Sentiment Analysis of Twitter to Optimise Fantasy Premier League Performance

By Lewis Watt

Loughborough University
Student ID: F125967
17COC251: 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 [?], and since then growth has been explosive, with the player count surpassing 10 million in the 2022/23 season [?]. 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 [?]. 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 [?], 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* [?], which launched in 2019 and has grown to over 40,000 paying users, with around £2.5m in annual revenue [?]. The platform uses a longitudinal multilevel regression model [?] 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 [?]. The FPL subreddit r/FantasyPL has over 700k members [?], and the official FPL Twitter account has a whopping 6.2m followers [?]. 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 *players* to refer to real-life premier league footballers.

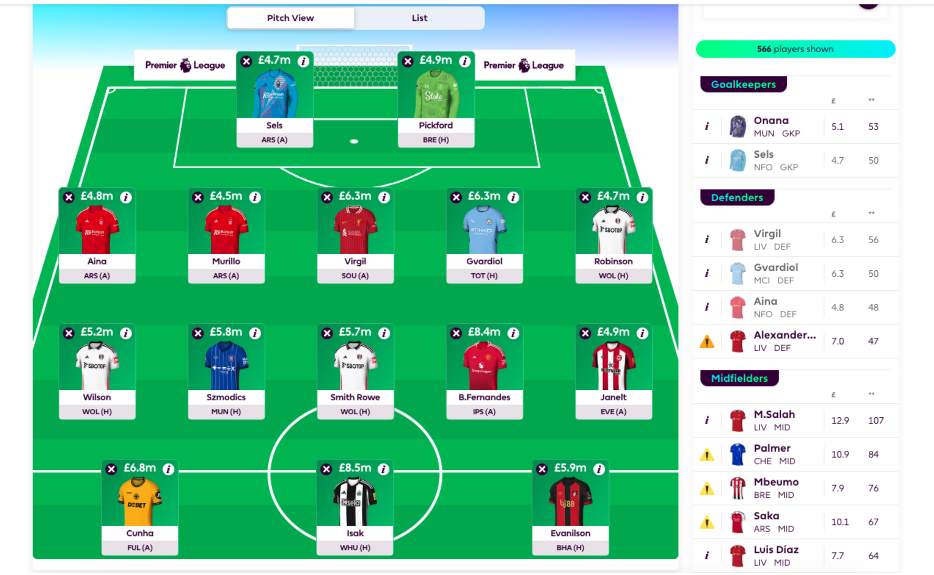
Fantasy Premier League is a mirror of the real-life English Premier League, meaning that each in game event (referred to as a *gameweek*) represents a set of real-life football matches. The game takes place over a *season* which usually runs from August until May of the following calendar year, and at 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



566 players shown

Goalkeepers	£	**
Onana MUN GKP	5.1	53
Sels NFO GKP	4.7	50

Defenders	£	**
Virgil LIV DEF	6.3	56
Gvardiol MCI DEF	6.3	50
Aina NFO DEF	4.8	48
Alexander... LIV DEF	7.0	47

Midfielders	£	**
M.Salah LIV MID	12.9	107
Palmer CHE MID	10.9	84
Mbeumo BRE MID	7.9	76
Saka ARS MID	10.1	67
Luis Díaz LIV MID	7.7	64

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 system that will take twitter data about English Premier League players as an input, and use a fine-tuned sentiment analysis model to analyse the twitter data. The sentiment for each player should be used to create an optimised forecast of a player's expected points (xP) for each *gameweek*. The system will then use a team selection algorithm to pick an optimal squad based on each player's xP and price. The team selection algorithm will also have to abide by the constraints placed on team selection by the official FPL rules.

Users should be able to interact with the system using commands via the console, however the data should be well organised to make it easy to integrate into a web or mobile app with a fully functioning user interface in the future. The system will be modular, meaning the sentiment analysis model, squad selection algorithm, and xP prediction models should be easily upgradable. This should allow the system to keep up with advancements in forecasting technologies that may arise in future.

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 system will be tested against existing research done with no sentiment analysis and the most popular AI FPL platform that charges users a subscription fee to use its services. 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 system will be simulated over a range of gameweeks and its overall score will be compared to see where it ranks in the global leaderboard against other prediction methods.

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 system by simulating 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 system is to achieve 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 mean squared error (MSE) or root mean squared error (RMSE) of models with and without sentiment analysis.
- Assess the impact of the sentiment analysis 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 stay within the top 10% of the global leaderboard during simulations.
- Benchmark the system'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 system 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 study will look at the existing models that have been created and review 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 [**?**]. 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 [**?**]. 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 [**?**]. 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 [**?**]. 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 [**?**]. 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 [**?**].

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 [?]. 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" [?]. This is illustrated in Francis Galton's *Vox Populi* where he describes a country fair in Plymouth in 1906 [?]. 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" [?].

An expansion on this theory is a new technique dubbed "surprisingly popular" [?]. 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% [?]. 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 [?]. 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 [?]. 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 [?]. 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 [?]. A paper by Droba published in the early 20th century outlines methods for collecting public opinion, explaining how early questionnaires were first deployed [?]. 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 [?].

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 [?]. 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 [?]. Shah outlines the different methods that can be used for sentiment analysis, explaining their pros and cons [?]. 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 [?]. The ability of BERT to understand 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 [**?**].

Targeted Aspect-Based Sentiment Analysis (TABSA) aims to determine sentiment toward specific targets, such as individuals or entities [**?**]. 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 [**?**]. 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 [**?**].

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

Methodology

3.1 Overview of the System

The main system is comprised of three components: the sentiment analysis model, the expected points model, and the team selection algorithm. The sentiment analysis model is used to label tweets about FPL players before a gameweek. The number of positive and negative tweets for each player are incorporated into an existing dataset, containing historical data for FPL players. Whilst many expected points models for FPL points predictions have already been developed, none have leveraged twitter sentiment. By adding this feature, it is hoped model performance will increase due to the additional knowledge and crowd wisdom the twitter data contains.

The expected points model outputs a final prediction for each player, representing its guess for how many FPL points that player will score in the upcoming gameweek. A team selection algorithm will take this information and identify the squad expected to maximise the number of points scored, whilst satisfying all budgetary, positional, and team constraints.

The sentiment analysis model is trained using a publicly available dataset of Premier League related tweets from the 2022/23 season. A fine-tuned BERT model is used to classify the data and then sentiment scores are aggregated to produce a final sentiment value for each gameweek. Respective historical data from the same time period is then taken from a community-run Github repository, and combined with the sentiment values for each player. A gradient boosting machine (GBM) is trained on the enhanced dataset, producing a final expected points value for each player.

Finally, a team selection algorithm takes the expected points values, and selects an optimal squad of 15 players that satisfies the FPL team constraints. This problem is formulated as an optimisation problem with the objective being to maximise total expected points within a given budget. Integer linear programming is used to find the best possible team composition.

Together these components form an intelligent support system for FPL managers that incorporates crowd wisdom as well as historical data. The final system should help managers to make better and more informed decisions that lead to an increase in their overall rank.

3.2 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 [**?**] 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 [**?**, **?**].

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 [**?**]. 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.

3.2.1 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 [**?**], 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 [**?**]. 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.

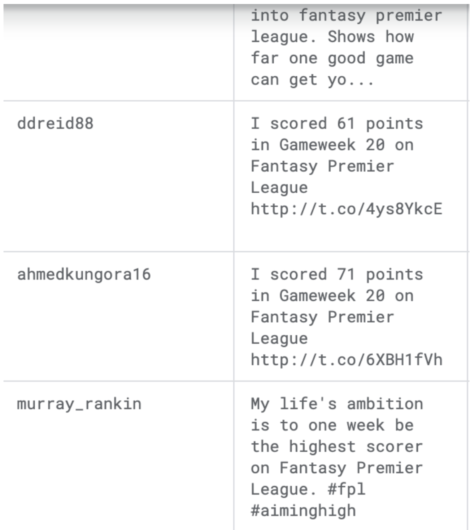
The chosen approach was to use an existing dataset published on popular machine learning forums such as Hugging Face and Kaggle. These sites have thousands of active users contributing to a wide range of research topics. As such, finding a dataset for a similar purpose to this study's was not difficult.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



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

The sentiment labels provided in the dataset were automatically created by the Vader [**?**], and TextBlob [**?**] 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.

3.2.2 Data Pre-Processing

The dataset was cut down to a more suitable size of around 20k tweets to keep training time reasonable (roughly 30 mins per epoch). This was done using a random sampling approach to reduce biases. 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. [**?**] 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.

3.2.3 Fine-Tuning BERT

To fine-tune a BERT model for sentiment analysis, the pre-trained BERT model was modified by adding a task-specific classification layer on top of the BERT encoder stack [**?**]. 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 [?] classifier, which takes as input a vector **h** and uses the following equation to predict the final class:

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

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.

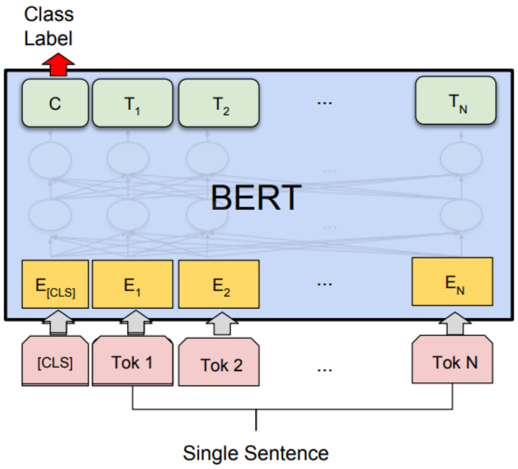


Figure 3.2: Single sentence classification [?]

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.

While standard sentiment analysis provides a single sentiment label per text, Targeted Aspect-Based Sentiment Analysis (TABSA) enables sentiment classification for specific entities within a sentence, such as individual players. The auxiliary sentence method introduced by Sun et al. [?] was used. BERT processes 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. This approach enables BERT to focus on the sentiment directed toward each specific player, rather than providing a single sentiment value for the entire sentence. A single value could be uninformative, as positive and negative sentiments might cancel each other out. By targeting individual players, this method ensures more accurate predictions, with players labeled according to their specific sentiment.

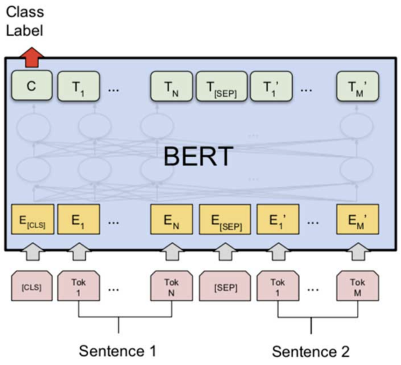


Figure 3.3: Sentence-pair classification [**?**]

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. 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.

3.2.4 Hyper-parameter Optimisation

In order to maximise the performance of the final model, hyper-parameters needed to be 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 [?]. 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 [?], 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 [?]. 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.

3.2.5 Model Evaluation

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:

- **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 in the overall FPL selection system. The model weights were saved using the transformers library and uploaded to Hugging Face. This allowed the trained model to be easily imported into other components of the system, facilitating further use and integration with the system's expected points model pipeline.

3.3 Expected Points Model

In order for the system to identify the best team of players ahead of an FPL gameweek, a prediction model is needed that can accurately forecast the number of points each player is going to score. 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. [**?**] 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 on by traditional news outlets.

3.3.1 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 [**?**]. 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 [**?**], 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

Once the base dataset had been constructed, sentiment information about each player was added into the dataset. To achieve this, unseen Twitter data was run through the sentiment analysis model to get a sentiment value for each row in the dataset. Finally, each tweet was 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, each containing the attributes 'player_name', 'gameweek', and 'sentiment'. These tweets were matched to their corresponding player and gameweek from the historical information dataset. For each match, two new columns, 'total_positive_tweets' and 'total_negative_tweets', were added to the dataset. The matching tweets for each player and gameweek were then counted, with the final counts of positive and negative tweets appended to the main dataset.

Finally the complete dataset was uploaded to a Hugging Face repository so it could be used by the GBM model for training, as well as later use for the overall evaluation of the complete system.

3.3.2 Data Pre-Processing

To pre-process the dataset before training the model, a number of steps were taken.

Firstly, the only text value feature in the dataset, the team name, was encoded using one-hot encoding. This is where a new column is introduced for every possible value the feature could take, so new columns for all 20 Premier League teams were introduced. Then if for example, the original row in the dataset had a team value of 'Manchester United', the 'manchester_united' column would have a value of 1, and all other team columns would have a value of 0.

Next, 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. Finally the dataset was split into train and validation sets, with around 5k rows being set aside for the final evaluation of the complete system.

3.3.3 Model Training and Evaluation

After the dataset was fully pre-processed and prepared, the Gradient Boosting Machine (GBM) model was trained to predict expected points for each player. The train and test datasets were checked 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 5.

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. After this model was observed to be behaving correctly, the hyper-parameters were tweaked to maximise performance.

To find the optimal hyper-parameters for the GBM model, Optuna [**?**] 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 point 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. During training, early stopping was implemented to prevent overfitting. If the validation loss did not improve for 10 consecutive iterations, training was halted. This sped up the training process and reduced computational demand, as each trial involved significantly less iterations.

Once training was complete, the best-performing set of hyper-parameters was selected based on validation set performance and was subsequently used to train a final model, which was implemented into the FPL team selection system. To ensure the inclusion of sentiment data lead to a significant performance increase, a further model was trained using the same set of hyper-parameters, however the data used to train it did not contain any sentiment information ('positive_tweets' and 'negative_tweets' were removed from the dataset).

Additionally, feature importance scores were analysed post-training to further ensure that sentiment-based features contributed meaningfully to the model's decision-making process.

3.4 Optimal Team Selection Algorithm

3.4.1 Constraints

3.4.2 Algorithm

3.5 Summary

Chapter 4

Sentiment Analysis Model

4.1 Theory Behind the BERT Model

Bidirectional Encoder Representation from Transformers or BERT [?], is a pre-trained language representation model based on the transformer architecture proposed in the now infamous 2017 paper ”Attention is All You Need” [?]. 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 [?].

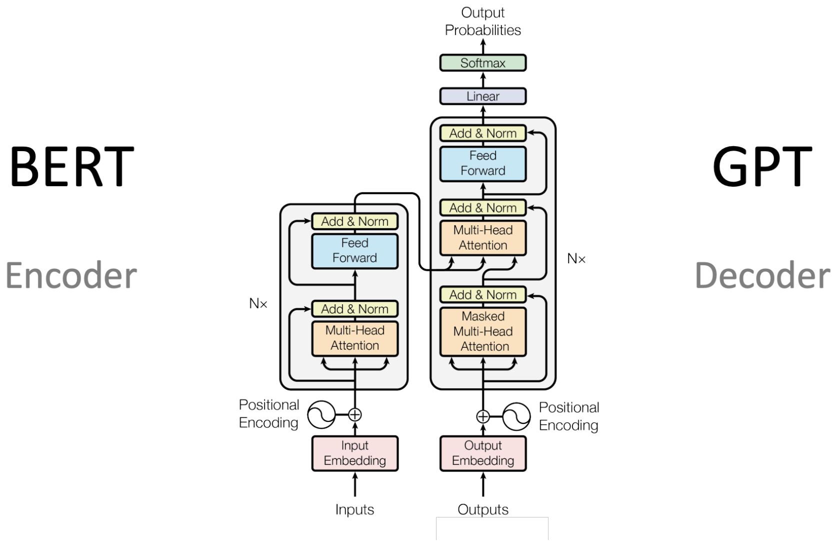


Figure 4.1: Architecture of a Transformer [?]

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) [?]. 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 [?] and BERT [?]. 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 [?].

In the BERT model, raw text is broken down into tokens using the WordPiece tokenisation algorithm [?]. 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 [?].

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 [?].

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 [?]. The SEP token is used to separate two sentences for tasks such as question-answering and dual-sentence classification [?].

Input Embeddings

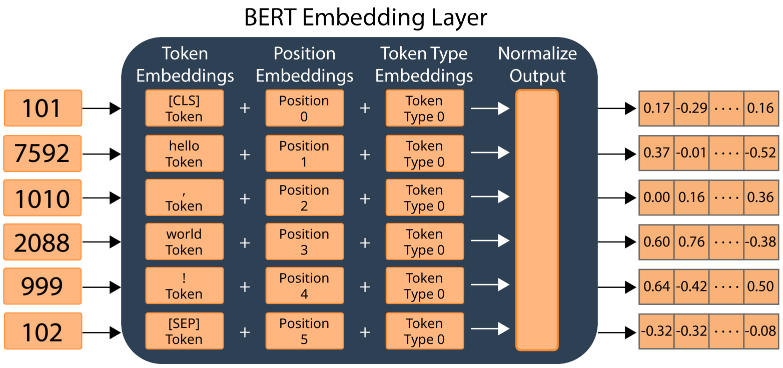


Figure 4.2: BERT Embedding Layer [?]

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 [?]. 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 [?].

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 [?].

- **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.
- **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' [**?**]. 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 [**?**]. 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).

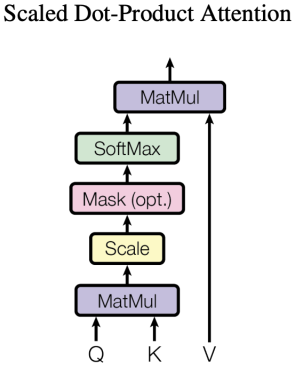


Figure 4.3: Scaled Dot Product Attention [**?**]

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

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 [?] 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 [?]), 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:

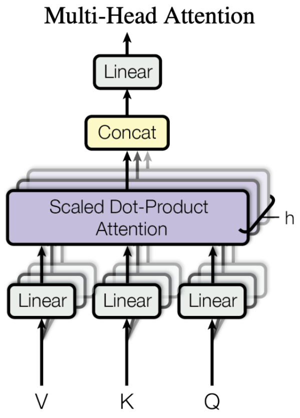


Figure 4.4: Multi-Head Attention [?]

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

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 [?].

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 [?].

The FFNN processes input embeddings by first projecting them into a higher-dimensional space. Specifically, the 768-dimensional input for each BERT token embedding [?] 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 [?].

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 $\text{BERT}_{\text{base}}$ which this study uses) [?]. 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 [?], 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 [?].

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) [?].

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 [?], 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 [?].

By leveraging these tasks across millions of sentences and performing millions of optimisation steps, BERT learns complex relationships between words and sentences [?]. The model's weights, including those for input embeddings and self-attention mechanisms, are updated via backpropagation [?] during this process.

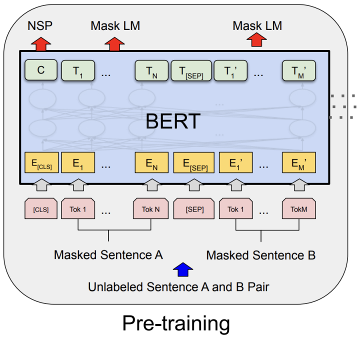


Figure 4.5: BERT Pre-Training [?]

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 [?].

4.2 Model Implementation

The BERT model used in this study to predict sentiment for FPL players was fine-tuned in Google Colab [?] 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 notebook format also allows for simple and clear separation of functions, allowing for single blocks of code to be independently executed. This allows for easy debugging and troubleshooting of code, improving the development experience. A Jupyter notebook was set up to create a pipeline for pre-processing data, training the model, optimising hyperparameters, and evaluation of the model. This format allows for easy and quick reproduction, as it includes markup to explain steps in more detail.

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 [?] which contains APIs for hundreds of pre-trained transformer models including BERT. The transformers library also includes tokenisers for each model and a PyTorch [?] 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 [?] 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.

4.2.1 Data Pre-Processing

The first step of the pipeline was to process the dataset, cleaning it of any noise and transforming it to include an auxiliary sentence suited to the TABSA approach chosen. To begin with any image links were identified and removed by searched for occurrences of 'a href' in the texts. Empty rows and duplicates were stripped out, and finally regular expressions were used to check if the player label for each tweet (who the tweet is supposedly mentioning) was actually mentioned in the text. For example, some entries might have a player label of 'Jordan Pickford' but the tweet makes no mention of him. This ensures the model can always see the player it is supposed to be predicting a sentiment value for in the input text.

The auxiliary sentence was then created for each entry in the dataset 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 data was converted into a format the model could understand. For the 'question' and 'context' columns this meant tokenisation. To do this the AutoTokenizer class from transformers 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.

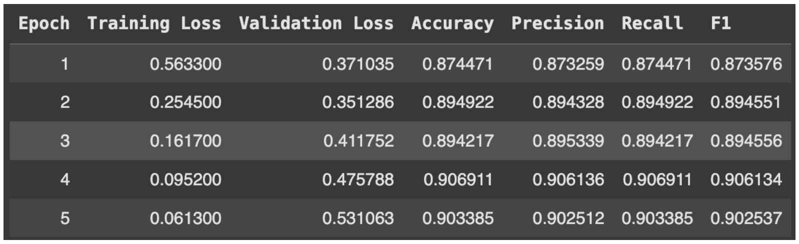
Once this was done the dataset was split into train and validation sets using the train_test_split method from scikitlearn. The datasets were then ready to be used for training of the model.

4.2.2 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 to optimise them properly later (see 4.2.4).

After these arguments were configured, the next step was to set up the Trainer object. The pre-trained `BertForSequenceClassification` model was loaded in from the transformers library. This model is configured for classification tasks on 2 input sentences with a suitable classification head already implemented, exactly what is needed for target-based sentiment analysis with an auxiliary sentence. The specific model used was 'bert-base-cased' which is just the base variant of the BERT model that is case-sensitive. The Trainer was initialised by passing in the model, datasets and training arguments. Once the Trainer had been initialised, the training process was started by calling the `train()` method.

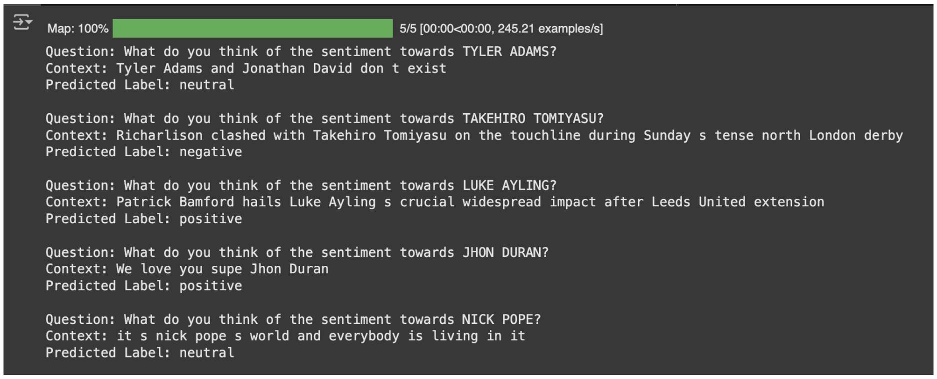
Training took around 90 minutes, and the resulting model scored an initial accuracy of around 90.34%. Other performance metrics such as precision and F1 score were configured and will be discussed further during final model evaluation. Focusing on the validation loss, it is clear that the initial model is starting to overfit the training data as the epochs go on, evidenced by the increasing values after epoch 2. This is something that will be refined using hyper-parameter tuning.



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 4.6: Performance metrics for the initial model

Once trained, the model was tested to ensure the it worked 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 the data used for model inference was pre-processed in the exact same way as the training data. This is crucial for achieving an accurate prediction from the model by ensuring the model recognises the inputs and can leverage knowledge learned during training to the fullest extent. Inference was done using the built in `eval()` method that the transformers library provides for every model.



```
Map: 100%  5/5 [00:00<00:00, 245.21 examples/s]
Question: What do you think of the sentiment towards TYLER ADAMS?
Context: Tyler Adams and Jonathan David don t exist
Predicted Label: neutral

Question: What do you think of the sentiment towards TAKEHIRO TOMIYASU?
Context: Richarlison clashed with Takehiro Tomiyasu on the touchline during Sunday s tense north London derby
Predicted Label: negative

Question: What do you think of the sentiment towards LUKE AYLING?
Context: Patrick Bamford hails Luke Ayling s crucial widespread impact after Leeds United extension
Predicted Label: positive

Question: What do you think of the sentiment towards JHON DURAN?
Context: We love you supe Jhon Duran
Predicted Label: positive

Question: What do you think of the sentiment towards NICK POPE?
Context: it s nick pope s world and everybody is living in it
Predicted Label: neutral
```

Figure 4.7: The model's predictions on unseen data

Model outputs appear to be mostly correct and are easily retrieved through the built-in methods provided by the transformers library. This allows for a quick and efficient way to generate a list of sentiment predictions for a large set of tweets, necessary for the final system.

4.2.3 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. It defines the search space and hyper-parameters available, and then feeds Optuna's suggestions into each model's training arguments. It also specifies which metric to use to evaluate this models - validation loss in this case. The validation loss is a measure of how far away the model's predictions are from the labelled values. 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.

An Optuna study was created using the *create_study* method, and then run for 10 trials. This amount should allow Optuna to explore a sufficient amount of configurations while keeping resource usage to a minimum. 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 4.1: Hyper-parameter configurations with associated validation loss

4.2.4 Model Evaluation

Whilst fully training the three best configurations, it was initially discovered that these configurations 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 great, so this approach is chosen as a compromise.

The configurations evaluated were the best 3 from hyper-parameter optimisation (see 4.2.4): Trial 5 (Configuration 1), Trial 1 (Configuration 2), and Trial 2 (Configuration 3). The results are as follows:

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 4.2: Classification report for Model 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 4.3: Classification report for Model 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 4.4: Classification report for Model Configuration 3.

As evident from the results, the models struggled with the negative class in all configurations. This was expected due to the class imbalance present in the dataset, which contains far fewer ”negative” samples compared to the ”neutral” and ”positive” samples. The model tends to mislabel ”negative” samples as ”neutral” due to the higher number of ”neutral” samples during training. This is reflected in the lower precision and recall for the negative class across all configurations.

The neutral class has high recall (close to 1.00) but suffers from lower precision in some configurations, particularly in Configuration 2, where the model incorrectly classifies a significant portion of the negative samples as neutral. The positive class also shows a reasonably balanced performance, but it is clear that the class imbalance impacts the model's performance, as seen in the discrepancies between recall and precision for the ”negative” and ”neutral” classes.

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. The F1-score of Configuration 1 was significantly higher than the other configurations, making it the most reliable model for classifying positive, negative, and neutral sentiments.

As such, the model trained using Configuration 1 was saved and uploaded to a newly created Hugging Face repository using the API provided. From there it was easily accessible to all other parts of the system.

Chapter 5

Expected Points Model

5.1 Gradient Boosting and Decision Trees

5.2 Model Implementation

The GBM model was implemented using the LightGBM library [?] 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 hyperparameters 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 [?].

Two more Google Colab notebooks were created, one to facilitate the creation of the dataset that contains data from the sentiment analysis model, 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 needed to be 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 for each gameweek of the 2022/23 season via GET request, and saving them. Once saved, irrelevant features were dropped, and a Pandas dataframe was created with the concatenation of all gameweek files.

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, and gameweek (there are no players with the same name on the same team, so this provides a unique record) and then `.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 dataset was examined to ensure features had been correctly calculated.

The next step was to add in the sentiment data that is so important to this study. 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 tweets for every player and every gameweek, labelled as 'positive', 'neutral', or 'negative'. To do this each tweet from the dataset was tagged with the gameweek it belonged to, using the tweet timestamp. The data was pre-processed using the same steps as model training, and then model was loaded using the Hugging Face API so the tweets could be run through it to get a sentiment prediction for each tweet.

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 this, it became clear there were some player names that contained typos and 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. This meant small typos were not an issue and 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 could be used for training, sklearn and Pandas were used. Pandas was used to remove the identifying features: name and gameweek, from the actual inputs, 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.

Inputs were scaled using the `MinMaxScaler` from sklearn which is a simple way of normalising the input values between 0 and 1. A different scaler was used for inputs and the target variable 'total_points' so that the predictions could be unscaled at the end. Additionally, the team name was encoded using Pandas `.get_dummies` method which simply adds a new column for every possible value of team name (one hot encoding). This meant 20 additional columns were added and the specific team column for each row was given a value of 1, with all others having a value of 0.

After this was done duplicates were dropped and the dataset was examined to ensure the correct columns existed, with appropriate values. Finally the dataset was split into test, train and final evaluation datasets. The purpose of the final evaluation dataset was to keep back data, so the entire system 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.

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, which is perfect for the needs of this study. A model object of the class was instantiated 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. Initial training was promising, with a Root Mean Squared Error (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-parameters to be explored were 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

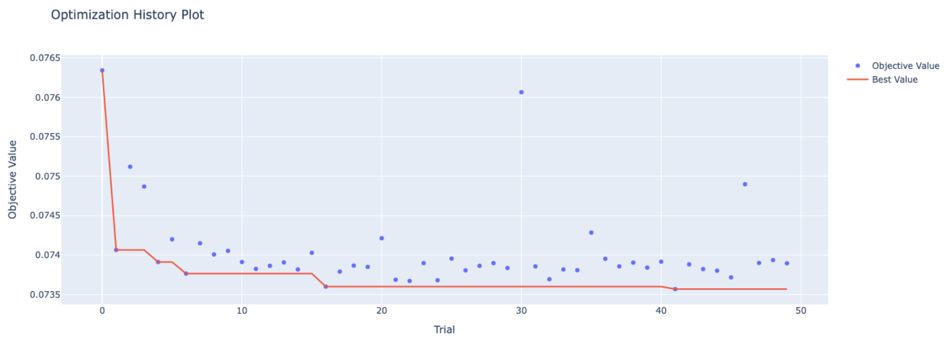


Figure 5.1: RMSE with each trial (hyper-parameter configuration)

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

The entire purpose of this study is to compare how models with sentiment information captured via twitter perform against models without this information. As such it made sense to train another model using the exact same dataset, minus the 'positive_tweets' and 'negative_tweets' features.

This way a valid comparison could be made between the two models, and any notable differences could be observed. The first comparison was the RMSE of validation data between the two models. The model trained with the sentiment data produced a RMSE of 0.07208, whilst the model trained without sentiment data produced 0.07221. This small difference favours the model trained with sentiment data, however it is hard to draw conclusions from such a small change.

To get a better understanding, matplotlib was used to visualise feature importance and the distribution of the residuals (real value - prediction) for each model.

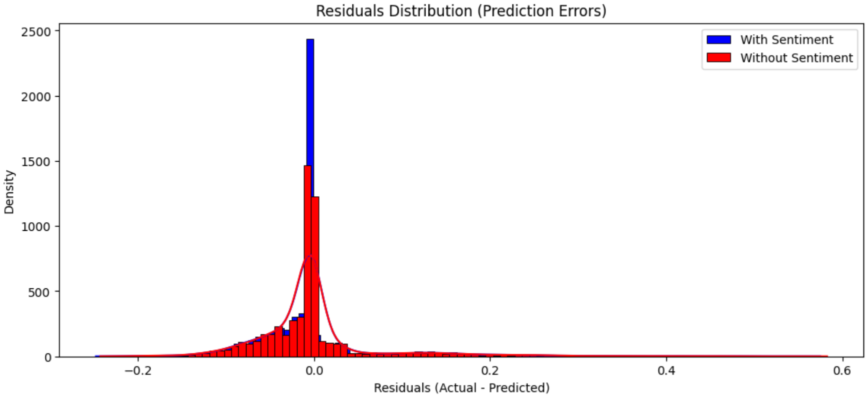


Figure 5.2: How residuals are distributed for each model.

From the residual distribution plot, it is clear the model with sentiment has a lot more predictions that are exactly correct or only slightly off (as indicated by the spike at 0.0). In contrast, the model without sentiment data has slightly more spread out predictions which indicate its predictions tend to deviate further from the true values.

This suggests that the incorporation of sentiment data helps the model to gather more information and make more accurate predictions, reducing any large errors. It seems that positive sentiment can act as a proxy for additional crowd wisdom. It is also worth noting the most important feature 'transfers_in_last_week' (how many people bought that player) is another metric of capturing crowd wisdom, so it is clearly important to predicting overall performance.

The wider spread from the model without sentiment data suggests it is less confident due to the lack of additional knowledge not captured by historical data.

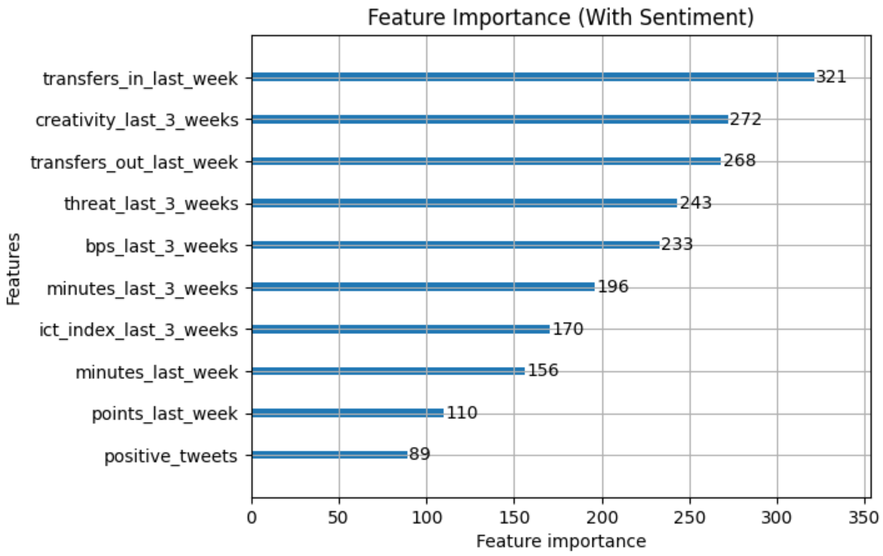


Figure 5.3: Top 10 features by importance

The feature importance analysis shows that 'positive_tweets' ranks 10th, just behind 'points_last_week'. This suggests that positive sentiment contributes meaningfully to model performance. Additionally, the ranking implies that positive sentiment has a stronger correlation with player performance than negative sentiment.

Chapter 6

Team Selection Algorithm

Chapter 7

Results and Discussion

Chapter 8

Conclusion and Future Work