F}^k = (f_1^k, ..., f_N^k)$ and $\mathbf{F} = (\mathbf{F}^1, ..., \mathbf{F}^K)$. Similarly the k^{th} entry in $\mathbf{a}_n$ as a_n^k and denote $\mathbf{A} = (\mathbf{a}_1, ..., \mathbf{a}_N)$. With this notation, the marginal likelihood of Kaiser et al.'s [2020] data association with Gaussian processes (DAGP) model is as follows

$$\begin{aligned} P(Y|X) &= \int p(Y|\mathbf{F}, \mathbf{A}) P(\mathbf{F}|X) P(\mathbf{A}|X) d\mathbf{A} d\mathbf{F} \\ P(Y|\mathbf{F}, \mathbf{A}) &= \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(\mathbf{y}_n|\mathbf{f}_n^k, (\sigma^k)^2)^{\mathbb{I}(a_n^k=1)} \end{aligned} \tag{4.1}$$

where σ^k is the noise for the k^{th} Gaussian likelihood and $\mathbb{I}$ is the indicator function.

As each process is assumed to be independent of the data and assignments, GP priors can be placed on the latent functions $P(\mathbf{F}|X)$ with mean functions and kernels. There is an assumption that the $\mathbf{a}_n$ are drawn independently from multinomial distributions with logit parameters $\boldsymbol{\alpha}_n = (\alpha_n^1, ..., \alpha_n^K)$. Next a relationship between the location in input space $\mathbf{x}$ and the associations is assumed, meaning independent GP priors can be placed on $\boldsymbol{\alpha}^k$. Any prior knowledge of the associations can be encoded into the choice of covariance function. The assignment probabilities is given by marginalising the $\boldsymbol{\alpha}^k$ that parameterise a batch of multinomial distributions

$$p(\mathbf{A}|X) = \int M(\mathbf{A}|\mathrm{softmax}(\alpha))p(\boldsymbol{\alpha}|X)d\boldsymbol{\alpha}. \tag{4.2}$$

Variational Approximation of Lower Bound

All GPs in the DAGP model use sparse approximations through inducing points and variables. They then derive a lower bound for the log-joint $\log p(Y, \mathbf{A}|X)$ given by

$$\begin{aligned}\mathcal{L}_{DAGP} =& \Sigma_{n=1}^{N}\mathbb{E}_{q(\mathbf{f}_n)}[\log p(\mathbf{y}_n|\mathbf{f}_n, \mathbf{a}_n)] + \Sigma_{n=1}^{N}\mathbb{E}_{q(\alpha_n)}[\log p(\mathbf{a}_n|\boldsymbol{\alpha}_n)] \\ &- \Sigma_{k=1}^{K}\mathrm{KL}(q(\mathbf{u}^k)||p(\mathbf{u}^k||\mathbf{Z}^k)) - \Sigma_{k=1}^{K}\mathrm{KL}(q(\mathbf{u}_\alpha^k)||p(\mathbf{u}_\alpha^k||\mathbf{Z}_\alpha^k))\end{aligned} \tag{4.3}$$

where $\mathbf{u}$ are the inducing variables and $\mathbf{Z}$ the inducing points. This bound has complexity $O(NM^2K)$ to evaluate.

4.3.2 Modulated Scalable Gaussian Processes

Lui et al. [2021] proposed a similar method with a tighter evidence lower bound (ELBO). Kaiser et al. [2020] keeps the indicator vector in the approximation of the lowerbound by approximating $\log p(\mathbf{y}, \mathbf{A})$ rather than the interested $\log p(y)$. Lui et al. [2021] directly approximates $\log p(y)$ by marginalising all the latent variables out.

Both Kaiser and Lui apply the reparameterisation trick when sampling from $\mathbf{a}_i \sim M(\mathbf{a}_i|\mathrm{softmax}(\boldsymbol{\alpha}_\mathbf{i}))$. Instead of sampling directly from the discrete distribution, the continuous relaxation proposed in Maddison et al. [2017] is employed to perform stochastic gradient descent. The trick enables sampling from a concrete random variable controlled by a temperature parameter λ. The value adopted by Lui is $\lambda = 0.01$. Samples from a concrete random variable are unit vectors and as $\lambda \longrightarrow 0$ the distribution approaches a discrete distribution.

4.3.3 Implementation

The implementation to mix gaussian processes was an iteration of the SMGP (scalable modulated Gaussian processes) model developed by Lui et al. [2021] [3]. The original model was developed in TensorFlow v1 using additional code copied over from GPFlow v1 [Matthews et al., 2017]. This model was updated to use the latest GPFlow API v2 and TensorFlow v2. In TensorFlow v1 only graph execution is possible, which is notoriously difficult to debug. Following this update the code could now be run in eager execution, enabling better debugging.

[3]The repository for the original implementation can be found here: `https://github.com/LiuHaiTao01/ModulatedGPs`

A new SMGPModified class was added following the lowerbound of Kaiser instead of Lui. This is because Lui's implementation of marginalising the latent variables, meant the assignment kernels parameters were never updated during optimisation if the assignment and prediction were using different likelihoods. This was a necessary change, to attempt to perform classification with the SMGP model. The original can be seen in C.1 and the modified in C.2.

The base code was also extended to be a GPFlow Module and ExternalDataTrainingLossMixin, adding the _training_loss method of the ExternalDataTrainingLossMixin class in GPFlow. With this the training loss could be passed into an optimiser's minimise function, in this case a TensorFlow Adam optimiser. The likelihood code can be see in C.3 and training loop in C.6.

To start the code creates the kernels and SVGP models (called "layers" in the code) with K number of latent GPs, before creating the SMGP model. This takes in your layers, and likelihood. The modified SMGPModified takes in two likelihoods: one for predicition and one for assignment. The code implemented by Lui et al. [2021] also features monte carlo sampling of 25 samples for each iteration of training. I also decided to batch the training set into minibatches of size 500, as anything larger could potentially crash my laptops GPU.

Various parameters can be set, including the seed value passed into numpy and TensorFlow for reproducability of results. The configuration of number of iterations, size of minibatches and number of monte carlo samples can be set. Relating directly to GPs and the SMGP model, K, the kernel hyperparameters and inducing points can all be changed.

4.4 XGBoost

The XGBoost [Chen and Guestrin, 2016] model implemented was a modification of the XGFootball model developed by Blumberg [2020]. In the code published by Blumberg [2020], there is an implementation of various boosting and decision tree models. Rather than train every classifier used in the original, only the XGBoost model was trained on the cricket dataset.

The model trained was a XGBoost classifier to classify the outcome of a delivery as either a boundary or not a boundary. For a classification tree with 2 classes, the value of the leaf node represents the score for one of the classes. It is then converted to a probability using the logistic function.

Hyperparameters were optimised using a log-loss evaluation metric (C.7). The optimised trees could then make predicitions on a validation set and the log loss values could be compared (C.8). A soft voting classifier was also employed, however the improvement in log loss was minimal (C.9). Training could then take place on the full training and validation sets, to produce a final model and make predictions.

4.4.1 SHAP

XGBoost models can be difficult to interpret. The SHAP package offers a solution to interpreting XGBoost models, using principles of game theory to determine the effect of each feature on the odds of each outcome. Essentially, SHAP considers how much the probabilities change when adding or removing features from the model. Figure 5.11 visualises the SHAP values for each feature, with a "high" shap value indicating a higher chance of scoring a boundary. The code for SHAP can be seen at C.10.

Chapter 5

Results and Discussion

The following is an exploration of the results yielded through the SMGP model and XGBoost model. For SMGP a review on toy data is provided first before moving to the cricket data. In the case of the John Doe dataset, 74.3% of the deliveries are not boundaries.

5.1 Mixture of Gaussian Processes

For the following examples the figures show in the top left samples drawn from the GP, the top right the ELBO maximisation during training (or in reality the optimisation of the lower bound), the bottom left the softmax assignments for each of the generative processes, and in the bottom right the predicitions of the generative processes. The predicitions show the upper and lower bounds within a 95% confidence interval for each process.

For the results with 2D input, two figures are provided for each example. The first figure represents the raw data, predicted samples, and a 3D plot of the assigments and predicitions. The second figure for each example takes slices of the output plane to plot. This shows the smooth prediction of the GPs for one axis, with the other axis set to a constant value. The second figure also plots the ELBOs during training. The first input axis is x_1 and the second input axis is x_2, except for the cricket data that labels appropriately the features used as input.

In each example, squared exponential kernels were used as priors for f^k and α^k and 25 inducing points for each SVGP layer.

5.1.1 1D Input Multimodal Toy Data

SMGP

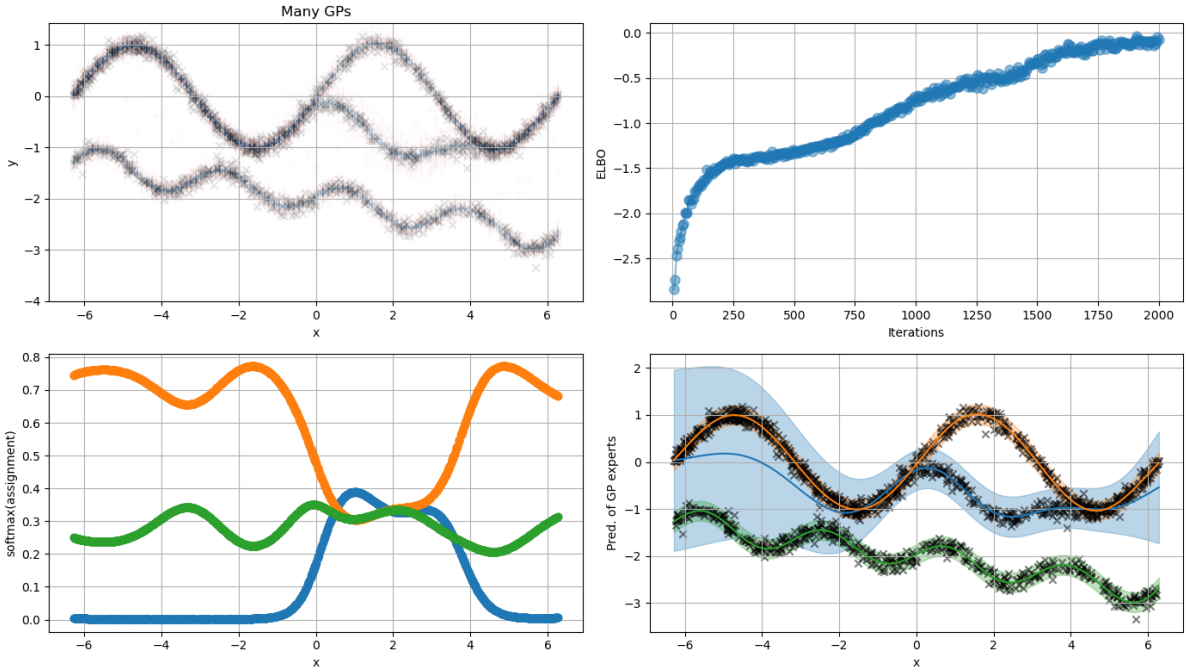

Figure 5.1: TensorFlow v2 SMGP results on multimodal toy data

Figure 5.1 recreates comparable results to Kaiser et al. [2020] multimodal toy example. The results were gathered using the GPFlow v2 upgraded version of Lui et al. [2021]. Uniformly spaced training points x are sampled to generate $y_1 = \sin(x) + \epsilon$, $y_2 = \sin(x) - 2\exp(-\frac{1}{2}(x-2)^2) + \epsilon$ and $y_3 = -2 - \frac{3}{8}\pi x + \frac{3}{10}\sin(2x) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 0.1^2)$. The resulting dataset is trimodal in the interval $[0, 5]$ and is otherwise bimodal with one mode containing double the amount of data than the other [Kaiser et al., 2020]. The SMGP has successfully learned that one of the generative processes has around twice the assignment probability of the other. In the tri-modal region the assignments become around equal, representing an equal chance of assignment for each process. This means it has successfully reconstructed the true generative process.

SMGP with Modified Likelihood

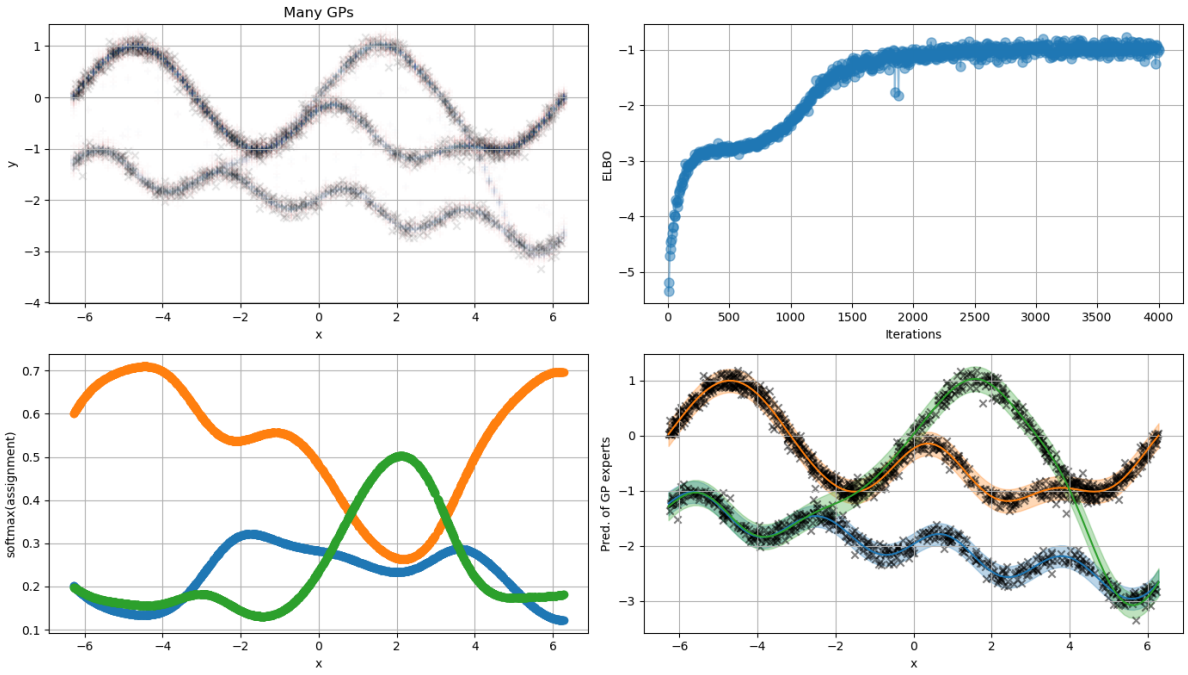

Figure 5.2: Modified SMGP results on multimodal toy data

Figure 5.2 uses the updated code (C.2) with the lowerbound updated to match Kaiser et al. [2020]. The three generating processes have been uncovered, however differently to before. The green process follows the blue process, before diverging in the trimodal region. At this point the assigments mix. However, in the bimodal region the orange process still has twice the assigment probability of the others.

5.1.2 1D Input Toy Data with Multiclass Likelihood

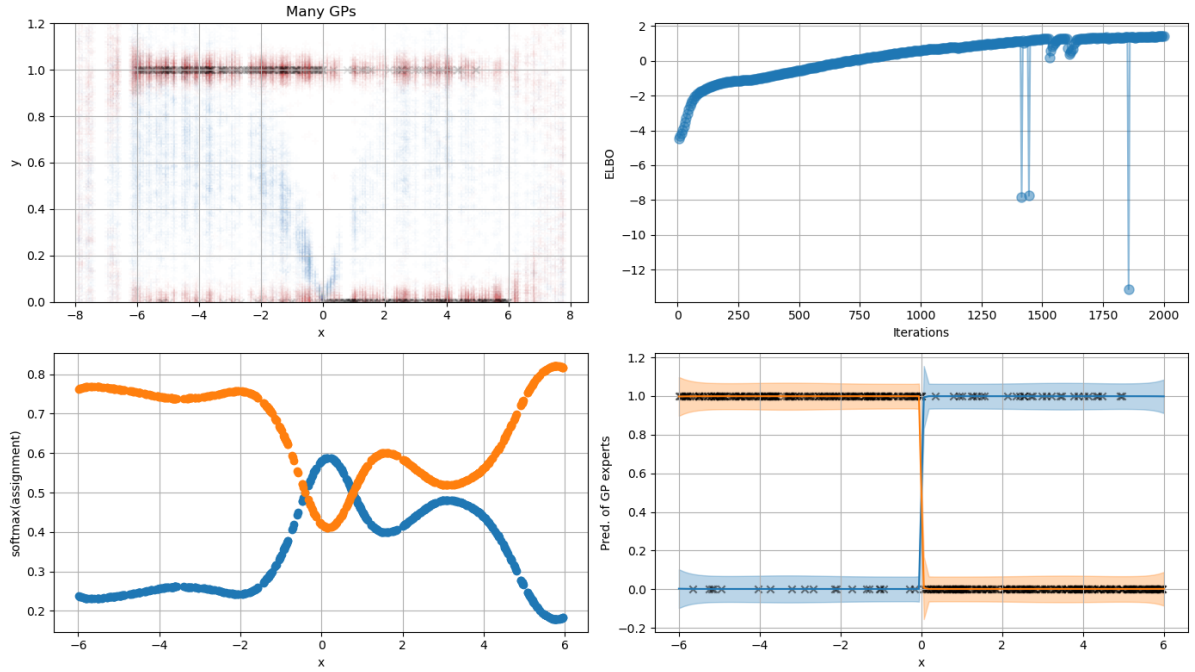

Figure 5.3: Modified SMGP results on toy data with a multiclass likelihood

Using the modified lowerbound different likelihoods could be applied to both the assignment and predicition. Figure 5.3 represents toy data that can take two values, 0 or 1. At input values less than 0, the process is 0 and at input values greater than 0 the process is 1. However, there is a probability of 0.1 that the process does not take it's proper value i.e. the process is stochastic. The SMGP has successfully learnt the real orange process and the stochasticity in the blue process. Furthermore, the orange process has far greater assignment probabilities, except at $x = 0$, where the process changes value. At this value the assignments of the two processes mix, before seperating again.

5.1.3 2D Input Toy Data

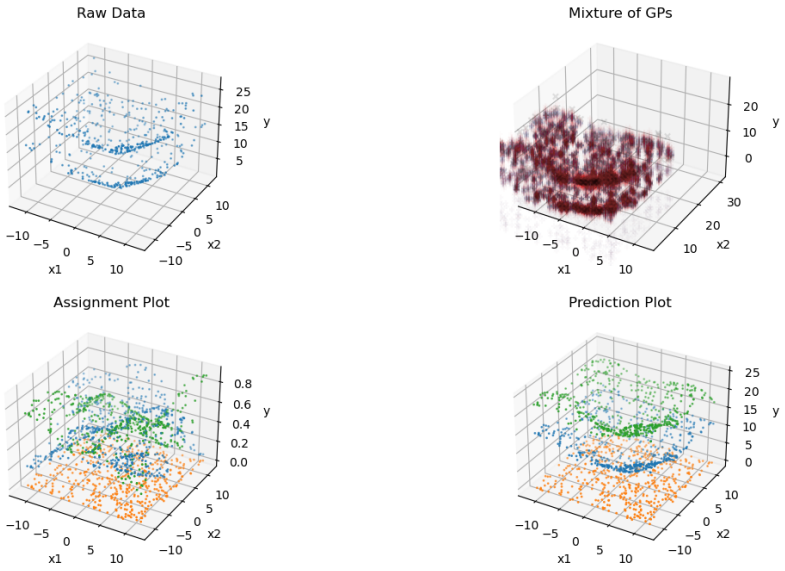

Figure 5.4: Tensorflow v2 SMGP results on 2d toy data

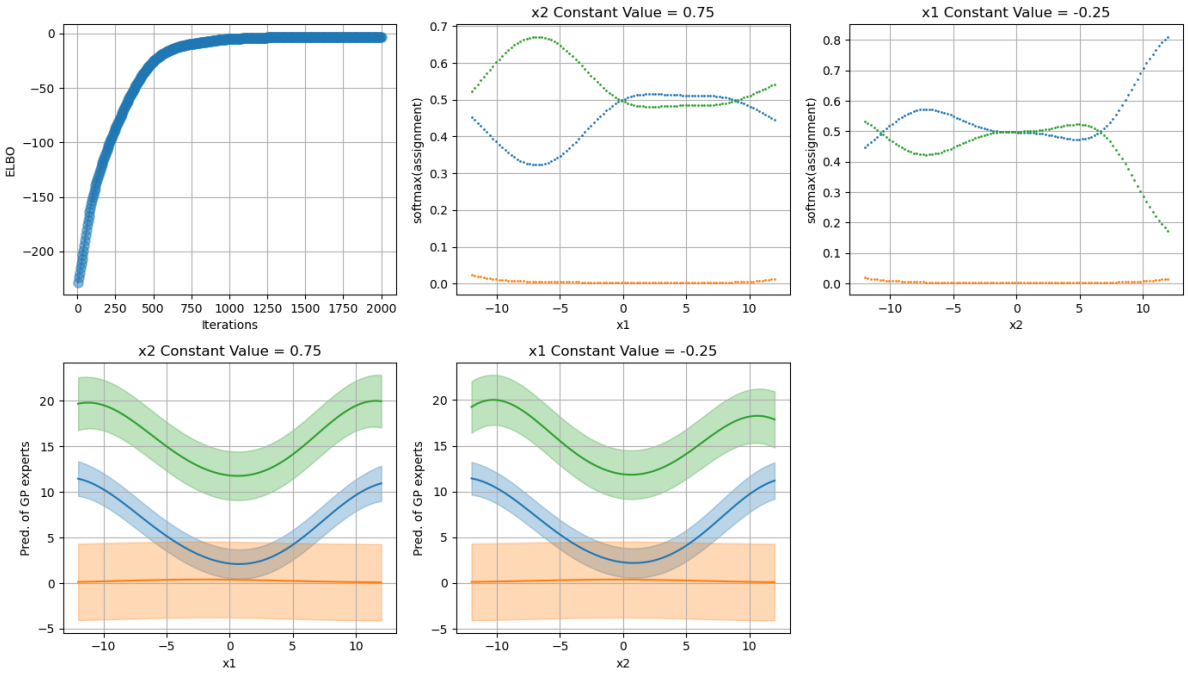

Figure 5.5: Tensorflow v2 SMGP results on 2d toy data

Figures 5.4 and 5.5 represent an example of a trained SMGP with 2D input. There are two processes, $y_1 = \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2}$ and $y_2 = y_1 + 10$. In addition, in this test, $K = 3$

meaning an additional third process was learnt. The SMGP has successfully learnt the two underlying processes. The assignment probabilities should be equal, given that each process has an equal number of points in the training set. However, at points the assignments of each process are not both 0.5, but are instead different. In some regions the assignments both equal 0.5 and it is possible that with further training they would eventually converge to be 0.5 across the entire input space. Furthermore, the SMGP has implicitly incentivised to explain the data using as few processes as possible, meaning the orange process has an assignment probability of 0.

5.1.4 2D Input Toy Data with Multiclass Likelihood

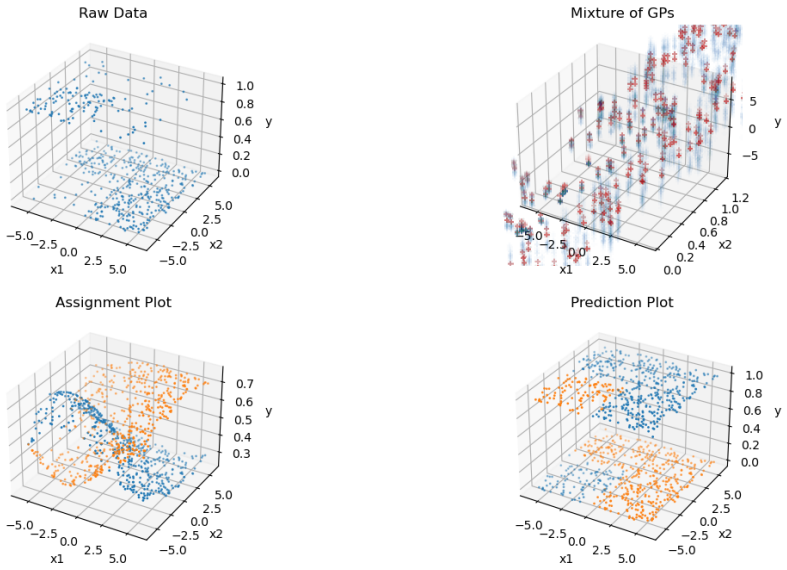

Figure 5.6: Modified SMGP results on 2d toy data with a multiclass likelihood

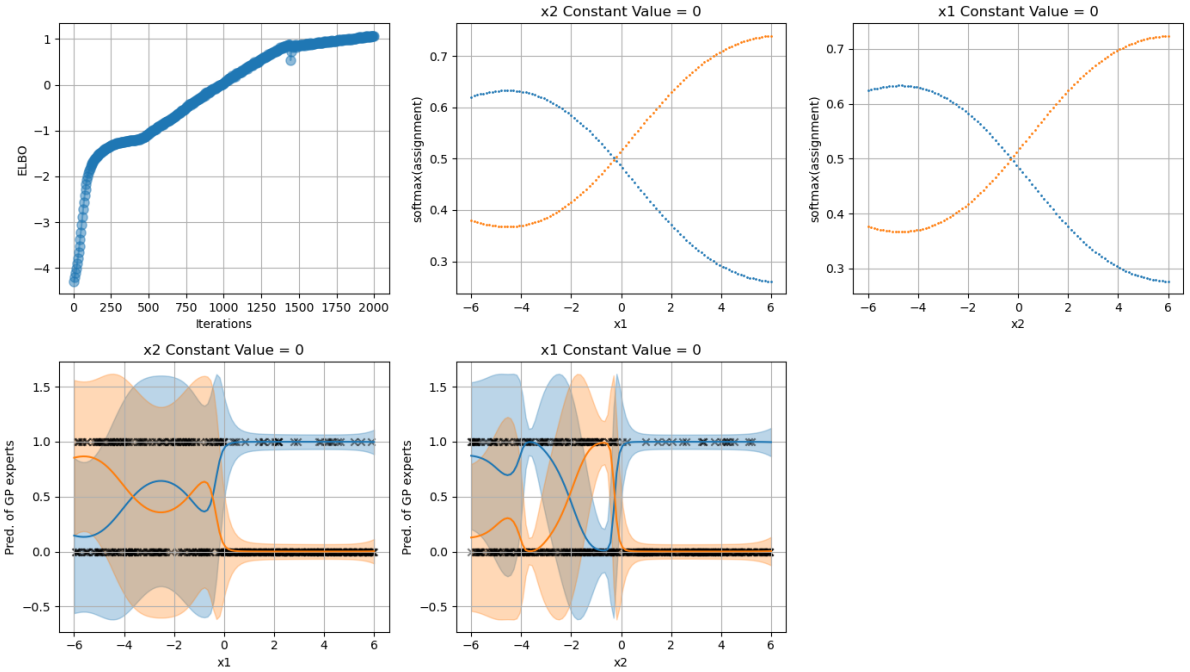

Figure 5.7: Modified SMGP results on 2d toy data with a multiclass likelihood

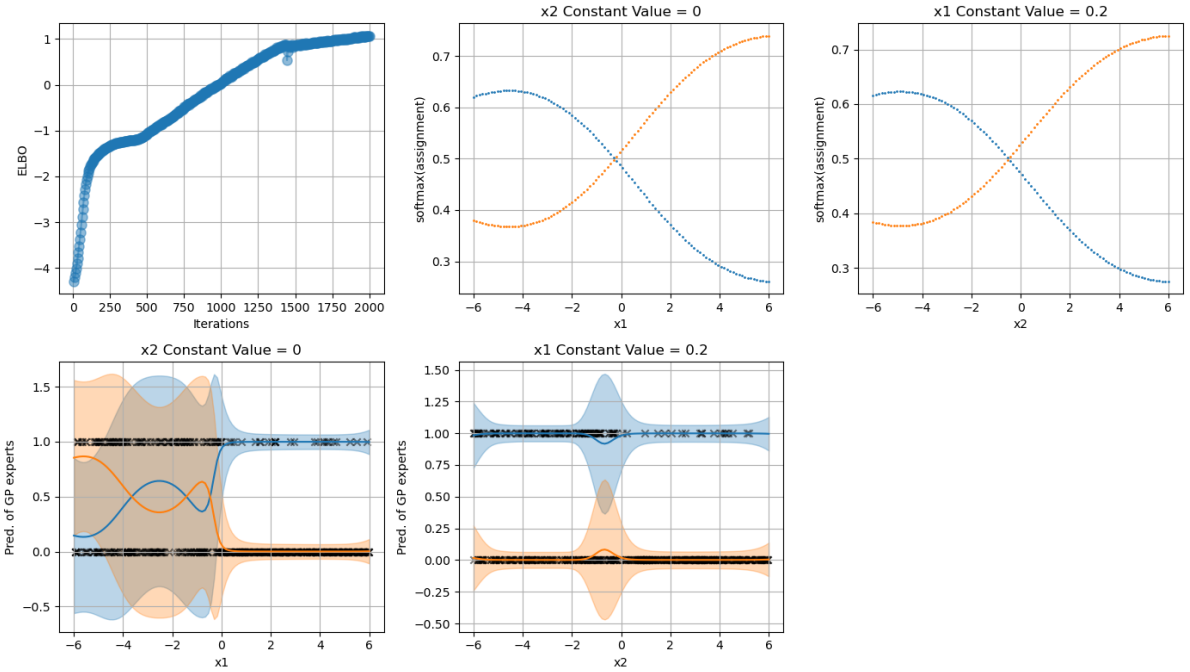

Figure 5.8: Modified SMGP results on 2d toy data with a multiclass likelihood. In this example the fixed value of x_1 has been shifted to 0.2 instead of 0.

Figures 5.6 and 5.7 show the results using the modified lowerbound, so that different likelihoods could be applied to both the assignment and predicition. The toy data is analogous to the 1D

multiclass example in that it can take two values, 0 or 1. At input values less than 0 for x_1 and x_2, the process is 0 and at input values greater than 0 the process is 1. However, there is a probability of 0.1 that the process does not take it's proper value i.e. the process is stochastic.

Discussion

It is clear here, that there is something wrong with the implementation somewhere. The predicitions in the regions where $x_1 < 0$ and $x_2 < 0$ are prediciting not whole numbers, which with a multiclass likelihood should be impossible. This is because a multiclass likelihood should only be able to predict for the different classes that exist (in this case 0 and 1). However, somehow the mixing of predicitions makes some sense. As the constant values for x_2 and x_1 are zero, this is right at the boundary where the raw data will be 0 or 1. So, if x_1 just went over 0, for example, $x_1 = 0.2$ the predictions should all be 1. It can be seen in figure 5.8 that doing so, does smooth the predictions more to each class, with the uncertainty only rising at around $x2 = -0.5$. So, in some ways it makes sense for the predicitions to be unsure to predict 0 or 1 right at this point. In any case, there must be a bug somewhere in the SMGP implementation or within GPFlow. In the other regions of input space, where $x_1 > 0$ and $x_2 > 0$ the predicitions flatten out to predict exactly the two possible classes.

Looking at the assignment probabilities, they also don't make sense. It should give a higher probability to the orange process at all times, given that this is the real process with more data. The only exception should potentially be at $x1 = 0$ and $x_2 = 0$ given that is where the two processes should "swap" (similar to figure 5.3).

5.1.5 Cricket Data with Multiclass Likelihood

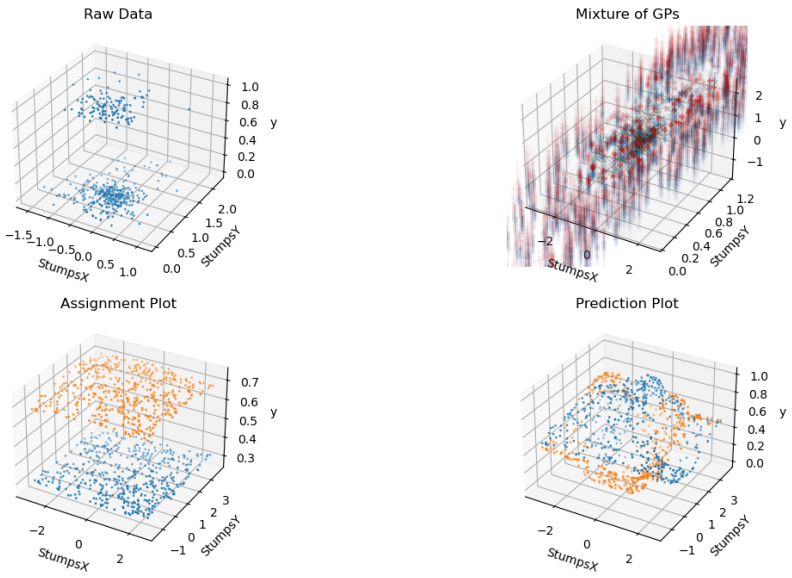

Figure 5.9: Modified SMGP results on John Doe dataset with a multiclass likelihood

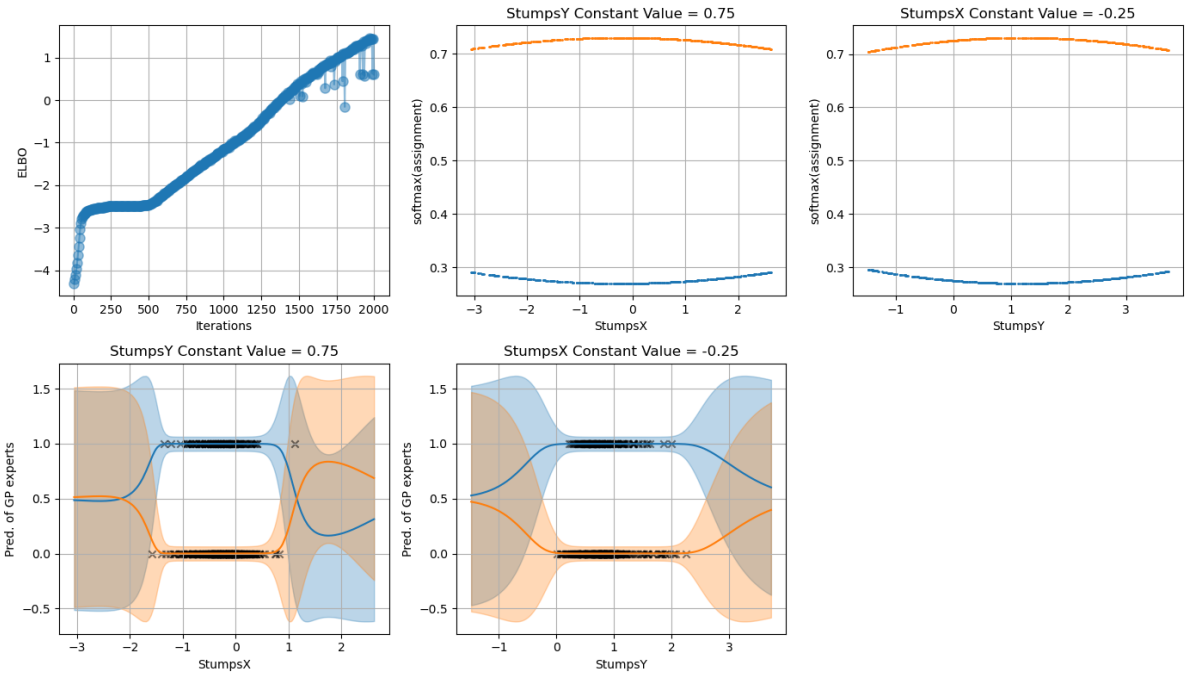

Figure 5.10: Modified SMGP results on John Doe dataset with a multiclass likelihood

Figures 5.9 and 5.10 show the results on the John Doe dataset for predicting whether a delivery would go for a boundary or not. This uses the SMGP implementation with the modified lowerbound, so that a multiclass likelihood could be applied to the predicition.

Discussion

The predictions have learnt the possible outputs of no boundary or boundary, with the uncertainty bars growing greatly outside the range of training data. Unfortunately, the assignments have not mixed in the desired fashion. The lengthscale of the assignment GP optimised to a large value, and the assignment essentially always has a high probability for no boundary being scored. This simply could be because, there are more boundaries scored than not. In the case of the John Doe dataset, 74.3% of the deliveries are not boundaries. As we can see, the assignment sits around 0.7 or slightly above for no boundary scored. It seems as though the assignments have learnt that not scoring a boundary is far more common than scoring a boundary, which is something we already knew.

The reasons for this is most likely that the outcomes are so mixed across the input data. In the case of the toy examples such as 5.1, the assignments only mixed in regions where there was equal chances of many outputs. In the majority of the input space however, the assignment probabilities reflected the proportions of the training data itself. It seems to be that this is the case here with the John Doe dataset. Because the outcomes are so mixed, it has simply learnt the probability of no boundary is higher than a boundary everywhere. It is possible that more inputs are needed to further seperate different data points in input space. Equally, more outputs (for example prediciting all the possible run outcomes), might produce better results than just a 2 class predicition.

5.2 XGBoost

Table 5.1: Results of XGBoost model

Model Features	% Accuracy	Log Loss
StumpsX & StumpsY	75.0	0.5796
StumpsX & StumpsY & PitchX & PitchY	71.43	0.5811

Table 5.2 shows the accuracy and final log loss for each of the models trained. The accuracies are less than desired, given that 74.3% of the dataset are not boundaries anyway. This accuracy could then in theory be achieved by predicting all deliveries to be not a boundary. This could be improved with a different model setup (different inputs and outputs) and this will be discussed more later.

5.2.1 StumpsX and StumpsY

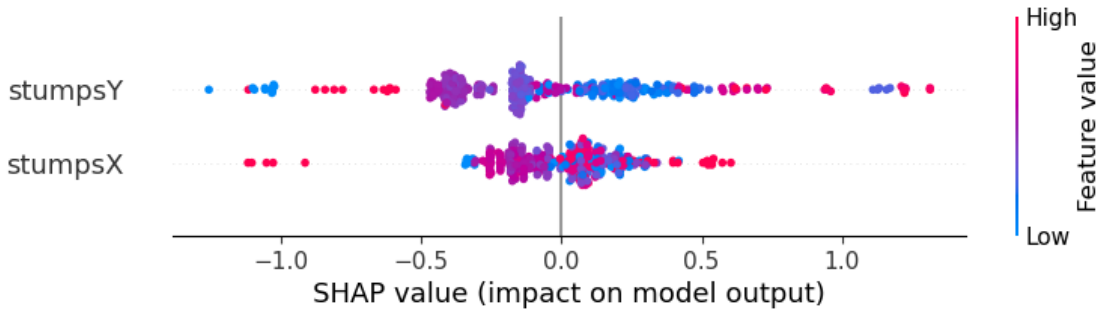

Figure 5.11: SHAP values for XGBoost model with stumpsX and stumpsY as features

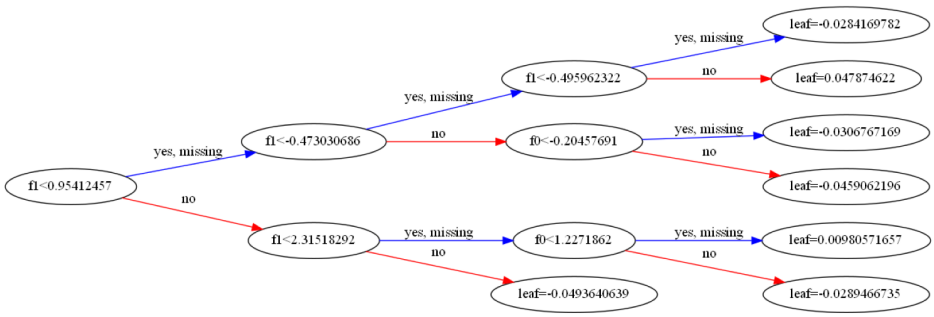

Figure 5.12: Decision tree for XGBoost model with stumpsX and stumpsY as features

Discussion

It can be seen that with just two features, the model hasn't learnt any clear relationships, as various features values result in various SHAP values. This potentially makes sense, given how mixed the outcomes are in the input space. This means potentially more input features are required to seperate the different outcomes more in input space.

Figure 5.12 also represents a decision tree for the model. It highlights how XGBoost will split the inputs into intervals to make decisions, potentially losing some of the richness of the continous features.

5.2.2 StumpsX, StumpsY, PitchX and PitchY

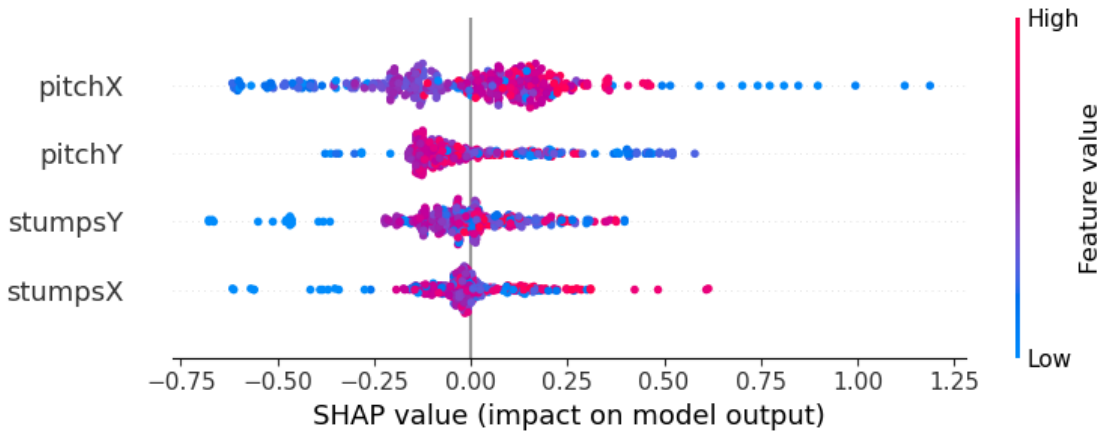

Figure 5.13: SHAP values for XGBoost model with stumpsX, stumpsY, pitchX and pitchY as features

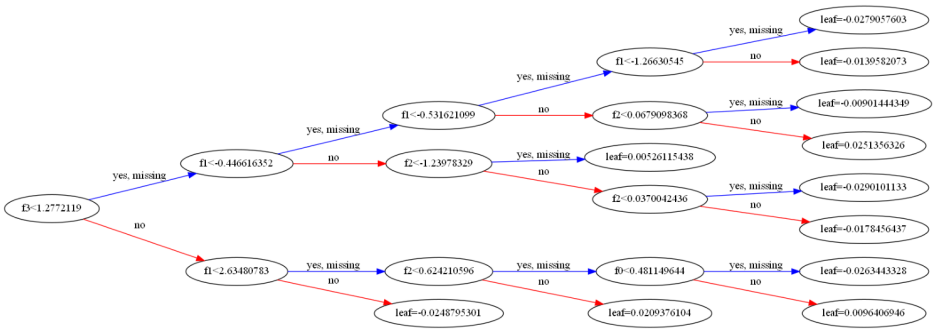

Figure 5.14: Decision tree for XGBoost model with stumpsX, stumpsY, pitchX and pitchY as features

Discussion

Following on from simply using the stump locations as features, pitch locations were considered as well. With 4 features things become clearer, but still not ideal. Lets look at each feature and discuss.

"PitchX" has not developed any clear relationship. Notably, low feature values of "PitchX" seem to be able to have a low and large SHAP value (and so impact on the models prediction). This could also be a potential drawback of the SHAP package and its colour scale and that a more numerical analysis is required. This is because as described in section 3.3.1 the relationship between pitch locations and outcomes is not simple or linear i.e. bigger "PitchX" does not necessarily linearly increase or decrease the chance of a boundary being scored. It's possible a more nuanced relationship with the "PitchX" feature has been discovered that can't be seen with the SHAP package. "PitchY" has a similar relationship, with low and high features values carrying potentially both low and large impact on the model.

"StumpsX" and "StumpsY" however have started to show some relationship. Lower values of "stumpsX" and "stumpsY" appear to lower the SHAP value, and therefore the odds of a boundary being scored. High values appear to increase it. This makes some sense for "StumpsY". This is because, figures 4.1 and 4.2 do highlight that low values of "StumpsY" have a lack of boundaries scored. However the intuition behind why it has done the same for "StumpsX" is not so clear. It also does not make sense that the same outcome did not happen for the model using just "StumpsX" and "StumpsY" as inputs.

Chapter 6

Conclusion

This project identified areas of new research in current sports analytics for cricket. In particular, there is a lack of exploration of alternative methods to those that are popular in current research. Therefore, this project explored using Data Association with Gaussian processes and XGBoost to predict the outcome of a whether a delivery in T20 cricket is a boundary or not. The research here provides a valuable first step towards new avenues of machine learning in cricket.

6.1 Future Work

There is a number of exciting potential future avenues from the work introduced in this project. Notably, further improvements to the SMGP model. Currently, the SMGP implementation by Lui et al. [2021] has limitations with using different likelihoods for the assignments and predictions. As a result, a revertion to the Kaiser et al. [2020] lowerbound was introduced as an attempt to handle different likelihoods for the assignment and predicition processes. As discovered, the classification is not behaving as expected. However there is potential to further advance the SMGP model for classification tasks. Any improvements to the model itself could then improve outcomes on the cricket data.

For example, all the possible outcomes (runs scored) of a cricket delivery could be modelled by a Multiclass classification SVGP. You could then have K numbers of these GPs, that are then mixed with an assignment GP. In this case, you first assign a GP to a region of input space and then predict using a multiclass GP that could predict any of the possible outcomes. This is different to the current implementation, as each process is no longer modelling just one outcome, but many. In any case, exploration of Gaussian processes in cricket should be explored further given the reasons provided in chapters 1 and 2.

It is also possible that predicting the exact specifics of what delivery to bowl is too challenging a task. Potentially predicting a broader idea of what bowler to pick for a certain over is more achievable, as it relies less on the specifics of what exact deliveries they bowl. This could be done with methods similar to Silver and Huffman [2021] explored in section 2.5.

Lastly, further refinement of all models implemented is possible. This includes a greater number of input features, as well as handling categorical inputs that can represent certain game states. For example, whether it is raining, overcast, pitch conditions, time of day etc. Such improvements would be applicable to both XGBoost or SMGP.

Bibliography

Rob Blumberg. A fully transparent expected-goals model, 2020. URL https://robblumberg.github.io/exp-goals/. Last accessed 05 May 2023.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Liverpool Echo. The unknown liverpool backroom analyst who convinced jurgen klopp to commission transfers of salah and keita, 2019. URL https://www.liverpoolecho.co.uk/sport/football/transfer-news/unknown-liverpool-backroom-analyst-who-16320524. Last accessed 05 May 2023.

ESPNcricinfo. How can the ipl become a global sports giant?, 2018. URL https://www.espncricinfo.com/story/tim-wigmore-how-can-the-ipl-become-a-global-sports-giant-1150459. Last accessed 05 May 2023.

Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization Theory and Neural Networks Architectures. *Neural Computation*, 7(2):219–269, 03 1995. URL https://doi.org/10.1162/neco.1995.7.2.219.

James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, 2015.

Tomislav Horvat and Josip Job. The use of machine learning in sport outcome prediction: A review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(5):e1380, 2020. URL https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1380?casa_token=aXIDtR-gOHQAAAAA%3A-YGsS9bhdiOkYkurvCmBCdZpbAx-MV4VF806rtp_h1CXJlCC0pg4hFFknpmc9dAb3wBAVu7NZ_FOBQ.

Scott Irvine and Rodney Kennedy. Analysis of performance indicators that most significantly affect international twenty20 cricket. *International Journal of Performance Analysis in Sport*, 17(3):350–359, 2017. URL https://doi.org/10.1080/24748668.2017.1343989.

Markus Kaiser. Incorporating uncertainty into reinforcement learning through gaussian processes. Master's thesis, 2017. URL https://github.com/mrksr/masters-thesis/blob/master/thesis/thesis.pdf.

Markus Kaiser, Clemens Otte, Thomas A Runkler, and Carl Henrik Ek. Data association with gaussian processes. In *Machine Learning and Knowledge Discovery in Databases: European*

Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II, pages 548–564. Springer, 2020.

Michael Lewis. *Moneyball: The art of winning an unfair game*. WW Norton & Company, 2004.

Haitao Lui, Yew-Soon Ong, Xiaomo Jiang, and Xiaofang Wang. Modulating scalable gaussian processes for expressive statistical learning. *Pattern Recognition*, 120:108121, 2021.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables, 2017. URL https://arxiv.org/pdf/1611.00712.pdf.

Alexander G. de G. Matthews, Mark van der Wilk, Tom Nickson, Keisuke. Fujii, Alexis Boukouvalas, Pablo León-Villagrá, Zoubin Ghahramani, and James Hensman. GPflow: A Gaussian process library using TensorFlow. *Journal of Machine Learning Research*, 18(40): 1–6, apr 2017. URL http://jmlr.org/papers/v18/16-537.html.

Liverpool FC News. Who is ian graham? liverpool suffer second transfer blow after julian ward departure, 2022. URL https://www.liverpool.com/liverpool-fc-news/features/ian-graham-liverpool-transfer-ward-25596815. Last accessed 05 May 2023.

Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Process for Machine Learning*, volume 2. MIT Press Cambridge, MA, 2006. URL http://gaussianprocess.org/gpml/.

Hector Ruiz, Paul Power, Xinyu Wei, and Patrick Lucey. " the leicester city fairytale?" utilizing new soccer analytics tools to compare performance in the 15/16 & 16/17 epl seasons. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1991–2000, 2017.

Joshua Silver and Tate Huffman. Baseball predictions and strategies using explainable ai. In *The 15th Annual MIT Sloan Sports Analytics Conference*, 2021. URL https://singlearity.com/static/assets/sloan-singlearity.pdf.

Fox Sports. The tampa bays are the undisputed kings of moneyball, 2019. URL https://www.foxsports.com/stories/mlb/the-tampa-bay-rays-are-the-undisputed-kings-of-moneyball. Last accessed 05 May 2023.

Dinesh Vatvani. Upgrading expected goals, 2022. URL https://statsbomb.com/articles/soccer/upgrading-expected-goals/. Last accessed 01 May 2023.

Indika Wickramasinghe. Applications of machine learning in cricket: A systematic review. *Machine Learning with Applications*, 10:100435, 2022. URL https://www.sciencedirect.com/science/article/pii/S2666827022001104.

Wikipedia. Cricket, 2023a. URL https://en.wikipedia.org/wiki/Cricket#. Last accessed 18 April 2023.

Wikipedia. Leg before wicket, 2023b. URL https://en.wikipedia.org/wiki/Leg_before_wicket#. Last accessed 18 April 2023.

Wei Yi. Sparse and variational gaussian process (svgp) — what to do when data is large, 2020a. URL https://towardsdatascience.com/sparse-and-variational-gaussian-process-what-to-do-when-data-is-large-2d3959f430e7. Last accessed 05 May 2023.

Wei Yi. Variational gaussian process (vgp) — what to do when things are not gaussian, 2020b. URL https://towardsdatascience.com/variational-gaussian-process-what-to-do-when-things-are-not-gaussian-41197039f3d4. Last accessed 05 May 2023.

Appendix A

Cricket Background

Whilst knowledge of cricket is useful to understand this project, I do not deem it necessary to appreciate the project and its applications. Any key ideas around cricket, T20 and IPL beyond the laws of the game are introduced in the main report. Despite this, I present a brief introduction to the game and its laws for those unfamiliar with the sport. Certain aspects of cricket have been introduced in limited detail, such as fielding and umpiring as they are unnecessary to understand in detail for this project.

A.1 A Brief Overview of Cricket

Cricket is a bat and ball game played between two teams of eleven players. It is played on a field, the centre of which is a pitch with wickets at each end comprising two bails balanced on three stumps. At the edge of the field is a boundary rope, see section A.3.2. The batting side scores runs by hitting the ball bowled at one of the wickets and then running between the wickets, or scoring a boundary. The fielding side tries to prevent this, by getting the ball to either wicket and dismissing the batters so they are "out", see section A.3.4. Which side bats/fields first is determined before the match starts through a coin toss between both team captains. When ten batters have been dismissed, the innings ends and the teams swap sides (the fielding side bat, and the batting side field). The game is referreed by two umpires. In the case of international or professional cricket, there is typically an additional 3rd umpire off the field for video or technology assisted reviews.

A.2 Additional T20 Cricket Details

The following section includes any additional details on T20 cricket not included in section 2.2.

A.2.1 Super Overs

On the rare case a T20 game finishes with both sides scoring the same number of runs, a tie breaker "super over" is played. In a super over, each team nominates three batsmen and one bowler to play a one-over-per-side "mini-match". If the Super Over also ends up in a tie, it is repeated until the tie is broken. An *over* is a set of 6 fair deliveries (so not wides and no balls, see section A.3.3) for the batting team to score.

A.3 Laws and Gameplay

A.3.1 Playing Field

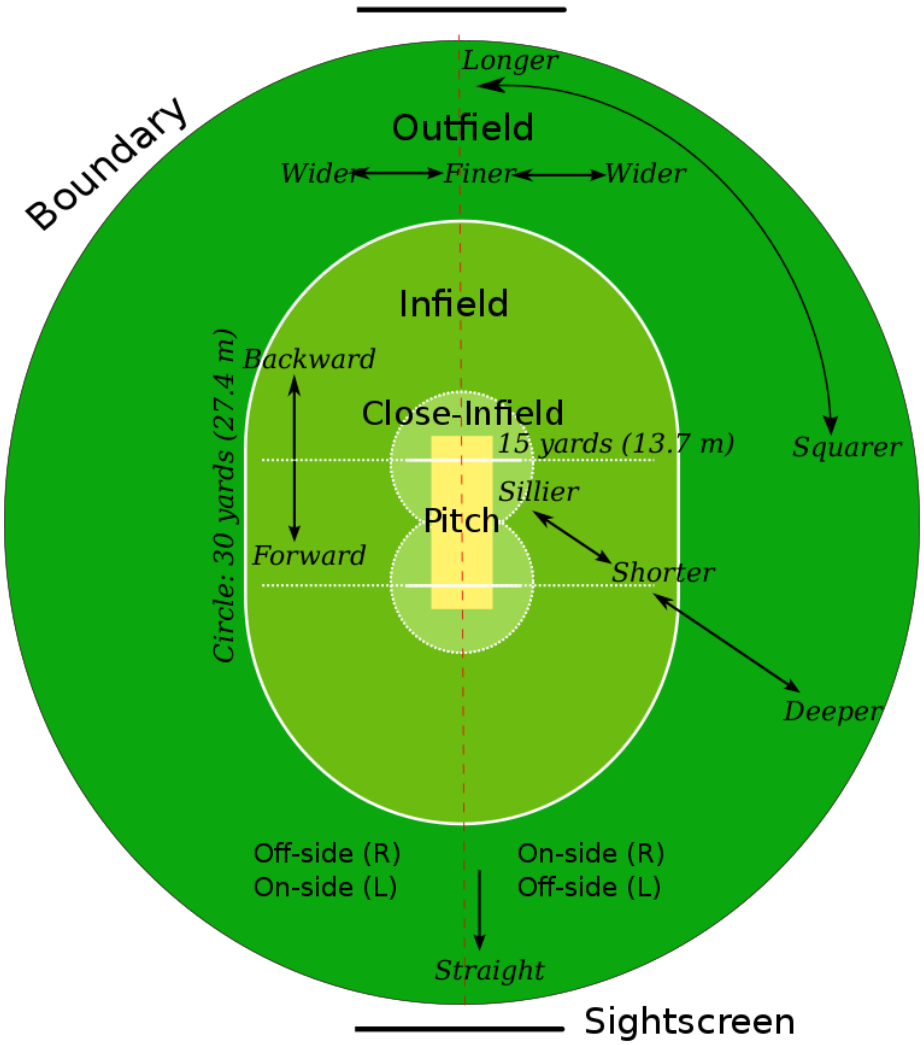

Figure A.1: A typical cricket field [Wikipedia, 2023a]

A cricket field as mentioned, comprises a central pitch with wickets at both ends of the pitch. A wicket is three stumps, with two bails balanced on top. See figure A.1 for a visual representation. The boundary surrounds the edge of the field and typically the field is oval in shape.

Figure A.2: A typical cricket pitch [Wikipedia, 2023a]

Figure A.2 shows a cricket pitch including the location of the wickets and the creases. As illustrated, the pitch is marked at each end with four white painted lines: a bowling crease, a popping crease and two return creases.

A.3.2 Runs

Runs are scored through either boundaries, or batsmen running between the creases. For the latter, the batter on strike attempts to score runs by hitting the ball whilst simultaneously not getting out. To score runs, after the batsmen strikes the ball there needs to be enough time for the batsmen at both ends to run to the other end of the wicket. If the fielding team get the ball to the wickets before the batsmen they could be *run out*, see section A.3.4. To register a run, both runners must touch the ground behind the popping crease with either their bats or their bodies (the batters carry their bats as they run). Each completed run increments the score of both the team and the striker. Infinitely many runs can be scored off one delivery this way, with the batsmen running back and forth between the wickets. Typically the maximum number of runs scored this way would be three, but it is possible to score more if there are misfields.

There are some terms for certain runs scored off a delivery. A *dot ball* is a delivery where the batter scored zero runs. A *single* is a delivery where the batter scored one run.

Boundaries

A boundary is scored when the ball is hit by the batsmen all the way to the boundary rope. Four runs are scored if the ball touches the ground at all prior to crossing the rope. If the ball is hit all the way over the boundary rope before touching the ground this is six runs. In T20 cricket the batsmen will typically aim to score boundaries, as they have the maximum return of runs.

Extras

Additional runs can be gained by the batting team as extras due to errors made by the fielding team. This is achieved in four ways: no-balls, wides, byes, and leg byes. A no ball occurs if the bowler bowls an illegal delivery, most commonly by overstepping the crease, but can also be called by an umpire for dangerous bowling. In T20 cricket and IPL, no balls award the batting team a *free hit*, a delivery in which they can only be dismissed via run out, hit the ball twice and obstructing the field (see section A.3.4). A wide has occured if the bowler bowls so that the ball is out of reach of the batter. Both a wide and a no ball have to be re-bowled

and award the batting team one run. Byes are any runs the batters achieve without hitting the ball, typically when the ball has been missed by the wicket keeper. Leg byes are any runs the batters achieve when the ball hits their body, but not their bat.

A.3.3 Overs

An over is 6 fair deliveries (not including wides and no balls etc.). A bowler can not bowl 2 overs in a row and at the end of an over the bowling end on the pitch changes. Despite this, typically bowlers will bowl in *spells*, where they will bowl alternate overs from the same end. In T20 cricket a bowler can bowl a maximum of 4 overs, meaning a minimum of 5 bowlers must be used. Typically a bowler might bowl two 2 over spells, however sometimes spinners in the middle of an innings might bowl all of their 4 overs in one spell. Unlike bowlers, the batsmen do not change ends at the end of each over.

A.3.4 Dismissals

There are nine ways a batter can be dismissed in cricket. The most common include being bowled, caught, leg before wicket (lbw), run out and stumped. Less common dismissals include hit wicket, hit the ball twice, obstructing the field, and timed out. For the purpose of this chapter I will only explain the most common dismissals.

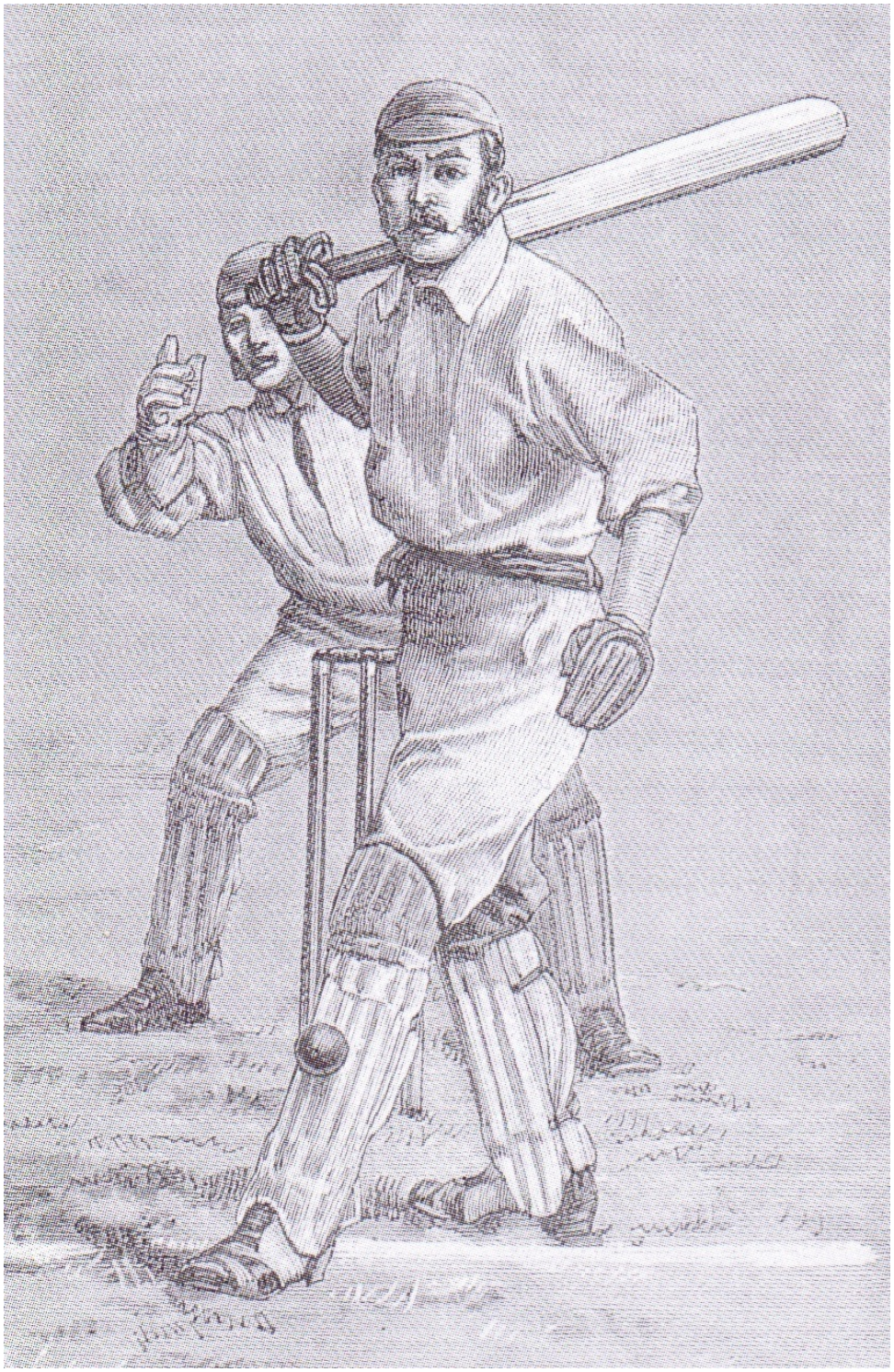

Figure A.3: A clear example of leg before wicket (lbw) [Wikipedia, 2023b]

Bowled

Being bowled is when the batter misses the delivery and the ball hits the stumps and takes the bails off the wickets. If the bails do not come off the wickets, the batter is not out, though this is very rare.

Caught

Being caught is when the batter strikes the ball in the air and a fielder catches this before the ball touches the ground. If the ball touches the ground at any stage (even in the fielders hands) it is not out.

Leg Before Wicket (lbw)

Leg before wicket is a more complicated dimissal type. Following an appeal by the fielding side, the umpire may rule a batter out lbw if the ball would have struck the wicket but was instead intercepted by any part of the batter's body (except the hand holding the bat). The umpire's decision will depend on a number of criteria, including where the ball pitched, whether the ball hit in line with the wickets, the ball's expected future trajectory after hitting the batsman, and whether the batter was attempting to hit the ball. Without explaining the finer details of lbw, a clear example of lbw is illustrated in figure A.3. Further information on lbw can be found on Wikipedia [2023b].

Run Out

A run out usually occurs when the batters are attempting to run between the wickets. The fielding team must successfully get the ball to one wicket and take the bails off with the ball (or their hands with the ball in them) before a batter has crossed the crease line near the wicket. The incomplete run the batters were attempting does not count.

Stumped

Lastly, being stumped involves the wicket-keeper catching the ball after the delivery and taking the bails of the stumps with the ball (or their hands with the ball in them) while the batsman is out of his ground (the batsman leaves his ground when he has moved down the pitch beyond the popping crease, usually in an attempt to hit the ball). A batter can only be stumped off a fair delivery (not a no ball or a wide). It is a special case of being run out and can only be performed by the fielding wicket keeper.

A.3.5 Basic Gameplay Example

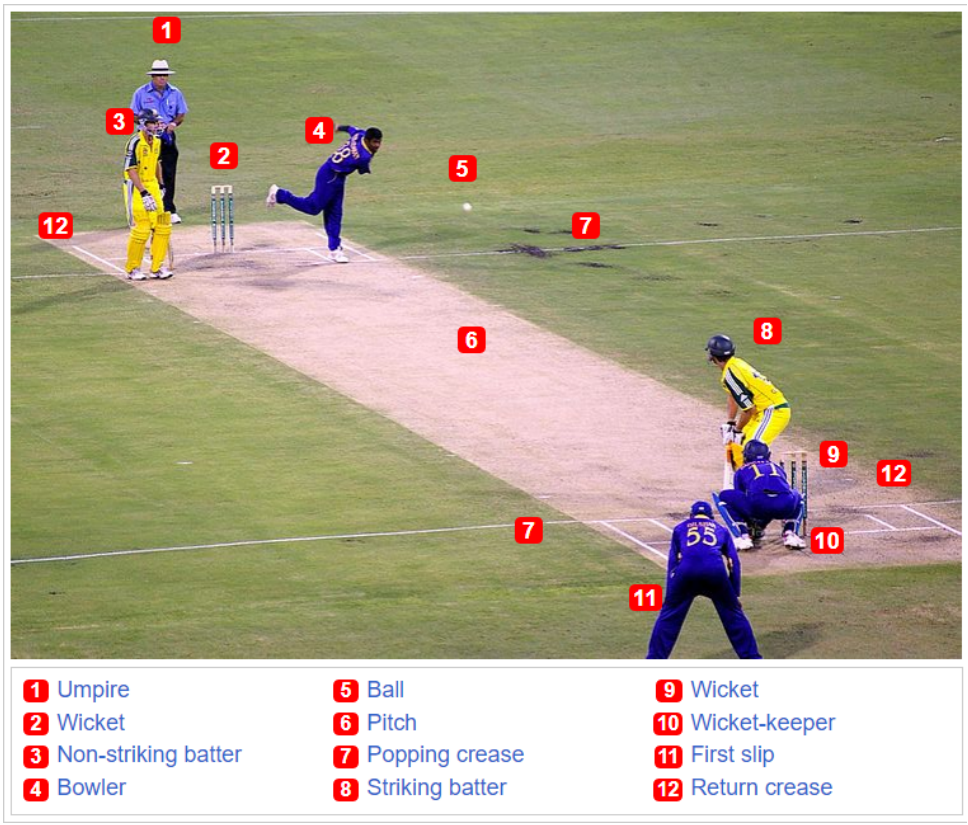

Figure A.4: An example of a ball being bowled and the components in play [Wikipedia, 2023a]

Figure A.4 shows a cricket delivery in play. The two batters (3 and 8; wearing yellow) have taken position at each end of the pitch (6). Three members of the fielding team (4, 10 and 11; wearing dark blue) are in shot. The bowler (4) is bowling the ball (5) from his end of the pitch to the batter (8) at the other end who is called the "striker". The other batter (3) at the bowling end is called the "non-striker". The wicket-keeper (10) is positioned behind the striker's wicket (9) [Wikipedia, 2023a].

The bowler (4) intends to dimiss the batsmen, or to prevent the striker (8) from scoring runs. By using his bat, the striker (8) intends to defend his wicket and hit the ball away from the pitch in order to score runs [Wikipedia, 2023a].

A.3.6 Player Roles

Typically players are selected to perform a specialised role.

Bowlers

As mentioned a minimum of five bowlers are required to bowl in a T20 match. Therefore, team selectors will typically choose 5-6 bowlers for the team for a given match. A bowler is someone specialised for bowling in one of two main ways: Seam (pace) bowling, or spin bowling. Seam bowlers typically use techniques such as *swing* or *seam* paired with a higher bowling speed (generally anywhere from 70 - 95 mph) to try and get wickets or reduce the

runs scored. However, in T20 cricket, seam bowlers will typically use a wider variety of delivery types, including slower balls, yorkers, bouncers, slower ball bouncers, wide yorkers, cross seam and more, to make prediciting their bowling more difficult and reduce the runs scored. Spin bowlers come in two main varieties: off spinners or leg spinners, and this essentially means what way they spin the ball. To spin the ball means to get the ball the ball to move dramatically in one direction after its bounce, by spinning it on release from the bowlers hand. Due to trying to spin the ball, spinners will typically bowl at a slower speed than pace bowlers (generally 45 - 65 mph). Like pace bowlers, a spin bowler in T20 cricket will use a variety of delivery types to try and reduce the runs scored from the opposing team.

Batters

The other main role is that of a batter. A batters job is to score runs for their team. In T20 cricket batters are generally specialised to score boundaries due to the short format, rather than be particularly good at protecting their wicket and staying in bat for many hours.

Wicket Keepers

The wicket keeper is a specialist fielder who stands behind the stumps to field the ball after it has been bowled. In the modern game, they are expected to also be reasonably good batsmen regardless of their wicket keeping skill.

All Rounders

All rounders are a special player who are good at both batting and bowling. All rounders are valued players, as they can provide the role of both a batter and bowler whilst only taking up one space on the teamsheet.

Captain

The last specialist role I will cover is that of the captain. The captain performs their captain duties as a batter, bowler, wicket keeper, or all rounder, while taking on the additional captains duties. The captain decides who will bowl each over and where each fielder will be positioned. While decisions are often collaborative, the captain has the final say. Captains in cricket typically shoulder more responsibility on the outcome of a cricket match than captains in other sports, given their level of responsibilty.

Appendix B

Mathematical Background

B.1 Cholesky Decomposition

The Cholesky decomposition of a symmetric, positive definite matrix A decomposes A into a product of a lower triangular matrix L and its transpose

$$LL^T = A, \tag{B.1}$$

where L is called the Cholesky factor [Rasmussen and Williams, 2006].

The Cholesky decomposition is useful for solving linear systems with symmetric, positive definite coefficient matrix A. To solve $A\mathbf{x} = \mathbf{b}$ for $\mathbf{x}$, first solve the triangular system $L\mathbf{y} = \mathbf{b}$ by forward substitution. Next solve the triangular system $L^T\mathbf{x} = \mathbf{y}$ by back substitution. Using the backslash operator, the solution can be written as $\mathbf{x} = L^T\backslash(L\backslash\mathbf{b})$ [Rasmussen and Williams, 2006].

B.2 Entropy and Kullback–Leibler Divergence

The *entropy* $H[p(x)]$ of a distribution $p(x)$ is a non-negative measure of the amount of "uncertainty" inherent to the variable's possible outcomes, given by

$$H[p(x)] = -\int p(x) \log p(x) dx. \tag{B.2}$$

The Kullback-Leibler (KL) divergence (also known as relative entropy) $KL(p||q)$ between two distributions $p(x)$ and $q(x)$ is defined as

$$KL(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx. \tag{B.3}$$

For two Gaussian distributions $\mathcal{N}(\mu_0, \Sigma_0)$ and $\mathcal{N}(\mu_1, \Sigma_1)$ we have

$$\begin{aligned} KL(\mathcal{N}_0||\mathcal{N}_1) =& \frac{1}{2}\log|\Sigma_1\Sigma_0^{-1}|+ \\ & \frac{1}{2}tr\Sigma_1^{-1}((\mu_0-\mu 1)(\mu_0-\mu 1)^T+\Sigma_0-\Sigma_1). \end{aligned} \tag{B.4}$$

When the log is the natural log entropy is measure in *nats*. The KL divergence can be viewed as the extra number of nats needed on average to code data generated from $p(x)$ under the distribution $q(x)$ [Rasmussen and Williams, 2006].

Appendix C

Code

C.1 Mixture of Gaussian Processes

```
def E_log_p_Y(self, Xt, Y, W_SND):
    Fmean, Fvar = self.pred_layer.predict_f(Xt, full_cov=False)
    var_exp = self.likelihood.variational_expectations(Xt, Fmean, Fvar, tf.
        cast(Y, dtype=tf.float64))
    var_exp *= tf.cast(W_SND, dtype=float_type)
    return tf.reduce_logsumexp(tf.reduce_sum(var_exp, 2), 0) - np.log(self.
        num_samples)
```

Listing C.1: E_log_p_Y for the original SMGP using Lui et al. [2021] lowerbound

```
def E_log_p_Y(self, Xt, Y, W_SND):
    Fmean, Fvar = self.assign_layer.predict_f(Xt, full_cov=False)
    var_exp = self.assign_likelihood.variational_expectations(Xt, Fmean,
        Fvar, tf.cast(Y, dtype=tf.float64))
    var_exp *= tf.cast(W_SND, dtype=float_type)
    E_log_p_A = tf.reduce_sum(var_exp, 2) - np.log(self.num_samples)

    Fmean, Fvar = self.pred_layer.predict_f(Xt, full_cov=False)
    var_exp = self.likelihood.variational_expectations(Xt, Fmean, Fvar, tf.
        cast(Y, dtype=tf.float64))
    var_exp *= tf.cast(W_SND, dtype=float_type)
    E_log_p_y = tf.reduce_sum(var_exp, 2) - np.log(self.num_samples)

    return tf.reduce_logsumexp(E_log_p_A, 0) + tf.reduce_logsumexp(
        E_log_p_y, 0)
```

Listing C.2: E_log_p_Y for the modified SMGP using Kaiser et al. [2020] lowerbound

```
def _build_likelihood(self, X, Y):
    Xt = self.integrate(X, self.num_samples)[0]
    # sample from q(w)
    W_dist = self.W_dist(Xt)
    W = W_dist.sample(1)[0, :, :]
    W_SND = tf.reshape(W, [self.num_samples, tf.shape(Xt)[1], self.K])
    # Expectation of lik
    L = tf.reduce_mean(self.E_log_p_Y(Xt, Y, W_SND))
    # ELBO
    # KL Divergence
    return L - (self.pred_layer.prior_kl() + self.assign_layer.prior_kl())
        / self.num_data
```

```
def _training_loss(self, data: Data) -> tf.Tensor:
    X, Y = data
    return -self._build_likelihood(X, Y)
```

Listing C.3: Likelihood for the modified SMGP

```
pred_kernel = gpflow.kernels.SquaredExponential(variance=0.5, lengthscales
    =0.5)
assign_kernel = gpflow.kernels.SquaredExponential(variance=0.1,
    lengthscales=1.0)
Z, Z_assign = kmeans(Xtrain, num_ind, seed=0)[0], kmeans(Xtrain, num_ind,
    seed=1)[0]

lik = GaussianModified(variance=0.5, D=K)

pred_layer = SVGPModified(kernel=pred_kernel, likelihood=lik,
    inducing_variable=Z, num_latent_gps=K, whiten=True)
assign_layer = SVGPModified(kernel=assign_kernel, likelihood=lik,
    inducing_variable=Z_assign, num_latent_gps=K,
                                whiten=True)

# model definition
model = SMGP(likelihood=lik, pred_layer=pred_layer, assign_layer=
    assign_layer, K=K, num_samples=num_samples,
              num_data=num_data)
```

Listing C.4: Creating kernels and inducing points to pass into SVGP layer. The SMGP model is then instantiated.

```
dataset = tf.data.Dataset.from_tensor_slices((Xtrain, Ytrain))
dataset = dataset.shuffle(buffer_size=num_data, seed=seed)
dataset = dataset.batch(num_minibatch).repeat()
train_iter = iter(dataset)
```

Listing C.5: Dataset batching

```
def run_adam(model, num_iter, train_iter, lr, compile=True):
    training_loss = model.training_loss_closure(train_iter, compile=compile
        )
    optimizer = tf.optimizers.Adam(lr)

    @tf.function
    def optimization_step():
        optimizer.minimize(training_loss, model.trainable_variables)

    print('{:>5s}'.format("iter") + '{:>24s}'.format("ELBO:"))
    iters = []
    elbos = []
    for i in range(1, num_iter + 1):
        try:
            optimization_step()

            if i % 5 == 0 or i == 0:
                elbo = -training_loss().numpy()
                print('{:>5d}'.format(i) + '{:>24.6f}'.format(elbo))
                iters.append(i)
                elbos.append(elbo)
        except KeyboardInterrupt as e:
```

```
            print("stopping training")
            break

    return iters, elbos
```

Listing C.6: Training loop for SMGP

C.2 XGBoost

```
def hyperparameter_opt(classifier, hyperparameter_dist, y_train, scoring="
    f1", n_iter=10):
    """
    Runs randomized hyperparameter search on preprocessed data for
        specified classifer.

    Arguments:
    ----------
    classifier (sklearn classifier object)
        - classifer we want to optimize
    hyperparameter_dist (dict or list of dicts)
        - dictionary with parameters names (string) as keys and
            distributions or lists of hyperparameters to try
    X_train (pd.DataFrame)
    y_train (pd.Series)

    Returns:
    --------
    (tuple) pd.DataFrame with hyperparameter cross validation results, and
        dict with best hyperparameters
    """
    print("Optimizing hyperparameters for {} model...".format(type(
        classifier).__name__))
    random_search = RandomizedSearchCV(estimator=classifier,
        param_distributions=hyperparameter_dist, cv=10, verbose=0,
                                       scoring=scoring, n_iter=n_iter)
    random_search.fit(X_train_preprocessed, y_train.astype('int'))
    print("Best hyperparameters {0}".format(random_search.best_params_))
    print("Best cross validation score {0}\n".format(random_search.
        best_score_))

    return pd.DataFrame(random_search.cv_results_), random_search.
        best_estimator_
```

Listing C.7: Hyperparameter optimisation for XGBoost classifier

```
for classifier_name, classifier_obj in cv_results.items():
    classifier = cv_results[classifier_name][1]
    y_train_pred = classifier.predict_proba(X_train_preprocessed)
    y_valid_pred = classifier.predict_proba(X_valid_preprocessed)

    validation_set_predictions[classifier_name] = y_valid_pred

    print(classifier_name)
    print("training set results: {0}".format(log_loss(y_train.astype('int')
        , y_train_pred, normalize=True)))
    print("validation set results: {0}".format(log_loss(y_valid.astype('int
        '), y_valid_pred, normalize=True)))
```

```
    print("")
```

Listing C.8: Compare log loss of training and validation sets for XGBoost classifier

```
ensemble_xg_model = VotingClassifier(
    estimators=[
        ('xgboost', cv_results["xgboost"][1]),
    ],
    voting='soft')
```

Listing C.9: Soft voting classifier for XGBoost

```
xgb = cv_results["xgboost"][1]
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_train_preprocessed)
shap.summary_plot(shap_values, X_train_preprocessed, feature_names=
    all_features_transformed)
plt.show()
```

Listing C.10: SHAP explainer for XGBoost

C.3 Pre-processing

```
from data_processing.query_utils import load_csv_data_mipl,
    filter_by_pitch_x_pitch_y

mipl_csv = load_csv_data_mipl()
mipl_csv = filter_by_pitch_x_pitch_y(mipl_csv)

batter_name = "ENTER␣BATTER␣NAME"
batter_data = mipl_csv[mipl_csv['batter'] == batter_name]

columns_containing_identifying_information = ['batter', 'batterId', '
    nonStriker', 'nonStrikerId', 'bowler', 'bowlerId',
                                                'dismissalDetails', 'matchId'
                                                    ]

batter = batter_data.drop(columns=
    columns_containing_identifying_information)

batter.to_csv('../Pre-processing-csvs/john_doe_dataset.csv')

print("Finished")
```

Listing C.11: Code to filter data for a batter and remove identifying information