Loughborough University

Sentiment Analysis of Twitter to Optimise Fantasy Premier League Performance

By Lewis Watt

Loughborough University
Student ID: F125967
24COC251: Computer Science Project

Supervisor: Magda Zajaczkowska
Submitted:

Contents

Chapter 1

Introduction

1.1 Background

Fantasy Premier League (FPL) is a popular online sports game based around the real-life top tier of English Football, the Premier League. Players of the game, often referred to as 'managers', are tasked with selecting a squad of 15 real-life premier league players every week. In game players are assigned a monetary value ranging from £4-15m, and each manager has a budget of £100m to build their team. The real-life performance of players determines how many points they are rewarded each week. Positive actions like scoring goals and providing assists increase points, whilst negative actions like receiving a red card or scoring an own-goal result in a decrease in points. Over the course of a 38-gameweek season, managers aim to score the most amount of points by rotating players in and out of their teams as they see fit.

The first version of FPL was launched in 2002 alongside the launch of the Premier League website, ahead of the 2002/3 season. Around 75,000 players participated in the first season [1], and since then growth has been explosive, with the player count surpassing 10 million in the 2022/23 season [2]. Each season the game has become more competitive, with the Premier League offering incentives such as a 7-night break inclusive of VIP hospitality tickets at two 2025/26 Premier League matches for the winner of the current season [3]. As well as the official prizes, many people participate in 'money leagues' where a small fee is paid for entry, and the winner takes home the prize pot. Alongside this more casual betting, professional gambling companies have started offering fantasy themed games where participants often stake money on day or weekend long periods, choosing a fantasy team to try and win them money by scoring the most points. With the global fantasy sports market valued at over $27 billion in 2022, and projected to reach $87 billion in 2031 [4], it is no surprise companies are scrambling to try and capitalise off of the immense popularity of the game.

With the increased popularity of FPL and a rapidly growing market full of potential customers, AI-based platforms aimed at increasing customers' overall ranks for little effort have started to emerge. These tools offer managers a quick and easy way to put together a team that they know will perform at a decent level, taking the effort out of the game for the managers who don't want to spend time diving through data and stats. AI tools also take the bias out of the game, as people tend to just pick their favourite players or players from their favourite teams, regardless of how well they are likely to perform in the game. Whether it be for monetary reasons, or for the simple pleasure of getting bragging rights over friends, the use of these platforms has shot up particularly in the last 2 years. One of the biggest AI platforms is *Fantasy Football Hub* [5], which launched in 2019 and has grown to over 40,000 paying users, with around £2.5m in annual revenue [6]. The platform uses a longitudinal multilevel regression model [7] in combination with the football data platform *Opta* to analyse each player's potential. The model is used to offer users of the platform recommendations for transfers, alongside a spreadsheet of points players are expected to score each week. By using this information to deliver customers clear and precise recommendations, *Fantasy Football Hub* has been successful in capturing a large customer base who can market their platform through word of mouth.

Another tool people consult when making FPL related decisions is social media, where users often share their teams with each other and discuss potential players to buy or sell. It was found that over 70% of FPL players find social media to be at least somewhat influential when it comes to making decisions around their team [8]. The FPL subreddit r/FantasyPL has over 700k members [9], and the official FPL Twitter account has a whopping 6.2m followers [10]. As well as the official accounts run by the Premier League themselves, many so called 'experts' have started to gain a large following on Twitter. Accounts such as `@FPLGeneral`, `@FFScout`, `@BenCrellin`, and `@LetsTalk_FPL` all have well over 250k followers, where useful tips and insights are often posted to help followers decide who to buy and sell each week.

However, using social media can often be a tedious process, sifting through thousands of posts trying to decide which ones to listen to - a stark contrast to the ease of use AI platforms bring. By leveraging sentiment analysis, this manual process can be automated, enabling AI to sift through vast amounts of data and extract meaningful insights. This study aims to explore how sentiment analysis can be applied to social media data and used in combination with existing AI models to enhance the accuracy of recommendations for FPL managers. In doing so it adds to the ongoing discussion around integrating human feedback into existing AI models used to forecast FPL performance. This study hopes to explore new areas with a particular emphasis on Twitter, a platform previously neglected when it comes to FPL research, due to the complicated jargon found in typical posts.

1.2   Rules of FPL

Note the terms *player* and *manager* are often confused when it comes to FPL, so please be aware that this study uses the term *manager* to refer to people who participate in the game of FPL, and *player* to refer to real-life premier league footballers.

Fantasy Premier League is a mirror of the real-life English Premier League, meaning that each in game event (referred to as a *gameweek*) represents a set of real-life football matches. The game takes place over a *season* which usually runs from August until May of the following calendar year, and at the end of the season the winner is crowned. There are 20 teams who play each other twice over the course of the season, resulting in a total of 38 gameweeks. Note that sometimes due to unforeseen circumstances like bad weather or domestic cup fixtures, games are postponed resulting in a team not playing in one gameweek (known as a *blank* gameweek), and playing twice in another gameweek (known as a *double* gameweek) to make up.

Upon the start of the season, each manager is tasked with choosing a team of players from a database of around 500. A team must be made up of 15 players, consisting of 2 goalkeepers (GKs), 5 defenders (DEFs), 5 midfielders (MIDs), and 3 strikers (STs). On top of these positional constraints, each player is assigned a monetary value in £m's (roughly according to how well they performed in the previous season), and the manager is given a budget of £100m from which they must select their team. A final constraint on team selection is that you cannot select more than 3 players from the same team, so it would not be possible to buy the whole Manchester United team, for example.

Replacing a player in your team with another is known as a *transfer*. Each manager is given 1 free transfer (*FT*) every gameweek, which can accumulate up to a maximum of 5. Managers who make a number of transfers that is higher than their free transfer balance must pay 4 points per additional transfer - referred to as a *hit*. Transfers cannot lead to illegal teams, i.e. teams resulting from transfers must still meet the positional, budgetary, and 3-player per team constraints mentioned above.

Another variable affecting team selection is price changes. Throughout the season player's values will go up and down depending on how popular they are among the player base. The more people buying a player, the more expensive they become and vice versa. This means managers can grow their budget by selling players who have gone up in value. When a manager sells a player who has gone up in value, the manager receives 50% of that increase rounded down to the nearest £0.1m. So if a manager bought a player for £5m and sells them at £5.5m, the manager will only receive £5.2m back. However, when a player goes down in value, the manager suffers the full price loss.

Every gameweek, managers must choose 11 *starters* (a *starting 11*) and 4 *substitutes* or *'subs'*. The starting 11 is the set of players who contribute to a
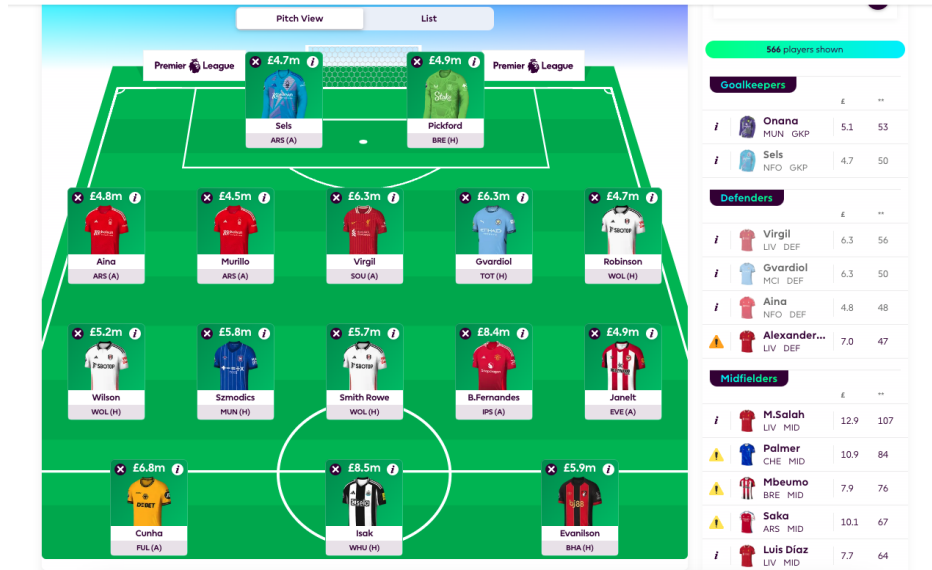
Figure 1.1: Initial Team Selection

manager's overall score (points scored by substitutes do not count towards their total). A starting 11 must consist of at least 3 DEFs, 3 MIDs, 1 ST, and exactly 1 GK. Subs will replace starters in the event that the starting player does not play a single minute in their real-life match. Before the start of the gameweek, the manager must choose a substitution preference, which is the order in which substitutes replace starters. This preference does not supersede the starting 11 constraints - that is, if a defender did not play in real-life and a midfielder was first in the substitute preference, the midfielder would not replace the defender if it resulted in a team with less than 3 defenders.

One of the most important choices a manger must make is who to *captain*. Giving a player captaincy places a 2x multiplier on their points, meaning if they were to score 10 points the manager would receive 20. A vice-captain is also chosen, and in the case that the captain is replaced by a substitute they will become the new captain.

A gameweek *deadline* is the final opportunity for managers to make changes to their teams for that gameweek. This deadline always occurs 90 minutes before the first real-life match of the gameweek. Once the deadline passes, any changes made will apply only to the following gameweek. Points are then awarded based on the outcomes of the matches over the next few days. This process of team selection, deadlines, and scoring continues throughout the season, resulting in a leaderboard ranking that determines the winner. The points scoring system can be found here

1.3 Project Aims and Objectives

1.3.1 Goals

The main goal of the project is to create a new model that will incorporate Twitter data about English Premier League players as an input. Using a fine-tuned sentiment analysis model to analyse the Twitter data, the sentiment towards each player should be incorporated into the new model to create a forecast of a player's expected points (xP) for each *gameweek*. A team selection algorithm to pick an optimal squad based on each player's xP and price will then be created to evaluate the overall performance of the model and compare it to existing models. These include models that use other sources of data for sentiment such as news articles and betting markets, as well as model with no sentiment information.

To the knowledge of the author, no research has been done surrounding sentiment analysis of Twitter for player performance forecasting in the FPL domain. This study aims to advance research in this area and further contribute to the existing fantasy sports forecasting research. The results of the model will be tested against existing research and the most popular AI FPL platforms. As well as this, models that have used sentiment analysis of other forms of media such as news articles will be compared to test the suitability of using Twitter as a data source. The model will be evaluated over a range of gameweeks and the team picked from the team selection algorithm will have its overall score compared against other prediction methods to see where it ranks in the global leaderboard.

1.3.2 Objectives

1. Analyse the project aims and break them down into key functional and non-functional requirements.

2. Conduct a literature review on fantasy sports forecasting and sentiment analysis techniques, as well as research on existing FPL forecasting platforms.

3. Fine-tune a sentiment analysis model specifically for capturing player sentiment in the FPL context, using labeled data for model training and validation.

4. Develop and validate a predictive model that uses Twitter sentiment, historical player data, and match fixtures to forecast a player's expected points (xP) for each gameweek.

5. Implement a team selection algorithm that optimises squad selection based on player xP, prices, and FPL constraints, ensuring compatibility with official game rules.

6. Test and evaluate the model by using the chosen team to simulate its

performance over multiple gameweeks against existing FPL forecasting methods, both with and without sentiment analysis.

7. Prepare a comprehensive report documenting the findings, challenges, and recommendations for future work by the end of the project.

1.3.3 Statistical Objectives

- The main goal of the project is to develop a model which achieves a higher overall rank than existing models over the course of an FPL season.

- Evaluate the effectiveness of Twitter sentiment analysis in improving xP prediction accuracy by comparing the root mean squared error (RMSE) of models with and without sentiment analysis.

- Assess the impact of the sentiment based model on team performance by comparing the total points scored by squads selected with and without sentiment-informed xP predictions.

- Measure the success of the team selection algorithm by analysing its ability to pick the best players from all combinations.

- Benchmark the model's results against popular AI-powered FPL platforms by comparing overall ranks, total points, and average weekly points.

- Compare data sources by conducting a statistical comparison against models using sentiment derived from other sources such as news articles.

- Ensure sufficient long-term prediction ability by analysing model performance across multiple gameweeks, accounting for variability in player form, injuries, and match conditions.

Chapter 2

Literature Review

Due to the huge popularity of not only Fantasy Premier League, but the multiple games available for different sports under the fantasy sports genre, there exists a wide plethora of research and discussion around the given problem of maximising performance in fantasy sports. This chapter looks at the existing models that have been created and reviews how different machine learning methods stack up against each other. The existing research suggests that whilst many different techniques have been applied to varying degrees of success, using Twitter sentiment to predict player performance over the course of a whole season is a novel approach to the fantasy sports optimisation problem. The theory of wisdom of crowds and human-feedback aided models are explored to verify the benefits of using Twitter as a source of information. Finally, existing sentiment analysis methods are analysed to see how natural language is processed and evaluated to gather an overall sentiment value. The combination of all the research gives a good understanding for the necessity and validity of the approach this study focuses on.

2.1   Fantasy Sports Forecasting

The problem of optimising a player's score over the course of a whole fantasy season is an extremely difficult task. The magnitude of this difficulty was illustrated by Kristiansen et al. who developed a mathematical model to describe fantasy premier league, and found that during the 2017/18 season the top manager in the world only managed to achieve a score equivalent to 51.75% of the optimal solution [11]. Furthermore, the mean gap between the optimal solution and the top manager was around 60 points per week, whilst the mean gap between the top manager and the average manager was only 20 points per week. This huge gap highlights the complex intricacies of fantasy sports optimisation, where the sheer number of variables like player mood, injury status, and team rotation make it almost impossible for human strategies to get anywhere close to

an optimal solution. The large discrepancy between optimality and the current peak of human performance shows the potential for advanced machine learning methods to bring new insights and strategies that could help bridge the gap between human strategies and optimal performance.

In terms of existing mathematical and machine learning models, many solutions for predicting performance have already been created by researchers dating back to the early 2000s. These models all largely focus on using historical data with objective, performance-based metrics that describe exactly what a player or team has done in the past few weeks, combined with some sort of score that represents how difficult their upcoming games are going to be. In 2012 Matthews et al. modelled the problem of choosing a team as a belief-state Markov decision process, where player selections were actions that had some reward value [12]. Using a Bayesian Q-learning algorithm actions were chosen to maximise long-term reward, resulting in a team expected to score the most points. This approach retrospectively would have ranked within the top 1% of all managers during the 2010/11 season of FPL.

In 2018 GS created a binary classification model to classify players into groups that are 1 - expected to score at least 4 points in the next gameweek, and 2 - expected to score less than 4 points [13]. He experimented with using different tree models but eventually concluded using a Gradient Boost model was the most accurate, using historical performance data to generate a number between 0 and 1 for each player. A threshold value was then determined to best split the players into the two separate categories. This approach proved successful, with the model achieving a precision of 83% when choosing players expected to score at least 4 points in the next gameweek.

In 2022 Rajesh et al. used random forests and gradient boosting machines to create a system enabling the "average interested person" to make better decisions about who to include in their teams [14]. A key feature found in this study was that training multiple models for each playing position (goalkeeper, striker etc.) drastically increased performance in comparison with using one model for each position. All models used in this study were found to outperform the average player by at least 20%, with Gradient Boosting Machines (GBMs) performing the best.

Also In 2022 Bangdiwala et al. compared three different methods - Linear Regression, Decision Trees, and Random Forests - for predicting the total number of points players would score in their upcoming match [15]. He found that over the course of a whole season, the Linear Regression model performed the best with a smaller root mean square error and mean absolute error than the other two models. The Random Forest model was a close second and the Decision Tree model performed the worst. Another study in 2024 by Papageorgiou et al. compared 14 models for fantasy basketball and found the most effective models to be Random Forests, Bayesian Ridge and AdaBoost [16].

2.2　Wisdom of Crowds

In his 2005 book *The Wisdom of Crowds*, James Surowiecki explains how collective groups are often much better at predicting things than individuals [17]. Aristotle is credited as the first person to write about this theory in his work *Politics*. He stated "it is possible that the many, though not individually good men, yet when they come together may be better, not individually but collectively, than those who are so, just as public dinners to which many contribute are better than those supplied at one man's cost" [18]. This is illustrated in Francis Galton's *Vox Populi* where he describes a country fair in Plymouth in 1906 [19]. During the fair, a contest was being held to guess the weight of an ox. Galton observed that the median guess of a crowd of 800 people was within 1% accuracy of the correct number. Suroweicki uses this example to explain how "a crowd's individual judgement can be modelled as a probability distribution of responses, with the median value being close to the true value of the predicted quantity" [17].

An expansion on this theory is a new technique dubbed "surprisingly popular" [20]. This technique was discovered during a study at MIT's Sloan Neuroeconomics Lab in collaboration with Princeton University. In the study, participants were asked a series of questions for which they had to provide what they thought was the correct answer, alongside what they thought the most popular answer would be. Researchers found that taking the average difference between the two responses as the correct answer, reduced errors by 21.3 in comparison to simple majority results, and 24.2 percent in comparison to confidence weighted results, where participants gave a confidence score alongside their answer to express how confident they were with their answer.

Taking this knowledge back into an FPL context, research exists detailing how the use of crowd-based metrics such as a player's ownership percentage among other mangers can benefit overall performance. In the earlier mentioned model built by GS, he states how using the concept of wisdom of crowds by adding features like player ownership%, transfers in, and transfers out to his gradient boost model improved the precision from 75% to 83.33% [13]. Another study in 2019 by Bonello et al. details how expanding on existing models with human feedback such as news articles and betting markets led to a performance increase of over 300 points over the course of a whole season, ranking within the top 0.5% of players in the world [21]. This is compared to a standard statistical model that only placed within the top 13%. The Bonello study also details how they explicitly decided against the use of Twitter posts in their model, as they found it hard to accurately derive sentiment from tweets due to grammatical errors, emoji usage, and football specific jargon. However, this study aims to use proper pre-processing and more accurate sentiment analysis methods that have emerged since 2019 (detailed in subsequent chapters) to eliminate these concerns, as there is a huge amount of data available on Twitter from a diverse crowd of people that should not be overlooked.

This is backed up by a 2022 study where Whittaker found that 72% of FPL players found social media to be at least somewhat influential in their decision making when it comes to their FPL team [8]. Twitter was also central to a 2019 study by Bhatt et al. where crowd wisdom was put to the test using Twitter sentiment as a metric for creating diverse crowds [22]. FPL related tweets were collected, and then matched to real people's FPL accounts. User's historical player selection was then collected, focusing specifically on who they chose to captain each week. The Twitter users were then clustered into diverse crowds based on the semantic diversity of their tweets, and further sampled from the clusters to create a final set of diverse crowds. Analysing the captaincy choice from these groups found that on average, the captaincy choice from a random group of 6 outperformed 73% of individuals, and the choice of a diverse group of 6 outperformed 93% of people.

2.3 Sentiment Analysis

The desire to capture and give meaning to public opinions has long been seen throughout history, with democratic voting as a measure of public opinion first appearing in early Greek civilisation in the 5th century BCE [23]. A paper by Droba published in the early 20th century outlines methods for collecting public opinion, explaining how early questionnaires were first deployed [24]. With the emergence of the internet in the last few decades, methods of gathering public opinion have shifted online making it much more efficient for organisations to gather relevant information. Companies have become interested in gathering opinions about their products or services through online reviews. With the explosion in popularity of social media, individual researchers have been able to gather information for tasks like predicting elections, stock market trends, and natural disasters [25].

Liu defines sentiment analysis or opinion mining as the field of study that analyses people's opinions, sentiments, evaluations, attitudes, and emotions from written language [26]. Patel et al. adds to this definition explaining it is a type of classification in which machine learning techniques are used to identify positive and negative words or reviews in text-driven databases [27]. Shah outlines the different methods that can be used for sentiment analysis, explaining their pros and cons [28]. In her article she explains the different machine learning models that are commonly used like the Naive Bayes algorithm that uses probability to determine whether a piece of text should be classified as positive, negative, or neutral. Recurrent Neural Networks (RNNs) and their variants Long Short-term Memory (LSTM) are mentioned due to their ability to handle long term dependencies often found in natural language.

In 2019, a new language representation model called Bidirectional Encoder Representation from Transformers (BERT) was introduced by Devlin et al. with a focus on pre-training deep bidirectional representations by jointly conditioning on both left and right context in all layers [29]. The ability of BERT to un-

derstand language context from both directions (unlike LSTM) more accurately meant it set new benchmarks in the natural language processing community, and has become a cornerstone of modern NLP research. BERT can be fine tuned on a multitude of tasks including sentiment analysis, and a study by Elankath et al. found that when used for sentiment analysis of Malayalam tweets, BERT was the top performing model in terms of accuracy when compared to other models like LSTM [30].

Targeted Aspect-Based Sentiment Analysis (TABSA) aims to determine sentiment toward specific targets, such as individuals or entities [31]. This idea is particularly important in the context of FPL sentiment analysis because tweets often mention multiple players. Identifying sentiment toward specific players is far more valuable than assessing the overall sentiment of a tweet. Sun et al. proposed a methodology using BERT to address TABSA by framing it as a sentence-pair classification task [32]. Their approach involves constructing an auxiliary sentence, such as "What do you think of the safety of location - 1?", and pairing it with the relevant context to identify the sentiment toward the specific target. Building on this idea, Hoang et al. advanced the methodology in 2019, achieving state-of-the-art results across various sentiment analysis benchmarks [33].

2.4   Summary

From the research explored, it is clear that there is a large gap between the points scored by the top performing manager in the world and the ceiling of potential points available. This gap highlights the potentially undiscovered strategies that machine learning models could discover to help bridge that gap and outperform other human managers. The existing models used for optimising FPL performance have been able to perform well with some ranking inside the top 1% of all players in the world. Of these models, some of the best performing include Random Forests, Linear Regression and Gradient Boosting Machines. It is also clear that incorporating human feedback and crowd wisdom into these models helps to improve their performance by a significant margin. Using high-performing deep learning models like BERT can give accurate sentiment analysis of tweets, whilst targeted aspect-based sentiment analysis can further improve this by capturing sentiment towards people and groups.

Chapter 3

Theory

The purpose of this chapter is to provide a deeper understanding of the fundamental machine learning methods and models used in this project. Additionally, it highlights the key theoretical concepts learned during the project's development. Whilst a model could technically have been developed without any of this knowledge, its addition has been extremely helpful in shaping the direction of development.

The implementation makes use of various APIs, which often function as 'black boxes' offering predefined methods for training and evaluation without revealing their internal workings. Gaining insight into the underlying processes of these API calls has been essential in refining the model. It has not only improved interpretability but also allowed for more effective debugging, as errors and unexpected behaviours become clearer when understood in context.

A strong theoretical foundation has also helped when it comes to model selection, feature engineering and hyper-parameter optimisation. For example, understanding the key differences in gradient boosting methods with their pros and cons has helped to ensure consideration for bias-variance trade-offs by avoiding overfitting. By including detailed explanations of these concepts it is hoped the reader gains a greater appreciation for the choices taken in this project.

3.1   The BERT Model

Bidirectional Encoder Representation from Transformers or BERT [29], is a pretrained language representation model based on the transformer architecture proposed in the now infamous 2017 paper "Attention is All You Need" [34]. The transformer is a deep learning architecture that is the backbone of some of the most prominent achievements in AI in recent years, including OpenAI's ChatGPT [35].
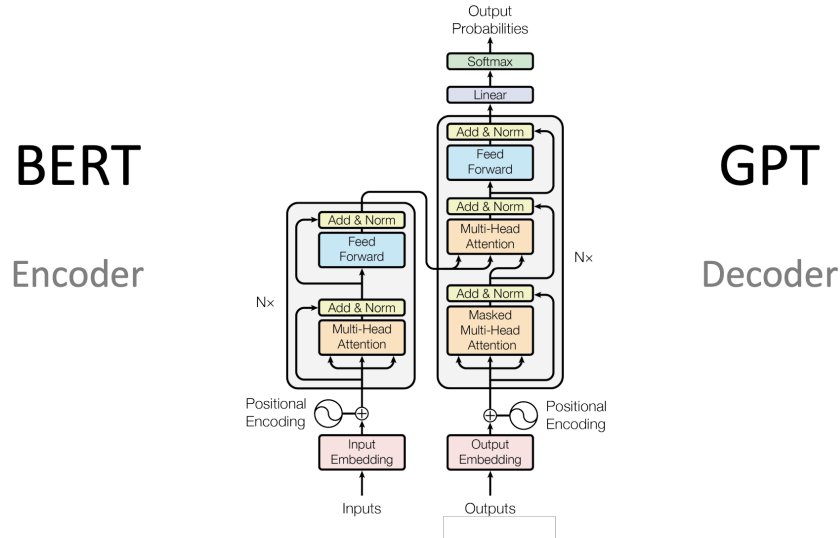
Figure 3.1: Architecture of a Transformer [34]

The original transformer architecture was designed for language translation, but has since been adopted and used for a vast array of other tasks, including for pre-trained systems like BERT and Generative Pre-trained Transformers (GPTs) [36]. The transformer is made up of two parts; the encoder, and the decoder. A few months after the transformer architecture was proposed, researchers started experimenting with the idea of separating the encoder and the decoder resulting in some incredible breakthroughs. The creation of encoder-only and decoder-only transformers is what has resulted in the AI boom that we have seen today with large language models like GPTs [35] and BERT [29]. BERT is made up of deeply bidirectional encoder-only transformers which although have been more understated than their decoder-only siblings (used in ChatGPT), are still extremely powerful for natural language tasks like sentiment analysis.

Tokenisation

To split large pieces of text into more manageable chunks of data so that natural language models can process and understand them better, a process known as tokenisation is carried out on the training data [37].

In the BERT model, raw text is broken down into tokens using the Word-Piece tokenisation algorithm [38]. As opposed to word-level or character-level tokenisation, where whole words or individual characters represent tokens, the WordPiece algorithm is a subword-level tokenisation algorithm. This means words are split into one or more tokens such as 'token' and 'isation'.

The algorithm starts by generating an initial vocabulary from the training data,

by splitting each word in the data into individual characters. Each character that does not start a word is prefixed with '##' to indicate it is a sub-word, so the word 'word' would be split like this: 'w' '##o' '##r' '##d'. WordPiece then computes a score for each pair that occurs in the training data using the following formula:

$$\text{score} = \frac{\texttt{pair\_freq}}{\texttt{first\_element\_freq} \times \texttt{second\_element\_freq}}$$

The pair with the highest score is merged into 1 token and added to the vocabulary, and the same merge is applied to the set of pairs of tokens. This process is then repeated until the vocabulary reaches a desired size. The BERT vocabulary is sized around 30k tokens [29].

Once the vocabulary has been generated, words can be tokenised. This is done by iterating through each word in a piece of text, and looking for the longest available token at the start of a word. If one is found, the algorithm does the same for the next part of the word, and so on until the whole word is tokenised. If the algorithm finds a part of a word that has no matching token in the vocabulary, the **entire word** is given the special token [UNK] or unknown [38]. So if you had a vocabulary of ['hel', 'lo', 'worl'] and the sentence 'hello world', the resulting tokenisation output would be ['hel', 'lo', [UNK]].

The BERT model also uses the special tokens [CLS] (classifier) and [SEP] (separator). The CLS token is placed at the very start of the input, and for sentiment analysis tasks it has a hidden state associated with it that will be passed to a classifier to predict sentiment after the input has been processed [39]. The SEP token is used to separate two sentences for tasks such as question-answering and dual-sentence classification [29].

Input Embeddings

After text has been tokenised, it needs to be converted into numerical values using text encoding. Text encoding allows raw text to be handled by neural networks which can only take numerical values as input [37]. The raw text is converted into a set of vectors called word embeddings, which can be processed by the encoder's neural network. Embeddings give meaning to tokens, with similar tokens like 'great' and 'awesome' closer together in vector space, and opposites like 'sad' and 'happy' further apart [41].

The embedding layer creates word embeddings (or subword embeddings, in the case of BERT) for each token in the input using 3 different types of embeddings: token embeddings, positional embeddings, and token type embeddings [40].

- **Token embeddings** are vectors that have been learned during the model's pre-training phase to place similar tokens close together in vector space, they are stored in an embedding matrix which maps each token to a specific embedding.

17

BERT Embedding Layer

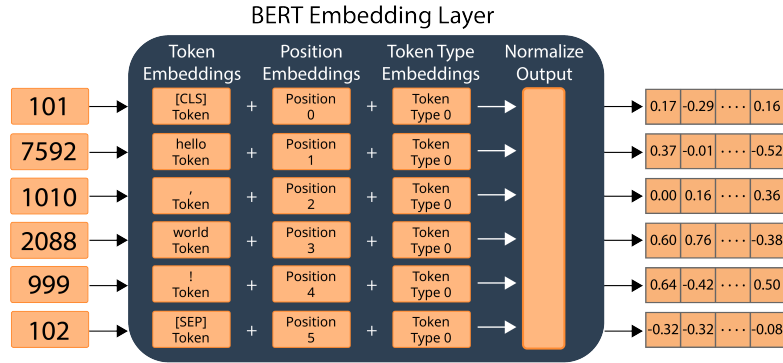|	Token Embeddings		Position Embeddings		Token Type Embeddings	Normalize Output				
101	[CLS] Token	+	Position 0	+	Token Type 0	→	0.17	-0.29	····	0.16
7592	hello Token	+	Position 1	+	Token Type 0	→	0.37	-0.01	····	-0.52
1010	, Token	+	Position 2	+	Token Type 0	→	0.00	0.16	····	0.36
2088	world Token	+	Position 3	+	Token Type 0	→	0.60	0.76	····	-0.38
999	! Token	+	Position 4	+	Token Type 0	→	0.64	-0.42	····	0.50
102	[SEP] Token	+	Position 5	+	Token Type 0	→	-0.32	-0.32	····	-0.08

Figure 3.2: BERT Embedding Layer [40]

- **Positional embeddings** are also learned vectors given to each token that represent their positions in the input sentence.

- **Token type embeddings** are often used for two-sentence NLP tasks like question-answering to identify which sentence a token belongs to. For one-sentence tasks like next word prediction, all tokens are assigned the same vector.

The embedding layer then sums these embeddings together, and applies normalisation to their sum. The resulting vector output contains meaningful information about each token and its position in the input. These embeddings are passed into the subsequent transformer layers of the BERT model for processing.

Multi-Head Attention

A fundamental concept of the transformer architecture is 'self-attention' [34]. Given the sentence: *'I took the pizza out of the oven and then ate it.'* it is apparent to any human that 'it' refers to the pizza and not oven, based on the surrounding context. Neural networks however lack the ability to identify this relationship, and require mechanisms like self-attention in order to resolve such ambiguities.

Self-attention allows neural networks to identify the importance of each token in relation to the other tokens in the input. To calculate attention values, BERT uses multiple attention 'heads' which all focus on different linguistic features of the input. Each head involves 3 components: the query (Q), the key (K), and the value (V) matrices [34]. Each of these matrices are made up of the input embeddings of the previous layer (a vector for each token), and are identical to ensure that every token in the input interacts with every other token, including itself (the idea of self-attention).
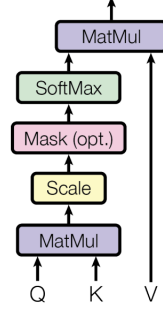
Figure 3.3: Scaled Dot Product Attention [34]

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Here QK^T computes the similarity between the key and query matrices, scaled by $\sqrt{d_k}$ (the dimension of the embedding vectors), and then the softmax function [42] normalises these values into attention weights. These weights are applied to the value matrix, resulting in an attention net matrix representing the contextual relationship between tokens.

For all attention layer heads (BERT uses h=12 attention heads [29]), there is a linear layer that uses learned weights to project the input embeddings into separate Q, K, and V matrices for each head. This way different linguistic features can be explored by the model to improve its learning ability. The outputs (attention nets) from each head are then concatenated to form a unified representation:

$$\text{MultiHead(Q, K, V)} = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O$$
$$\text{where head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right).$$

Here, W_i^Q, W_i^K, W_i^V are the learned projection matrices for the i-th head, and W^O is the projection matrix for the final linear layer. This final layer aggregates the information from all attention heads to produce a final self-attention representation for the input [34].
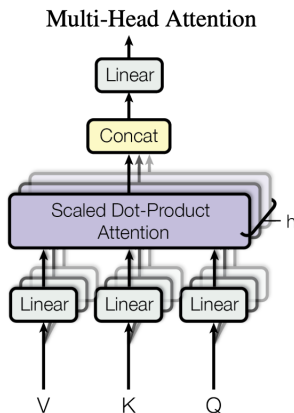
Figure 3.4: Multi-Head Attention [34]

Feed-Forward Network

The feed-forward neural network (FFNN) is the final stage of the encoder. It is a type of multi-layer perceptron (MLP), used for transforming the output of the self-attention mechanism into a refined representation that can capture deeper relationships within the input sequence [43].

The FFNN processes input embeddings by first projecting them into a higher-dimensional space. Specifically, the 768-dimensional input for each BERT token embedding [29] is transformed into a 3072-dimensional space via a linear projection. This expansion allows the model to explore more complex interactions among the input features. Following this, a ReLU activation function is applied to allow the model to identify more intricate non-linear patterns in the language. Finally, another linear projection reduces the representation back to its original dimensionality, ensuring compatibility with the encoder's output structure [43].

The FFNN is essential for capturing patterns and interactions that the self-attention mechanism alone cannot model. Its reliance on simple matrix operations makes it computationally efficient and highly parallelisable, leveraging modern hardware accelerators like GPUs and multi-core CPUs to significantly reduce training time. To ensure stable training and enhance convergence, the FFNN's output undergoes normalisation, producing the final output representation for the encoder.

Final BERT Architecture

The previous sub-sections have explored the components of the encoder, but the BERT model actually makes use of multiple encoders stacked together (12 for BERT$_{base}$ which this study uses) [29]. This is what makes BERT a deep-learning model, as it incorporates many layers into its structure. The stacking of multiple encoders allows the model to progressively learn more high-level representations of the input data as it passes through the layers.

Each encoder layer outputs a 768-dimensional matrix capturing increasingly complex patterns and relationships in the data at every subsequent encoder layer. The output of the final encoder layer represents the most refined and meaningful representations for each token [29], which can then be used for downstream tasks. For sentence classification tasks like sentiment analysis, the special token ([CLS]) is used, and its final vector representation serves as the input to the classification head. The classification head takes the input vector and outputs a score representing the model's sentiment prediction [39].

The depth of the architecture, combined with the self-attention mechanism, enables BERT to model long-range dependencies effectively. This is a huge benefit for processing natural language, where relationships between words are often spread across entire sentences or paragraphs.

Pre-Training

BERT was pre-trained using two tasks: Next Sentence Prediction (NSP) and Masked Language Modelling (MLM) [29].

The first task, MLM, is a technique for training deeply bidirectional models like BERT. Typical language models process tokens sequentially left-to-right, using preceding tokens in a sentence to predict the next. BERT, however, processes all tokens in a sentence simultaneously and can see both preceding and following tokens for each word. To predict a missing token in the middle of a sentence, BERT could "cheat" by leveraging full context, including the token to predict. To address this, word "masks" are applied to randomly hide tokens in a sentence [29], tasking BERT to predict them using only surrounding context. This technique enables BERT to learn bidirectional relationships, resulting in a more robust model than typical left-to-right ones.

The second task, NSP, teaches BERT to understand relationships between two sentences, useful for downstream tasks like Question Answering (QA) and Natural Language Inference (NLI). In this task, BERT is fed two input sentences, A and B, and must guess whether sentence B follows A or is unrelated. BERT is trained on an even split of positive and negative cases. This simple task produces remarkable results, with the final model achieving 97% accuracy on NSP tasks [29].

By leveraging these tasks across millions of sentences and performing millions of optimisation steps, BERT learns complex relationships between words and

sentences [29]. The model's weights, including those for input embeddings and self-attention mechanisms, are updated via backpropagation [44] during this process.
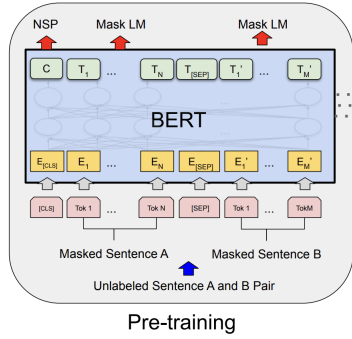

Figure 3.5: BERT Pre-Training [29]

This extensive pre-training enables BERT to capture nuanced patterns and relationships in text, which are then fine-tuned on downstream tasks like sentiment analysis, question answering, and named entity recognition [39].

3.2   Gradient Boosting and Decision Trees

Gradient Boosting is a machine learning technique used for regression and classification tasks that focuses on creating a large group of small, relatively poor models and then combining them into a single powerful and efficient model. These smaller models are typically what is known as a decision tree, a simple structure that makes predictions by splitting data based on feature values.

In order to improve the model iteratively, each new decision tree is trained to correct the residuals (actual - predicted values) of the last one. All models' answers are weighted and then summed, with the result taken as the final prediction. The process of creating new trees is repeated until a set limit is reached, or additional trees fail to improve the fit.

Compared to other ensemble methods (ones which combine multiple models) like bagging and random forests which train multiple models independently on different parts of the data, gradient boosting differs as it trains new models sequentially, based on the errors of the previous one. Each tree learns the mistakes of its predecessor, making boosting a more adaptive technique.

Decision Trees

Decision trees [45] are simple structures used for both classification and regression, that contain 3 important parts: the root node, decision nodes, and leaf nodes. These components make a up a flowchart-like tree where given some data, the tree can be traversed according to attribute values e.g. "age > 50" until a terminal or leaf node is reached that contains the prediction e.g. "is bald" or "is not bald".
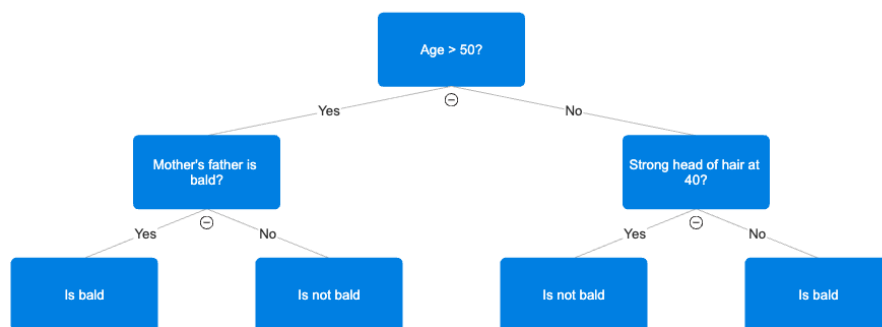

Figure 3.6: A simple decision tree to predict baldness.

Decision trees can be built using various different algorithms, although they all share the goal of trying to find the attribute that best splits the data into groups. This means trees are built in such a way as to ensure they are as small as possible whilst still providing an accurate prediction. A few metrics used to measure how well an attribute splits data include: Gini impurity, entropy and mean squared error (MSE) [45].

Whilst decision trees are simple and easy to understand, they are vulnerable to overfitting as complex trees can result in a scenario where trees start to memorise the training data rather than picking decisions that result in good general performance. To combat this techniques such as pruning can be used, and ensemble methods that build multiple trees such as gradient boosting are preferred for more complex datasets.

The Gradient Boosting Algorithm

The gradient boost algorithm builds on decision trees by using multiple decision trees that are sequentially improved to create a final model that avoids overly complex trees prone to overfitting data.

Initially an input dataset consisting of n rows is taken. Each row contains some predictor values (x_i) and the target value (y_i).

To measure how well the model's predictions stack up against the true values, a **loss function** is used. This compares the model's predictions $F(x)$ with the observed values y. For regression tasks, mean squared error is typically used:

$$\frac{1}{2}(Observed - Predicted)^2$$

The $\frac{1}{2}$ is used at the front, so that calculations are simplified when derivatives are taken (the chain rule means the squared power cancels out with $\frac{1}{2}$).

Step 1: Initial Prediction

The initial model prediction is simply the average of all known values. This is chosen as it minimises the total error across the dataset, and is determined using the formula:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$$

which simply states the predicted value $F_0(x)$ should be the one that minimises the loss function $L(y_i, \gamma)$. To calculate this the derivative of the loss function is taken with respect to F_0 and set to 0, resulting in a formula for the average:

$$F_0(x) = \frac{\sum_{i=1}^{n} y_i}{n}$$

Step 2: Building Decision Trees

After the initial prediction, a set number M (typically 100) of decision trees is created. To create a new tree m, first the residuals of the old tree are calculated:

$$r_{im} = (Observed - Predicted) \text{ for } i = 1, ..., n$$

Then a regression tree is trained to predict these residuals, with each residual value placed in a leaf or terminal value of the tree according to feature values. Often there are more residuals than leaves, so for each leaf with multiple residuals in it, the average of the residuals is taken.

After constructing the new decision tree, the model's prediction is updated. This is done by adding the tree's output values weighted by a learning rate ν (value between 0 and 1), to the initial prediction (the average). In other words, the model takes the previous prediction $F_{m-1}(x)$ and adjusts it using the corrections suggested by the new tree.

$$F(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$$

$\sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$ just means add up the output values γ_{jm} for all the leaves R_{jm} that a sample x can be found in.

Step 3: Making Predictions

Once all M trees have been built, the final model outputs the initial prediction plus the adjustments made by each tree. For new data, predictions are made by passing the input through all M trees and summing their contributions:

$$F_M(x) = F_0(x) + \text{corrections from 100 trees}$$

This process allows the model to gradually improve its accuracy, until it reaches an optimised state.

The LightGBM Algorithm

LightGBM [46] is a leaf-wise gradient boosting algorithm developed by Microsoft that focuses on fast performance speeds and efficiency over large datasets. It works by using a histogram-based technique which divides the data into discrete bins based on continuous values. As a result, instead of using the entire dataset to measure which split is the best, a bin can be used which approximates the whole dataset whilst using far less computational power and memory. The result is a much quicker process that still keeps a high accuracy.

LightGBM also uses leaf-wise tree growth instead of level-wise growth which means that instead of expanding all nodes at a given level in the tree before moving down to the next level, the node that reduces loss the most is expanded. This results in much deeper trees which are often more accurate, but also increases the risk of creating trees which overfit the data. The resulting trees are usually also harder to understand as they don't follow a balanced structure with the same number of nodes at each level.
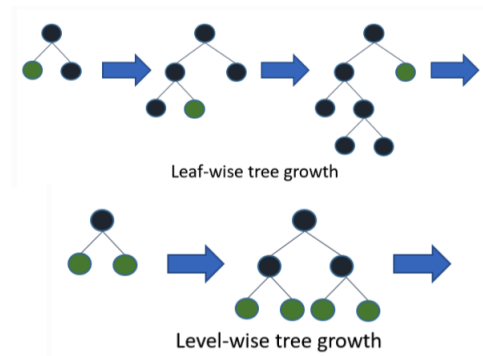

Figure 3.7: Level-wise and Leaf-wise decision tree growth [46].

Overall lightGBM has much faster training speeds and better efficiency when it comes to large and complex datasets. Its leaf-wise node expansion approach also allows for more complex and accurate decision trees than those built using normal gradient boosting methods.

Chapter 4

Methodology

This chapter outlines the approaches taken towards developing the components that make up the final prediction model. The choices taken for the specific sentiment analysis and expected points models are outlined, including the model selection, data identification and processing, training and hyper-parameter tuning, as well as evaluation methods. The team selection algorithm is explained, and the manner in which these components combine to form and evaluate the final model are detailed.

4.1 Overview of the Approach

This project focuses on developing and evaluating a machine learning model for predicting FPL player performance using twitter sentiment and historical footballing data. The methodology consists of three main stages: 1 - creation of a sentiment analysis model, 2 - creation of an expected points prediction model, and 3 - model performance evaluation.

The first stage involves fine-tuning a sentiment analysis based BERT model to classify tweets about FPL players ahead of each gameweek. The resulting sentiment predictions for tweets are aggregated into sentiment scores for each player, then integrated into a dataset containing historical player performance data. Whilst existing FPL expected points models rely solely on structured historical data, this approach investigates whether incorporating Twitter sentiment can improve predictive performance.

In the second stage, a gradient boosting machine (GBM) model is trained on the combined dataset to estimate expected points for each player. The model's output represents its prediction of how many FPL points a player is likely to score in the upcoming gameweek. The model's predictions for the final 8 gameweeks of the 2022/23 FPL season are saved as a dataset to evaluate the performance of the model.

Finally, the model's effectiveness is assessed by applying an algorithm that simulates FPL team selection to the prediction dataset. The team selection process is formulated as an optimisation problem, where integer linear programming is used to construct a fifteen-player squad and eleven-player starting team that maximises expected points whilst adhering to budget and constraint rules. This evaluation aims to quantify the impact of sentiment analysis on decision-making in FPL under real-world conditions.

By incorporating Twitter data into player performance predictions, this project explores the extent to which social media data can enhance existing FPL forecasting models. The findings may provide valuable insights for FPL managers seeking data-driven strategies to improve their decision-making.

4.2   Sentiment Analysis Model

To capture crowd wisdom through Twitter, a sentiment analysis model is needed to automatically label tweets and gauge user's opinions towards each player. Whilst it is possible to train a sentiment analysis model from scratch, this study uses a pre-trained BERT model [29] available for free from Google. This approach leverages access to a model that has been trained on billions of texts, giving it an extremely complex understanding of natural language. A model trained from scratch could not hope to replicate such complexity given the timeframe of this project. It would also be unreasonable to expect such a large task to be completed without the use of advanced computing machinery unavailable to individual researchers without the backing of a large corporation. The BERT model has been chosen specifically because it has been proven to outperform other state-of-the-art models like LSTM in sentiment analysis tasks [30, 33].

This study uses fine-tuning to refine BERT for the specific task of sentiment analysis on FPL tweets. Fine tuning is the process of adapting a pre-trained model for specific tasks or use-cases [47]. This is a much less computationally demanding task than fully training a model, and only requires a relatively small dataset which is ideal for a project of this scale. By combining the extensive natural language knowledge that BERT already has with a suitable dataset of FPL tweets, a sophisticated and accurate sentiment analysis model that suits the needs of the project will be achieved.

4.2.1   Data Collection

Identifying and collecting suitable data for fine-tuning the model is a critical step in creating a model with optimal performance. The quality of the fine-tuning data directly affects the model's output quality [48], so it was important to chose a dataset carefully.

The first approach considered was manual data collection directly via Twitter, however, exploring this approach led to many dead ends. The first problem was

the financial cost of using the official Twitter API, which has seen a price increase from $3 per month to $100 per month since the company was sold in 2022 [49]. This change has led to many free tools aimed at helping researchers leverage Twitter data for their studies being shut down, impacting the entire research field by limiting the options they have when it comes to collecting data. Without being able to use these tools, and with the official API unaffordable, using the API to efficiently collect data was not an option. Whilst other methods of collecting data exist, such as writing code scripts to scrape data from the Twitter website, or using 3rd party browser extensions, these methods are against the Twitter terms of service and could result in a permanent account ban if caught.

Manual collection of data on Twitter is an acceptable approach within the terms of service, however it is an extremely tedious process. When considering a dataset needed to fine tune a BERT model should contain thousands of entries, as well as the time constraints of this project, manually collecting and then labelling thousands of tweets was also not an option.

The chosen approach was to use an existing dataset published on popular machine learning forums such as Hugging Face and Kaggle. These sites have thousands of active users contributing to a wide range of research topics. As such, finding a dataset for a similar purpose to this study's was not difficult. 4 potential datasets were identified on Kaggle, and include:

- [Fifa World Cup 2022 Tweets with Sentiment Labels](#) - (30k tweets)
- [Premier League Teams Tweets with Sentiment Labels](#) - (460k tweets)
- [FPL Tweets from 2012 - 2023, unlabelled](#) - (110k tweets)
- [Premier League Players Tweets with Sentiment Labels](#) - (167k tweets)

A further inspection of these 4 datasets was carried out to identify the most suitable given the needs of the model being built.

The first dataset, whilst labelled, only contained tweets for the first day of the world cup. This meant a majority of tweets were centered around the event as a whole and the controversy of Qatar hosting, with little mention of players or actions they had carried out during football matches. The second dataset was much more suited to the needs of the model, however had the drawback of being centred around teams rather than individual players. The third dataset was focused specifically on FPL which appeared ideal at first. However it was found that this dataset contained a lot of unrelated tweets from people tweeting about their own performance in FPL, instead of expected performance of actual players.

The fourth and final dataset was identified as the most suitable for this study. This dataset contains a far greater number of relevant tweets, focused on individual Premier League players from the 2022/23 season (it is almost impossible to find more recent data, due to the Twitter API price changes coming into effect from 2023). As well as the relevant focus of the tweets themselves, this

	into fantasy premier league. Shows how far one good game can get yo...
ddreid88	I scored 61 points in Gameweek 20 on Fantasy Premier League http://t.co/4ys8YkcE
ahmedkungora16	I scored 71 points in Gameweek 20 on Fantasy Premier League http://t.co/6XBH1fVh
murray_rankin	My life's ambition is to one week be the highest scorer on Fantasy Premier League. #fpl #aiminghigh

Figure 4.1: Examples of noisy tweets irrelevant to player performance, such as personal FPL commentary

dataset contains player labels for each row which is perfect for construction of an auxiliary sentence for target-based sentiment analysis. Instead of manually searching for players in each tweet, the existing *player_name* column in the dataset can be used. Finally, this dataset comes with the advantage of being almost fully cleaned of any noise including emojis, links, images, etc. reducing the amount of pre-processing required.

The sentiment labels provided in the dataset were automatically created by the Vader [50], and TextBlob [51] models. These models are fairly accurate however it is worth noting they do incorrectly label some tweets and therefore this dataset is not perfect. A potential solution to this problem is manually labelling the tweets, however this would require some criteria to match certain labels to tweets, and would also be extremely time consuming. Given the time constraints of this project and the limited human resources available for labelling, the dataset's size would have to decrease significantly to use a manual labelling approach. It is hoped that using the existing labels will be more beneficial due to the much larger dataset size available with this approach, which should result in a more accurate final model.

4.2.2   Data Pre-Processing

The decision to cut the dataset down to a more suitable size of around 20k tweets was made, in order to keep training time reasonable (roughly 30 mins per epoch). Reducing the dataset size will most likely affect the model's performance negatively, however, given the time constraints of this project the computation time required to train the model on a dataset over 100k rows would have been too large.

Many tweets in the dataset mention multiple players, with some even containing conflicting sentiments e.g. "Haaland was poor today, massively outshone by Salah". In order to capture the separate sentiment values (target based sentiment) the method introduced by Sun et al. [32] was used, where an auxiliary sentence was created posing a question to the model. The chosen format was: "What do you think of the sentiment towards [player]?". Using this approach, the model inputs and outputs will look like:

- **Input:** What do you think of the sentiment towards Haaland?
- **Output:** Negative
- **Input:** What do you think of the sentiment towards Salah?
- **Output:** Positive

While a more FPL-specific auxiliary sentence (e.g., 'Should [player] be in my FPL squad?') was considered, it risked introducing noise due to sentiment labels from Vader/TextBlob. For example, 'Jamie Vardy has great hair!' might be labeled as 'yes' (suggesting he should be in an FPL squad) despite being unrelated to performance. Due to this potential issue, a more generic question was used with the intuition being that positive overall sentiment should closely correlate with positive FPL performance.

After adding the auxiliary sentence to the dataset, the rest of the tweets were cleaned of any noise. Removing noise from tweets ensures the model learns only relevant information, ultimately improving performance. Additionally, duplicates and empty rows were removed, and any other unnecessary columns were dropped.

4.2.3   Fine-Tuning BERT

To fine-tune a BERT model for sentiment analysis, the pre-trained BERT model can be modified by adding a task-specific classification layer on top of the BERT encoder stack [29]. This final layer transforms the output weight values (represented as a vector $\mathbf{h}$) into a final class 'positive', 'negative', or 'neutral' sentiment by using probabilities to predict which class the model's output most closely represents.

The classification layer is a softmax [42] classifier, which takes as input a vector **h** and uses the following equation to predict the final class:

$$p(c|\mathbf{h}) = \text{softmax}(W\mathbf{h})$$

The vector **h** corresponds to the token [CLS] which is a special token added to the input sentence before it is processed by the model. This token is modified as the input passes through the model's layers, forming a condensed representation of the entire input sequence. By fine-tuning the model specifically for sentiment analysis, this token learns to focus on sentiment information. W represents the task-specific weight matrix, which is updated during fine-tuning to minimise classification loss.
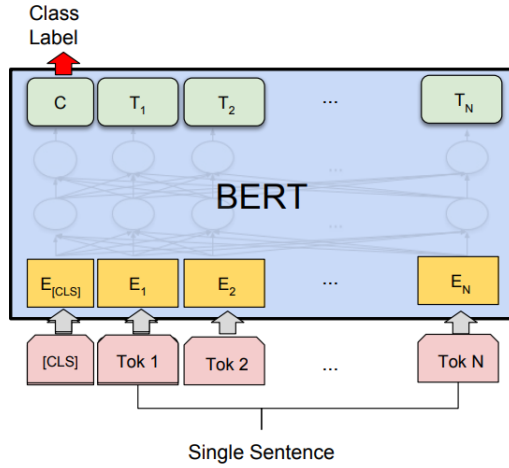

Figure 4.2: Single sentence classification [52]

To minimise classification loss, a labeled dataset of texts and their corresponding sentiment labels (e.g., 'positive,' 'negative,' 'neutral') was used. During training, the model was fed these texts and its predictions were compared to the labels. Weights were updated iteratively to improve accuracy. This process was repeated for a set number of epochs (complete passes through the dataset) to improve accuracy whilst preventing overfitting.

In order to fine-tune BERT with the auxiliary sentence method outlined above, BERT is passed the auxiliary sentence and the original tweet as a sentence pair, with a [SEP] (separator) token placed in between them. Classification is done in the same way as single sentence sentiment analysis with the hidden vector **h** used to predict the label.

The fine-tuning process not only adapts BERT to specific targets but also allows it to capture domain-specific language features that differ from general usage.

31
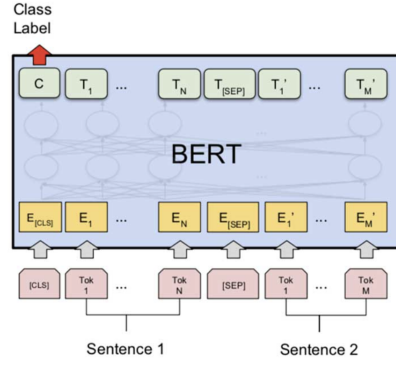
Figure 4.3: Sentence-pair classification [53]

For instance, while the word "soft" might generally convey positive sentiment (e.g., a soft blanket), in the FPL context, calling a player "soft" is often a negative sentiment. By fine-tuning BERT on domain-specific tweets, the model can learn such nuanced language patterns, improving its accuracy in the target domain.

Fine-tuning BERT is computationally efficient compared to training a model from scratch and requires a relatively small amount of labeled data. As such it was a practical approach that suited the needs of this project given the time restraints.

4.2.4 Hyper-parameter Optimisation

In order to maximise the performance of the final model, hyper-parameters were optimised. Selecting a suitable set of hyper-parameters is crucial for model performance, and adjusting these values during training can help the model to avoid overfitting the training data, minimise validation loss, and maximise accuracy [54]. Typically when implementing a machine learning model, researchers manually tweak hyper-parameters and spend significant time exploring configurations that may not improve performance meaningfully. This manual process is inefficient and resource-intensive, so by using an automatic framework, less computational resources and time is spent developing the model.

The framework chosen was Optuna [55], an automatic hyper-parameter optimisation framework for Python that finds the most optimal hyper-parameter values via trial and error. It conducts a series of trials and uses the previous trial to identify promising potential tweaks that could be made to improve performance. This process is repeated and a history of trials is kept throughout the process to guide the next changes made to the hyper-parameters. Through enough trial and error, optimal hyper-parameter values are found.

The search space for hyper-parameters was defined based on the values recommended in the original BERT paper [29]. The chosen hyper-parameters along with their respective search space were:

- Learning Rate (2×10^{-5} to 5×10^{-5}) - determines the step size at which the model updates its weights during training. A higher learning rate can speed up training time but risks overshooting the optimal parameters, whilst a lower learning rate ensures stability but may result in slower training.

- Batch Size (16 or 32) - Batch size refers to the number of training samples processed simultaneously before updating the model's weights. A smaller batch size can introduce noise in the gradient updates, which sometimes aids in escaping local minima but can lead to slower convergence. A larger batch size offers smoother gradient updates but may require more computational resources and risks overfitting.

- Number of Epochs (2 to 4) - specifies how many times the entire training dataset will be processed by the model during fine-tuning. Too many epochs can cause the model to focus on specific patterns in the training data, leading to overfitting. Too few epochs can limit the model's ability to learn from the data and lead to poor accuracy.

The datasets used during this stage were slimmed down to reduce training time. Evaluating models trained with the entire dataset would have taken too long (90 mins per trial) and used too much computing resource, so instead a dataset 25% of the size was used. It is hoped this approach resulted in a decent general view of the best configurations, of which a small amount were then explored further using the whole dataset.

4.2.5   Model Evaluation

In order to ensure the best configuration was chosen, the three best configurations discovered during hyper-parameter optimisation were used to fully train 3 models.

These models were evaluated using the following performance metrics to ensure the best model was chosen:

- **Accuracy** - The percentage of predictions that the model correctly makes across all classes.

- **Precision** - The combined percentage of predictions that are correct for each class, weighted by the number of instances in each class. E.g. if 30 samples are labelled positive, and the model correctly predicts them all, but also incorrectly predicts 10 more samples as positive the precision for that class will be $30/40 = 75\%$.

- **Recall** - The combined percentage of true instances that were correctly identified by the model, also weighted by the number of instances in each class. For the same example where all 30 positive samples were correctly predicted, the recall would be 100% for the positive class.

- **F1 Score** - The harmonic mean of precision and recall, calculated as:

$$F_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Using the F1 score as a performance metric should reward models that are both precise and able to recall relevant rows of data effectively, ensuring a robust model. F1 score is particularly relevant for sentiment analysis with multiple labels, as it ensures a model is not just performing well on the majority class, and can accurately predict less frequent classes of data.

The three models were evaluated on approximately 1000 samples taken randomly from the unseen data, ensuring no prior knowledge from training could influence the evaluation, and providing a fair assessment of the models' performance. The unseen data was processed in the exact same way as the training data to ensure consistency.

After evaluation of the three models, the best model was selected to be used to create the enhanced dataset to train the expected points model.

4.3 Expected Points Model

In order to identify the best team of players ahead of an FPL gameweek, a prediction model that can accurately forecast the number of points each player is going to score was developed. Predicting player performance in Fantasy Premier League is a well-explored problem. Many studies have used a variety of machine learning and statistical methods in order to achieve this goal. The study most similar to this one, carried out by Bonello et al. [21] in which news articles and betting markets were used to improve traditional model performance, found that Gradient Boosting Machines (GBMs) were the most effective. They state "ensemble methods show[ed] far higher suitability to the task when compared with alternate supervised learning approaches such as SVMs."

Due to the similar nature of this study (the main difference is the source of crowd wisdom coming from Twitter instead of news articles and betting markets) GBMs were chosen as the approach for expected points model. In selecting GBMs, this study builds on the success of previous studies where they have been proven to handle complex datasets with lots of interacting features. The selection of Twitter data adds a unique advantage, capturing opinions from millions of fans around the world instead of just 'expert' journalists. As well as this, FPL-specific information such as when a double gameweek is coming up for certain teams will typically not be reported by traditional news outlets.

4.3.1 Data Collection and Engineering

The core of the dataset containing historical information for each player was taken from a community-run Github repo set up by Anand [56]. This repository contains data for each FPL season dating back to 2016, including information for every player and every gameweek. This dataset was used by GS when he created his binary classification model in 2018 [13], so it made sense to combine this approach with the sentiment data collected, in order to get a fair comparison. Data for the 2022/23 FPL season was used, as this matches the timeline of the most recent Twitter data available (more recent data locked behind $100 per month API).

As well as the raw data available, rolling features were computed such as 'average_points_last_3_weeks', and 'minutes_last_3_weeks', in order to capture a greater range information about each player ahead of a gameweek. In total 21 input parameters were chosen, with roughly 3 main categories for the type of information captured by each input:

- **Information known before kick-off** - e.g., home or away, opponent difficulty
- **Player specific data** - e.g., average points last 3 weeks, goals scored last 3 weeks
- **Team specific data** - e.g., team form, team goals scored last 3 weeks

To incorporate sentiment data into the dataset, unseen Twitter data was run through the sentiment analysis model to get a sentiment value for each row in the dataset. Each tweet was then tagged with the FPL gameweek it belonged to by taking start dates for each gameweek and matching tweets to the closest start date.

This process resulted in a dataset of unseen tweets with a corresponding sentiment label, for each player and gameweek. These tweets were matched to their corresponding player and gameweek from the base dataset. For each match, two new columns, containing the number of positive and negative tweets for each player were added to the dataset, resulting in the final dataset which included sentiment information.

4.3.2 Data Pre-Processing

To pre-process the dataset before training the model, each numerical value was normalised (scaled to a value between 1 and 0). This was done so all features in the dataset had the same scale, which is important for the convergence of GBMs. Normalisation was performed using Min-Max scaling, which rescales each feature by subtracting the minimum value and dividing by the range (maximum value - minimum value). This ensures that no feature dominates the learning process due to differing scales, and that the model can more efficiently optimise during training.

After this any rows with empty values or duplicate rows were dropped from the dataset completely. Then the dataset was split into train and test datasets, with an additional evaluation set kept back for final evaluation of the model. The approach taken for this was to keep back data for the last 8 gameweeks of the season (gameweeks 31-38) so that the final model could be evaluated for a consecutive 8 weeks in a row. With more data available, it would have been better to test the model over the course of a whole season, or even in real time to see how it copes with the most up to date information. However, as mentioned many times previously, the limited amount of twitter data available for free makes this impossible to do. There may be some bias in the data with certain events like double gameweeks happening at the end of the season, but this was chosen as the best approach with the limited amount of data available.

4.3.3 Model Training and Evaluation

A Gradient Boosting Machine (GBM) model was trained to predict expected points for each player. The train and test datasets were stratified to ensure an even distribution of data from different gameweeks. Making sure data was evenly distributed was important to avoid model bias towards a particular period of the season, improving the model's ability to generalise.

Training of a GBM is slightly different to a typical neural network like the one used in the final layer of the BERT model. A GBM works by building a 'tree' (a

simple decision-making structure) that initially outputs the mean of the target values, and then calculates the residual errors (difference between the model's prediction and the true values) to build a new better tree. Instead of updating model weights like a neural network, new trees are added to reduce the loss progressively. For a more detailed explanation of decision trees and GBMs, see chapter 3.

An initial GBM model was trained with default hyper-parameters, and the resulting performance was checked to make sure there were no obvious errors or unexpected results. To find the optimal hyper-parameters for the GBM model, Optuna [55] was once again used just as for the sentiment analysis model. Root Mean Squared Error (RMSE) was chosen as the evaluation metric, as this directly quantifies the accuracy of points predictions. Due to the much faster training speeds seen when training a GBM as compared to a BERT model, 50 different combinations of hyper-parameters were explored. Early stopping was implemented to prevent overfitting so that iff the validation loss did not improve for 10 consecutive iterations, training was halted. This was to speed up the training process and reduce computational demand.

The hyper-parameter space was defined as:

- Number of Leaves (20 - 100) - controls the maximum number of leaves per tree (complexity). Higher complexity can increase performance but leaves the model susceptible to overfitting.

- Learning Rate (0.01 - 0.1) - controls the step size during gradient boosting. Lower learning rate leads to slower convergence but better generalisation, while higher learning rate leads to faster convergence but risks overshooting.

- Min Data in Leaf (100 - 1000) - controls minimum number of data points in a leaf. Lower values risk overfitting, whilst higher values capture more information per leaf so are better at generalising.

- Max Depth (3 - 15) - controls the maximum depth of each tree. Deeper trees are more complex but risk overfitting

The best-performing set of hyper-parameters was selected based and was used to train the final model. An additional model trained with no sentiment data was also saved, so that a comparison in performance could be made and any differences between the two models could be observed. The hope being the model trained with sentiment data would show some improvements to reflect the additional data made available to it.

Finally, the final evaluation data was then run through the model to get expected points predictions for each entry in the dataset. This resulting dataset containing model predictions and actual points scored for the last 8 gameweeks of the season was used for model evaluation with the team selection algorithm.

4.4   Optimal Team Selection Algorithm

To fully evaluate the performance of the final prediction model, it needs to be tested under real-world conditions. This means using unseen data to gather predictions for real FPL gameweeks. This prediction data needs to be transformed from a points prediction score for each player in the game, to a useful team recommendation.

In order to do this and form real teams that satisfy the rules of the game, a selection algorithm was constructed to find the best possible team from the information available. This selection algorithm used integer linear programming in order to select the most optimal 15 players (maximise expected points) whilst satisfying the budgetary, positional, and squad size constraints. The pool of available players contained around 500 individuals making the optimisation computationally significant and non-trivial.

4.4.1   Constraints

Detailed in section 1.2, the rules of FPL place many constraints on how a squad of players can be formed. The constraints placed on an initial squad formation are as follows:

- **Squad Size** - The number of players in a squad must be equal to 15.

- **Squad Value** - The total cost of all players in a squad must not be more than £100M.

- **Positional Constraints** - A squad must be made up of 2 goalkeepers, 5 defenders, 5 midfielders, and 3 strikers.

- **Team Constraints** - A squad cannot have more than 3 players belonging to the same real-world premier league team

As well as selecting an initial squad of 15, the algorithm needs to select a 'Starting XI' whose points will count towards the final score each gameweek. Positional constraints placed on the 11 are:

- **One Goalkeeper**
- **Between Three and Five Defenders**
- **Between Three and Five midfielders**
- **Between One Three Strikers**

As such, after running data through the team selection algorithm to get the best 15 players, an additional function to pick the best 11 players from a squad of 15 (fitting the positional constraints) was constructed.

4.4.2 Algorithm

The algorithm uses linear integer programming which is a mathematical optimisation technique. For the context of FPL team selection, the problem is defined with each player from the available pool of 500 being assigned a decision variable (set to 1 if player selected, 0 if not). An objective function is then defined which is to maximise the sum of the expected points for all players selected.

Additionally, the constraints mentioned above are modelled as a linear set of rules for the optimiser to follow when solving the problem. The PuLP library is utilised to explore all possible solutions in an effective way using branch-and-bound techniques. If a solution is possible, the PuLP solver class will output the optimal solution after finishing the search through the solution space.

Once the squad has been selected, the problem of choosing a starting eleven is simple. Initially the minimum amount of players in each position are chosen based on points - e.g. best GK, 3 best DEFs, 3 best MIDs and best ST. Then the final 3 players are chosen by taking the 3 players with the highest expected points, completing the eleven.

4.4.3 Final Evaluation with Selection Algorithm

With the team selection algorithm in place, a set of teams for each gameweeFthe k (31-38) and each model (all data, no sentiment data, and true values) were created.

To do this, the dataset created from the predictions of the final GBM models was split by gameweek, resulting in 8 gameweeks of data. Then the expected points predictions for the no sentiment model, the all data model, and the actual points scored were extracted. The data for each model and gameweek was then run through the team selection algorithm to get a squad, starting eleven, and total points scored.

For an accurate comparison against baselines like the average manager score, the captaincy was also simulated. The captain chip is used by managers to pick one player from their starting eleven whose points are doubled. In this case the player with the highest expected points had their actual points doubled to simulate them being chosen as the captain.

The resulting data for each gameweek and model was used for the final evaluation of the model.

4.5 Summary

To summarise, this chapter has outlined the three main components that have been developed in order to create and evaluate the final prediction model. First, the sentiment analysis BERT model was fine-tuned to classify FPL related tweets, generating aggregated sentiment scores for each player. Secondly, a

gradient boosting machine (GBM) model was trained on historical footballing data combined with the aggregated sentiment scores to predict how many points each player would score in an upcoming gameweek. Finally an integer linear programming algorithm was implemented to select an optimal team based on expected points predictions, simulating real world FPL decision making under the constraints of the game.

Together these components make up a complete pipeline for FPL decision making based on historical and Twitter-based sentiment data. The results and evaluation of the final model are detailed in chapter 6.

Chapter 5

Implementation

This chapter describes the technical implementation of the FPL decision making model and evaluation pipeline, providing a comprehensive overview of how each component was created. It details the specific tools, libraries, and APIs used throughout the project such as Hugging Face Transformers for sentiment analysis, LightGBM for points prediction, and PuLP for team selection. Each stage of the development pipeline is explained, with steps taken for data collection, preprocessing, model training, and hyperparameter tuning included.

On top of this, this chapter includes details of how each component was combined to form a cohesive and well-structured pipeline. It also explains how different evaluation metrics were used to assess model performance at intermediate stages throughout the development process. The techniques used to improve these metrics along with considerations such as bias-variance tradeoffs are explained, detailing how imbalanced data and generalisation across gameweeks was handled. The goal of this chapter is to provide a clear and reproducible account of how the theoretical methodology was converted into a working decision making model.

5.1   Sentiment Analysis Model

The BERT model used in this study to predict sentiment for FPL players was fine-tuned in Google Colab [57] using Python. Google Colab is a cloud computing platform that was chosen as it allows free GPU and TPU access, much better suited to fine-tuning a deep learning model than a CPU found in a typical laptop - helping reduce training time. Its Jupyter notebook format also allows for simple and clear separation of functions with markup text support, allowing for single blocks of code to be independently executed and described with headings and text. This allowed for easy debugging and troubleshooting of code, improving the development experience. Multiple Colab notebooks were set up

to create a pipeline for pre-processing data, model training, hyper-parameter optimisation, and evaluation of the model. This format allowed for easy and quick reproduction of results, making it easy to tweak parts of code and try new approaches.

Python was chosen as the preferred language due to the extensive number of libraries that exist supporting the development of deep learning models. Some of the most popular include Hugging Face's Transformers library [58] which contains APIs for hundreds of pre-trained transformer models including BERT. The transformers library also includes tokenisers for each model and a PyTorch [59] Trainer class that allows fine-tuning to be done with just a few lines of code. The model's hyper-parameters can be fine tuned and optimised using the Optuna library [55] to explore the best possible combinations. Use of these libraries ensure correct approaches are taken, and massively reduces development time by simplifying the entire process.

5.1.1   Data Pre-Processing

To start pre-processing the dataset, as much noise as possible was removed. Any image links were identified and removed by searching for occurrences of 'a href' in the texts. Empty rows and duplicates were stripped out, and regular expressions were used to check if the player label for each tweet (which player the tweet is supposedly about) is actually mentioned in the text.

For example, some entries in the dataset contained player labels such as 'Jordan Pickford' but when inspecting the tweet content, no mention of him was found. To ensure a consistent format and to remove noise likely to disrupt model training, these entries were removed from the dataset.

The auxiliary sentence needed for the target-based sentiment analysis approach outlined in the methodology was then created for each entry in the dataset. This was done by using the `player_name` column to construct a sentence in the form "What do you think of the sentiment towards [`player_name`]". The column names were adjusted and reordered so that the final dataset contained just three rows: 'question', 'context', and 'label'.

Finally, all the text data was converted into numerical format so the model could understand via tokenisation. To do this the `AutoTokenizer` class from Hugging Face's Transformers library was used to load in the standard BERT tokeniser. The columns were tokenised and then saved. The 'label' class was mapped to integers, with 0 representing positive, 1 representing negative, and 2 representing neutral.

Once this was done the dataset was split into train and validation sets using the `train_test_split` method from scikitlearn with a `test_size` of 0.1 and `seed` set to 42. The datasets were then ready to be used for training of the model.

5.1.2 Training the Model

The next step in the pipeline was to use the processed datasets to train the model's classification layer used to fine-tune it. The first step of training was to configure the `TrainingArguments` from the Transformers library. This class contains the hyper-parameters for the model such as learning rate, batch size, epochs, and weight decay. These hyper-parameters can be tweaked to improve model performance and prevent overfitting during training. Initially, these were set to default values suggested by the Hugging Face tutorial, with the intention of optimising them later (see 5.1.3).

After these arguments were configured, the Hugging Face `Trainer` class was set up. To do this the pre-trained BERT model was loaded in from the transformers library, with the `BertForSequenceClassification` variant used - because a sequence of text (the auxiliary sentence and tweet content) was being used to classify tweets. The specific model used was `'bert-base-cased'` which is the case-sensitive base variant of the BERT model. The case sensitive variant was used due to tweets often containing capitalisation to convey extreme sentiment (e.g. 'COME ON WHAT A GOAL!!'). The `Trainer` class was initialised by passing in the model, train and validation datasets, and `TrainingArguments`. Once the Trainer had been initialised, the training process was started by calling the `train()` method.
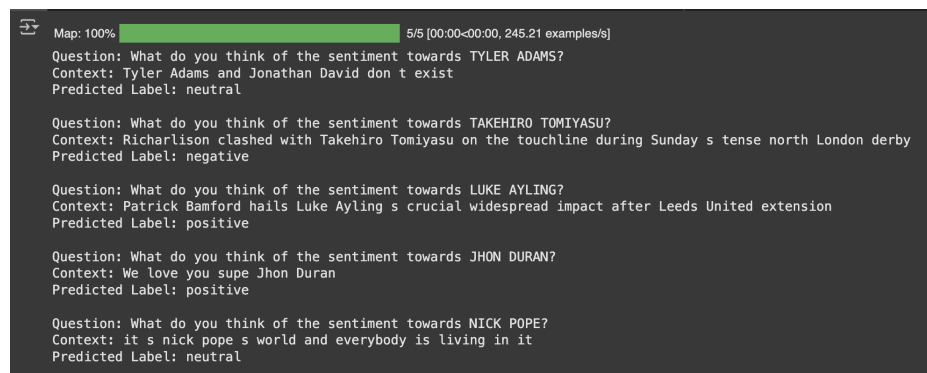
Training took around ninety minutes, and the resulting model scored an initial validation accuracy of around 90.34%. Other performance metrics such as precision and F1 score were configured and will be discussed further during final model evaluation. Focusing on the validation loss, it appeared that the initial model was starting to overfit the training data as the epochs increased, evidenced by the increasing in value after epoch 2.

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.563300	0.371035	0.874471	0.873259	0.874471	0.873576
2	0.254500	0.351286	0.894922	0.894328	0.894922	0.894551
3	0.161700	0.411752	0.894217	0.895339	0.894217	0.894556
4	0.095200	0.475788	0.906911	0.906136	0.906911	0.906134
5	0.061300	0.531063	0.903385	0.902512	0.903385	0.902537

Figure 5.1: Performance metrics for the initial model

Once trained, the initial model was tested to ensure that it produced outputs as expected. 5 samples from the unseen data were taken and passed through the model to infer an output. The unseen data was processed by the same tokeniser function used for the training data to ensure it was pre-processed in the exact same way as the training data. This was important for achieving an accurate prediction from the model as it ensured the model recognised the inputs and could leverage knowledge learned during training to the fullest extent. Inference

was done using the built in `eval()` method for the `model` class provided by the Transformers library.

Figure 5.2: The model's predictions on unseen data

Model outputs appeared to be reasonable, and were easily retrieved through the built-in methods provided by the transformers library. This validated that generating a list of sentiment predictions for a large set of tweets would be feasible, which was necessary for building the final model.

5.1.3 Hyper-parameter Optimisation

Optuna was the chosen library for tuning hyper-parameters, due to its extensive documentation and computationally efficient approach.

The first step in configuring Optuna was to specify the hyper-parameters to be tweaked and the space to be explored. To do this, an objective function was defined to guide the hyper-parameter tuning process. This function evaluates the performance of each 'trial' which is a model trained on a set of hyper-parameters suggested by Optuna.

Firstly the search space and hyper-parameters available were defined following the ranges specified in 4.2.4, for Optuna's to generate suggestions for the model in each trial. The metric used to evaluate models was also defined, with validation loss being used in this case. The validation loss is a measure of how far away the model's predictions are from the true labelled values of the validation dataset. To achieve an accurate model, this metric should be as low as possible to indicate the model's guesses are very close to the truth, thus the goal of the Optuna `study` was set to minimise validation loss.

An Optuna study was created using the `create_study` method, and then the `optimize` method was called, with the objective function passed and `n_trials` set to 10. This number of trials should allow Optuna to explore a sufficient amount of configurations while keeping resource usage to an acceptable level.

Because Optuna chooses each subsequent configuration based on the previous one, each new configuration should explore meaningful search space and therefore enable a near optimal configuration to be found in 10 trials.

The configurations and resulting validation loss for each trial were:

Trial	Learning Rate	Batch Size	Epochs	Validation Loss
1	3.895×10^{-5}	16	3	0.626
2	3.996×10^{-5}	16	4	0.629
3	3.946×10^{-5}	32	3	0.783
4	2.464×10^{-5}	32	3	0.847
5	4.504×10^{-5}	16	3	**0.524**
6	2.609×10^{-5}	16	2	0.876
7	3.742×10^{-5}	32	2	0.890
8	2.007×10^{-5}	16	2	0.844
9	3.494×10^{-5}	32	3	0.736
10	3.710×10^{-5}	16	3	0.705

Table 5.1: Hyper-parameter configurations with associated validation loss

5.1.4   Model Evaluation

The three best model configurations were fully trained in order to select the best. Whilst fully these configurations, it was initially discovered that they started to overfit due to an increase in data samples, so to compensate for this training was reduced by one epoch. Although optimising the hyper-parameters with the full dataset would give the best results, the time taken to do this (around 15 hours for 10 trials) would have been too large, so this approach was chosen as a compromise.

The configurations evaluated were the best 3 from hyper-parameter optimisation: Trial 5 (Configuration 1), Trial 1 (Configuration 2), and Trial 2 (Configuration 3).

After evaluating the models, Configuration 1 (Trial 5) stood out as the best model based on the F1-score, which is a better reflection of the model's overall ability to balance precision and recall. As such, the model trained using Configuration 1 was saved and uploaded to a newly created Hugging Face repository using the API provided, making it easily accessible to all other components of this project.

The full results and performance metrics for each configuration are discussed in Chapter 6.

5.2   Expected Points Model

The GBM model was implemented using the LightGBM library [46] due to its comprehensive documentation, simple APIs, and impressive performance aided by the use of GPU learning. LightGBM is a gradient boosting framework that uses a leaf-wise tree splitting approach rather than a level-wise one, which improves overall model performance as more complex trees can be created. More complex trees can however lead to overfitting, so ensuring that the model hyper-parameters are configured correctly is an even more important task than it would be for frameworks using a different algorithm. LightGBM has been shown to outperform other popular algorithms such as XGBoost, making it an appropriate choice for this task [60].

Two more Google Colab notebooks were created, one to facilitate the creation of the dataset combining data from the sentiment analysis model and historical footballing data, and a further to create the model training and evaluation pipeline. Again, Python was chosen as the development language due to its wide range of machine learning libraries such as Pandas and Numpy. Additionally, Matplotlib was used to create visualisations for the final evaluation of the models with and without sentiment data.

5.2.1   Dataset Creation and Feature Engineering

In order to train the GBM model, a dataset was created with the most relevant features possible. The first stage in achieving this was downloading the historical dataset from the [Github repo](). This was a simple case of downloading the raw files from Github for each gameweek of the 2022/23 season via GET request, and saving them.

From this point the 'rolling' features were computed, such as 'starts_last_3_weeks' and 'points_last_3_weeks', by using the inbuilt `.shift` and `.rolling` methods from Pandas. The dataset rows were grouped by name, team, position and gameweek to obtain unique records. From here `.shift(1)` could be used to look back 1 gameweek, whilst `.shift(1).rolling(3)` would look back at 3 previous gameweeks. This allowed for easy and efficient calculation of features, without having to use any loops or array methods. Once these calculations were done, any empty rows were dropped and the relevant features were saved to a dataframe.

The list of relevant features used for the dataset is as follows:

```
'points_last_week', 'points_2_weeks_ago', 'points_3_weeks_ago',
'avg_points_last_3_weeks', 'expected_goal_involvements_last_3_weeks',
'ict_index_last_3_weeks','creativity_last_3_weeks',
```

```
'threat_last_3_weeks', 'bps_last_3_weeks', 'minutes_last_week',
'minutes_last_3_weeks','starts_last_3_weeks',
'transfers_in_last_week', 'transfers_out_last_week',
'value_change_last_week', 'was_home_last_week',
'team_goals_scored_last_3_weeks','team_clean_sheets_last_3_weeks',
'total_points', 'value'
```

After saving these features the dataset was examined to ensure features had been correctly calculated.

index	name	position	team	gameweek	points_last_week	points_2_weeks_ago	points_3_weeks_ago	avg_points_last_3_weeks
109	Aaron Ramsdale	GK	Arsenal	34	1	2	1	1.33
110	Aaron Ramsdale	GK	Arsenal	35	3	1	2	2.0
111	Aaron Ramsdale	GK	Arsenal	36	10	3	1	4.67
112	Aaron Ramsdale	GK	Arsenal	37	2	10	3	5.0
113	Aaron Ramsdale	GK	Arsenal	38	2	2	10	4.67
117	Aaron Wan-Bissaka	DEF	Man Utd	4	1	0	0	0.33
118	Aaron Wan-Bissaka	DEF	Man Utd	5	0	1	0	0.33
119	Aaron Wan-Bissaka	DEF	Man Utd	6	0	0	1	0.33

Figure 5.3: The dataset with rolling features computed.

The next step was to add in the sentiment data. This was done by taking the unseen Twitter data from the dataset identified in the sentiment analysis chapter, and running those tweets through the sentiment analysis model to get a list of labelled tweets for every player and every gameweek. To do this each tweet from the dataset was tagged with the gameweek it belonged to, using the tweet timestamp. The start date of each gameweek was taken from the same Github repo that contains the historical data, so matching the start dates with the tweet timestamp was trivial.

Next the data was pre-processed using the same steps as model training, and then the saved sentiment analysis model was loaded using the Hugging Face API so the tweets could be run through it to get a sentiment prediction.

The positive and negative sentiment values for each player for each gameweek were then summed, resulting in a 'positive_tweets' and 'negative_tweets' counter for each player and gameweek. The player and gameweek columns were then matched with the same record in the dataframe containing historical information, and merge was performed to get a final dataset.

When implementing name matching, it became clear there were some player names that contained typos and abnormal characters (see Figure 5.4) so matching player names was not behaving as expected. To fix this, the `rapidfuzz` library was used, which is a matching library that accounts for typos. When

	player_name	text	gameweek	sentiment
1397	Paul Dummett	Mate wait till you find out about a href PaulD...	1	neutral
1398	N'Golo Kant�	Administrator you who have good contacts we ne...	1	positive
1399	N'Golo Kant�	I remember when Arsenal fans compared this guy...	1	neutral
1400	Diego Carlos	It s very careless to ignore the fact that Vil...	1	positive
1401	Cafu	Maicon is the best RB EVER and even better tha...	1	positive

Figure 5.4: The tweets after being run through the sentiment analysis model.

checking if two names are the same, it creates a confidence score based on how similar the texts are. To account for typos and missing characters a confidence threshold of 80% was set. The inclusion of this matching approach meant the sentiment data was correctly appended to the main dataframe.

This final dataframe was saved as a CSV file and then uploaded to a Hugging Face repository so that it could be used by the second notebook for training and evaluating the GBM model.

5.2.2 Data Pre-Processing

To process the dataset before it was used for training, sklearn and Pandas were used. Firstly empty rows and duplicates were removed from the dataset using the `dropna` and `drop_duplicates` methods belonging to the Pandas dataframe.

Next the data was split into training and validation sets, but first the data from gameweeks 31-38 was kept back for a final evalution dataset. The remaining gameweeks' data was split into train and validation sets using the `train_test_split` method from sklearn. The `test_size` parameter was set to 0.15. The purpose of the final evaluation dataset was to keep hidden data, so the final model could be fairly evaluated without any data leakage during training. The test and train set were used as normal for model training, with around 15k training samples and 3k test samples.

Inputs were scaled using the `MinMaxScaler` from sklearn which is a simple way of normalising the input values between 0 and 1 to prevent large values from one input being overrepresented by the model weights during training. A different scaler was used for inputs and the target variable 'total_points' so that the predictions could be unscaled at the end. The scaler was fit on the training dataset and then used to scale the final evaluation and validation sets. This way no data from evaluation and validation sets influenced the scaler values.

Finally pandas was used to remove the identifying features: name, gameweek, position, and value from the actual features, with the intent to add them back to the dataset once final model predictions had been generated. This way a dataset of expected points for each player and gameweek could be generated, with enough information to enforce team selection constraints to form a team.

5.2.3 Model Training and Hyperparameter Optimisation

Initially the model was trained with default hyper-parameters, using the class `LGBMRegressor` from LightGBM. This class makes it extremely simple to create a regression model and train it, perfect for the needs of this project. A model object of the class was instantiated passing in 'rmse' as the metric, and then the `.fit` method was called passing in the training data inputs and the target variable.

The `.predict` method was called on the same model with the validation data to get a performance overview of the model. To get the metrics sklearn and numpy were used, calculating the Root Mean Squared Error (RMSE) using the `sqrt` function from numpy and the `mse` function from sklearn. Initial training was promising, with a RMSE of just 0.0753 for the validation set. To further reduce this value, hyper-parameter optimisation was once again employed using Optuna.

The hyper-parameter search space was defined according to 4.3.3, and an Optuna study was set up and run. The results of the Optuna study are shown below:
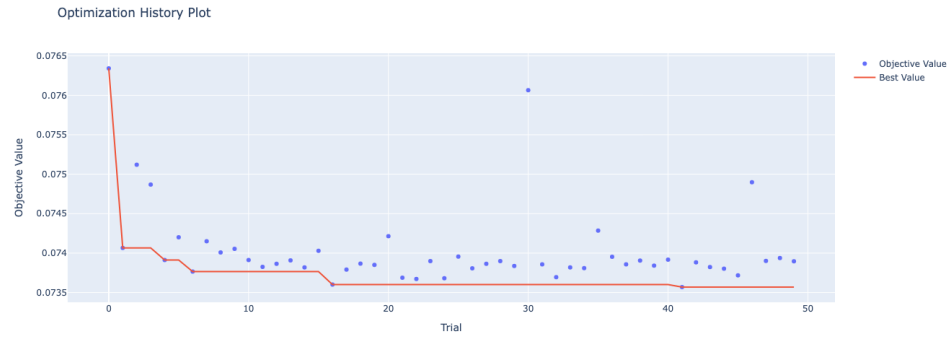

Optimization History Plot

Figure 5.5: RMSE with each trial (hyper-parameter configuration)

Because of the computational efficiency of LightGBM and the relatively speedy process of training a GBM model, 50 different combinations were explored in a short time period. Additionally the use of early stopping (where training is cut short if a model doesn't improve after a certain number of steps) after 10 steps was used to speed up training. The Optuna study was run and each trial recorded, with the best RMSE value shown as 0.07389 and best configuration identified as: {'num_leaves': 67, 'learning_rate': 0.08101, 'min_data_in_leaf': 110, 'max_depth': 7}. These hyper-parameters were saved into a 'best_params' variable so they could be used later in the pipeline to train the final models.

Additionally, the Optuna visualisation tools were used to plot a visualisation of how RMSE was reduced progressively as the search space was explored.

5.2.4 Evaluation of Sentiment and Non-Sentiment Based Models

To evaluate how sentiment data affected the model, it made sense to train another model using the same dataset minus the 'positive_tweets' and 'negative_tweets' features.

This set up allowed for a fair comparison between the two models, and any notable differences could be observed. To create both models, new dataframes for the training and evaluation data were created by taking copies of the originals and dropping the sentiment data columns. Two model objects were then created and trained on the data using the hyper-parameter configuration obtained from the Optuna study.

Initially the RMSE was calculated manually like for the initial model and then compared between the two models. By comparing the RMSE for the same target

variables in both models, we can get an idea of how far each model's predictions are from the true values. After examining the difference between the RMSE of the two models, it was found the difference was fairly small and likely within the margin of error. To combat this, cross validation was performed using the `cross_val_score` method from sklearn with a `cv` value of 5. This method splits the data into 5 training and test sets, trains a model on each split, and then averages the RMSE across all splits. Using cross-validation ensures that any observed differences between models are not due to random variability in specific train-test splits, but more general differences in model performance.

To visualise the differences between the predictions of each model, a histogram of residuals (true values - predicted values) was plotted using matplotlib. As well as this, a feature importance chart was created using lightGBM's `plot_importance` method.

These visualisations, along with RMSE comparisons are detailed in section 6.2.

5.3   Team Selection Algorithm

The team selection algorithm was also implemented using Python in Google Colab. Python was chosen because of the PuLP library that can be used to solve linear integer programming problems, and Colab was used for consistency with the rest of the project, along with the added readability that markup text provides.

PuLP allowed for the easy definition of the problem and constraints, with a simple API for solving the problem with just a few lines of code. The Colab environment allowed for quick experimentation and visualisation of results, and its familiar format used for the other components of the prediction model meant it was an ideal platform for the development of the team selection pipeline.

5.3.1   Squad Selection

The first stage of the team selection pipeline was to define a function that returned an optimal squad of 15 players based on some points predictions.

The problem was defined by creating an object from the `LpProblem` class in PuLP, with the aim set to `pulp.LpMaximize`, meaning the goal is to maximise the total number of predicted points. For each player in the dataset, a binary decision variable was created using the `LpVariable` class. This variable is set to 1 if the player is selected for the team, and 0 otherwise.

The objective function was then added to the problem. It calculates the total predicted points of all selected players by multiplying each player's predicted points by their decision variable, and summing the results. This was done using the `lpSum` function which tells the solver to pick a set of players such that the total expected points is as high as possible.

Next the constraints were added to make sure the selected squad follows the FPL rules. Each constraint was added using `lpSum` to ensure constraints were satisfied based on the player data in the input DataFrame. After defining the full optimisation problem, `problem.solve()` was used to compute the optimal squad. The final selected players were extracted by checking which decision variables were set to 1. Finally a dataframe containing the name, team, position, value, expected points, and actual points for all 15 selected players was returned.

5.3.2 Team Selection and Real-World Performance Evaulation

The second stage of the selection pipeline was to use the squad selection function to generate squads for each model and each gameweek. Then a starting eleven team was chosen from each squad, a captaincy chip was applied and the final points total was calculated so each model could be evaluated.

For selection of a starting eleven a function was defined that took the squad of 15 as input, then sorted players into dataframes for each position ordered by expected points. An initial team of 8 was constructed by using the `.head()` method for each position's dataframe:

```
starting = pd.concat([
      gks.head(1),
      defs.head(3),
      mids.head(3),
      fwds.head(1)
   ])
```

To fill the rest of the available positions, the remaining players were combined into one dataframe again sorted by expected points, and the top three players were selected to form a team of eleven.

With both functions defined, the teams and resulting points for each model and each gameweek could be generated. To do this a dictionary was created with arrays for the `all_data`, `no_sentiment_data`, and `optimal_score` models. Then the gameweeks were looped through, extracting the relevant columns for expected points for each model and the true points scored, this information was added to the arrays in the dictionary.

Then for each gameweek and model, the `select_fpl_team` function defined earlier was called with the relevant data to get the optimal squad. This was run through the `select_starting_xi` function to get the best eleven players, and then the actual points scored by that eleven was calculated using `.sum()`. The best player (player with highest expected points) from the eleven had their points added to the total again to simulate their points being doubled.

The resulting teams and points scored for each week were saved for evaluation, with visualisations being generated using matplotlib. The results of final evaluation are detailed in chapter 6.

Chapter 6

Results and Discussion

The goal of this project is to evaluate the performance of a new FPL prediction model constructed using existing methods, but with the inclusion of a novel sentiment data source - Twitter. As such, this chapter presents the results of two main evaluation stages: the performance of the sentiment analysis model used to extract player sentiment from tweets, and the downstream evaluation of the FPL prediction model that incorporates this sentiment data.

The final model is compared against the baseline average scores of all managers, a Gradient Boosting Machine model without twitter data, the final GBM model that includes twitter data, and the best possible score that could have been achieved. The results suggest that the lack of Twitter data available currently prevent any solid conclusions from being drawn about how effective it is as a sentiment data source. As well as this the limited timescale from which to draw comparisons (just 8 gameweeks) further limit the extent to which any concrete conclusions can be made. The GBM models constructed both perform almost identically with minimal differences.

6.1   Sentiment Analysis Model Evaluation

To assess the effectiveness of the sentiment analysis model, three hyper-parameter configurations were selected for final evaluation based on their performance during tuning. These models were trained on the full dataset (with one fewer epoch to avoid overfitting) and then evaluated using standard classification metrics: precision, recall, F1-score, and overall accuracy.

The full list of configurations explored during hyper-parameter tuning are detailed in section 5.1.3, with the top 3 performing configurations (based on validation loss) detailed below.

Configuration	Learning Rate	Batch Size	Epochs	Validation Loss
1	4.504×10^{-5}	16	3	0.524
2	3.895×10^{-5}	16	3	0.626
3	3.996×10^{-5}	16	4	0.629

Table 6.1: Hyper-parameter configurations and associated validation loss

The following tables show the classification performance for each configuration on the validation data set:

Class	Precision	Recall	F1-Score	Samples
Positive	0.95	0.75	0.84	483
Negative	0.86	0.62	0.72	190
Neutral	0.71	0.99	0.83	402
Accuracy		0.82		1075
Weighted Avg	0.85	0.82	0.81	1075

Table 6.2: Classification report for Configuration 1.

Class	Precision	Recall	F1-Score	Samples
Positive	0.96	0.44	0.61	483
Negative	0.87	0.32	0.46	190
Neutral	0.51	0.99	0.67	402
Accuracy		0.63		1075
Weighted Avg	0.77	0.63	0.61	1075

Table 6.3: Classification report for Configuration 2.

Class	Precision	Recall	F1-Score	Samples
Positive	0.94	0.55	0.69	483
Negative	0.81	0.59	0.69	190
Neutral	0.61	0.99	0.76	402
Accuracy		0.72		1075
Weighted Avg	0.79	0.72	0.72	1075

Table 6.4: Classification report for Configuration 3.

The results indicate that the model consistently struggled with the negative class across all configurations. This was expected, given the class imbalance in the dataset - negative samples were significantly underrepresented compared to neutral and positive samples. As a result, the model frequently misclassified

negative instances as neutral, leading to lower precision and recall scores for the negative class.

Conversely, the neutral class consistently achieved high recall (close to 1.00), especially in Configurations 2 and 3, although at the cost of lower precision. This suggests that the model tended to over-predict the neutral label, particularly when uncertain. The positive class maintained reasonably balanced scores across all three configurations.

Overall, Configuration 1 outperformed the others, with the highest F1-scores and balanced performance across all classes. This made it the most reliable model for classifying sentiment in this task. As such, this model was used to classify all Twitter data that was later fed into the expected points model.

6.2   xP Model Performance Evaluation

As outlined in section 5.2.4, Root Mean Squared Error (RMSE) was used to evaluate the performance of the expected points model. RMSE measures the average magnitude of prediction errors, where lower values indicate better performance. Additionally, to evaluate how much impact the sentiment data had on the model, another model was trained using a modified dataset that excludes all twitter sentiment information, and the RMSE of the two models compared.Since both models use the same GBM architecture and hyper-parameters, any differences in RMSE can be attributed to the presence or absence of sentiment data.

Cross validation was used to ensure differences in RMSE for each model were not due to randomness or a favourable train test split. The results showed that the average RMSE for the model trained with sentiment data was **0.0711 ± 0.0060**, while the model without sentiment data achieved an average RMSE of **0.0711 ± 0.0059**. Although the RMSE for the sentiment model was marginally lower, the difference falls well within the standard deviation. This seems to suggest that sentiment data did not significantly improve model performance in terms of RMSE alone.

To further explore the differences between models, and to ensure there is no significant difference that was not captured by RMSE alone, additional analysis was performed. This included a plot of the residual distribution between both models (6.1), as well as a feature importance chart (6.2).

The histogram of residuals (6.1) shows a general trend with most prediction errors concentrated towards the center and then a gradual tail-off as the predictions get further from the true values. The plot for the sentiment-informed model almost exactly fits the plot for the model with no sentiment information, suggesting no meaningful difference between the two models. If anything, there is a larger peak of exactly, or almost exactly, correct predictions (at 0.0 on the x-axis) for the model trained using no twitter sentiment information. The blue bars slightly to either side representing the sentiment-informed model's predic-
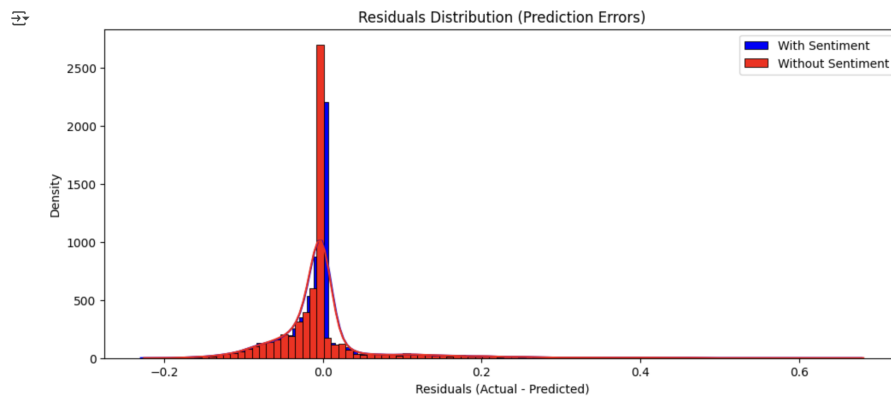
Figure 6.1: Distribution of residuals for each model

tions could suggest it tends to miss the true values by a small amount more than the non sentiment-informed model. However, the differences shown in the histogram are so marginal that they could easily be due to favourable train test splits, and thus it is hard to draw any conclusions about how they differ in terms of performance.
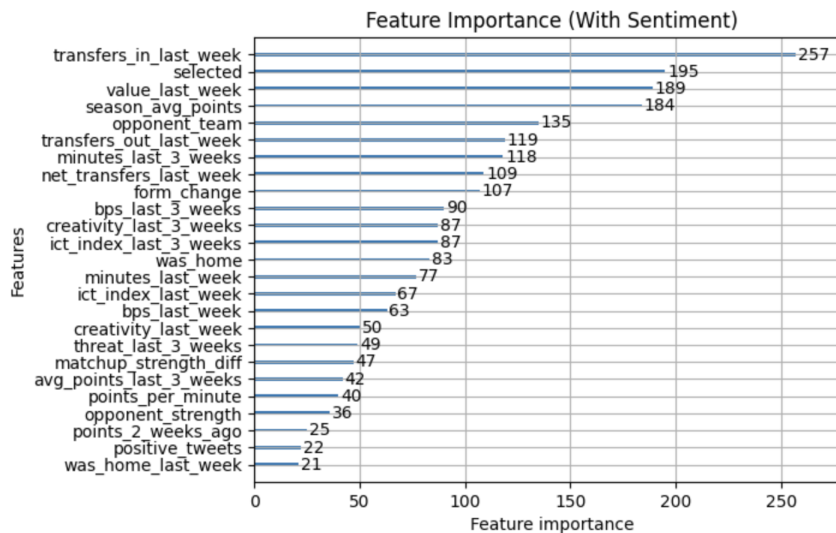

Figure 6.2: Feature importance

Plotting the feature importance of the final model, the new `positive_tweets` feature added to the dataset ranked 24th in importance, with a score of just 22. Given the fifty feature columns that make up the entire dataset, this places it within the top half of features suggesting its inclusion does add some meaningful

benefit to model performance. This may explain the small differences in RMSE between the two models. However, with a gap of over 150 importance points from the top 3, the benefit of its inclusion appears minimal.

Interestingly, the top three features all represent some form of crowd perception.or behaviour. With `transfers_in_last_week` and `selected` indicating the recent and overall popularity of a player among managers' teams, and `value_last_week` tracking how a player's price (directly tied to popularity) changes.

This suggests crowd wisdom and sentiment information are extremely important for predicting player performance - as expected. The gap between the top three features and the `positive_tweets` feature may be explained by the lack of Twitter data available during training. The importance gap could also indicate that Twitter is not a suitable data source for capturing sentiment information, perhaps due to the noise and non-FPL related content.

Overall, while crowd-based signals such as transfers and price movements appear to provide strong and reliable sentiment information, improving model performance, data taken explicitly from Twitter appears to offer only marginal additional value.

6.3   Team Selection Evaluation

Detailed in section 4.4, the final model was evaluated on unseen data over the course of an 8 gameweek period, in order to evaluate how it performed under real-world conditions. The results of simulating model performance for the final 8 gameweeks of the season were plotted (6.3, and 6.4). The final model was compared alongside the best possible score, the average score for all players, and the model trained without Twitter data.
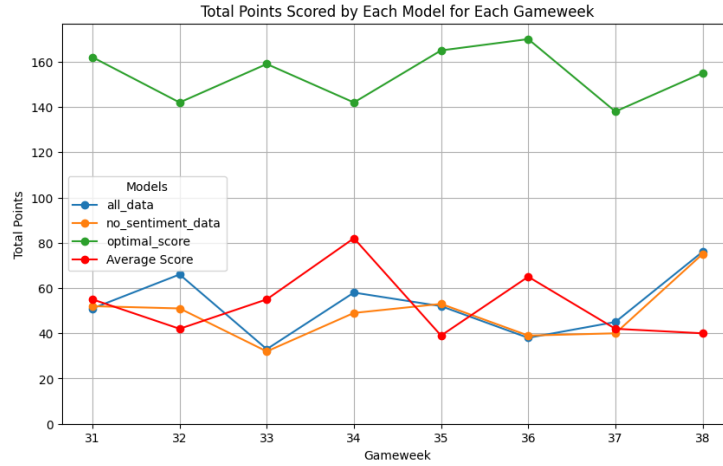
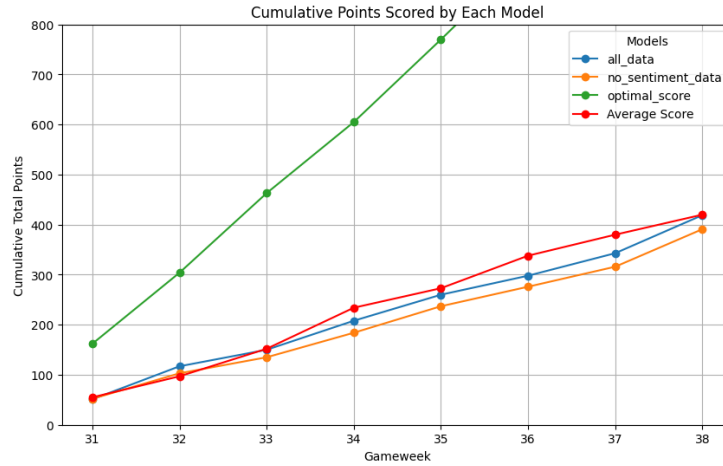

Figure 6.3: Point scores per gameweek

Figure 6.4: Point scores per gameweek (cumulative)

The gap between the optimal scores and model predictions remains extremely large - with both models and the average scoring around 90 points less in each

gameweek. Interestingly, the average manager slightly outperformed both models, although this can be mainly attributed to the time period in which evaluation took place. In the 2022/23 season, double gameweeks (where teams play two matches and can earn double the points) occurred during gameweeks 34 and 36 [61]. Managers typically make use of their *chips*, one-time power-ups that can be used to score extra points, during these gameweeks. Since the prediction model only focuses on predictions for one gameweek in the future, chips were not factored into the simulation.

This hypothesis matches up with figure 6.3 where a noticeable gap between the average score and the model scores appears during gameweeks 34 and 36, driven by chip usage. During almost all other weeks both models outperformed the average manager. With more data and a full-season simulation, the models would likely show a clearer performance advantage over the average score.

Focusing back on the prediction models, the final model trained with Twitter data outperformed the model trained without by around 25 points over the course of the 8 gameweek period. This could suggest a positive impact from including Twitter data, however the time period for evaluation is not long enough to draw a solid conclusion. Zooming in on some of the point differences between the two models, gameweeks 32 and 34 stand out to be the largest gaps. Exploring the teams selected by each model during these weeks provides some more insights:

Table 6.5: Players Exclusive to Final Model (GW32)

Player	Expected Points (xP)	Actual Points
José Sá	4.1	1
Mohamed Salah	5.3	7
Alexander Isak	4.8	13

Table 6.6: Players Exclusive to Model with no Twitter data (GW32)

Player	Expected Points (xP)	Actual Points
Nick Pope	4.4	2
Bruno Guimarães	4.7	2
Dominic Solanke	4.6	2

For gameweek 32, both models agreed on eight players for their starting eleven. The remaining three players selected exclusively by the final model outscored those chosen exclusively by the non-sentiment model by 15 points thanks to the 7 and 13 point hauls from Mohammed Salah and Alexander Isak.

To investigate if the selection of these players could be directly attributed to the twitter sentiment information, tweet counts for each player were examined:

Table 6.7: Positive and Negative Tweets for Differential Player Selections (GW32)

Player	Positive Tweets	Negative Tweets
Nick Pope	14	6
Bruno Guimarães	17	6
Dominic Solanke	2	0
José Sá	0	0
Mohamed Salah	19	2
Alexander Isak	6	6

While Mohammed Salah had a large number of positive tweets (19), which may have contributed to his selection, both Nick Pope and Bruno Guimarāes had a similarly high number of positive tweets (14 and 17) but were not chosen by the final model. The highest scorer in Alexander Isak actually had the same amount of negative tweets as positive (6) so it seems unlikely his selection by the final model was influenced by the Twitter features.

A similar analysis was conducted for gameweek 34, which had similar results. Whilst there were differences in player selection favouring the final model, tweet sentiment alone did not consistently explain these choices. This evidence seems to indicate the observed performance differences between the final model and the model trained with no Twitter information are likely not due to the inclusion of Twitter sentiment data alone.

Chapter 7

Conclusion and Future Work

This project found no clear evidence that crowd sentiment sourced from Twitter significantly improves the performance of FPL points prediction models. While crowd wisdom derived from verifiable in-game statistics - such as transfers in, transfers out, and selection percentage - appeared highly valuable during model training, the Twitter-based sentiment data did not demonstrate the same level of importance.

The exact reason why the Twitter data failed to make a significant impact remains unclear, though several likely explanations can be suggested. For one, the amount of Twitter data available was far from ideal. While the dataset contained around 147,000 tweets, dividing this over 38 gameweeks for roughly 650 players results in a negligible amount of tweets per player. Many players are not mentioned at all, likely due to their lack of popularity among fans, while a small number of high-profile players dominated the conversation (see 7.1). This unequal distribution is expected, but it poses a significant challenge as without coverage for a large percentage of the available player pool it becomes difficult to uncover potential breakout or differential picks outside of the usual high-profile selections. This reduces the likelihood of the model selecting players not already selected by existing prediction models.

Unfortunately, obtaining more Twitter data was a major challenge throughout this project due to the increasing costs of using the Twitter (now X) API. Given a larger financial backing and unlimited usage of the Twitter API, a much larger dataset would have been easy to create and a real-time prediction system would have even been possible. A richer dataset containing hundreds or thousands of tweets per player per gameweek, rather than just the few dozen observed for this project, would have likely produced a stronger signal and had a more meaningful impact of model performance.

Another factor that could have contributed to the limited impact of the Twitter data is the accuracy of the sentiment analysis model. The model's outputs are not 100% accurate, and its predictions were influenced by noise during model training. As mentioned in the methodology for the model, the model was trained using data automatically labelled by a VADER sentiment model. Using manually labelled tweets would have likely improved accuracy but was unfeasible due to the time constraints of the project. Given that manual labelling was not feasible for the already too small dataset, increasing the dataset volume and manually labelling data is not a practical solution going forward.
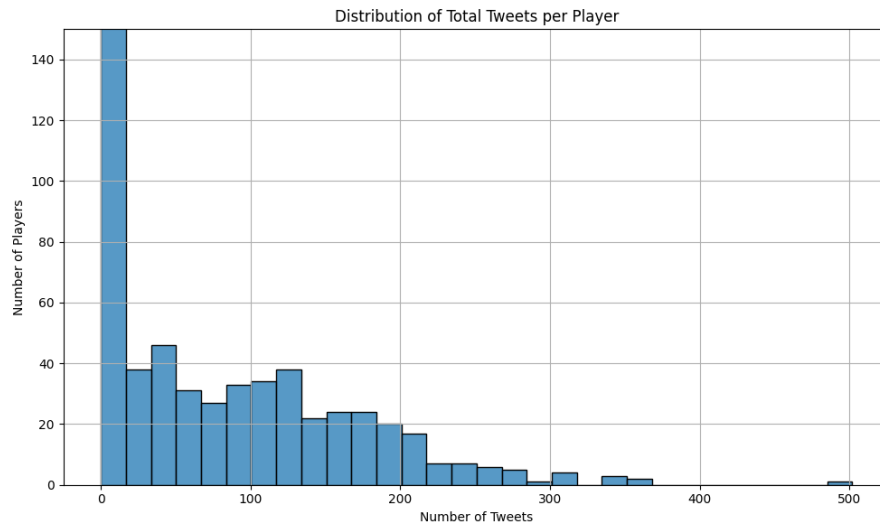

Figure 7.1: Distribution of Tweets per Player.

Finally, model evaluation was also limited by the lack of Twitter data. In order to simulate real-world conditions for final evaluation, 8 consecutive gameweeeks worth of data was kept back, leaving 30 gameweeks worth for model training.

Bibliography

[1] https://fpltips.com/the-fpl-era-the-history-of-fantasy-football/, June 2023.

[2] https://www.attackingfootball.com/how-many-people-play-fantasy-premier-league-fpl/, November 2023.

[3] https://fantasy.premierleague.com/prizes, November 2024.

[4] https://www.skyquestt.com/report/fantasy-sports-market, April 2024.

[5] https://www.fantasyfootballhub.co.uk/, November 2024.

[6] https://thenextweb.com/news/ai-is-killing-fantasy-football-fpl, September 2024.

[7] J. L. Peugh, "A practical guide to multilevel modeling," *Journal of School Psychology*, vol. 48, no. 1, pp. 85–112, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022440509000545

[8] D. Whittaker, "A study of information behaviour in the fantasy premier league community," Ph.D. dissertation, C, 2022.

[9] https://www.reddit.com/r/FantasyPL/, November 2024.

[10] https://x.com/officialfpl, November 2024.

[11] B. K. Kristiansen, A. Gupta, and W. Eilertsen, "Developing a forecast-based optimization model for fantasy premier league," Master's thesis, NTNU, 2018.

[12] T. Matthews, S. Ramchurn, and G. Chalkiadakis, "Competing with humans at fantasy football: Team formation in large partially-observable domains," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 26, no. 1, pp. 1394–1400, Sep. 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/8259

[13] R. GS, "Building an fpl captain classifier," https://medium.com/datacomics/building-an-fpl-captain-classifier-cf4ee343ebcc, Sep 2018.

[14] V. Rajesh, P. Arjun, K. R. Jagtap, S. C. M, and J. Prakash, "Player recommendation system for fantasy premier league using machine learning," in *2022 19th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2022, pp. 1–6.

[15] M. Bangdiwala, R. Choudhari, A. Hegde, and A. Salunke, "Using ml models to predict points in fantasy premier league," 10 2022.

[16] G. Papageorgiou, V. Sarlis, and C. Tjortjis, "An innovative method for accurate nba player performance forecasting and line-up optimization in daily fantasy sports," *International Journal of Data Science and Analytics*, 2024. [Online]. Available: https://doi.org/10.1007/s41060-024-00523-y

[17] J. Surowiecki, *The Wisdom of Crowds*. Knopf Doubleday Publishing Group, 2005. [Online]. Available: https://books.google.co.uk/books?id=hHUsHOHqVzEC

[18] Aristotle, *Politics*, ser. Loeb classical library. Heinemann, 1932. [Online]. Available: https://books.google.co.uk/books?id=a5y5DgAAQBAJ

[19] F. GALTON, "Vox populi," *Nature*, vol. 75, no. 1949, pp. 450–451, 1907. [Online]. Available: https://doi.org/10.1038/075450a0

[20] D. Akst, "The wisdom of even wiser crowds," https://www.wsj.com/articles/ the-wisdom-of-even-wiser-crowds-1487265722?tesla=y&mod=vocus, Feb 2017.

[21] N. Bonello, J. Beel, S. Lawless, and J. Debattista, "Multi-stream data analytics for enhanced performance prediction in fantasy football," *arXiv preprint arXiv:1912.07441*, 2019.

[22] S. Bhatt, K. Chen, V. L. Shalin, A. P. Sheth, and B. Minnery, "Who should be the captain this week? leveraging inferred diversity-enhanced crowd wisdom for a fantasy premier league captain prediction," in *Proceedings of the international AAAI conference on Web and Social Media*, vol. 13, 2019, pp. 103–113.

[23] J. Thorley, *Athenian Democracy*, ser. Lancaster Pamphlets in Ancient History. Taylor & Francis, 2012. [Online]. Available: https://books.google.co.uk/books?id=-ANoUXtMbIAC

[24] D. D. Droba, "Methods used for measuring public opinion," *American Journal of Sociology*, vol. 37, no. 3, pp. 410–423, 1931.

[25] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysis—a review of research topics, venues, and top cited papers," *Computer Science Review*, vol. 27, pp. 16–32, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013717300606

[26] B. Liu, *Sentiment Analysis and Opinion Mining*, ser. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2012. [Online]. Available: https://books.google.co.uk/books?id=AZBfAQAAQBAJ

[27] A. Patel, P. Oza, and S. Agrawal, "Sentiment analysis of customer feedback and reviews for airline services using language representation model," *Procedia Computer Science*, vol. 218, pp. 2459–2467, 2023.

[28] S. H. Shah, "Top 5 techniques for sentiment analysis in natural language processing," https://medium.com/illumination/top-5-techniques-for-sentiment-analysis-in-natural-language-processing-c07ba5b83f64, Dec 2023.

[29] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[30] S. M. Elankath and S. Ramamirtham, "Sentiment analysis of malayalam tweets using bidirectional encoder representations from transformers: a study," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 3, pp. 1817–1826, 2023.

[31] M. Saeidi, G. Bouchard, M. Liakata, and S. Riedel, "Sentihood: Targeted aspect based sentiment analysis dataset for urban neighbourhoods," 2016. [Online]. Available: https://arxiv.org/abs/1610.03771

[32] C. Sun, L. Huang, and X. Qiu, "Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 380–385. [Online]. Available: https://aclanthology.org/N19-1035

[33] M. Hoang, O. A. Bihorac, and J. Rouces, "Aspect-based sentiment analysis using BERT," in *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, M. Hartmann and B. Plank, Eds. Turku, Finland: Linköping University Electronic Press, Sep.–Oct. 2019, pp. 187–196. [Online]. Available: https://aclanthology.org/W19-6120

[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[35] "Chatgpt architecture," https://medium.com/@ashish.sharma1981/chatgpt-architecture-exploring-the-inner-workings-of-the-language-model-41731fc05483, October 2023.

[36] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von

Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Q. Liu and D. Schlangen, Eds. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://aclanthology.org/2020.emnlp-demos.6

[37] D. Jurafsky, "Speech and language processing," 2000.

[38] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. [Online]. Available: https://arxiv.org/abs/1609.08144

[39] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" 2020. [Online]. Available: https://arxiv.org/abs/1905.05583

[40] G. Novak, "Bert embeddings," https://tinkerd.net/blog/machine-learning/bert-embeddings/, March 2023.

[41] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [Online]. Available: https://arxiv.org/abs/1301.3781

[42] I. Goodfellow, Y. Bengio, and A. Courville, "Softmax units for multinoulli output distributions. deep learning," *Preprint at*, 2018.

[43] K. Gomez, "The feedforward demystified: A core operation of transformers," https://medium.com/@kyeg/the-feedforward-demystified-a-core-operation-of-transformers-afcd3a136c4c, December 2023.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*. Cambridge, MA, USA: MIT Press, 1988, p. 696–699.

[45] IBM, "What is a Decision Tree?" Retrieved March 2025 from https://www.ibm.com/think/topics/decision-trees, 2025.

[46] G. for Geeks, "Light GBM," Retrieved February 2025 from https://www.geeksforgeeks.org/lightgbm-light-gradient-boosting-machine, 2025.

[47] D. Bergmann, "What is fine-tuning?" https://www.ibm.com/topics/fine-tuning, March 2024.

[48] R. Awati, "Garbage in, garbage out (gigo)," https://www.techtarget.com/searchsoftwarequality/definition/garbage-in-garbage-out.

[49] J. Porter, "Twitter announces new api pricing, posing a challenge for small developers," https://www.theverge.com/2023/3/30/23662832/twitter-api-tiers-free-bot-novelty-accounts-basic-enterprice-monthly-price.

[50] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," 01 2015.

[51] "Textblob," https://textblob.readthedocs.io/en/dev/, December 2024.

[52] H. Nuraliza, O. Pratiwi, and M. Lubis, "Metaverse tweet sentiment text classification using bert algorithm and tunning hyperparameter," 08 2023, pp. 207–212.

[53] T. Chang, Y.-C. Fan, and A. Chen, "Emotion-cause pair extraction based on machine reading comprehension model," *Multimedia Tools and Applications*, vol. 81, 05 2022.

[54] X. Du, H. Xu, and F. Zhu, "Understanding the effect of hyperparameter optimization on machine learning models for structure design problems," *Computer-Aided Design*, vol. 135, p. 103013, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010448521000245

[55] "Optuna," https://optuna.org/, December 2024.

[56] V. Anand, "FPL Historical Dataset," Retrieved February 2025 from https://github.com/vaastav/Fantasy-Premier-League/, 2025.

[57] "Google colab," https://colab.google/, December 2024.

[58] "Hugging face transformers," https://huggingface.co/docs/transformers/en/index, December 2024.

[59] "Pytorch," https://pytorch.org/, December 2024.

[60] Pranjal, "Light GBM vs XGBOOST," Retrieved February 2025 from https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/#h-lightgbm-vs-xgboost, 2017.

[61] FPL, "Which teams have Double Gameweeks in FPL?" Retrieved April 2025 from https://www.premierleague.com/news/3090425, 2023.