Real-time Social Media Content Recommendation for Live Sports Events

by

Renbin Liu

B.S. Computer Science and Engineering, Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
April 28, 2022

Certified by .....
Tomás Palacios
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by .....
Feifei Peng
Senior Data Scientist at Sky, 6-A Thesis Supervisor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Real-time Social Media Content Recommendation for Live Sports Events

by

Renbin Liu

Submitted to the Department of Electrical Engineering and Computer Science
on April 28, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The presence of social media is getting greater in the sports arena. Many people who watch live sports games also follow social media platforms for live coverage and commentaries. Although these additional content can enrich the watching experiences of the audience, they may become distractions to the audience from some key events in a live sports game. In this thesis, we propose a system that will automatically present relevant and engaging social media content for a live game. We will employ techniques in Natural Language Processing to filter social media posts to select the best ones for users to follow while watching the game. With an engagement prediction model augmented with other metadata of the post and of its author, the audience can enjoy the game without missing out on important game coverage and reactions on social media.

Thesis Supervisor: Tomás Palacios
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Feifei Peng
Title: Senior Data Scientist at Sky, 6-A Thesis Supervisor

Acknowledgments

I would like to thank Jian Li and Feifei Peng for their mentorship, feedback, and guidance throughout my thesis project. Both of them provided extraordinary insight and practical advice on my thesis. The results presented here would not be possible without their efforts. Additionally, they were instrumental in ensuring that I had a great experience while interning at Sky.

I would also like to thank Prof. Tomás Palacios for agreeing to supervise my thesis. Finding a thesis advisor at MIT turned out to be much more difficult than my anticipation. He provided me with great advice and future directions for my project.

I want to recognize all colleagues of whom I have collaborated with at Sky, including engineers who offered incredible advice and resources on the deployment aspect of my project, as well as product managers for their valuable insights and advice. This thesis project would not have been possible without their support.

I am very grateful to the 6-A program for providing me with this opportunity to participate in this internship program. During my early years at MIT, I knew from my heart that this program would be a great fit for me. I would like to thank Kathy and Myung-Hee for their great support on the logistic side of this program. Their support made my dream a reality during my final year at MIT.

Relocating to London for the program is quite an experience, and I want to pay tribute to everyone who has made my time in London memorable. I would like to thank:

- Dana, Gordon, and Rajan, of One Lampton Road, to make my stay there a fantastic experience.

- Everyone in the Lubiao Christian Student Fellowship for providing me with a spiritual home. I have made many good friends there, and I look forward to seeing them again when I visit London in the future.

Finally, I would like to thank all of my family members and friends who supported me during my journey at MIT. Studying at one of the greatest academic institution

in the world is certainly challenging and rewarding. They are the ones who lifted me up during the most difficult times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The COVID-19 pandemic gives rise to virtual substitutes for in-person activities, including live sports events. Enhancing user experience during these events is important to keep the users engaged. No viewers would find watching a soccer game on their own a satisfactory substitute for in-person alternatives. There are many avenues for solving this problem. For example, one can set up poll questions for users to quiz their knowledge. In this project, we will investigate the direction of incorporating social media content during those live events.

Besides watching the live game, social media platforms are useful places to view live coverage and other important updates. Many viewers will share their opinions as these live events go on. Seeing relatable content can connect viewers with one another and create a sense of community among them. However, navigating between watching the game and using social media applications can be rather distracting and may cause users to miss important game events. Adding selected content that interests users will avoid this problem and thus enhance their overall experiences.

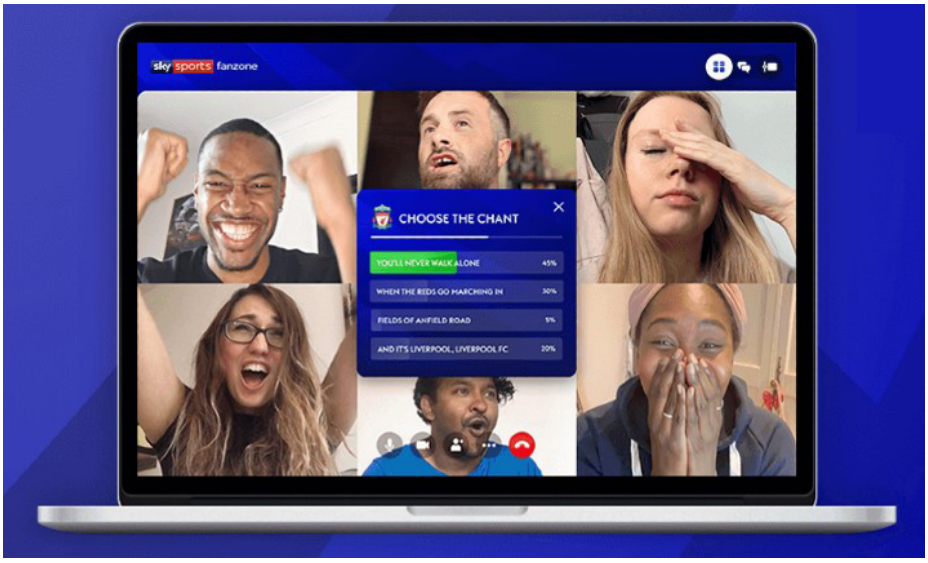


Figure 1-1: User interface of Sky Sports Fanzone, an application providing interactive watch-together experiences for viewers.

1.2 Current Experience with Live Sports

There are many ongoing efforts to enhance interactive experiences during live sports events. We will focus our discussions on three specific components: watch-together experiences, live news feeds, and social media content.

The watch-together experience offers an alternative to in-person watching parties, which can be popular when COVID-related restrictions are in effect. The essence of this experience is to watch a live sports game together through a video conference software, as demonstrated in Figure 1-1. However, developments of additional features can enhance this virtual activity. For example, viewers can put on some fan gears as virtual filters to show support for specific teams. They may also participate in pregame quizzes to demonstrate their knowledge about players and teams involved in a game. These new features make up for the lack of face-to-face experiences and can build connections among users.

A live news feed for a sports event is the go-to place to see the perspective of sports commentators. Figure 1-2 shows examples of such highlights. The vast majority of them are descriptions of some important moments during the game. They may also contain some news reports to set up the context for the game. Although

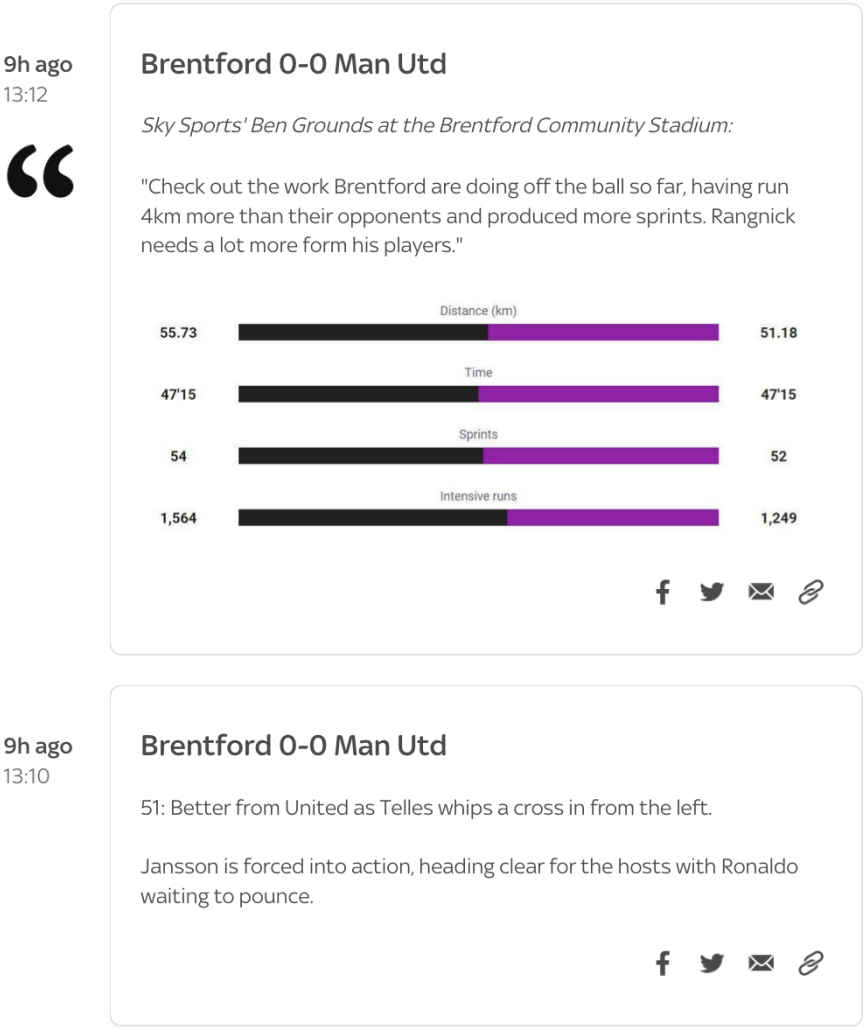


Figure 1-2: Some example posts from the live news feed of the Brentford v. Manchester United game on January 19, 2022.

Figure 1-3: An example tweet showing a satire on the performance of the Arsenal team.

the feature may help users to get a new perspective, most of these descriptions are things that viewers probably can see for themselves. This feature has the risk of not adding anything substantial to user experiences. Additionally, this feature can become a distraction for viewers who are watching the live video stream of the game simultaneously.

Content from social media platforms can perhaps better complement the experience of watching a live game. It is no surprise that viewers check their social media accounts for updates frequently. Specifically, they look for updates, as well as interesting opinions and commentaries. Figure 1-3 shows an example of such a tweet. It attracted many engagements for its satire on Arsenal's performance during a game in August 2021.

This observation inspires us to improve the experience of these viewers with social media content during a live sports game. Most users don't have access to a feed that contains relevant content for a specific game exclusively. Take Twitter as an example. Almost all Twitter users will have other irrelevant content in their feed, making it difficult for them to locate content specifically related to the game they are watching. Furthermore, checking social media applications can be distracting for viewers, causing them to miss important moments during a game. With those considerations in mind, building a system to deliver a better companion social media feed for a live sports game can improve their experiences watching sports coverage. This system can also help them to feel connected with a community of sports fans who are paying attention to the same game.

1.3 Main Contribution

1.3.1 Problem Statement

The main problem is to build a system that delivers relevant and interesting social media content to a user watching live game coverage. We narrow the domain of the problem to only include English Premier League games. We also choose to focus on social media data on Twitter. Given a set of n tweets provided by our clients, we would like to find top k tweets that are most suited for a Premier League game, or for a specific moment within such a game, where $k \leq n$. We provide further definitions of a "suited" tweet:

- A suited tweet is **relevant**. We want to ensure that any tweets that are shown to the users must be relevant. For instance, users looking for tweets about a foul probably will not be pleased to see tweets about a goal that happened 20 minutes ago. The system doesn't need to rank the outputs by the extent of relevance, but it must make sure that any tweets returned are related to the game or a game moment that the client is looking for.

- A suited tweet is **popular**. A popular tweet gets lots of attention on Twitter,

and the attention provides a strong signal that such a tweet contains good content for our users to see. An example may be a tweet written by a player thanking the supporters for their extraordinary achievements during the game. Users don't want to miss out on those trends on social media platforms while watching live coverage.

- A suited tweet is **sensational**. Lots of sports reporting focus on facts and figures about players and teams. These numbers and statistics do not include the emotional aspects of a game. A committed foul can make some viewers angry, while a missed goal will induce much disappointment. We can find lots of emotions related to those Premier League games on social media platforms like Twitter. Having sensational content will be an excellent complement.

1.3.2 List of Contributions

The main achievements of the thesis project are listed below:

- Created a basic quality control filter to remove incomplete tweets or those in foreign languages.
- Created a relevance filter using keyword matching to remove any irrelevant tweets provided by the clients.
- Implemented a sentiment analysis system via VaderSentiment[11] to choose tweets with specific types of sentiment.
- Designed and trained a content moderation model with TF-IDF feature extraction algorithm to remove offensive and inappropriate content from the user feed.
- Designed and trained an engagement prediction model using a multimodal Transformer-based algorithm[10] to predict the number of likes that a tweet is expected to receive.
- Deployed the content moderation model and integrated it with our system.

- Deployed the engagement prediction model as an API endpoint using Amazon SageMaker and integrated it with our system.

- Implemented a Kedro training pipeline[2] for the engagement prediction model to automate the model training process.

1.4 Related Work and Literature Review

The success of this thesis project relies heavily on two aspects. The first one is a solid framework for the engagement prediction task to achieve our main objective. Additionally, the engagement prediction task itself requires a set of appropriate Natural Language Processing(NLP) techniques to maximize its performance. NLP also plays a critical role in other aspects of this project, including sentiment analysis and content moderation.

1.4.1 Engagement Prediction

There has not been too much work on engagement prediction overall. Instead, a lot of the work has been focused on predicting individual engagement tendency on a piece of content, rather than the overall engagement level. The RecSys challenge is a great example of that. It featured user behavior prediction on Twitter content [4]. The problem setup focuses on a specific target user and a tweet. The question they are trying to address is: How likely is it for the target user to engage with this tweet? On the other hand, we are trying to predict the overall level of engagement of a tweet, and we are not concerned about the behaviors of each individual user. Nevertheless, the methods used in these top submissions can inspire us on choices of features and model architectures.

Many of the top submissions [15] [9] [8] [6] to this challenge involve BERT and its variant to obtain a text embedding of the content. Some of them decided to train the model with the original language modeling objective to get the model familiarized with the language of Twitter. Other features from those papers include past behaviors

of tweet authors and of the target users. Some of these features can be useful for our thesis project. In fact, many of the features listed in Chapter 3.2 are motivated by these papers. There is also a variety of choices in terms of models. One of them decided on a full deep learning approach [15] while others tried ensemble models involving gradient boosting trees [9] [8]. Our engagement prediction model adopts a full deep learning approach, as we have discovered methods to combine different forms of features without using a boosting tree.

We will now closely analyze the system presented in [15], as it is the most similar to the main component of our system. It utilizes features about the content creator and the target user. These features are directly fed into the feedforward layer in their model. The system creates a history embedding that represents all the tweets of which the target user interacted with in the past. This embedding can give us substantive information about the target user, including their past behaviors and their interests. This history embedding is then combined with the embedding of the tweet content through an attention layer.

The system we propose in this thesis serves a different purpose. For example, we don't have access to past user behaviors and cannot make predictions on individual level. However, many design ideas in this thesis are inspired by the system presented in [15], including the use of features about the tweet creator and an attention layer for combining different types of features.

1.4.2 NLP Techniques

A performant engagement prediction model relies on state-of-the-art NLP techniques. In the RecSys challenge, the contestants were provided with the BERT embeddings of the tokens in the tweet. Unsurprisingly, many of those papers above used Transformer-based models to obtain a vector representation of the tweet text, which proved to be helpful in the task.

In recent years, there has been a significant breakthrough in NLP with the advent of the Transformer-based models. BERT[7] is the prime example of that. The success of these models heavily depend on the success of attention modules[14]. The most

significant contribution of those models is that they can be used for all types of downstream classification and regression tasks with a customized dataset.

Although BERT[7] is an excellent choice for a NLP model, it comes at the cost of longer training time and higher demand for computational resources. In some cases, we would like the performance of Transformer-based models like BERT, but we also need more efficient training and inferences. As a result, further optimizations on Transformer-based models are discovered, including distilBERT[13]. Our thesis project will use this model for extracting embeddings from textual data in the engagement prediction model.

1.5 Thesis Organization

The thesis starts with an overview of the system. It then covers the engagement prediction task in detail. Afterwards, experimental content and engineering work will follow.

Chapter 2 gives an overview of our proposed system. It first enumerates our design values. The chapter then describes each component of our proposed system and specifies how the design choices with those components meet with our design values.

Chapter 3 discusses the dataset we have built for the engagement prediction model. It starts by describing the process of data collection. We then show some features of the dataset that may be helpful for our task. These features are validated by our exploratory data analysis that follows. Finally, we move on to considerations and implementations for feature engineering.

Chapter 4 tells the story about the evolution of our engagement prediction model. It covers a range of models, including a ridge regression model, a Transformer-based model involving textual features, and other models that incorporate textual and tabular data.

Chapter 5 is about the experiment setups and results of machine learning models in our system. For both content moderation and engagement prediction, we describe

the experiment setup to train the model and show the results of those experiments. More discussions on the experimental results follow afterwards.

Chapter 6 focuses on the engineering work for this thesis project. It discusses the deployment strategies for both of our machine learning models. We also discuss a training pipeline for our engagement prediction task.

Chapter 2

System Overview

In this chapter we propose a system that will provide tweets with the highest engagement potential given a Premier League game or event. We will provide an overview of this system in this chapter and discuss each component briefly in each section. Figure 2-1 shows the architecture diagram of our system.

2.1 Design Values

Before we discuss each component of our system, we would like to discuss some important design requirements.

- The first obvious requirement of our system is being **accurate**. To be more concrete, we would like to return tweets that are relevant and are with high engagement potential. Or it needs to be sufficiently sensational. This relies on our system to accurately identify this information. If we cannot achieve this objective, users may be frustrated and become disinterested in this feature.

- The other major requirement is the system's **efficiency**. Since we expect the input tweets are often real-time. Typically, those are posted only for up to a few minutes. It is essential for our system to perform analysis efficiently so that the data we provide to our clients are still relevant and up-to-date. Clients won't

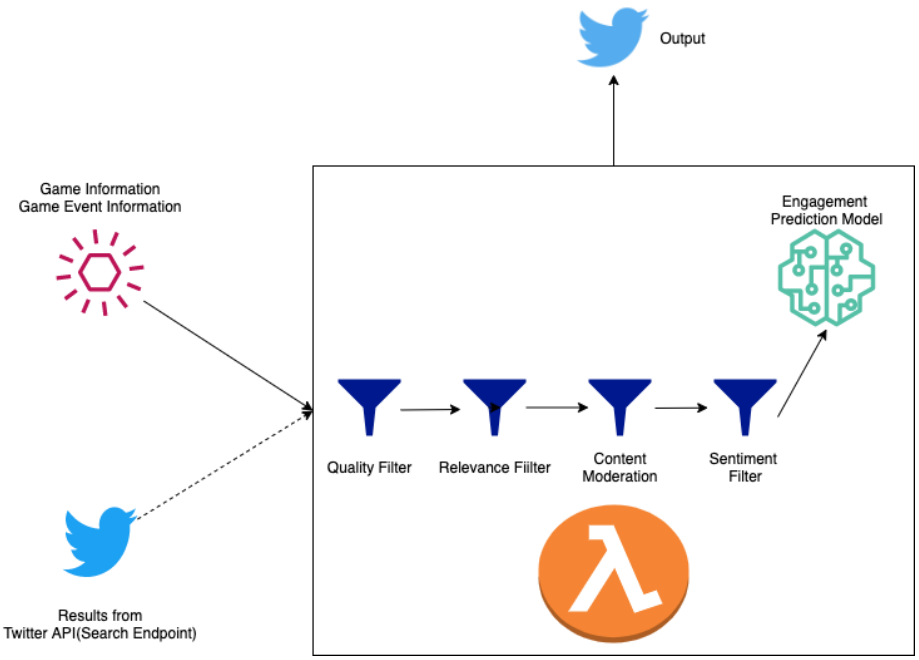


Figure 2-1: The architecture diagram of the system. Given game or event information, the input will pass through several filters sequentially. Afterward, the remaining data will be fed into an engagement prediction model to predict their popularity scores. The system will then return tweets sorted by predicted popularity scores in descending order.

like it if our analysis takes minutes to complete. By then, a lot of the tweets provided by clients will become out-of-date in the views of our users.

2.2 Quality and Relevance Filter

The first component is the quality and relevance filter. The main purpose of this component is to filter out any tweets that may be irrelevant to the game that the client is interested in, or any tweets that are truncated or not in English. For the quality filter, there are specific attributes in the tweet objects that can be checked. The relevance filter uses a simple rule-based approach by checking for presence of at least one team name in the tweet content. We acknowledge that this may not be the best method as it can potentially miss out on relevant tweets that do not include any team names. It may also falsely introduce tweets with a team name with other connotations. A tweet containing the word "Chelsea" may not necessarily be relevant to the Chelsea FC team. Instead, it may be about Chelsea as a neighborhood in West London.

Although the rule-based term matching heuristic has its faults, this may be the best that we can do. One possible alternative is to use machine learning algorithms to classify relevance, but we do not have any labeled data for doing this. In addition, formulating this problem as an unsupervised learning problem is also challenging and perhaps unnecessary. The specific way of which we will approach this unsupervised learning problem is to encode our tweet text into an embedding. We need to do the same to obtain the text embedding of a game event or of a Premier League game. This can be difficult because encoding text such as "Arsenal v. Newcastle" or "Goal by Player X" is not informative enough to produce a useful embedding. Without a high quality game(or event) embedding, it is likely that our relevance filter will end up looking for keywords in the tweet body, and that is very similar to our current approach.

Since this component of our system does not involve any heavy computations, it is the best idea to put this component as the first filter. The other components

in the system use machine learning algorithms and may become the bottlenecks of our system in terms of its efficiency. By removing irrelevant tweets or those with poor quality, we can minimize the workload of our machine learning algorithms and optimize the performance of our system.

2.3 Content Moderation

Although our system does not create any new social media content from scratch, considering the consequences of the content that we produce from this system is indispensable. Twitter can be a great tool to connect people, but any careless misuses can have negative consequences. For example, our system may accidentally become a spreader of misinformation. It is also possible for our system to output offensive and inappropriate content that will turn users away. Wrong content can have huge negative consequences for user experiences and any applications that use this system.

Having a content moderation model in our system is a responsible thing to do. It is beneficial in two ways. First, the content moderation model will filter out a vast majority of any offensive content. This can help improve user experiences by ensuring that none of the users are offended or distracted. On top of that, removing inappropriate content helps facilitate a healthy online environment for our users to engage with live sports coverage.

We use a machine learning model to classify tweets and label them as "toxic" or "not toxic". If our model predicts a sufficiently high probability score for a tweet to be "toxic", we will filter it out before the tweet makes it to the next stage.

For our content moderation model, we use a TF-IDF count vectorizer as a feature extraction algorithm. The feature vector is then fed into a logistic regression classifier to produce the final prediction. Chapter 5.1 describes more experiment details and reasons for choosing this specific model.

2.4 Sentiment Filter

The next stage of our system is the sentiment filter. Depending on the needs of the clients, it will filter out tweets with specific types of sentiments. There are three types of sentiments: positive(score $>$ 0.2), neutral(score between -0.2 and 0.2), and negative(score $<$ -0.2).

A sentiment filter may be necessary due to client requirements. Our system selects the same tweets for all types of clients regardless of their personal preferences, and this may be the only place where clients have some flexibility. Imagine a client may be a fan of the Manchester United team. That client may have a preference to see tweets about Manchester United with positive sentiment. The sentiment filter gives them such flexibility for customization.

We use the vaderSentiment package[11] in Python to perform sentiment analysis. This package employs a rule-based lexical analysis framework, and it is best suited for Twitter data. Due to the fact that incoming data are most likely to be about Premier League games, this package is generally effective, but it may be too generic for our use case.

2.5 Engagement Prediction Model

All of the components covered so far provide basic quality assurances of the content curated and satisfy some user preferences. We still need a component to rank all the content and push out the most ideal one from Twitter. The engagement prediction model will take up this responsibility.

After tweets have made it past the sentiment filter, all of them are sent as inputs to the engagement prediction model. The model will provide a score for each tweet, and the score represents the expected popularity level of that tweet. After obtaining scores for all the tweets, we will rank them by the predicted popularity score in descending order.

This method is necessary for our problem for two reasons. The first reason is

that the engagement level of a tweet is a good proxy for its quality. Our clients will probably want to capture trending content on social media and send them to users in real-time. The second rationale is based on the lack of foresight on a tweet's engagement level in the future. We assume that all tweets provided by clients will be real-time, and their current engagement metrics will not be reliable. A model will be necessary to predict those metrics.

Chapter 3

Engagement Prediction Dataset

A successful machine learning model for predicting engagement level requires good data. That is the reason we invested in building the dataset for this task. To collect data, we utilize the Twitter Search API to gather historical data from Twitter. Then we identify some specific features that will intuitively impact the engagement level. We then perform exploratory data analysis to verify our hypotheses and make final decisions on features that can make it to our model. Finally, we will make some feature engineering choices to represent all the features properly in our model and to facilitate an efficient run of the optimization algorithm during the training process.

3.1 Data Collection

Since there are no existing datasets for performing the engagement prediction task on Twitter data, we need to build a dataset from scratch. We do so by leveraging the Twitter Search API for obtaining relevant tweets. Since England Premier League games happen on a weekly basis, this effort is ongoing until late November 2021.

During each week, we identify the games that have taken place. For each game, we build a query to pull all the tweets that contain at least one of the team name. We note that this approach is not perfect as it may miss out on relevant tweets that don't have references to a specific team name. At the same time, it may also introduce some irrelevant content. However, irrelevant tweets will not be a concern as there are

Figure 3-1: An example tweet in the dataset.

very few of them in our dataset. As our dataset grows larger, the influence of those irrelevant tweets on the model parameters will be insignificant.

Once we complete pulling all the tweets, we will pass all of our collected tweets through the basic filter described in chapter 2.2. Our model is only intended to work with complete tweets in English, so anything that doesn't match this standard should be discarded. With all the tweets collected, we remove any duplicates that we have collected. We do not need to worry about repetitions across different weeks because only tweets posted in the 7-day period before the time of which data is collected are accessible through the Twitter Search API. All the tweets that we have collected for this week will not overlap with anything from the previous week. We upload all unique tweets as a file after deduplication. Finally, a crawler will process the uploaded data and add them into a SQL database.

By the time of which this thesis is written, there are around 100,000 tweets related to England's Premier League games in the database. Figure 3-1 shows an example of a tweet in our dataset.

3.2 Features Overview

As we inspect the metadata of our dataset, we identify a list of features that can potentially be helpful in predicting engagement level. We provide a brief justification

for each feature below:

- Text of the tweet(`tweet_text`): Obviously, what people write in their tweet will matter a lot. More sensational content is more likely to attract other users.
- User bio(`ud_text`): The user bio in a profile helps us understand who a user is. Influential figures such as football players of a Premier League team, or a renowned sports journalist, can bring higher level of engagement for their tweets.
- Time of day of which the tweet is posted(`hour`): We can imagine a tweet that is posted at 8pm right after a game is likely to receive more engagements than a midnight tweet.
- Number of hashtags(`num_hashtags`): Hashtags are helpful tools to reach out to a broader audience group. However, having too many hashtags may work against the goal of getting more attention.
- Number of URLs(`num_urls`): Having URLs in a tweet suggest that it has external content that can attract more engagements. At the same time, users are generally wary of a tweet full of URLs.
- Number of user mentions(`num_mentions`): Mentioning too many users suggests that the author can be reaching out to their small bubbles and may not care about engagements. Users can also be turned away by tweets with many user mentions.
- Number of media(`num_media`): Having media-rich content, such as images and videos, can attract users for further engagements.
- Author's verification status(`user_verified`): A verified user is likely a celebrity or a reputable source of a subject area. In our context, those are most likely associated with official team accounts, players of Premier League teams, and verified sports reporters and commentators. These figures will have a broader audience than those with unverified Twitter accounts.

- Author's number of followers(`num_followers`): More followers implies reaching a broader audience.

- Number of lists that contain author's account(`num_lists`): Newcomers to Twitter can follow some lists of accounts with different topic focus. If one's account is in more lists, then that person is more likely to be followed by a new user on Twitter. This will create more engagement opportunities.

- Number of tweets posted by the author in the past(`num_past_tweets`): This is an indicator of user activity. Having posted more tweets in the past can be a signal that the user is active. An active user can receive more engagements for their tweets.

- Whether the profile/profile image is default (`default_profile, default_profile_image`): Having a default profile or profile image can suggest that the user may not be active and thus have fewer engagements for the tweets that they post.

3.3 Exploratory Data Analysis

After identifying potential features, we want to analyze the data to verify our intuitions. We focus on analysis on two aspects of data: textual data(`tweet_text` and `ud_text`) and tabular data(everything else).

3.3.1 Textual Data

During exploratory data analysis, we investigate the most common tokens people use in their tweet and their user bio. We perform this analysis by grouping all the tweets into two categories. The popular tweets are the ones that have received at least 500 likes, and the normal ones do not meet this requirement. Table 3.1 shows the wordclouds for the most frequent n-grams within the tweet content and the user biographies.

(a) Unigrams in content for popular tweets

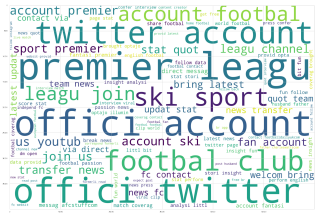
(b) Unigrams in content for normal tweets

(c) Bigrams in content for popular tweets

(d) Bigrams in content for normal tweets

(e) Unigrams in user bio for popular tweets

(f) Unigrams in user bio for normal tweets



(g) Bigrams in user bio for popular tweets

(h) Bigrams in user bio for normal tweets

Table 3.1: A table with wordclouds representing the most frequent n-grams for tweet content and user biographies. Each column represents a specific type of tweets. The caption under each image describes the type of tweets, the feature, and number of words that each wordcloud shows.

	log_fav_count	num_hashtags	num_user_mentions	num_urls	num_media	user_description_present	user_verified	log_followers	log_lists	log_tweets	using_default_profile	using_default_profile_image	log_retweet_count
log_fav_count	1.000000	-0.018645	-0.092515	-0.111871	0.226312	0.113878	0.381696	0.576554	0.518409	0.212748	-0.107855	-0.066920	0.858443
num_hashtags	-0.018645	1.000000	-0.073116	0.175042	0.208939	0.065023	0.041756	0.039224	0.066269	-0.025735	-0.046329	-0.020302	0.022269
num_user_mentions	-0.092515	-0.073116	1.000000	-0.145970	-0.078002	-0.066570	-0.088121	-0.154902	-0.149587	-0.138548	0.088443	0.051335	-0.109559
num_urls	-0.111871	0.175042	-0.145970	1.000000	0.110490	0.061649	0.133013	0.129513	0.186706	0.189251	-0.154196	-0.046871	-0.061470
num_media	0.226312	0.208939	-0.078002	0.110490	1.000000	0.064396	0.110539	0.143773	0.161988	0.016284	-0.061770	-0.047581	0.234647
user_description_present	0.113878	0.065023	-0.066570	0.061649	0.064396	1.000000	0.091323	0.283138	0.189573	0.220295	-0.117631	-0.259809	0.096922
user_verified	0.381696	0.041756	-0.088121	0.133013	0.110539	0.091323	1.000000	0.542759	0.612269	0.270622	-0.279323	-0.037667	0.346197
log_followers	0.576554	0.039224	-0.154902	0.129513	0.143773	0.283138	0.542759	1.000000	0.829410	0.693217	-0.388405	-0.231021	0.529797
log_lists	0.518409	0.066269	-0.149587	0.186706	0.161988	0.189573	0.612269	0.829410	1.000000	0.608321	-0.488697	-0.107905	0.482921
log_tweets	0.212748	-0.025735	-0.138548	0.189251	0.016284	0.220295	0.270622	0.693217	0.608321	1.000000	-0.406782	-0.196591	0.210376
using_default_profile	-0.107855	-0.046329	0.088443	-0.154196	-0.061770	-0.117631	-0.279323	-0.388405	-0.488697	-0.406782	1.000000	0.087715	-0.103179
using_default_profile_image	-0.066920	-0.020302	0.051335	-0.046871	-0.047581	-0.259809	-0.037667	-0.231021	-0.107905	-0.196591	0.087715	1.000000	-0.049838
log_retweet_count	0.858443	0.022269	-0.109559	-0.061470	0.234647	0.096922	0.346197	0.529797	0.482921	0.210376	-0.103179	-0.049838	1.000000

Figure 3-2: The correlation matrix of the tabular data for the tweets collected.

An interesting trend we observe is that the most frequent tokens used for popular tweets and regular tweets are similar within the tweet body. This suggests looking for the presence of specific words within tweet content may not be the best for separating popular and normal tweets. Therefore, a statistical learning model with word frequency as features may not be effective. We may need more a sophisticated approach to extract a semantic embedding for tweet content. On the other hand, we observe words like "official" and "account" float around in the user bio. These are official accounts for teams and players and tend to have more followers. It may not be surprising to see those tweets having higher levels of engagement.

3.3.2 Tabular Data

On the tabular features, we plot a correlation matrix to identify features that may be helpful in predicting engagement level. Figure 3-2 shows the correlation matrix among features and labels. The label we are using here is `log_fav_count`, which is the logarithm of the number of likes a tweet has received. Some of the features here are log-transformed. We will provide details on this transformation and the rationale for doing so in Chapter 3.4

We can observe that number of followers and number of content lists of which the author is in are important signals for predicting engagement level. This observation is expected as authors with more followers are likely to have more content exposure. This helps them to get a higher level of engagement for the content they produce on Twitter. The rest of the features listed in Chapter 3.2 have weak correlations with our label. We decide to incorporate all of the features without worrying too much about

some features being highly correlated with each other. This is because we will have a large quantity of training data, so overfitting will not be a concern. In addition, our primary goal is to predict engagement level with our machine learning model. We are not interested in performing any causal analysis to identify factors that **cause** a tweet to have high engagement levels.

3.4 Feature Engineering

Now that we have decided on a set of features to use for our machine learning model. The next step is to transform the features so that they are ready for the model to use. This is an essential step to ensure that our features meet the input specifications of our machine learning model.

The first set of decisions we need to make are with the textual features. We decided to remove all hashtags, URLs, and user mentions in both `tweet_text` and `ud_text`. We do so by applying a regular expression to identify all the relevant elements in the textual data and then replacing them with empty strings. These decisions are made in conjunction with our model choice. Since we are using a Transformer-based model [14][7][13] for predictions, we rely on a tokenizer to break up the text into small word units. These Transformer-based tokenizers are not specifically designed for handling tweets, especially the hashtags, URLs, and user handles. Including these may hurt model performance because the model will treat hashtags and other elements as some unknown tokens. This will distract the model from learning to extract a good semantic embedding for the entire tweet. Furthermore, we also have tabular features for the number of such elements in a tweet, so including tweet-specific elements without cleaning can be repetitive. Figure 3-3 shows an example output of the text cleaning algorithm.

Besides textual features, we need to process other tabular features. We describe the main decisions we have made for different groups of tabular features and justify our choices below:

We use one-hot encoding to encode the hour of the day of which a tweet is posted.

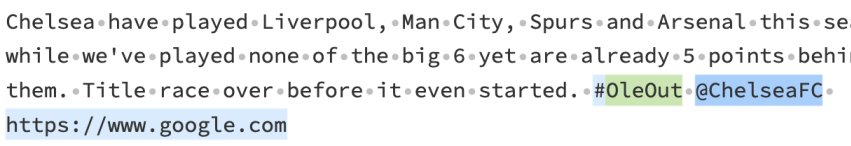


Figure 3-3: A demonstration of the text processing algorithm. The highlighted areas are the text that will be removed. As shown in this figure, URLs, hashtags, and user mentions are highlighted by a regular expression.

As discussed in Chapter 3.2, the time of which a tweet is posted can make an impact on the engagement level. It will be problematic to use the hour as a raw categorical feature because there is no direct linear relationship between engagement level and increasing hour of day. To consider an example, a tweet posted at 8pm(hour=20) may have a higher engagement level than a tweet posted at 6am(hour=6), as well as a tweet posted at 11pm(hour=23). To resolve this issue, we use one-hot encoding to convert the hour into a list of 0-1 features, with 1 representing the hour of which the tweet is posted. This method enables us to analyze each hour interval independently without losing information. Although this increases the dimension of our tabular feature by around 23, having a large training set will alleviate this concern.

We applied log transformations on features and labels with large values, including number of likes, retweets, followers, lists, and past tweets of which a user wrote in the past. The rationale for applying this transformation is to standardize the range of various features and labels. Consequently, this will facilitate a more efficient run of our optimization algorithm during training. Without it, we can imagine some categorical features with value of 0 or 1 while a user may have tens of millions of followers. If we just pass in features with such a large magnitude, it will take forever for our machine learning model to converge during training. The other reason that makes this transformation reasonable is that all the features that we mention are non-negative, which is a requirement for log transformations. To address the concern that taking logarithm of 0 will be negative infinity, we use the t function for transformation.

$$t(x) = log_{10}(x + 1) \tag{3.1}$$

This transformation ensures that zero value will be mapped to zero and there will not be any infinities in our transformed dataset.

We decide to leave other tabular features as-is because they are either categorical features with 0/1 value, or numerical features with a range between 0 and 20.

Chapter 4

Engagement Prediction Model

With a good dataset, we can now start building a machine learning model to predict the engagement level of a given tweet. In particular, we try to predict the number of likes a tweet will eventually get. We can typically observe these metrics a few days after a tweet is posted, but this is apparently not realistic as we need to make real-time predictions. As a result, it is important for us to extrapolate this metric based on the information we have.

4.1 Linear Regression Model

Based on Occam's Razor, the simplest model that does the job is typically the best one. This philosophy motivates us to start with the simplest model, and we can then attempt more sophisticated models later on. For our engagement prediction task, the simplest model will be a linear regression model that uses all of our tabular features.

Although our feature dimension is significantly lower than the number of training data points, we still need to avoid overfitting. Therefore, we fit our linear regression model with L2 regularization. In essence, our linear regression model attempts to minimize the following objective function:

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} (\theta^T x_i + \theta_0 - y_i)^2 + \lambda \|\theta\|^2 \tag{4.1}$$

In the equations above, θ and θ_0 represent the parameters of our linear regression model. Specifically, θ_0 is the zeroth-degree constant offset. x_i and y_i are the features and label values, respectively. n is the number of data points in the training set. λ is the regularization parameter. The first term in the equation is the typical Mean-Squared Error(MSE) loss function for a linear regression model. The second term avoids overfitting by penalizing any regressor with large parameters. Large parameters suggest the model may be overfitting to the training data. Intuitively, large λ will result in stronger regularization effect by favoring θ with small magnitude. In our use case, we have sufficient training data and therefore a small λ will suffice.

4.2 Text-based Model

So far we have built a baseline model relying on tabular features. The result from the exploratory data analysis in Chapter 3.3 suggests textual features may be helpful for predicting engagement level. On top of that, any methods depending on word counts may struggle because the most frequent words are similar for tweets with different levels of engagement. These insights point us toward using state-of-the-art language models to perform this regression task.

Transformer-based models are one of the most common choices for state-of-the-art language models[7]. These models are first pretrained on a large corpus for language modeling, and they can be then finetuned on a domain-specific dataset for a narrower task. In our case, we finetune a Transformer-based regression model for predicting engagement level.

Figure 4-1 shows the architecture diagram for our text-based model. We use two separate distilBERT models[13] to extract semantic embeddings for the tweet text and the user biography. We then concatenate the embeddings and feed it into a simple linear layer. It is worth noting that the MLP regressor here only has a linear layer to speed up training and inference. The reason that distilBERT is picked over other variants is for its performance. DistilBERT models are more efficient than the standard BERT model with marginal performance decline.

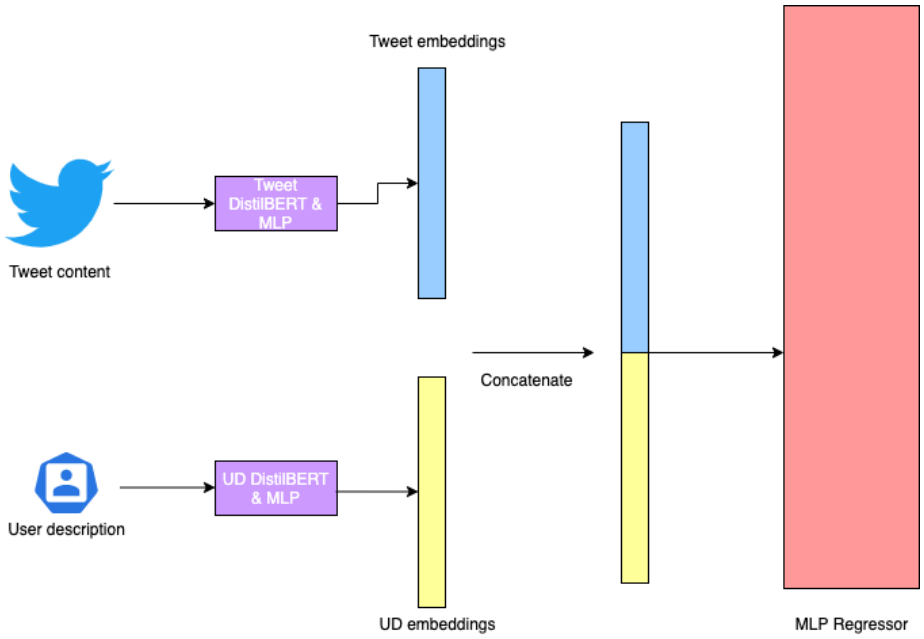


Figure 4-1: The architecture diagram of the text-based model. The tweet content and the user bio is fed into a distilBERT model and we obtain their respective embeddings. We then concatenate the embeddings and feed the new feature vector into a linear MLP to obtain the result.

4.3 Feature Combination

Now that we have created a model with two different sets of features. The next step is to find a way to put everything together. We made two relatively simple attempts in this section.

4.3.1 MLP Regressor

To make our feature combination method successful, we need an effective tool to properly weigh semantic embeddings of textual features and other tabular features. A simple linear layer for text regression task from Chapter 4.2 is insufficient and will not result in significant model performance improvement for any feature combination methods. This observation is reasonable given the complexity and the variety of our features. To use all of them together, we need a more sophisticated module to extract useful information from the feature vector generated by our feature combination methods. The MLP regressor described below serves our use case well.

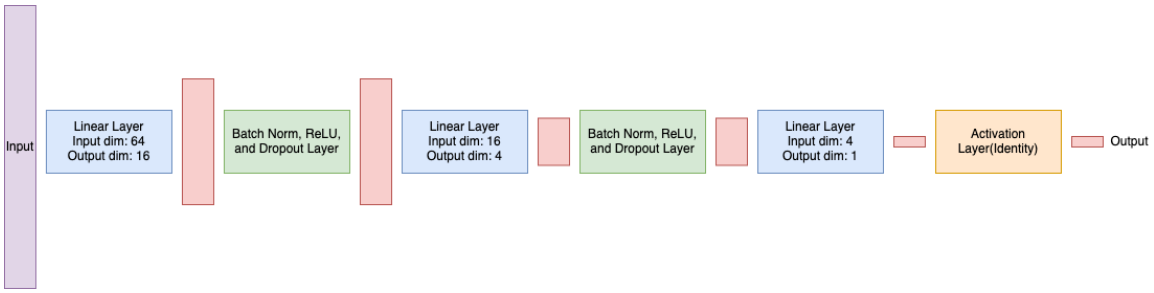


Figure 4-2: The architecture diagram of the MLP module. This figure demonstrates a MLP unit with input dimension 64 and output dimension 1. The first linear project the input down to 16-dimensional vector. Then it goes through batch normalization, ReLU, and the dropout layer. The entire process is repeated again to get a 4-dimensional hidden vector. Finally, this hidden vector goes through a linear layer to a 1-dimensional output. For our regression task, we use the identity function for the activation module.

Now we will describe the architecture of our MLP regressor, as shown in Figure 4-2. The MLP has several modules. Each module consists of a fully-connected layer. The input dimension of the fully-connected layer is at least 4 times larger than the output dimension. Following the fully-connected layer is a batch normalization module. Then the output will be fed into the ReLU activation function. Finally, the result goes into a dropout layer with a dropout probability of 0.1. The number of modules depending on the dimension of input, the dimension of output, and a specified ratio between input dimension and output dimension for each module. For the last module, only a fully-connected layer is present.

4.3.2 Concatenation

Figure 4-3 shows the most straightforward feature combination method: concatenating all the feature vectors. In this approach, we use the distilBERT model to obtain text embeddings from tweet content and user bio. We concatenate these text embeddings together, along with all the tabular features. This is the simplest way to combine textual data and tabular data as the MLP regressor can now properly weigh both types of data together.

The apparent weakness of this method is that the MLP regressor may not be able to find the right balance between different types of features. This has to do with the

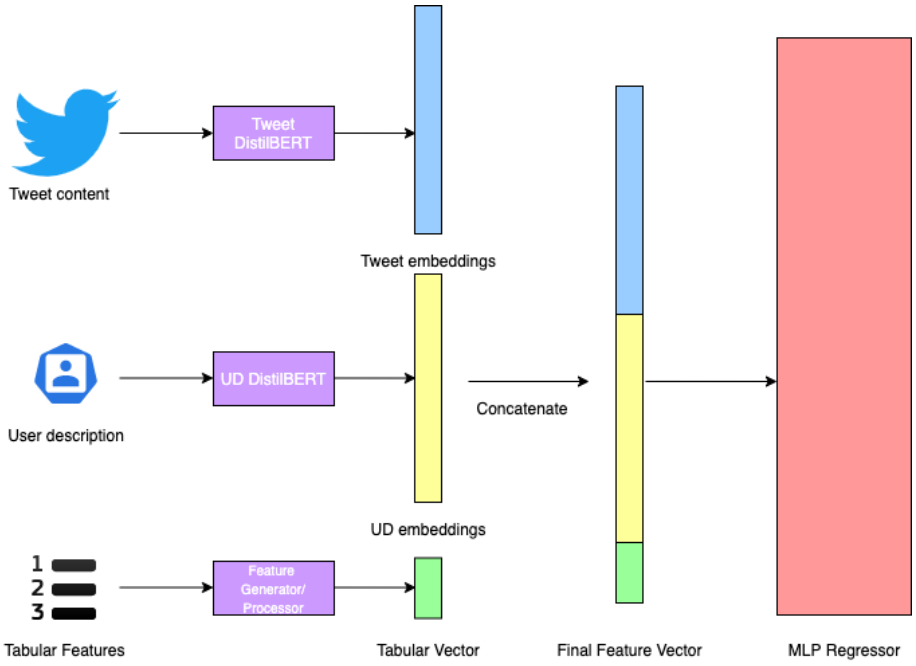


Figure 4-3: The architecture diagram of the model using the concatenation feature combination method. The tweet content and the user bio is fed into two separate distilBERT models and we obtain their respective embeddings. We then concatenate all the text embeddings with the tabular features and feed the result into the MLP regressor.

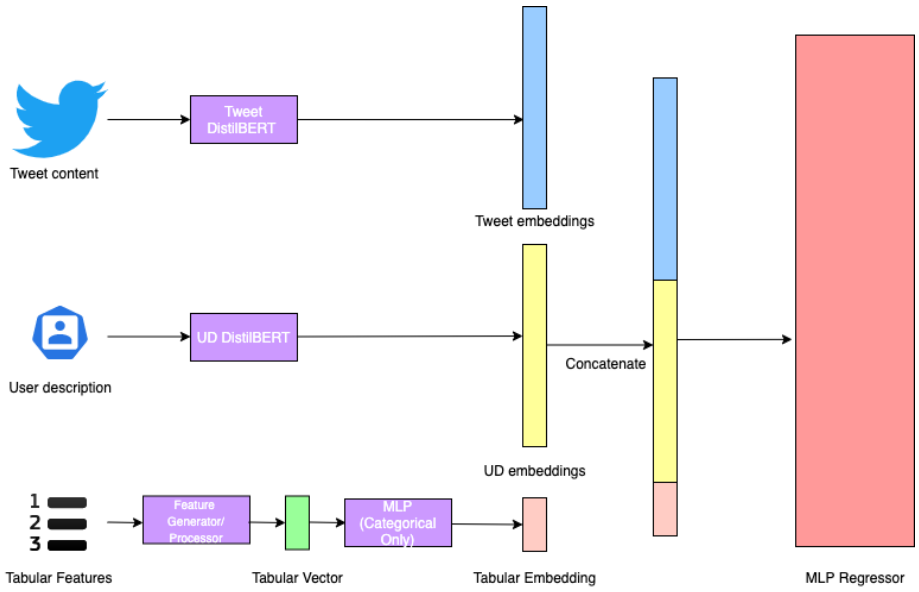


Figure 4-4: The architecture diagram of the model using the concatenation feature combination method. The tweet content and the user bio is fed into two separate distilBERT models and we obtain their respective embeddings. We feed to categorical feature vector into the MLP. Note that the numerical feature vector is unchanged moving out of that MLP for categorical features. We then concatenate all the text embeddings with the tabular features and feed the result into the MLP regressor.

dimensions of those features. For our dataset, the dimension of the tabular vector is around 30-40. This is very small compared to a 1528-dimensional concatenated text embeddings for tweet content and user biography. On top of that, a right balance between categorical features and numerical features within the tabular vector may be difficult. We attempt to solve this problem in the next section.

4.3.3 MLP on Categorical Features

In the previous section, we mentioned that the model may not be able to find a proper balance between categorical and numerical features. In particular, the dimension of our numerical feature is less than 10 while our categorical feature vector is at least 20-dimensional. It may be sensible to add a small MLP module for categorical features to limit its dimension.

As shown in figure 4-4, this architecture is pretty much the same as the one in Chapter 4.3.2, except we have a small MLP unit for the categorical features. Note

that this MLP does not apply to the numerical features. The architecture of this MLP is the same as the one described in Chapter 4.3.1, except for the number of output units is 8 for our case. The additional MLP unit is intended to refine and extract the most useful information out of the categorical feature vector, and the output will be concatenated with the numerical feature vector to form the new tabular feature vector.

This modification does not solve the problem of weighing tabular and textual features properly. Perhaps the above addition may help, but we need something else to achieve a good balance for all types of features. This leads to an attention module as described in the next section.

4.4 Attention-based Multimodal Model

A question arises from the combination approach presented in section 4.3.2: We know that the dimension of the text embeddings completely outweigh the dimension of tabular features. With a simple fully-connected layer, the model may not be able to learn the optimal balance in terms of weighing between textual features and tabular features. We need to adopt a method that enables the model to figure out a fine balance between the two types of features. We made an attempt in the previous section to find a balance within two types of tabular features, but that may not be suitable once we involve textual features. To solve this problem, we use an additional attention layer.

The idea of an attention module has been discovered a while ago for machine translation tasks [3]. The success of subsequent state-of-the-art Natural Language Processing models, including the Transformer-based models, heavily rely on the attention mechanism. The following equations describe the setup of our attention module.

$$m = \alpha_{t,t} W_t x_t + \alpha_{t,c} W_c x_c + \alpha_{t,n} W_n x_n \tag{4.2}$$

where

$$\alpha_{i,j} = \frac{exp(\mathbf{LR}(a^T[W_i x_i, W_j x_j]))}{\sum_{k\in\{t,c,n\}} exp(\mathbf{LR}(a^T[W_i x_i, W_k x_k]))} \tag{4.3}$$

In the above equations, x_t represents the text embeddings, x_n represents the numerical feature vector, and x_c represents the categorical feature vector. m is the final embedding vector output by the attention module, and it will be fed into a MLP to produce the final output. a is a trainable weight vector in the attention module. W_i represents a trainable weight matrix for different types of feature vectors. $[x, y]$ represents the concatenation of vectors x and y. $\mathbf{LR}$ represents the leaky ReLU function.

We note that the final embedding m that is fed into the MLP is a linear combination of the textual, categorical, and numerical features. We can think of α as the extent of which the model needs to pay attention to a specific type of features, as all the α values will add up to 1. This setup ensures that the model will learn the proper way to distribute its attention to different types of features for different input data points.

Now that we are done describing the attention layer. We present the architecture diagram of the attention-based model as shown in Figure 4-5. The tweet content and the user bio is fed into two separate distilBERT models and we obtain their respective embeddings. Afterwards, the attention module that we have described will take the concatenated text embedding and the tabular embeddings to produce the final tweet embedding. The tweet embedding is then fed into the MLP regressor for the final output.

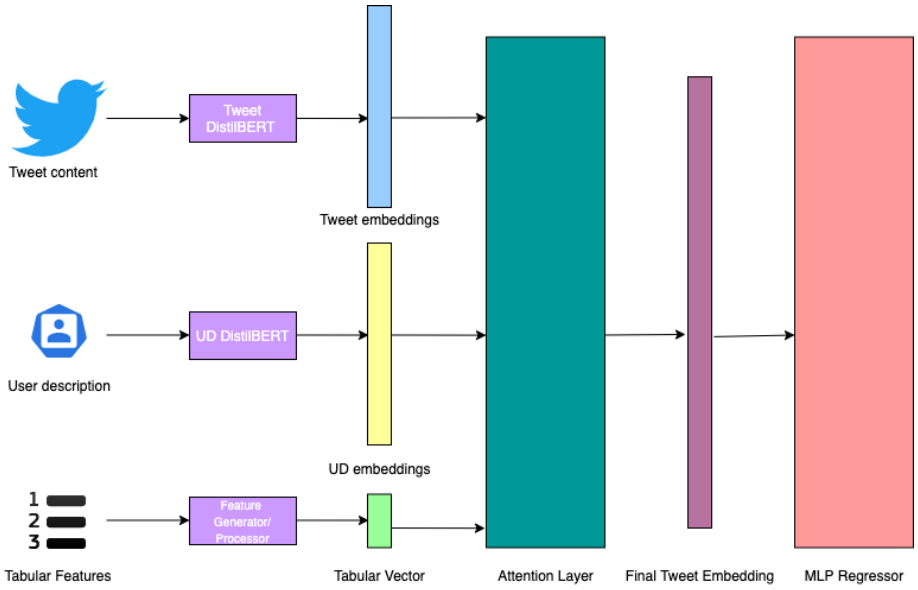


Figure 4-5: The architecture diagram of the model using the attention-based method. The tweet content and the user bio is fed into two separate distilBERT models and we obtain their respective embeddings. We then have a attention layer for the concatenated text embedding and the tabular embeddings to produce the final tweet embedding.

Chapter 5

Evaluation and Discussion

In this chapter, we will focus on the results of machine learning models in our system. The first part of this chapter will focus on the content moderation model as presented in Chapter 2.3. Then we will move on and discuss the engagement prediction model. For each part, we will describe the experiment setup, present the performance of our trained models, and discuss those results.

5.1 Content Moderation Model

As described in Chapter 2.3, we use machine learning to perform content moderation. In particular, we want to classify a tweet as "toxic"(positive label) or "not toxic"(negative label).

5.1.1 Experiment Setup

We use an existing toxic tweet dataset [1] on Kaggle, as it is the dataset closest to our model use case. We split our dataset and only use 70 percent of the original dataset as the training data. The rest of the dataset becomes testing data. Once the dataset split is done, we preprocess the text by removing all URLs, hashtags, and user mentions in the tweet text as they do not provide additional information on the toxicity of tweets.

Model	Precision	Recall	F1 Score	Accuracy	Throughput(tweets/sec)
TF-IDF Feature	0.9358	0.9247	0.9301	0.9404	452
BERT	0.9388	0.9550	0.9468	0.9548	38

Table 5.1: The result on the test set for different models after training.

We use two approaches for this problem. The first method uses a TF-IDF count vectorizer as a feature extraction algorithm. The feature vector is then fed into a logistic regression classifier to produce the final prediction. Besides URLs, hashtags, and user handles, we also remove common stopwords defined by the NLTK library[5]. All of the common stopwords are generally non-toxic and may dilute signals of offensive content.

The second method is to use BERT[7] to get a semantic embedding of the tweet. Then we feed the semantic embedding to a classification layer to get a prediction. Since BERT models are successful in many text classification tasks after being fine-tuned a specific dataset, we choose to adopt this method. To train the BERT model, we do so for 3 epochs with initial learning rate 2×10^{-5} with an AdamW optimizer[12]. We adopt a linearly decaying learning rate schedule. The dimension of the semantic embedding is 768. The dropout probability is 0.1. The Transformer encoder within the BERT model has 6 layers and 12 heads, Within the Transformer encoder, the dimension of the internal hidden layer is 3072.

In the next section, we will see that the Transformer-based BERT model performs better than the TF-IDF model. However, we decide to use the TF-IDF model due to efficiency considerations and its performance on real-time data. To further probe the performance of our content moderation models on real data, we collect an additional 1000 tweets related to specific Premier League games as an external dataset to test our model. We label all the 1000 tweets manually.

5.1.2 Model Performance

Table 5.1 shows the performance of different models after being trained on the Kaggle dataset. Although the BERT model has a higher F1 score than the TF-IDF model, its

Threshold	Precision	Recall	F1 Score	Accuracy
0.5	0.9389	0.3275	0.4856	0.7134
0.3	0.9411	0.3721	0.5333	0.7310
0.2	0.9358	0.3953	0.5559	0.7390
0.1	0.9320	0.4516	0.6084	0.7598
0.05	0.9291	0.5078	0.6566	0.7806
0.01	0.8692	0.6182	0.7225	0.8038
0.005	0.7965	0.6977	0.7438	0.8014

Table 5.2: The result of the BERT content moderation model on the external data set with different separating score thresholds.

Threshold	Precision	Recall	F1 Score	Accuracy
0.5	0.9586	0.5388	0.6898	0.7998
0.3	0.9251	0.6705	0.7776	0.8414
0.2	0.8836	0.7209	0.7940	0.8455
0.1	0.8152	0.8295	0.8223	0.8519
0.05	0.7516	0.8857	0.8131	0.8319
0.01	0.5664	0.9670	0.7144	0.6805
0.005	0.5163	0.9845	0.6773	0.6125

Table 5.3: The result of the TF-IDF content moderation model on the external data set with different separating score thresholds.

inference speed is significantly slower. One of the main requirements for our system is to be efficient as we are working with real-time data. Given that we will spend much time for engagement prediction, we cannot afford to spend extra time for content moderation.

We then evaluate both models on the newly collected tweets and try out different score thresholds for toxicity classification. Table 5.2 shows the performance of the BERT model on the collected data, and Table 5.3 shows the performance of the TF-IDF model.

5.1.3 Discussion

We observe that the performance on the collected data is worse than the one on the held-out test data for both models. This is reasonable with a few explanations. First, the Kaggle dataset contains data from Twitter, but they are not constrained to a specific topic. The content of the tweets in the training data is slightly different

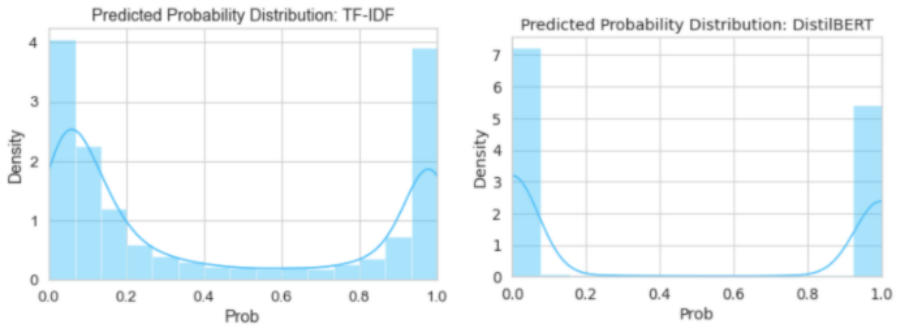


Figure 5-1: The distribution of prediction scores for content moderation models. The one to the left is for the TF-IDF model and the right one is for the BERT model. The y-axis measures the density of prediction distribution.

compared to the external dataset we have collected. This variation in data content may give our models a more difficult time performing this classification task because these models are working with data of a new context.

The second reason, and perhaps a more important one, is the discrepancies in dataset labeling instructions between the Kaggle dataset and the external dataset. For labeling the external dataset, we apply a high level of scrutiny on tweets. Even if a tweet is appropriate overall and is not intended for harm or offense, we label a tweet as "toxic" as long as we see a single inappropriate word in there. On the other hand, the labeling standards are much more relaxed for the Kaggle dataset. Even though we do not have direct access to their labeling standards, an appropriate tweet with a swear word may be labeled as "not toxic". This difference in labeling standards can negatively impact the performance of our models on the real-life dataset, as our model is learning from a more relaxed standard and cannot adapt to stricter scrutiny without extra training.

Finally, it can be shocking to learn that our BERT model is doing worse compared to the TF-IDF model on the external dataset. This probably has to do with the overconfidence of the Transformer-based model. As illustrated in Figure 5-1, we see the predictions made by the BERT model are more polarized. In other words, the BERT model tends to make more confident predictions. This overconfidence hurts the BERT model in particular once we see the label shift as discussed earlier. Even setting different thresholds will not save BERT model's performance as the predictions are

Name	Description	Value	Models Applied
`dropout`	Dropout probability for BERT embeddings	0.3	2,3,4
`mlp_dropout`	Dropout probability for MLP layer	0.3	3,4
`epochs`	Number of epochs of model training	12	2,3,4
`lr`	Learning rate of the model	3×10^{-5}	2,3,4
`max_len`	Maximum number of tokens in a text	240	2,3,4

Table 5.4: A list of hyperparameters used for training the engagement prediction model.

too inflexible for the external dataset.

5.2 Engagement Prediction Model

The goal of the engagement prediction model is to predict the number of likes a tweet will get at the end. In this experiment, we set up different models using various sets of features and feature combination methods.

5.2.1 Experiment Setup

We use the engagement prediction dataset that we have built as described in Chapter 3. We split our engagement prediction dataset and use 80 percent of data for training, 10 percent of data for validation, and the other 10 percent for testing. We set up four models for the experiment. The first one is a ridge regression model as described in Chapter 4.1. We will only use the tabular features for this model and all the default hyperparameters as defined in scikit-learn. The second model is a distilBERT model as described in Chapter 4.2 that only uses textual features. The third model uses a distilBERT model that concatenates textual embeddings and tabular features. The fourth model uses a distilBERT model with an attention layer for different types of features.

During training, we train each model for up to 12 epochs. At the end of every epoch, we evaluate and save our model using our validation set. We then pick the model with the lowest loss on the validation set and run it on the test set.

Table 5.4 shows the hyperparameter configurations we use for all the models men-

Number	Model Name	Text	Tabular	Combine Method	R2 Score	MSE
1	Ridge Regression		Yes	-	0.5294	0.4204
2	DistilBERT	Yes		-	0.6251	0.3350
3	DistilBERT	Yes	Yes	Chapter 4.3.2	0.7264	0.2445
4	DistilBERT	Yes	Yes	Chapter 4.4	0.7385	0.2337

Table 5.5: The result on the test set of the engagement prediction dataset. MSE here is the mean-squared-error of the model.

tioned above. Those hyperparameters are selected with our validation dataset as well. It follows the process described in the previous paragraph, except that we are running training different sets of hyperparameters. We choose the hyperparameter set that achieves the lowest validation loss during the entire training process. Note that if anything is missing from the table, then we are using the default configurations set by the HuggingFace[16] and Multi-Modal Transformer[10] module.

5.2.2 Model Performance

Table 5.5 shows the performance of different models laid out in the previous section. We note that relying on the textual features alone can be more effective than relying on tabular features only. This can be that ridge regression is not a sufficiently sophisticated model to extract all information from the tabular features. It can also be that the BERT model can build effective embedding for the tweet content and for the user biography, both of which are important signals for engagement levels.

A simple concatenation of textual features and tabular features enhance model performance further. This is expected as combining both pieces of information gives us better ideas about engagement. In this case, the MLP regressor does most of the heavy lifting and deserves the most credit for this improvement.

Finally, the attention-based concatenation method performs the best among all models. One simple reason can be that the model has extra parameters in the attention layer that don't otherwise exist. We will discuss more about the attention layer in the next section.

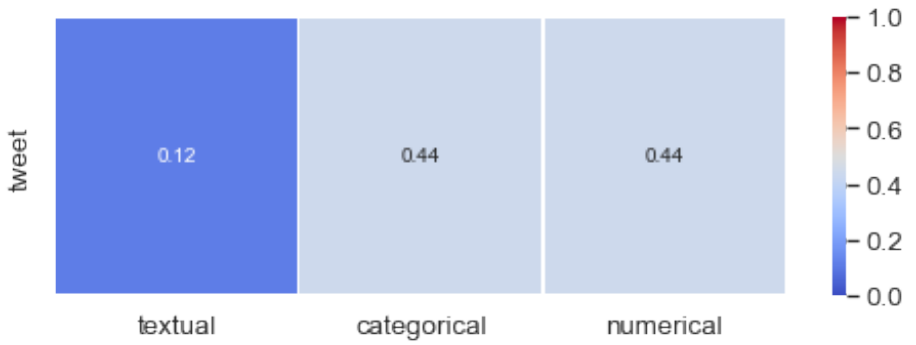


Figure 5-2: The average attention distribution over all data in the testing set.

5.2.3 Discussion

Our engagement prediction model tends to struggle with tweets that put much more emphasis on media content over text. For example, a tweet with a single word "GOAL" and a few photos highlighting a quite sensational moment may get a high level of engagement. However, our models do not consider specific media content or perform any analysis on them. It won't be surprising then to observe our models producing a much lower prediction.

To further understand the importance of the attention layer, we attempt to visualize the attention parameters. We do so by feeding in our held-out testing data into our model. While the model is running inference, we collect the attention distribution for each data point. Finally, we take the average for each element of the distribution. Figure 5-2 shows the resulting attention parameters.

The result presented by the above figure may seem counterintuitive at first. The performance of our models suggest we may be relying on textual features more. The mean attention distribution suggests the contrary. To understand this distribution, we need to keep in mind the dimension of various types of features. The tweet content embedding and the user bio embedding have a dimension of 768 each. On the other hand, we only have a 35-dimensional tabular feature space. By depending on the concatenation method in model 3, it tends to overemphasize the textual features because it will weigh everything in the final embedding equally. This creates an inherent advantage for the textual features caused by their high dimensions. Interestingly, we

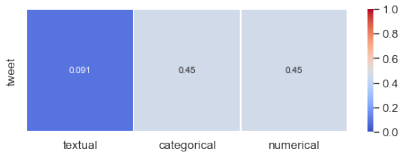


Figure 5-3: The right figure shows the attention distribution of the example tweet of the left.

note that tweets with higher levels of predicted engagement tend to pay slightly less attention to textual data compared to an average tweet. An example of this is shown below in Figure 5-3.

5.3 Testing the Entire System

We are interested in seeing how the engagement prediction model can help us obtain good content for the entire system. We do so by collecting around 5000 tweets related to the Arsenal v. Newcastle game on November 27, 2021. We feed all the tweets into the system by a batch of 500 and ask for the top 3 tweets that summarize the entire game. Figure 5-4 shows the output from the system responding to this request.

All of the top 3 tweets suggested by our system have at least thousands of engagement actions. The first two serve as a round-up for the entire game with a photo depicting one of the highlighted moment during the game. The third tweet points to an important event during the game: the first goal. It is worth noting that all of three tweets come by users with verified accounts. In fact, all of these accounts have the word "official" in their user biography. Finally, the tweets shown in Figure 5-4 are certainly not the top three tweets in terms of their levels of engagement, but it is still useful for us to surface needed content to some extent.

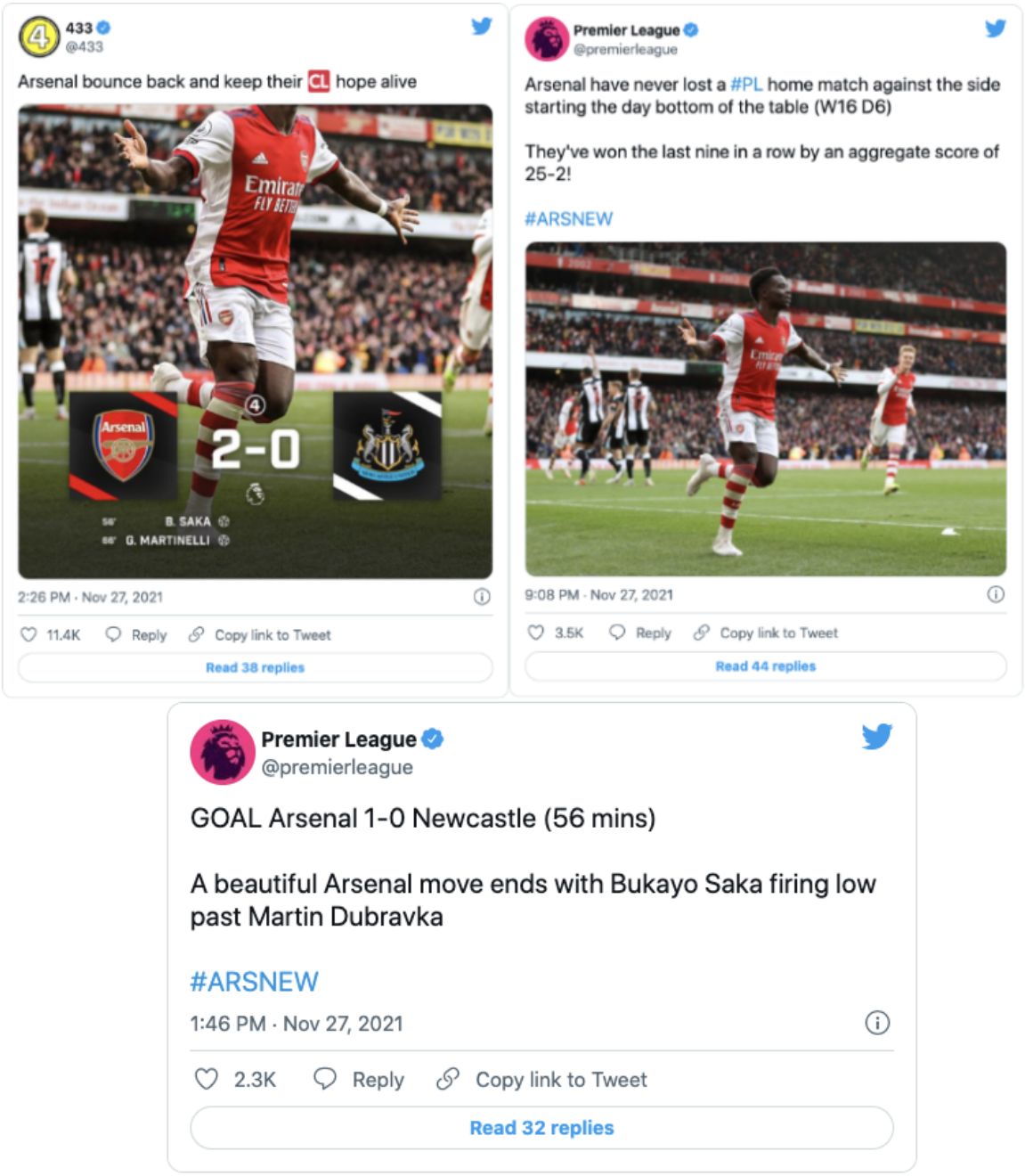


Figure 5-4: A sample output of our system when it is asked to provide the top tweets for the whole Arsenal v. Newcastle game on November 27, 2021.

Chapter 6

Engineering and Implementations

Now that we have trained our models, we need to integrate them with our system to bring in their contributions. We will now discuss some specifics on the engineering aspect of this project, especially on the deployment of machine learning models and the creation of a streamlined machine learning training pipeline. The engineering work is critical not only for its necessity for converting research ideas to practical products, but it also paves the way for further research and improvements on this project. All of the work described in the chapter is implemented and available for internal use at Sky.

6.1 Deploying Content Moderation Model

The first model that we need to deploy is our content moderation model. Based on the results presented in Chapter 5.1.2, we decide to deploy the TF-IDF model. To do so, we upload our model file onto cloud storage. Whenever the system needs to use the content moderation model, it will load the model file from the storage unit and then perform inference on incoming data. We make the following considerations for this deployment strategy:

- The methods we have described above are the easiest. We don't need to write extra code, besides the one for invoking the model. Setting up the right cloud

environment on Amazon Web Services aside, there is not much effort involved in making this happen. In comparison, alternative approaches such as making a separate endpoint for the machine learning model will involve much more implementations and become more time-consuming and may potentially take more computing resources.

- It may not always be wise to load the model every time the system needs to access it. Our current strategy forces us to load the model every time the system receives incoming data. This is not a big deal for us. Once the model is initialized and loaded for the first time after deployment, it will be cached and future loads will become much more efficient. The time for model loads after the cold start is negligible. In addition, the first load will take at most a few seconds because the model file is relatively small.

One specific challenge with respect to this deployment strategy is to set up the `scikit-learn` package, of which the model is implemented in. Our cloud environment does not provide native support for this Python package, and it is impossible for the automatic deployment script to upload a local version of the `scikit-learn` package because of space constraints. We need to work around this issue by creating a custom layer that supports `scikit-learn`. We then augment this layer to our system so that the package is supported.

The deployed model performs efficiently given our demand. We observe that there is approximately a 1-second cold-start period when the model receives an input request. The system is loading the content moderation model during that 1-second period. After loading the model, our system can make about 120 predictions per second. This is reasonably efficient and will not slow down our entire system.

6.2 Inference Endpoint for Engagement Prediction

Contrary to the content moderation model, the situation with respect to the engagement prediction model is very different. We work with model files around 500 MB,

thus making the model loading process painfully long. Furthermore, there are other files of which the model depends on to be successfully loaded. Finally, our engagement prediction model has hundreds of millions of parameters and needs to be run with a GPU environment to preserve its efficiency.

Flexibility is another critical property of our framework of choice. The content moderation model has a single file, but we have more than that. For example, we have a configuration file that helps with properly initializing the model. We also need support for a tokenizer to perform real-time data preprocessing before model inference. Then a specialized environment is necessary so that all the Python packages we have used, such as HuggingFace[16] and Multi-modal Transformers[10], are supported.

All of these considerations point us to create a new API endpoint to serve the model. Clients will put in data through the API endpoint in the request body, and the endpoint will handle user input and output predictions toward the input data. This framework has several benefits. First, we can choose GPU as part of our hardware support to significantly accelerate inference. Second, implementing an endpoint will provide more flexibility.

Figure 6-1 shows the architecture diagram of our engagement prediction model endpoint. Once an incoming inference request is accepted, the data in the request body is extracted and passed into the model handler pipeline within the model image. The model image is an environment specifically set up for so that our engagement prediction model can run. The model handler pipeline contains three nodes. The Preprocess node converts the input data to the form of which the model can accept. The inference node invokes the engagement prediction model and obtains raw predictions from it. If the model is not initialized, we will load the model from an S3 bucket, our cloud storage unit. The Postprocess node will then convert predictions produced by the model into a more workable data structure and send it back to clients.

While we set our eyes on this endpoint framework, there are some internal choices we need to face. We had to choose between a PyTorch-based framework and a generic framework. The PyTorch-based framework is easier to implement and more automated, while the generic framework is more flexible. At the end the flexible nature

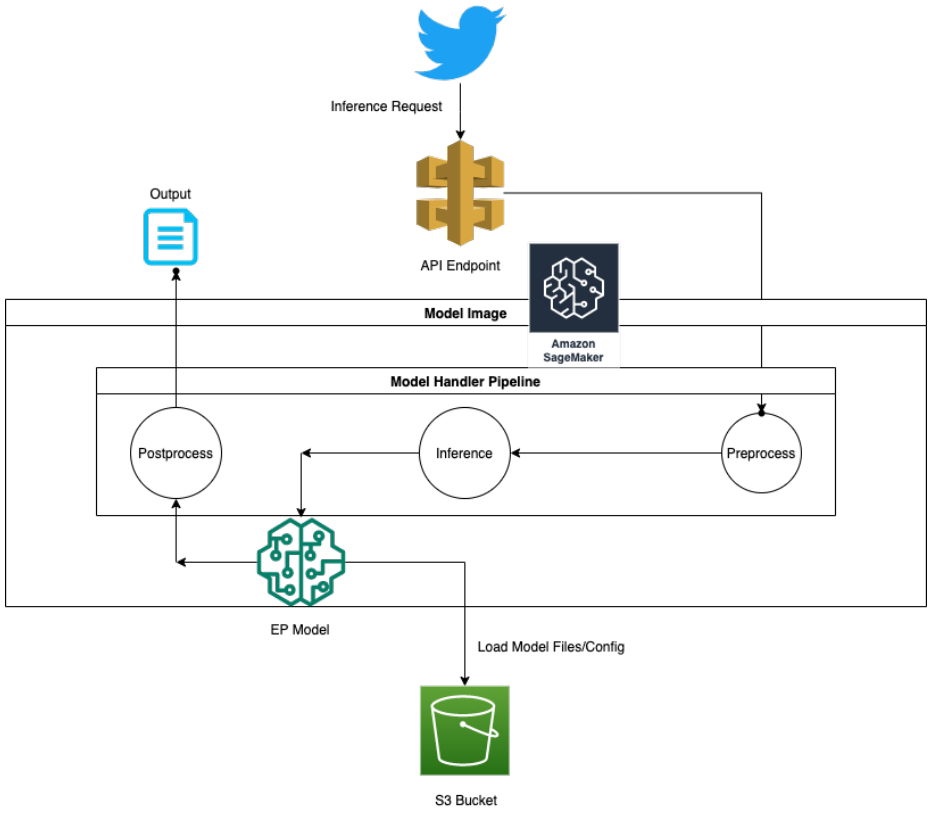


Figure 6-1: The architecture diagram of the deployed endpoint for the engagement prediction model.

of the generic framework wins. The ease of implementation for the PyTorch-based framework comes at a cost of flexibility. It relies on message-passing between different nodes of the model handler pipeline, and our model is way too big to be passed around. The generic framework works around the limit in message passing by allowing us to define our model and tokenizer objects as class attributes. This opens the door for one-time only initialization at the beginning, which can be done by tracking the model state.

There are more benefits with the generic model framework. A major one is its modularity. We show the code structure of our model handler pipeline below:

```
class EPModelHandler:
    def __init__(self):
        self.initalized = False
        self.ep_model = None
        self.tokenizer = None

    def initialize(self, context):
        # Code to initialize model
        self.initialized = True

    def preprocess(self, request):
        # Code to preprocess data

    def inference(self, model_input):
        # Code to run model and get predictions

    def postprocess(self, result):
        # Code to convert model predictions into a Python list.

    def handle(self, data, context):
        model_in = self.preprocess(data)
```

```
        model_out = self.inference(model_in)
        output = self.postprocess(model_out)
        return output
```

This structure can be helpful as we can run a subset of nodes in the model handler pipeline. It also simplifies implementations as we can easily integrate some of the code written for data processing and inference in Chapter 6.3. On the other hand, the PyTorch-based framework provides limited modularity because we cannot define class attributes. This forces most of the data processing work to be done in the `inference` function, which is rather awkward from a software engineering point of view.

Our API endpoint that serves the engagement prediction model can now take in a batch of tweet objects and output their predicted engagement levels. Our endpoint can handle approximately 35 tweets per second. Due to the limitation on the size of the request body, the API endpoint can handle up to about 100 tweets per request.

The lack of automation is the main limitation with this model endpoint. While the endpoint is running and fully functional, the deployment process is not fully automatic. There is an existing script for deploying the pipeline, but people need to fill in their authentication information manually before getting permission for deployment. Automating this process will accelerate the deployment process and aid with future improvements discussed in Chapter 6.3.

6.3 Training Pipeline for Engagement Prediction

One of the headaches that data scientists need to face when they are experimenting with machine learning methods is the way to organize all the code. Since our code intended for research and experiments is not subject to the same standard as production-ready code written by software engineers, machine learning code can become chaotic and disorganized very quickly without regular maintenance. For production-level machine learning models, we need to ensure that the code is organized and modular. These properties will streamline the training and the deployment

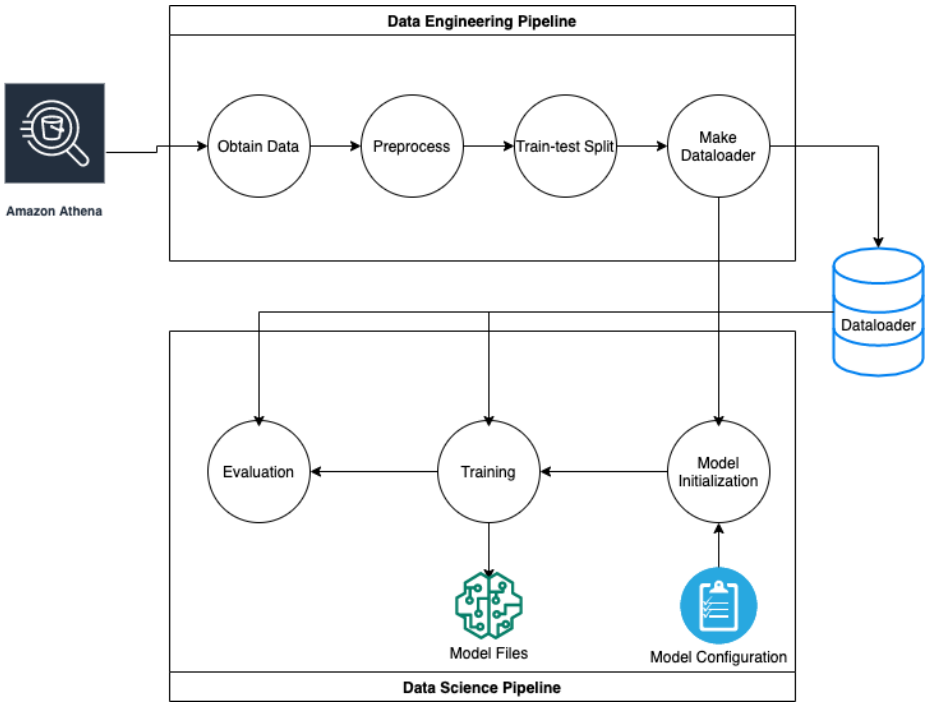


Figure 6-2: The architecture diagram of the end-to-end Kedro pipeline for training an engagement model. The pipeline will take the data from a database and output a trained machine learning model for engagement prediction.

process.

Fortunately, we have an existing tool that helps with organizing our machine learning code and converting it into a full training pipeline. We use the Kedro package[2] in Python to create our training pipeline for the engagement prediction model.

Figure 6-2 shows the architecture diagram of the Kedro pipeline. It is composed of a data engineering pipeline and a data science pipeline. The data engineering pipeline takes in raw data from a database, preprocesses the data, and converts them to "dataloaders", a format of which the our machine learning model can accept. The data science pipeline is intended for training and evaluating a new model and output a model file. It takes "dataloaders" that are preexisting or generated by the data engineering pipeline, initializes and trains a new model, and stores the trained model into a local file.

The data engineering pipeline first loads data stored from a database on Amazon Athena, a database service provided by AWS. Then the loaded data goes through the

"Preprocess" node so that we can obtain new features and clean existing features. Details about the data processing are discussed in Chapter 3.4. The "Train-test Split" node then splits the processed dataset into training, validation, and testing dataset. We have a random seed to ensure the split is consistent across multiple runs. Finally the "Make Dataloader" node converts split datasets into dataloader objects.

The first step in the data science pipeline is to initialize an untrained model based on an internal model configuration file. Then it will take the dataloader and run the training algorithm. Once training is complete, it will produce a model file and send the model for evaluation. The evaluation node will run the trained model on test data and report out all the metrics.

The Kedro and the design of this pipeline satisfies the important quality of modularity. Most users will run the entire pipeline at once. However, the design also enables users to run the data engineering pipeline or the data science pipeline alone. To take a step further, they can even decide to run a single node in the pipeline. The Kedro framework allows us to modularize our training script by breaking it down by different components and stitching them together as modular pipelines.

One limitation of this pipeline is that it can only perform computations locally. If one needs to run the pipeline with the goal of having a trained model deployed. They have to run the pipeline, upload the model file to cloud storage, and follow instructions in Chapter 6.2 to make this happen. A good future direction for this pipeline is to deploy it to the Amazon Web Services so that it can run on the cloud. We can then configure the pipeline to save the trained model file to cloud storage. We will then create a separate pipeline for deployment that takes in a model file and produces an API endpoint for invoking the model.

Chapter 7

Conclusion

Overall, we were able to build a Twitter Content Recommendation API specifically for English Premier League games. The API endpoint will help with advice on the best tweets for a companion feed during live game coverage. One can imagine using this API endpoint for other use cases. For example, we can use this API endpoint to gather all the tweets during the first half of a game during the half-game break. We can also use this endpoint at the end of each game by identifying tweets that act as a good summary of all the events during a live Premier League game.

The driving forces to make this API endpoint successful in its mission are the content moderation filter and the engagement prediction model. The content moderation filter is critical to ensure that viewers will not see any inappropriate and offensive content, which will cause damage in terms of usage of the overall Sky application that employs this API endpoint. The engagement prediction model helps with identifying content with the highest engagement potential in the future, and this is indispensable given that the tweets of which the clients do not have reliable engagement metrics.

7.1 Limitations and Future Work

There are several challenges that will limit the performance of this recommendation API. In terms of its structural limit, the endpoint has to limit the number of tweets a client input. This is because of the inherent restrictions of the amount of the data

that Amazon API Gateway can handle per request. Having too much data will cause the API endpoint to crash. Without changing the infrastructure of which we rely on, there is no way to work around these limitations. The clients will be responsible for developing a wise strategy to query the endpoint multiple times to achieve their desired use case.

The other major challenge is the relevance filter. For now, we have a simple rule-based system checking for matching terms. This approach works fine with most content, but it can sometimes be too restrictive. Suppose we want to find the top tweets about a goal that has just happened. The API endpoint will exclude all tweets without mentioning "goal" or any players who are involved with that specific game event. We can imagine such a system will miss out on a lot of valuable content. For instance, a tweet saying "Brilliant!" with an image of the player who has scored the goal will not make it past the relevance filter. On the other hand, this relevance filter may accidentally include irrelevant content as well. A tweet that may be talking about Brentford, an area in southwest London, can make it past the relevance filter for a game involving the Brentford team. With these reasons above, the event endpoint of the recommendation API may not necessarily work well. To resolve this issue, we need a more sophisticated approach. The problem with using machine learning methods is the lack of access to data. In particular, we need data that represents games and events to match with tweet content. We can then use NLP models to solve the relevance classification problems. Another possible solution is to create a feature representation for images, which can be helpful for the relevance classification problem if we have enough domain-specific image data.

Besides the inherent limitations with our API endpoint, making good use of this API endpoint will be challenging without high quality data. Due to current limitations with the relevance filter mentioned above, the burden of finding relevant tweets falls more on clients. The process of acquiring such relevant tweets is nontrivial. Clients can take advantage of the Twitter Search API or the Twitter Powertrack API to customize their tweet feed for their input data. Nevertheless, the interface for designing rules to match specific tweets is challenging to work with. On top of that,

most of the rules focus on specific attributes of the tweet and of the authors. Clients may also make very specific rules to match tweets based on the text, but no tools can help them with determining relevance of their media content. This issue further reinforces the need to refine the relevance filter.

7.2 Going Beyond Sports

For this thesis project, the social media content that we are working with is very targeted in terms of its scope. We specifically focus on content related to the England Premier League games. However, we can appropriate this tool to other domains with some more work. Obvious alternatives include other sports, such as tennis and cricket games. If we want to make this happen, we will need to retrain our engagement prediction model on domain-specific data depending on our use cases, as our current model will not work well with out-of-domain data. Although the tweet and author attributes may still be reliable, out-of-domain text features will cause our engagement prediction model to have suboptimal performance. With this method, we can even go beyond sports.

We rely on collecting domain-specific data to empower our engagement prediction model at this moment. Another interesting direction to take with the engagement prediction model is to generalize it with a broader domain. For example, we may build an engagement prediction model that can be used for all sports in England. This generally results in lower model performance as the engagement prediction model needs to learn about many domains during training. This can be overcome with sufficient data across domains of our choices.

7.3 Personalization

The current system will provide the same set tweets to all users regardless of their personal preferences. There are some existing features such as the sentiment filters to provide some degrees of customization, but those are manual processes and are far

from personalizations. One of the next steps is to add personalization into this system. This can involve turning our machine learning model into a recommender system. Rather than predicting the overall level of engagement, we can predict whether a specific user will engage with a specific tweet. There is some previous work done on individual-level tweet engagement prediction. Our version of the problem should be slightly easier because all the tweets are from the same domain. The more challenging part is for data collection. One potential proposal is to use this current system to broadcast a set of tweets and ask users for feedback on specific tweets. The other option is to add an engagement feature to enable users to interact with specific tweets, and we can use those engagement data for training our new machine learning model.

Bibliography

[1] Toxic tweets dataset. https://www.kaggle.com/ashwiniyer176/toxic-tweets-datasetm. Accessed: 2021-09-15.

[2] Sajid Alam, Lorena Bălan, Gabriel Comym, Yetunde Dada, Ivan Danov, Lim Hoang, Rashida Kanchwala, Jiri Klein, Antony Milne, Joel Schwarzmann, Merel Theisen, and Susanna Wong. Kedro, December 2021.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[4] Luca Belli, Sofia Ira Ktena, Alykhan Tejani, Alexandre Lung-Yut-Fon, Frank Portman, Xiao Zhu, Yuanpu Xie, Akshay Gupta, Michael Bronstein, Amra Delić, Gabriele Sottocornola, Walter Anelli, Nazareno Andrade, Jessie Smith, and Wenzhe Shi. Privacy-aware recommender systems challenge on twitter's home timeline, 2020.

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[6] Amine Dadoun, Ismail Harrando, Pasquale Lisena, Alison Reboud, and Raphael Troncy. Two stages approach for tweet engagement prediction, 2020.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[8] Nicolò Felicioni, Andrea Donati, Luca Conterio, Luca Bartoccioni, Davide Yi Xian Hu, Cesare Bernardis, and Maurizio Ferrari Dacrema. Multi-objective blended ensemble for highly imbalanced sequence aware tweet engagement prediction. In *RecSysChallenge '20: Proceedings of the Recommender Systems Challenge 2020*, pages 29–33. ACM, 2020.

[9] Shuhei Goda, Naomichi Agata, and Yuya Matsumura. A stacking ensemble model for prediction of multi-type tweet engagements. In *RecSysChallenge '20: Proceedings of the Recommender Systems Challenge 2020*, pages 6–10. ACM, 2020.

[10] Ken Gu and Akshay Budhkar. A package for learning on tabular and text data with transformers. In *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*, pages 69–73, Mexico City, Mexico, June 2021. Association for Computational Linguistics.

[11] C.J. Hutto and Kenny Joseph. Vader Sentiment Analysis, March 2021.

[12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[13] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. June 2017.

[15] Maksims Volkovs and Zhaoyue Cheng et al. Predicting twitter engagement with deep language models. In *RecSysChallenge '20: Proceedings of the Recommender Systems Challenge 2020*, pages 38–43. ACM, 2020.

[16] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020.