Universitat de Barcelona

Fundamental Principles of Data Science Master's Thesis

Twitter engagement model for RecSys Challenge 2021

Author:
Marcos Moreno Blanco
Adrià Torralba Agell

Supervisor:
Dr. Santi Seguí Mesquida
Pere Gilabert Roca

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamental Principles of Data Science

in the

Facultat de Matemàtiques i Informàtica

July 1, 2021

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Twitter engagement model for RecSys Challenge 2021

by Marcos Moreno Blanco
Adrià Torralba Agell

Recommendation systems is an interesting and wide field of research and it is present in a huge amount of different areas in our daily life. The RecSys ACM conference is the most important conference in the recommendation area and every year they organise a competition: the RecSys Challenge. The work presented here aims to solve the RecSys 2021 Challenge which consists of giving a probability to two Twitter users that interact. In this project we have worked in the development of a model which uses the power of Gradient Boosting Trees to combine multiple hand-crafted features in an aim to represent the interaction between the users. Our team reached the 14th place in the overall challenge leaderboard and is placed between the 7th and the 9th place in terms of Like overall performance.

Acknowledgements

To our project managers Santi and Pere for their dedication in this work and interest in the competition.

To our family and friends for the support and confidence in the realisation of this project.

Contents

Chapter 1

Introduction

1.1 Introduction

During the past decades mankind has evolved so rapidly that the amount of data and information we generate has become impossible to manage as we used to. Streams and rivers of data flow into seas and oceans of immeasurable size, causing an overload of information in which more data does not necessarily mean an improvement of the system. Too much information can sometimes mean less sales, and companies know that: according to Forbes, there are 2.5 quintillion bytes of data created each day and more than 3.7 billion humans use the internet. On average, Google processes more than 40000 searches every second, and Spotify adds 13 new songs every minute to its platform. In a world with so much information available at any time, the need for an external tool that helps downsize the amount of information we need to deal with has grown exponentially.

1.1.1 Recommender Systems

Due to the reasons stated above, there has been a rising interest in the so called *recommender systems*, the goal of which is to predict users' preferences to help filter the most suitable items for them . They were first mentioned in a technical report in 1990 ([1]), and then implemented at a large scale during the whole decade (e.g. [2] and [3]). Nowadays however, they can be found in a large number of domains, ranging from music or film services, to social media apps and online stores. From Amazon, to Spotify, Twitter and Booking, recommender systems are now everywhere, and if there is someone to 'blame' for, that is Netflix. From 2006 to 2009, the company organised the *Netflix Prize*, a competition with a prize of $1,000,000 to a team that could improve the accuracy of their own recommender system. This competition is believed to have been a turning point that increased the momentum of research on recommender systems, that today revolves around all kinds of topics: from bias, fairness, bubbles and ethics of recommender systems, to security and privacy, and economic models and the consequences of recommender systems.

1.1.2 Types of Recommender Systems

Recommender systems can be formulated mainly as two types of problems: prediction and ranking problems. In prediction problems, recommender systems aim to predict, be it the score a user would give to a film after inferring the user's tastes, be it whether a user will like a certain song or tweet or not (binary classification). Ranking problems consist in returning a list of the top-k items available instead.

At the same time, recommender systems can be classified into non-personalised methods if they provide the same results to all users, collaborative filtering methods if they build a model based on the users' previous decisions or content-based methods

if they use features of the different items to make recommendations, among others. However, nowadays it is common to see hybrids of the aforementioned methods rather than single approaches.

1.2 RecSys Challenge 2021

The ACM Conference on Recommender Systems (RecSys) is an international event in which the most advanced techniques, results and systems in terms of recommender systems are presented. It is considered the most important conference at global level in the field of recommender systems research. Each and every year the convention revolves around an unique theme or role, being it this year "A place to meet and exchange". In addition, every year an international company partners with the event to organise a challenge that focuses in solving a real-world task within the scope of recommender systems.

The RecSys Challenge 2021 is organised by Politecnico di Bari, ETH Zürich, Jönköping University, and the data set is provided by Twitter. The challenge focuses on the practical task of tweet engagement prediction in a dynamic environment. In particular, the challenge implies predicting four different engagement types: Likes, Retweet, Quote, and replies. The goal is to predict the probability of the different engagement types of a target user for a set of tweets based on heterogeneous input data while at the same time providing fair recommendations.

1.3 Motivation

Doing this project we can apply first hand all the knowledge gathered during this last year course in which we have completed the Fundamentals Principles of Data Science MSc. At the same time, we see this as an opportunity to face and overcome obstacles that can only appear when working in real-world problems and datasets; and participating in a top international competition like this one is enough of a reason to wanting to try our best. It is an unique and enriching opportunity to culminate the masters in the best possible and practical way.

1.3.1 Goals

The main objective of this project is to develop a complex recommendation system to solve the problem proposed. For instance, we want:

- To extract general features from the given dataset.
- To extract text features from the given dataset.
- To create a Non-text related model using some State-of-the-Art techniques.
- To create a Text Model using some State-of-the-Art model that works in the multilingual case.
- To create a powerful model where we join all the hand-crafted features.
- To submit regular solutions to the RecSys Platform and compete with top teams.

Since one of the requirements for the RecSys Challenge 2021 is *fairness*, we are also interested in build a fair model in our implementation.

1.3.2 Document Structure

This document has the following structure:

- Chapter 1: Introduction to the present topic.
- Chapter 2: Background. Here we introduce all the theoretical insights needed to follow the outline of the project.
- Chapter 3: Dataset. Here we explain and analyse the given dataset.
- Chapter 4. Model Architecture. Here we present the problems encountered and we explain how we have sorted them. Additionally, we present here the architecture of the implemented model.
- Chapter 5: Text Model. All the features regarding the text of the Tweet are explained here.
- Chapter 6: Non-Text Model. All the non-text related features and models are explained here.
- Chapter 7: Results. Here we explain the embedding created in order to improve the particular scores of the two different models.
- Chapter 8: Conclusions. In this last Chapter, we summarize and conclude our project and suggest ideas for further work.

Chapter 2

Background

2.1 RecSys Challenge 2020

Last year RecSys Challenge was very similar to this one. In fact, the goal was the same: to predict whether if two different Twitter users interact or not, and if they interact, which interaction they will yield. The main difference between this year's challenge and last years is that this year we have an additional requirement about Fairness. (See Section 7.1.3 for details).

Figure 2.1 shows the competition leaderboard of the participants from last year challenge.

Final Leaderboard

Rank	Team/User Name	Overall Score	Method Name	PRAUC Retweet	RCE Retweet	PRAUC Reply	RCE Reply	PRAUC Like	RCE Like
1	NVIDIA RAPIDS.AI	9	rapids.ai v2	0.6111	37.91	0.2185	24.23	0.9108	53.01
2	learner	14	Layer 6 AI	0.5395	30.96	0.2218	21.94	0.7761	25.42
3	Team Wantedly	18	Team Wantedly	0.5266	30.06	0.1918	20.44	0.7716	24.76
4	learner_recsys	20	ffm+sgd+prior+embeddings	0.4529	22.50	0.1161	10.42	0.7221	18.28
4	BanaNeverAlone	20	last_test_sub_2	0.5042	27.21	0.1850	18.71	0.7531	21.20
4	AskMiso & CLIP	20	askmiso & CLIP v1	0.5135	28.57	0.2069	21.09	0.7650	23.65
5	wsm_LLLLL	21	Method 1: Expected + Tweet	0.5516	-0.04	0.5135	-0.02	0.6666	3.52
6	POLINKS	22	baseline	0.5516	-0.03	0.5135	-0.05	0.7131	-0.00
6	Los Trinadores	22	factored.ai Golduck	0.3673	16.93	0.5135	-0.04	0.7310	20.09
7	Better	23	Better	0.4992	26.98	0.1879	19.35	0.7484	21.17
8	myaunraitau	27	myaunraitau	0.4916	25.65	0.1622	16.76	0.7387	19.74
9	Not_Last_Place	28	last-combined	0.4891	21.80	0.1761	18.60	0.7511	17.64
10	narsil	29	quite random	0.4865	24.21	0.1570	14.98	0.7464	20.01
11	Chilling	30	AI	0.3088	12.33	0.1137	11.82	0.6602	12.70
12	stathisch	34	Basic NN	0.4370	21.22	0.1293	13.83	0.7199	17.58
13	Marysia	35	sample method	0.3232	11.81	0.1116	8.79	0.6038	6.65
14	rly	36	rforestbaseline	0.5516	-401.05	0.5135	-250.74	0.6161	-785.57
15	kagurazaka	38	gb_topic	0.3644	14.43	0.1175	12.33	0.6940	11.69

FIGURE 2.1: Competition Leaderboard RecSys 2020 Challenge.

2.2 Classifiers

After the revision of the results from last year RecSys Challenge [4], we have decided to use LightGBM [5]. Which is a well known classification algorithm based on the Gradient Boosting technique.

2.2.1 LightGBM

In order to define what LightGBM is, we need to define two different other concepts: Decision Trees and Gradient Boosting.

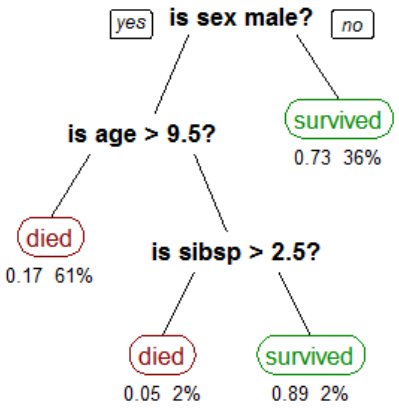


FIGURE 2.2: Decision Tree

On one hand, a decision tree is a common structure used in Machine Learning that serves as a classifier. You can see in Figure 2.2 an example of a decision tree: it creates a sequence of chained splits in order to classify a sample of the training dataset into a defined category. Furthermore, each leave node of the tree corresponds to a label that you aim to correctly classify. Its relative simplicity enhances this method interpretability provided by the fact that one can express any data sample by a combination of its features or characteristics.

On the other hand, we have Gradient Boosting and in particular Gradient Boosting Decision Trees (GBDT). The main idea behind them is that instead of considering one tree each time, there are multiple trees considered simultaneously in order to make our prediction. Thus, through many iterations the data is fit to the trees and a selected loss function is minimised. In terms of training, this kind of models follow a rather simple approach: they split the data if the splitting improves the performance of the model(i.e., it minimises the loss function). Due to its nature, this method can easily lead us to an overfitted model. Nonetheless, a good way to prevent this from happen is to limit the number of splits each node can have. Doing so, we are minimising the risk of our model to overfit. When expanding and building a tree, one can consider two different strategies:

1. **Level-wise**: in this case we expand by levels. This method yields a balanced tree in which the search time is minimal.

2. **Leaf-wise**: in this case we expands the node that optimises the most. In this setting, the resulting tree is more likely to be overfitted to the training dataset.

However, this usually provides faster training time and better performance when the dataset grows horizontally.

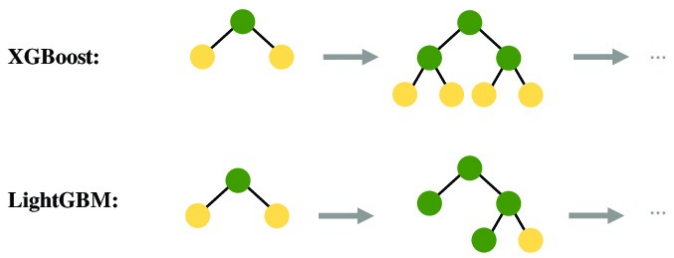


FIGURE 2.3: Comparison between XGBoost and LightGBM. Source: [6]

Figure 2.3 shows a comparison between these two approaches.

LightGBM splits the data at each point by grouping the data characteristics and using this grouped features instead of the original ones. Doing so, the algorithm is significantly faster.

2.3 Natural Language Processing

Another important topic in order to do this project is Natural Language Processing.

Natural Language Processing is the field inside Machine Learning that provides with algorithms for processing languages and text in general. There are plenty of interesting problems and topics in this field such as Optical character recognition (OCR), Speech recognition and synthesis, Sentiment Analysis and Named Entity Recognition (NER). In our particular case, we are interested in the BERT tokenization.

2.3.1 BERT tokenization

The BERT Model, which stands for Pre-training of Deep Bidirectional Transformers for Language Understanding, [7] is a text model developed by Google. A text model helps encode words into tokens, and vice versa. It was presented in 2018 and gathered a lot of attention in a very short span of time. It was trained using Wikipedia and BookCorpus [8], and there were released many versions of it. To be precise, the BERT variant of our interest is the *Bert-Multilingual-Base-Cased*.

The functioning of the BERT model is pretty straightforward: it uses 12 fully-connected layers combined with 12 *represent patterns*. These patterns capture contextual information, such as interaction between a word and the previous or the posterior one, the grammatical function words develop in different contexts, or how the words are correlated to punctuation marks. The result of this setup is a network with more than a hundred interacting patterns, to help encode and decode words in the most precise and accurate way. Figure 2.4 shows the structure and architecture of the BERT model.

To input the text from the tweets into the version of BERT of our choosing, first it is important how the input format of the sentences is structured: they always begin with the token [CLS] and end with [SEP], indicating the start of a new line and the end of the current line, respectively. Additionally, there is another special token ([UNK]) that is used whenever the BERT decoder find an unknown token or word.

To encode the words into tokens, a conversion using three different embeddings is used:

1. Token embeddings, in this phase transformations are made in order to find the root of the verbs and other transformations in order to reduce the word complexity.
2. Segment embeddings, in this phase we determine the number of segments we need for each sentence. We will have only one segment in case the input is only one sentence and two segments if the task requires two input sentences.
3. Position embeddings, in this last embedding we encode the position of the word within the sentence.

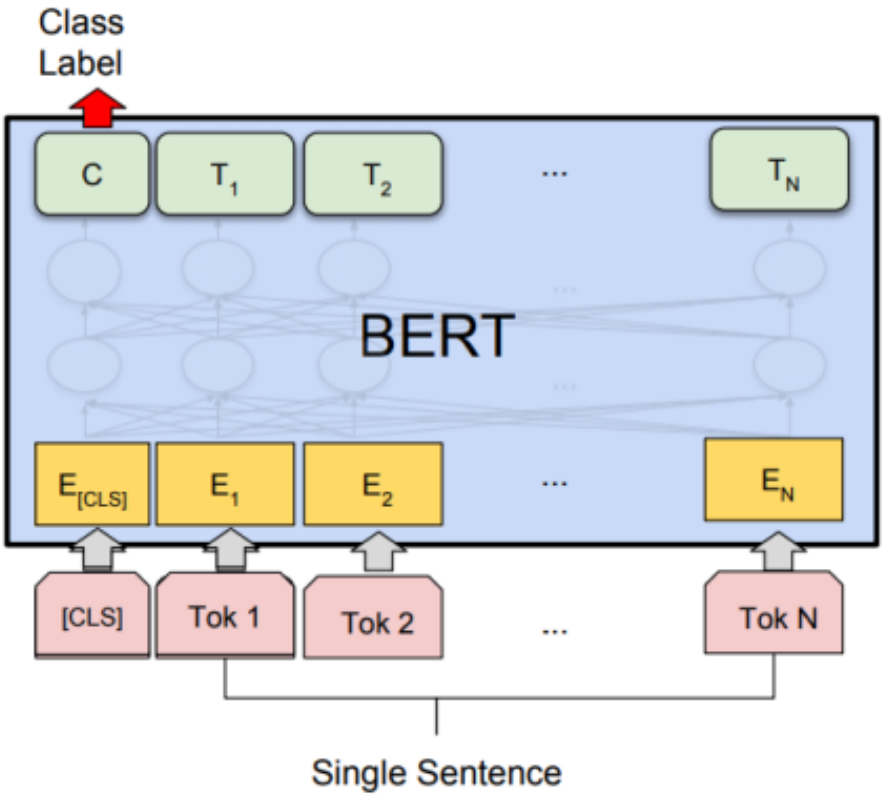


FIGURE 2.4: BERT model. Source: [9]

2.4 Encodings for categorical features

Categorical variables are those that contain two or more categories. These variables can be either nominal, with no intrinsic ordering to its categories, or ordinal, with a clear order. As many Machine Learning algorithms are unable to operate on categorical data but are able to learn from it, it is common to convert it to a numerical form. Since the data set contains a number of categorical features, such as `type` or `media`, here we explain only the two main types of feature encoding we have explored when implementing our models.

2.4.1 One-Hot Encoding

One-hot encoding consists in creating a new feature for each category, in which we place a 0 or 1 depending if the category is present in each sample or not. This is one of the most common types of encoding categorical variables and that is an advantage when implementing it: most libraries can provide a fast working function even for large data sets. Figure 2.1 shows this type of encoding.

Color	Red	Yellow	Blue
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Blue	0	0	1
Yellow	0	1	0

TABLE 2.1: One-hot encoding of a variable into three different ones

2.4.2 Target Encoding

Target encoding is a bit more complex: for each distinct category in the starting feature we compute the average of the corresponding values in the targeted label. Then, each value is replaced by the mean of its corresponding category. This method has the downsize that it can lead to overfitting very easily, specially in small data sets, when it could translate badly to a bad performance in the test set. Figure 2.1 shows this type of encoding.

Color	Target	Encoded Color
Red	1	0.5
Red	0	0.5
Yellow	0	0
Blue	1	1
Yellow	0	0

TABLE 2.2: Target encoding of a variable

Chapter 3

RecSys Challenge 2021 Dataset

The data available for the challenge consists in a public dataset of close to 1 billion data points, making roughly 40 million each day over 28 days.

Let us explain the different columns for each data point. They can be grouped in four different groups.

- **Tweet features:** Those columns contains the data of the tweet itself with all its related important information, such as a list with the hashtags existing in the tweet, the type of the tweet and the timestamp of the creation of the tweet, among others.
- **Author features:** Those columns contains information of the author of the tweet, such as the follower and following count of the account or if the user is verified, among others.
- **Reactor features:** Those columns contains information of the reactor of the tweet, such as the follower and following count of the account or if the user is verified, among others.
- **Engagement features:** In this group we have the columns that we aim to predict with the implementation of our recommender system. In particular it contains four different columns; one for each interaction we aim to predict; that contains a timestamp if a particular action was triggered by the reactor user. In addition to this, it also contains a columns with a flag indicating if the author of the tweet follows the account of the reactor user.

See Table 3.1 to see the complete details for each column.
The dataset was released in the following way:

1. First, on March 30th, the validation server opened and data from Week 1-3 was released, adding up to a total of $\simeq$ 270 GB.
2. Later, on June 9th, the test server opened, and data corresponding to Week 4 (validation data) was released, composed of two files of $\simeq$ 6.5 GB each, one with labels (part-00000) and another one without them (part-00001 (1)).

	Feature Name	Feature Type	Feature Description
Tweet Features	Text Tokens	List[Long]	Ordered list of Bert ids corresponding to Bert tokenization of Tweet text.
	Hashtags	List[String]	Tab separated list of hashtags (identifiers) present in the tweet.
	Tweet id	String	Tweet identifier
	Present media	List[String]	Tab separated list of media types. Media types can be Photo, Video or Gif.
	Present links	List[String]	Tab separated list of links (identifiers) included in the tweet.
	Present domains	List[String]	Tab separated list of domains included in the tweet (e.g. twitter.com)
	Tweet type	String	Tweet type, can be either Retweet, Quote, Reply or Toplevel.
	Language	String	Identifier corresponding to the inferred language of the tweet.
	Timestamp	Long	Unix timestamp, in seconds of the creation time of the tweet.
Author User Features	User id	String	User identifier.
	Follower count	Long	Number of followers of the user.
	Following count	Long	Number of accounts the user is following.
	Is verified?	Bool	Is the account verified?
	Account creation time	Long	Unix timestamp, in seconds, of the creation time of the account.
Reactor User Features	User id	String	User identifier.
	Follower count	Long	Number of followers of the user.
	Following count	Long	Number of accounts the user is following.
	Is verified?	Bool	Is the account verified?
	Account creation time	Long	Unix timestamp, in seconds, of the creation time of the account.
Engagement Features	Does author follows reactor?	Bool	Does the author account follows the reactor account?
	Reply timestamp	Long	If there is at least one, unix timestamp in seconds, of one of the replies.
	Retweet timestamp	Long	If there is one, unix timestamp in seconds, of the retweet of the tweet by the reactor user.
	Retweet with comment (Quote) timestamp	Long	If there is at least one, unix timestamp in seconds, of one of the retweet with comment of the tweet by the reactor user.
	Like timestamp	Long	If there is one, unix timestamp in seconds, of the like.

Table 3.1: Details for the different columns of the dataset

3.1 Subsampling

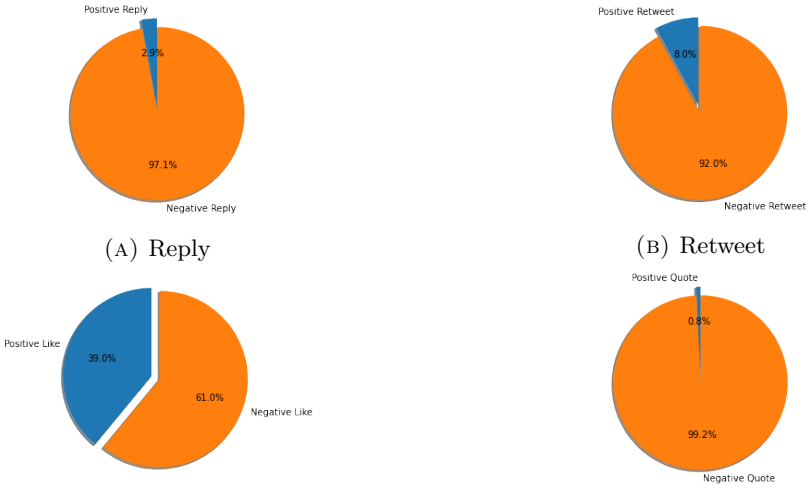
At a first stage, due to the huge size of the released dataset (from Week 1-3), we were forced to train and validate our models with a subset of samples of smaller size

($\simeq$ 4.11GB) filtering the dataset by the English language.

3.1.1 Distribution of the sub-sampled training dataset

Figure 3.1 shows the distribution of the positive and negative samples over the aforementioned sub-sampled training dataset.

We can clearly see that, except in the case of the Like, we have a *small class problem*. Indeed, the hardest problem to predict here is the Quote, since it has less than 1% of positive samples.

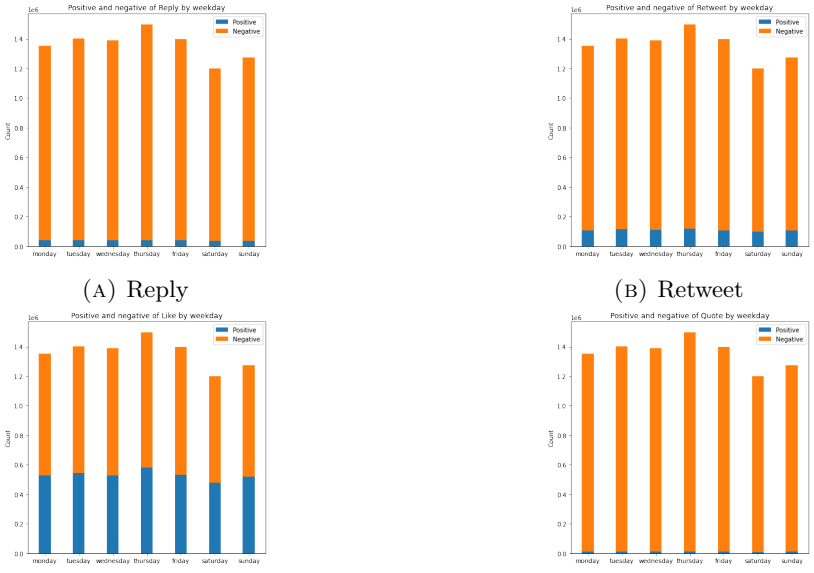


(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 3.1: Distribution of positive and negative samples for the 4 different targets in the sampled train set

Figure 3.2 shows the same distribution as in Figure 3.1 but grouped by the day of the week.



(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 3.2: Distribution of positive and negative samples for the 4 different targets grouped by the day of the week in the sampled train set

3.2 Validation data

Later on, validation data from Week 4 was released. As we have commented before, the size of this second dataset is much more manageable, being it divided into two files of $\simeq$ 6.5 GB each.

3.2.1 Distribution of the validation data

Figure 3.3 shows the distribution of the positive and negative samples over the released validation dataset.

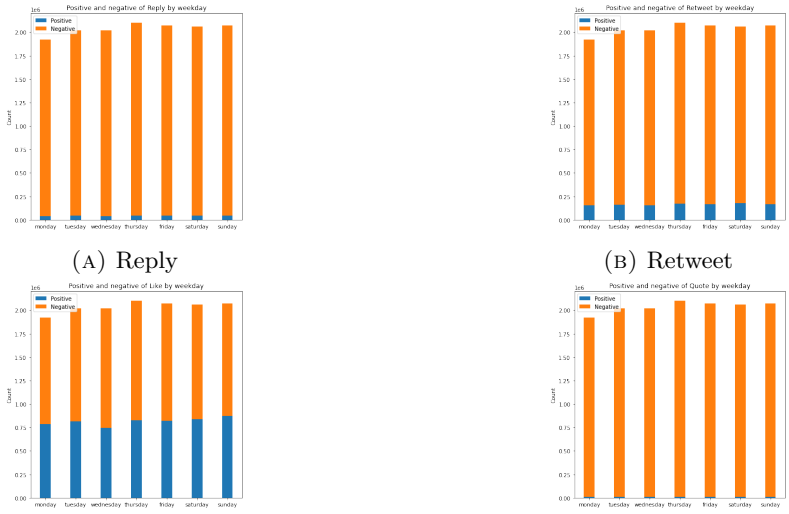
We can see here as well the *small class problem* for the Reply, Retweet and Quote.

(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 3.3: Distribution of positive and negative samples for the 4 different targets in the validation set

Furthermore, on Figure 3.4 we can see tha same distribution as in Figure 3.3 but grouped by the day of the week.



(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 3.4: Distribution of positive and negative samples for the 4 different targets grouped by the day of the week in the sampled train set

3.2.2 Train-validation-test split

As stated in Section 3.1, we have considered two different data sets in order to train and evaluate the models locally. In each of them we have performed two splits, the first one related to the training and test sets, and the second one to the training and validation sets:

- **Training subsampling:** In the training subsampling, we focused in obtaining the best performing model targeting Like engagement splitting the data set with *sklearn*'s `train_test_split` function set as

  ```
  df_train, df_test = train_test_split(df, test_size=0.5, random_state=17)
  ```

  with a fixed random state for reproducibility. After that, due to the limitations of our personal computers, we would also downsize the data sets with *pandas*' `sample` function

  ```
  df_train = df_train.sample(frac=0.4, replace=True, random_state=1)
  ```

  with again a fixed random state, and the fraction varying depending on the memory we had available and how much would the transformations applied to the data increase its size. Since the data is made up of tweets shown to reacting users over a period of time, we faced several problems when validating our models:

  - A time series is a series of data points ordered in time. In our case, tweets can be shown to different users at different points of time, meaning the tweet's popularity can fluctuate over time, and also, that the majority of tweets are repeated many times in the data set.
  - Due to the nature of social media, some topics can have a huge importance in recommendations one week and be completely irrelevant the next week. Thus, we decided to split our training and validation set making emphasis on its temporal properties, in particular, splitting according to a specific date that would return us a training set of $\simeq 80\%$ its original size, and a validation set of $\simeq 20\%$ of the original training set size.
  - That way we could see if our models would perform well in both predicting engagement over the same training period, and in short term engagement prediction.

- **Validation released set:** For the case of the validation released set, aside from the aforementioned splits we incorporated a new one in which we splitted training and test set with the condition that no tweet could be in both sets at the same time. This approach helped us establish a threshold for our models, since the data set in which they were going to be tested in the competition was of the same exact period of time, but we did not know how the splits had been done. If the splits had taken into account a similar approach our models would perform as expected and, if not, our models would perform better because they would have already seen the evaluated tweets at different moments of time.

Chapter 4

Model Architecture

In this chapter we explain how we have decided to structure our model and the reasoning behind it.

4.1 Limitations

In the realisation of this project, we faced several issues and limitations.

4.1.1 Local limitations

On one hand, all the trained models for the submissions have been trained on premise using our personal computers.

In particular, we have used an MSI GS66 Stealth 10SE-616XES (Intel Core i7-10875H, 32GB RAM, 1TB SSD, Nvidia RTX 2060-6GB) under a Windows 10. In addition to this, we have been using an ASUS ZenBook UX410 (Intel Core i7-7500U, 256 GB SSD, Mesa Intel HD Graphics 620 (KBL GT2)) under Pop!_OS. Furthermore, since the RAM of this last laptop was pretty low for some of the models, we used Google Colab with a specific setup that allowed for 25 GB of usable RAM instead of the usual 12.5 GB.

Our main limitation in order to process the data was the RAM memory. As we have mentioned earlier, we have been working over a sample in the training dataset for weeks 1 to 3 due to a lack of memory when processing the dataset.

4.1.2 Submission limitations

On the other hand, we have faced important issues and limitations when trying to place a submission on the challenge.

To be precise, they defined the way to submit results as follows. Instead of downloading the test dataset, transform it, evaluate it and submit the predictions from local, they have set up instances on Google Cloud Platform (e2-highmem-16 [10] instances) in which we have submitted our results and we have run our models from this computer on the cloud. Moreover, you can not access the Internet from this remote computer.

To be precise, the submission should contain the following:

- A path to a docker image from Docker Hub that should contain all the required packages you have been using in order to load, process and store the results.
- A `run` file (without extension) that is the entry point of the Docker image. I.e. this is the file that is executed once the instance is up.
- All the needed files and models in order to process and evaluate the test data.

The test data of the submission has approximately 10 millions rows.

Furthermore, the main limitations of this models have been the memory limit of 64GB of RAM and the execution time limit of 24 hours. Additionally, each user participating in the challenge had one chance to run a submission every 24 hours.

Finally, let us state that in the submission environment we did not had any GPU in order to speed up the data processing. For this reason, we had to dismiss using complex trained models such as sentiment analysis for feature creation over the test data.

4.2 Architecture

Since this is a team project, we needed to find the most optimal way to split the different tasks that at the same time would help us achieve the best possible results in the competition. We concluded that:

- Due to the size of the dataset and the complexity of the problem, we should focus on a simpler problem to start optimising. As one of the most spoken languages in the world, and also one of the most representative languages of the data set, we decided to focus our efforts in training and testing models with tweets only in English.
- Another way of simplifying the problem was to, instead of focusing in improving our models' performance in all four engagement prediction types, to pay more attention to one of them in particular. Thus, with Like being the target label with the best positive and negative samples proportion, we chose to optimise those predictions.
- Besides, our main intention was to extract all the possible information from the starting feature set. In order to do so, we split the starting set into text and non-text features: this would allow us to get the most out of each subset so in the end we could ensemble our best models and maximise the predictive power of the model we could achieve. With each of us focusing on a particular problem, we could end up with far more powerful models rather than if we worked on the same problem.

With this reasoning, Adrià worked in a Text Based Model, which is explained in Chapter 5, and Marcos worked in a Non-Text Based Model, which is explained in Chapter 6.

After working in both models with the subsampling from the training data released, when the validation labelled set was released we started working in a Mixed Model that Incorporated the best features from each of our best models. The results of the ensemble can be found in Chapter 7.

Chapter 5

Text Based Model

This Chapter is devoted to explain the features implemented in the classifier using only the encoded tweet and its timestamp. Moreover, we introduce and explain the problems and issues found during the implementation of those features, how we have solved the problems and finally we present the results obtained using only this model.

5.1 Feature Set

In this section we explain the set of features we have implemented and designed in order to take advantage of the text of the tweet to improve the metrics of our trained models.

The following list groups and explains the different features we have designed.

- **Counts**
  - **length_tweet**: counts the number of words the tweet has.
  - **num_unique_words**: counts the number of unique words the tweet has.
  - **number_of_punctuations**: counts the number of punctuation symbols are in `!"#$%&'()*+,-./:;<=>?@[\]^_‘{|}~`
  - **number_of_UNK**: counts the number of UNK tags the tweet has. See Section 2.3.1 for details.
  - **num_of_keywords**: counts the number of "keywords" present in the tweet. To be precise, it checks if a word has length greater than 2 characters and if the word is not in a list of common English words [11].
  - **num_of_keywords_bis**: this is a modification of the previous feature that only takes into account if the a word is not in the list of common English words.
- **Statistics about tokens and words**
  - **max_token_len**: computes the max between all the tokens in the BERT column.
  - **min_token_len** computes the min between all the tokens in the BERT column.
  - **mean_token_len** computes the mean between all the tokens in the BERT column.
  - **token_len_2**: counts the number of tokens that has length 2.
  - **token_len_3**: counts the number of tokens that has length 3.
  - **token_len_4**: counts the number of tokens that has length 4.

  - **token_len_5**: counts the number of tokens that has length 5.
  - **token_len_6**: counts the number of tokens that has length 6.
  - **mean_token_value_as_int**: mean value of the tokens of a tweet as integers (e.g. '106' as 106).
  - **max_token_value_as_int**: maximum value of the tokens of a tweet as integers.
  - **min_token_value_as_int**: minimum value of the tokens of a tweet as integers.
  - **mean_token_char_value_as_int**: mean of the tokens of a tweet as a sum of the numbers that form them (e.g. transforming '106' to 7).
  - **max_token_char_value_as_int**: maximum value of the tokens of a tweet as a sum of the numbers that form them.
  - **min_token_char_value_as_int**: minimum value of the tokens of a tweet as a sum of the numbers that form them.
  - **huge_words**: following the procedure of transforming the tokens of a tweet to the sum of the numbers that form them, number of tokens the transformation of which adds up to a larger value than 35.
  - **big_words**: following the procedure of transforming the tokens of a tweet to the sum of the numbers that form them, number of tokens the transformation of which adds up to a value larger than or equal to 20 but smaller than 35.
  - **medium_words**: following the procedure of transforming the tokens of a tweet to the sum of the numbers that form them, number of tokens the transformation of which adds up to a value larger than or equal to 10 but smaller than 20.
  - **small_words**: following the procedure of transforming the tokens of a tweet to the sum of the numbers that form them, number of tokens the transformation of which adds up to a value smaller than 10.

- **Flags**
  - **contains_worst_words**: flag to check if the tweet contains one of the worst words. Find more information on [12].
  - **contains_best_words**: flag that checks if the tweet contains some word that is very likely to have a retweet. Find more information on [12].
  - **contains_reply**: flag that checks if the tweet contains the char '@'.

- **Ratios**
  - **ratio_unique_words**: `num_unique_words` divided by `length_tweet`.
  - **ratio_of_UNK**: `number_of_UNK` divided by `tweet_length`.
  - **ratio_of_punctuations**: `num_of_punctuations` divided by `length_tweet`.

5.2 Problems and limitations encountered

The features explained in the previous section were the ones that we used in the final submission.

Due to the limitations explained in Section 4.1, we were forced to optimise the amount of resources the creation of these features needed, especially *RAM usage*.

To be precise, originally we had another additional column with the decoded text using the BERT tokenizer explained above. However, holding on memory all this information was very demanding. So, in order to relax the memory requirements, we have implemented the aforementioned features decoding the tokens *on-the-fly*. Moreover, those features that needed to compare two different words are implemented in a way that we compare the two items in the token format. So, we do not need to transform them into a string format and then compare it wasting memory and computation time.

Finally, we have implemented an additional feature that we could not include in the final submission due to lack of RAM memory in the submission environment.

- `tf_idf`: computes the sum of the TF-IDF [13] algorithm, sum its columns and normalises the result using the MinMaxScaler [14].

5.3 Results

In this section we present the results obtained using only text features to predict the 4 different targets proposed in this challenge.

In particular, we have evaluated the trained models following the instructions explained in Section 3.2.2.

5.3.1 Evolution of the text models

Figure 5.1 shows the evolution of the results obtained on the text model.

It is important to state that models from v1 to v5 where trained using the sub-sampling training data (See Section 3.1) while the rest of the reported scores were obtained from the released validation set (See Section 3.2).

Nonetheless, all of them were trained splitting the data using sklearn's `train_test_split` (see Section 3.2.2). We can see a slow improvement with the sampling of the training set.

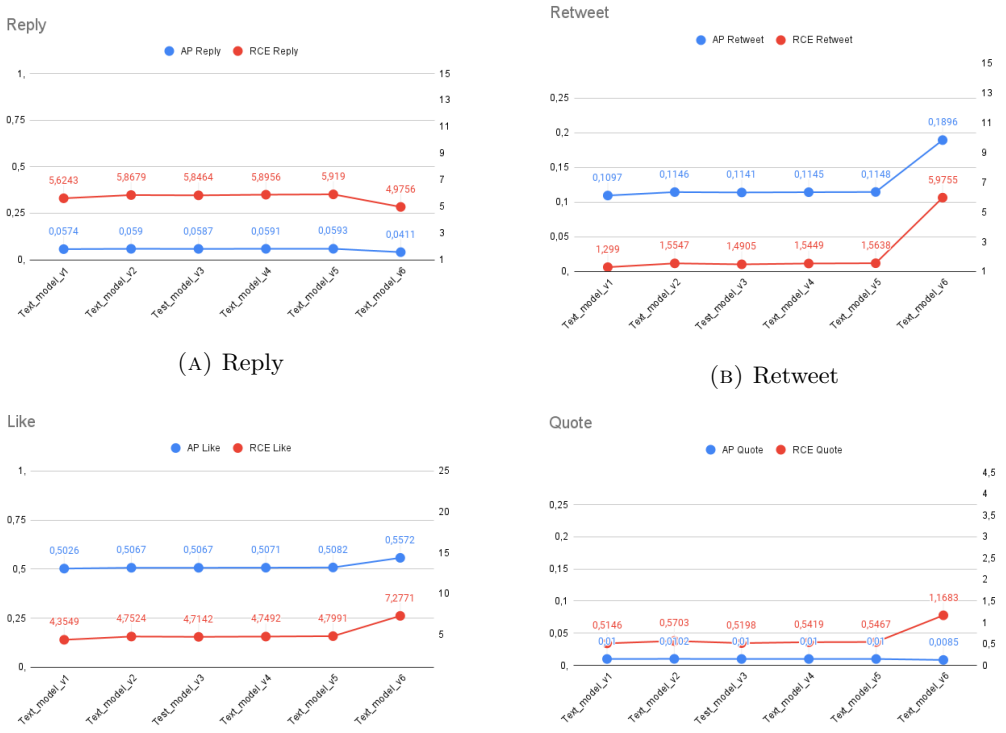
Finally, we can see that the version 6 of the text models (the one trained using the validation dataset) with the addition of some additional text features we can see a huge improvement in the case of the Retweet, a moderate improvement in Like and Quote models and a small decrease in the score of the Reply model. However, we have applied hyper-parametrization tuning over the Reply model in order to adjust it and minimise the loss of score.

5.3.2 Best text model

The results of the best Text Model can be found on Table 5.1.

	AP Reply	RCE Reply	AP Retweet	RCE Retweet	AP Like	RCE Like	AP Quote	RCE Quote
Text model v6	0,0411	4,9756	0,1896	5,9755	0,5572	7,2771	0,0085	1,1683

TABLE 5.1: Results for the best text models implemented.



(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 5.1: Evolution of the text models.

5.3.3 Feature importance of the best text models

The feature importance of the best text model for the four different targets can be found in Appendix B (Figure B.1 to B.4).

We can see that the best feature for all the models (except the Like) is the `mean_word_len`. While the best feature for the Like model is the `ratio_of_keywords`.

After the top ones we find usually that the ratios are really relevant in the prediction and that `min_word_len`, `contains_worst_words` and `contains_best_words` are usually the worst ones.

Chapter 6

Non-Text Based Model

In this chapter we discuss the features and models created from the remaining ones of the initial set i.e., all of them except for the tweet content. We also comment on the different limitations we have had to deal with and show the evolution of the models' performance.

6.1 Feature Set

From the initial set of features we created some more of them, here we describe the ones we ended up using on our final models as they helped the model predict and score best on all metrics.

- **Ratios**
  - `EWUF_follow_ratio`: ratio between people number of people followed and number of followers of the engaged with user.
  - `EWUF_follow_ratio`: ratio between number of people followed and number of followers of the engaging user.
  - `cross_ratio1`: ratio between number of people followed by the engaged with user and followers of the engaging users.
  - `cross_ratio2`: ratio between number of people followed by the engaging user and followers of the engaged with user.
  - `cross_follower_ratio`: ratio between number of followers of the engaged with user and number of followers of the engaging user.
  - `cross_following_ratio`: ratio between number of people followed by the engaged with user and number of people followed by the engaging user.
  - `ratioxratio`: product between the ratio of followers of the engaged with user and the ratio of followers of the engaging user.
- **Features from timestamps**
  - `hours`: hour of the creation of the tweet.
  - `weekday`: weekday of the creation of the tweet.
  - `hoursweekday`: time of the week of the creation of the tweet (value from 0 to 1, as a proportion).
  - `EUF_time_from_creation`: time in minutes since the creation of the account of the engaged with user.
  - `EWUF_time_from_creation`: time in minutes since the creation of the account of the engaged with user.

  - **`EUF_time_from_creation`**: time in minutes since the creation of the account of the engaging user.
  - **`diff_from_creation`**: difference in minutes between the time of creation of the account of the engaging user and the creation of the account of the engaged with user.

- **Number of elements**
  - **`numHashtags`**: number of hashtags inside the tweet.
  - **`numLinks`**: number of links inside the tweet.
  - **`numDomains`**: number of domains inside the tweet.
  - **`numMedia`**: number of media inside the tweet.
  - **`numVideo`**: number of videos inside the tweet.
  - **`numPhoto`**: number of images inside the tweet.
  - **`numGIF`**: number of GIFs inside the tweet.

- **Top categorical data**
  - **`topHashtags`**: encoding of whether the tweet contains at least one of the N most counted hashtags or not.
  - **`topHashtagsInternet`**: encoding of whether the tweet contains at least one of the hashtags stored in a list that contains the most useful hashtags to better engagement according to some sites (see [12].).
  - **`topLinks`**: encoding of whether the tweet contains at least one of the N most counted links or not.
  - **`topDomains`**: encoding of whether the tweet contains at least one of the N most counted domains or not.
  - **`topLanguage`**: encoding of whether the tweet contains at least one of the N most counted languages or not.
  - **`nottopLanguage`**: encoding of whether the tweet contains at least one of the N least counted languages or not.

- **Relevance**
  - **`EUF_verified`**: whether the engaging user is verified or not
  - **`EWUF_verified`**: whether the engaged with user is verified or not.
  - **`relevance`**: whether both interacting users are verified or not.
  - **`engagee_follows_engager`**: whether the author of the tweet follows the reactor or not.

Aside from the aforementioned features, the models in their distinct versions incorporated either one-hot or target encoding of some of the initial features, such as **`type`** or **`media`**.

6.2 Problems and limitations encountered

The features exposed in Section 6.1 are the ones that have been used as a part of the final mixed model of the competition, but a number of them were discarded before due to different types of limitations:

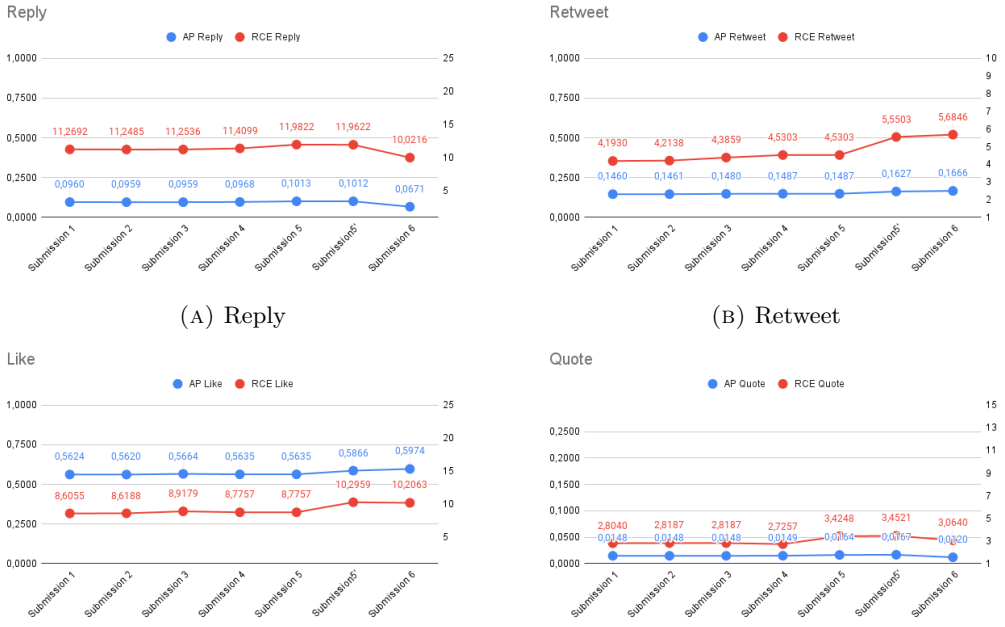
- We tried to apply a one-hot type of encoding to some categorical values such as **`hashtags`**, **`domains`** and **`links`**. Our intention was to create features for the most frequent hashtags and domains, but even if we could make it work for the training of the models in our computers, the size of the competition test set would make our submissions throw a memory usage error, exceeding the 64 GB limitation.
- In a similar fashion we tried performing the same transformation on the hashtags from **`topHashtagsInternet`**, but we encountered the same problem as before.
- Another transformation we experimented with was standardise the data, but we did not see any significant improvement in performance and we ended up dismissing it.
- Finally, we also dismissed the use of target encoding in our final submissions for most of the categorical variables. Although it was useful that the competition test set was from the same period of time than the validation set released; it did not improve significantly our models' performance compared to one-hot coding and was significantly slower. Since we were focusing on reducing computation time of the text features, we chose not to incorporate that method.

6.3 Results

In this section we present the evolution of the performance of the 4 type of engagement models built from the features explained in Section 6.1 and Section 6.2.

6.3.1 Evolution of the models

In particular, Figure 6.1 displays the results obtained from the different models. From Submission 1 to Submission 5', the models were trained with the subsampling training data, while Submission 6 was trained with the released validation set, all of them splitting the data using sklearn's **`train_test_split`** (see Section 3.2.2). As it can be seen, all of them improved gradually when working with the subsampling of the training set, and both Reply and Quote did worse with the validation set and Like and Retweet saw an improvement with the change. However, this could easily be solved with some parameter tuning, which we actually did after mixing models, so the decay does not need to be considered as something significant.



(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 6.1: Evolution of the Non-Text Based Features models.

6.3.2 Best non-text based features model

The results of the non-text based model can be found on Table 6.1. These results correspond to Submission 5'. As it can be seen, in general AP is complicated to predict and our model fails to obtain significant results. RCE is relatively high in Reply and Like when comparing to Retweet and Quote. It seems only natural that we obtain the worst results for Quote, since it constitutes the most difficult problem, and that we obtain optimal results for both AP and RCE in Like, which we were specifically targeting.

	AP Reply	RCE Reply	AP Retweet	RCE Retweet	AP Like	RCE Like	AP Quote	RCE Quote
Submission 5'	0,1012	11,9622	0,1627	5,5503	0,5866	10,2959	0,0167	3,452

TABLE 6.1: Results for the best non-text features model implemented.

6.3.3 Feature importance of the best non-text based models

The feature importance plot of all four engagement types of the best Non-text Based Model can be seen in Appendix B. For all four models we observe a dominance of ratios and features created from timestamps, specially the ones encoding the age of the account, and the ones encoding information about the point in time of creation of the tweet during the day or during the week. Number of elements features from **`Hashtags`**, **`media`** and **`Domains`** also seem to contribute to good predictions, in addition to the one-hot encoding features from **`type`**. Finally, relevance features seem to have low predictive power in all models, probably due to verified users being a minority in the data set.

Chapter 7

Results

This Chapter is devoted to explain the final mixed implemented model used in the final submission of the challenge as well as the results obtained both locally and in the submission environment.

7.1 Evaluation

The submissions are evaluated against two metrics: Average Precision (AP) and Relative Cross-Entropy (RCE). Both are implemented in a **metrics_recsys_2021.py** file, available in the official page of the challenge.

7.1.1 AP

The first of the metrics used to evaluate the predictions resulting from the models is the Average Precision. The AP is a useful metric when dealing with imbalanced classes, and as we have seen when taking a look at the dataset it fits this problem perfectly. It is defined as the area under the Precision-Recall curve, which is the curve that shows the trade-off between precision and recall for different thresholds:

- The binary predictions of a system can fall into 4 categories: true positive if the sample was predicted as positive and its label was indeed positive, false positive if the sample was predicted as negative but its label was positive, false negative if the sample was predicted as positive and its label was negative, and true negative if both the label and prediction were negative.
- Precision is defined as the number of true positives over the number of total real positives (true positives plus the number of false positives):

$$P = \frac{T_p}{T_p + F_p}$$

- Recall is defined as the number of true positives over the number of total predicted positives (true positives plus the number of false negatives):

$$R = \frac{T_p}{T_p + F_n}$$

- Varying the threshold of a classifier, different values of Precision will be obtained, while Recall does not depend on the threshold. The relationship between both variables can be visualised in a plot featuring the Precision-Recall curve.

The Average Precision condenses the results of the Precision-Recall curve, being it the weighted mean of the Precision achieved at each threshold with the increase in Recall from the previous threshold as weight:

$$AP = \sum_n P_n(R_n - R_{n-1})$$

In this case, the sklearn implementation of the AP is used to evaluate the submitted models' predictions.

7.1.2 RCE

For this problem, the second one of the metrics they use to evaluate the resulting predictions is the Relative Cross-Entropy. It is a variation of the Cross-Entropy or log loss function, commonly used to measure the performance of a model predicting a probability value between 0 and 1, which is defined as

$$CE = -\sum_n y_n log(p_n)$$

for n classes where y_n are the labels and p_n the predicted probabilities. For the case of binary classification, and averaging along N the samples,

$$CE = -\frac{1}{N}\sum_n [y_n \log(p_n) - (1 - y_n)\log(1 - p_n)]$$

would be the correct expression. At the same time, with a similar structure, the Strawman Cross-Entropy (SCE) can be defined as

$$SCE = -\frac{1}{N}\sum_n [y_n \log(ctr) - (1 - y_n)\log(1 - ctr)]$$

where ctr is the constant true rate i.e., the fraction of positive labels. With both the CE and the SCE, the RCE can be expressed as

$$RCE = 100(1 - \frac{CE}{SCE}).$$

7.1.3 Fairness

One of the innovations that this year's RecSys Challenge brings, is its approach to the ranking and evaluation system. Although the most frequent methods focus on accuracy and the ranking of the results, this year they wanted to emphasise the concept of *fairness*. As they state, their objective was to:

- introduce the concept in an easy way,
- but at the same time making it a concrete and real problem,
- without adding any extra data external to the dataset.

In order to achieve so, they suggest using as a popularity-based metric the number of followers the author of a tweet has. Doing so, they want to reflect on how the popularity of a user should not influence how good the recommendations of the system are, and definitely should not produce worse recommendations for a less popular user than for a popular one. Specifically, what they do is dividing the authors of tweets

into 5 categories according to their number of followers (according to the quantiles of said feature) and compute the *RCE* and *AP* across each of the groups. Averaging of those metrics along the groups ends up providing the final score. To help participants check how do their models perform in this aspect, they do in fact provide them with a **fairness.py** with a coded implementation.

In Table 7.1 we can see how our model predicts with similar success for the first three popularity groups, with no significant difference, but does better for groups 4 and 5. It could be argued that we are providing fair recommendations for the first groups, but unfair recommendations when including groups 4 and 5, since having more followers apparently benefits them.

	AP (RT)	RCE (RT)	AP (NRT)	RCE (NRT)
Group 1	0,5934	10,8526	0,5927	10,8218
Group 2	0,5759	9,5229	0,5676	9,5388
Group 3	0,5919	10,1866	0,5909	10,0789
Group 4	0,6273	11,5873	0,6275	11,6936
Group 5	0,6754	15,4138	0,66042	14,3485

TABLE 7.1: AP and RCE for Like engagement and for each popularity group in both Mixed Model (RT) and Mixed Model (NRT).

7.2 Mixed features

In order to improve the results from the two separated implemented models, we have created another model combining the features from both sides: text features and non-text features under the same type of classifier (LightGBM).

In particular, we have trained it using the following parameters:

```
parameters = {
    'max_depth': 10,
    'objective':'cross_entropy',
    'metric': 'None',
    'is_unbalance': 'true',
    'boosting': 'gbdt',
    'num_leaves': 45,
    'learning_rate': 0.05
}
```

In this setting, we limit the max depth of the trees to 10, we configure the objective as the cross entropy function, we set to true the parameters that indicates to the model that the data is unbalanced and we set to 45 the maximum number of splits each node can make.

7.3 Final Results

As stated in Section 3.1, the easiest target to predict was the Like, since the distribution positive and negative samples is more or less equally distributed.

For this reason, and due to our lack of resources, we decided to focus our efforts in try to design features and models in order to aim to perform in this target really well.

Table 7.2 shows the results obtained with the final model, both running predictions locally and submitting it to the evaluation environment of the RecSys Challenge.

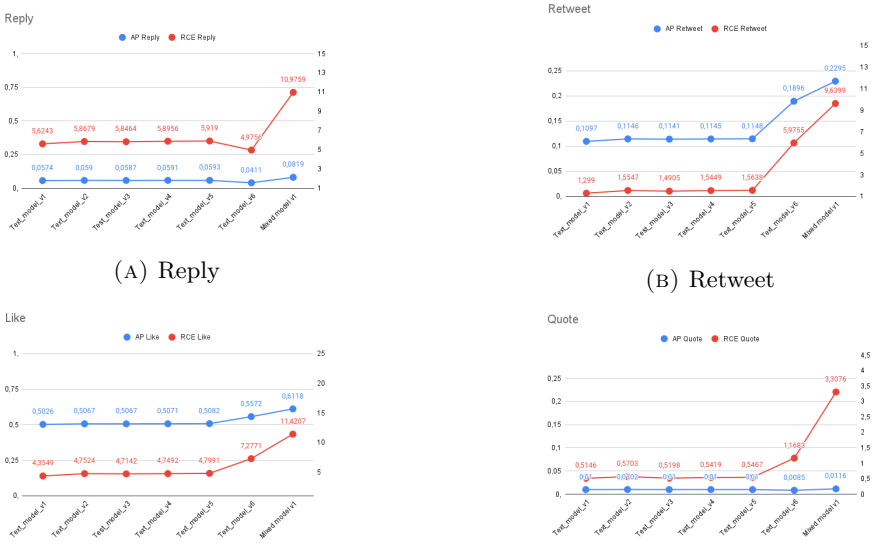
	AP Reply	RCE Reply	AP Retweet	RCE Retweet	AP Like	RCE Like	AP Quote	RCE Quote
Mixed Model Locally (NRT)	0,0819	10,9759	0,2295	9,6399	0,6118	11,4207	0,0116	3,3076
Mixed Model Locally (RT)	0,0817	11,1817	0,2311	9,7273	0,6165	11,6347	0,0119	3,3074
Mixed Model Submission	0,0969	11,9062	0,2633	11,6541	0,6015	11,3440	0,0132	3,6431

TABLE 7.2: Results of the final model, where RT refers to the results training and testing in a split that allows repeated tweets, and NTR stands for not repeated tweets.

As aforementioned in Section 3.2.2, we trained and tested models following two different approaches, one in which we performed a split in which tweets could be repeated in both the training and the test set (RT), and one in which we did not allow for tweets to be in both subsets. The first approach would then serve as an estimation of the best result (higher bound) we could achieve when submitting and evaluating in the competition data set, and the second one as the worst result we could achieve (lower bound). As Table 7.2 shows, both Reply, Retweet and Quote exceeded our expectations, while Like was the engagement type that we did best estimate. It was something to be expected since it was the target that we knew better and around which our models revolved most of the time.

7.3.1 Comparison with text only and non-text only models

In Figure 7.1 and 7.2 we want to reflect on the spike in performance that the models manifested after mixing both models and training with the validation set with the not repeating tweets approach (NTR). As it can be seen in Figure 7.1 the performance of all text models increases drastically. Although all non-text models also see an improvement, Retweet is the most benefited, in particular its *RCE*.



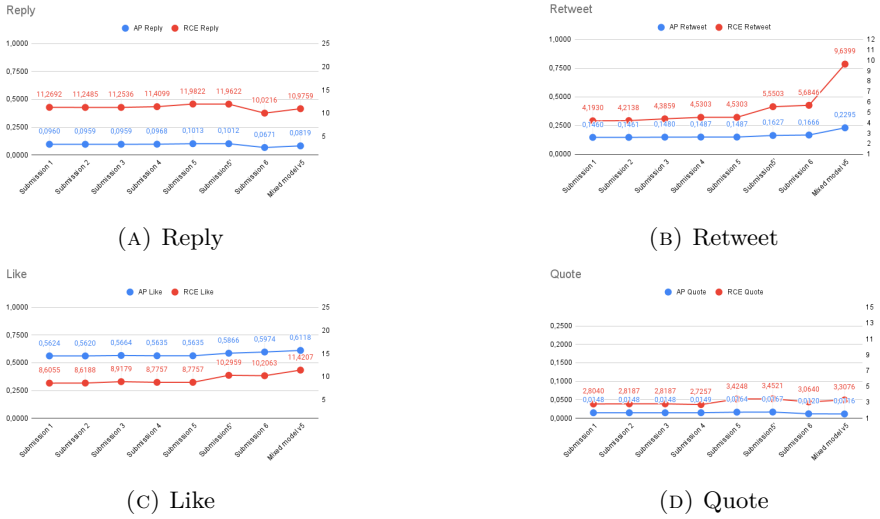
(A) Reply

(B) Retweet

(C) Like

(D) Quote

FIGURE 7.1: Transition of the Text Based Model into the Mixed Model.



(A) Reply (B) Retweet

(C) Like (D) Quote

FIGURE 7.2: Transition of the Non-Text Based Model into the Mixed Model.

7.3.2 Feature importance of the Mixed Model

The feature importance plot of all four engagement types of both Mixed Model (RT) and Mixed Model (NRT) can be seen in Appendix B.

From the plots, it becomes apparent that both versions of Mixed Model make great use of timestamp features, such as the time of creation of the accounts or the time of creation of the tweet. They also value greatly ratios relative to both the author of the tweet and the reactor followers, and to the content of the tweet, like the ratio of punctuations, the ratio of keywords and the ratio of unique words. Some of the most valued features overall are:

- Timestamp features: **`EWUF_time_from_creation`**, **`EUF_time_from_creation`**, **`diff_from_creation`**, **`hoursweekday`**, **`hours`**.
- Ratios: **`EWUF_follow_ratio`**, **`cross_following_ratio`**, **`cross_ratio1`**, **`cross_ratio2`**, **`ratio_of_keywords`**, **`ratio_of_punctuations`**.

On a second tier, there are features describing the overall properties of the tweet's content, as the number of tokens of a specific length, the features classifying tokens into four big categories (huge, big, medium and small) and the features counting the number of hashtags or media elements in the tweet:

- Content of the tweet: **`token_len_3`**, **`token_len_4`**, **`token_len_5`**, **`token_len_6`**, **`length_tweet`**, **`contains_reply`**.
- Number of elements: **`num_of_keywords`**, **`numHashtags`**, **`numMedia`**, **`numDomains`**, **`num_of_keywords`**, **`kinda_num_of_keywords`**, **`number_of_UNK`**

Finally, in the last third of the classification, there are features such as the ones indicating if the tweet contains some specific words that should be good or bad towards engagement, verified features and features indicating whether or not the tweet contains one of the most frequent hashtags, domains or links. Moreover, features counting the number of specific media elements, as the number of GIFs or videos do not seem to contribute much neither:

- Verified: `EUF_verified`, `EWUF_verified`.
- Contains words: `contains_best_words`, `contains_worst_words`.
- 'Top' Features: `topHashtagsInternet`,`topHashtags`, `nottopLanguage`, `topLinks`, `topDomains`.
- Number of media elements: `numPhoto`, `numVideo`, `numGIF`.
- Intrincate token properties: `min_token_char_value_as_int`, `min_token_value_as_int`, `max_token_len`.

7.4 Competition Leaderboard

June 23rd was the last day of the challenge to submit solutions to the proposed problem. Some days later they updated the leaderboard and the results can be seen on Figure 7.3.

We have obtained the 14th place in this competition. However, targeting Like engagement we are in a better overall position: our models AP can compete with the 7th submission, and RCE also lays between the 7th and 9th position.

Competition Leaderboard

The leaderboard shows the latest submission for each user, to see the full list of submissions please click here.

Rank	User	Method	AP Retweet	RCE Retweet	AP Reply	RCE Reply	AP Like	RCE Like	AP RT with comment
1	bennys101010	nvidia_rapidsai_final_ensemble_v2	0.4614	29.5127	0.2649	26.6123	0.7216	23.6124	0.0692
2	_mdaniluk	Synerise_v1	0.4514	28.5222	0.2559	25.7468	0.7046	22.0994	0.0662
3	s_jianing	LAYER6_AI	0.4317	27.4239	0.2490	25.3526	0.6836	19.8578	0.0660
4	SamueleMeta	test_lightgbm	0.4060	25.0928	0.2118	22.6491	0.6636	17.9193	0.0520
5	perecasxiru	final1	0.3940	24.0142	0.2077	22.1539	0.6559	16.9609	0.0459
6	sol142820574	stacking_gnn	0.3846	23.5012	0.2018	20.7757	0.6056	11.8858	0.0503
7	zhwsmile	version3-20210623-014546	0.3849	23.2816	0.1863	16.9172	0.6031	8.6238	0.0534
8	angrypark_	final_guess_2	0.3521	20.8042	0.1801	19.5385	0.6017	11.5249	0.0457
9	PanikaTM	Beta-1_1	0.3591	20.7594	0.1868	20.3206	0.5806	7.1209	0.0470
10	alexkrauck	User_Based_XGB_V9	0.3503	18.2654	0.1786	18.6721	0.5794	8.0251	0.0406
11	LivesInAnalogia	version_8	0.3279	18.4443	0.1742	19.1013	0.5743	7.8972	0.0366
12	MonicaL44466660	xgb_te_simple	0.3247	17.3254	0.1340	15.8511	0.5902	11.3813	0.0275
13	EricAnd34551278	MergedFeatures_Final	0.3281	14.3470	0.1410	11.9674	0.5543	5.5791	0.0345
14	mmorbla	mix_competition	0.2633	11.6541	0.0969	11.9062	0.6015	11.3440	0.0132
15	0in_kwon	DGANN	0.2367	9.0863	0.0917	11.0405	0.5794	9.3288	0.0132
16	swaroooooo	old_xgb	0.2036	6.9854	0.0658	3.9229	0.5404	6.8254	0.0080
17	reza1248	l32	0.2533	-150.7222	0.0849	-203.0445	0.4960	-192.5780	0.0190
18	hyez_o	recline_3	0.1131	2.2789	0.0592	5.4937	0.5277	6.2110	0.0076
19	IoannisPanagi10	Random_Model_Submission	0.0860	-242.1735	0.0282	-725.8259	0.3872	-49.9430	0.0057
20	alykhantejani	random_predictions	0.0859	-242.6079	0.0285	-719.3697	0.3868	-49.9752	0.0058

FIGURE 7.3: Result of our team in the RecSys2021 Challenge Leaderboard.

The reader can find the complete list of participants on `https://recsys-twitter.com/competition_leaderboard/latest`.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

In this project we have created a model capable of performing reasonably well in the proposed challenge. We consider we have accomplished most of our established objectives from Section 1.3.1:

- We have been able to extract interesting and useful features from timestamp, categorical and numerical features related to different aspects of the problem. We have explored a variety of different feature engineering techniques, such as binning, standardisation, the creation of interaction features and some types of categorical variables encoding that allowed us to enhance our model's performance. We faced many problems due to working with a real-world data set that helped us learn and overcome barriers we could not have faced otherwise.
- We have also developed features from the tweets' content, with techniques learnt in courses such as Machine Learning, Recommender Systems and Natural Language Processing. We have managed to solve technical difficulties of working with limited environments, both in our personal computers and in the submission server, which restricted our options.
- We have created an optimal performing model based on non-text features, and successfully tuned it using State-of-the-Art techniques, using binary *RCE* as a loss function and trying to avoid overfitting through early-stopping and tuning regularisation parameters, the number of leaves and the maximum depth of the models.
- We have built a State-of-the-Art model using text features and introduced minor and major changes to successfully decrease its required computation time and memory usage. We have been able to extract a significant predictive power from difficult to work feature that has had a big impact in the resulting final model.
- We have been able to merge our crafted models into one better performing and improved mixed model, successfully grouping our best working features.
- We have tested our models in a top international competition in the field and have achieved significant results according to our expectations and limitations, performing competitively in Like engagement which was our main objective. We managed to climb into top positions after a difficult start with poor submissions, improving with every model and familiarising ourselves with the data set.
- We have created fair models in order to fulfil the *fairness* requirements proposed in the challenge.

8.2 Future Work

Since the competition was only opened for a limited amount of time in addition to the technical limitations and difficulties suffered, we could not implement all the things we would liked in order to improve the performance of the models.

For instance,

- Implement and apply a sentiment analysis model to the decoded text in order to have a better understanding of what is going on in the tweet could improve the metrics proposed.
- Use the BERT model in more profound and wide aspects, analysing the most present languages in the data set and extracting information about the factors that influence engagement in each of them.
- Analyse better the importance of each of our created features and encodings applied, in order to combine them in a more powerful way. Since we could not spend much time studying minor changes in every model, we did not exploit some facets to its maximum capabilities.
- Work specifically with each engagement type. Due to time limitations we had to focus on only one type of engagement and we would have pr eferred to devote more resources to improving specifically the other three models.
- Implement a Deep Neural Network in order to perform feature extraction and use the last dense layer as the input for a further classifier.

Appendix A

Source code

The reader can find the source code with the implementation explained in this document in

https://github.com/adry26/RecSys2021.

There are three files uploaded to the github repository, which are:

- `Mixed_Model.ipynb`: a notebook with all the code necessary to load, train and test the final Mixed Model (NRT) and (RT), along with code used to perform feature importance plots.
- `Text_Based_Model.ipynb`: a notebook with all the needed code to load, train and evaluate the final Text Model. In addition to this, it provides the needed code to show the feature importance of the models and save and load trained models.
- `Non_Text_Based_Model.ipynb`: a notebook with all the code necessary to load, train and test the final Non-Text Based Model, along with code used to perform feature importance plots. It can be run sequentially or starting at either Train or Test sections.

Appendix B

Feature importance of Mixed Model

Here you will find the feature importance plots of the Text Based Model, the Non-Text Based Model, and the Mixed Model, both the one trained and tested without restrictions on repeated tweets, and the one with the restriction that tweets cannot be in both subsets.

B.1 Feature importance of the Text Based Model

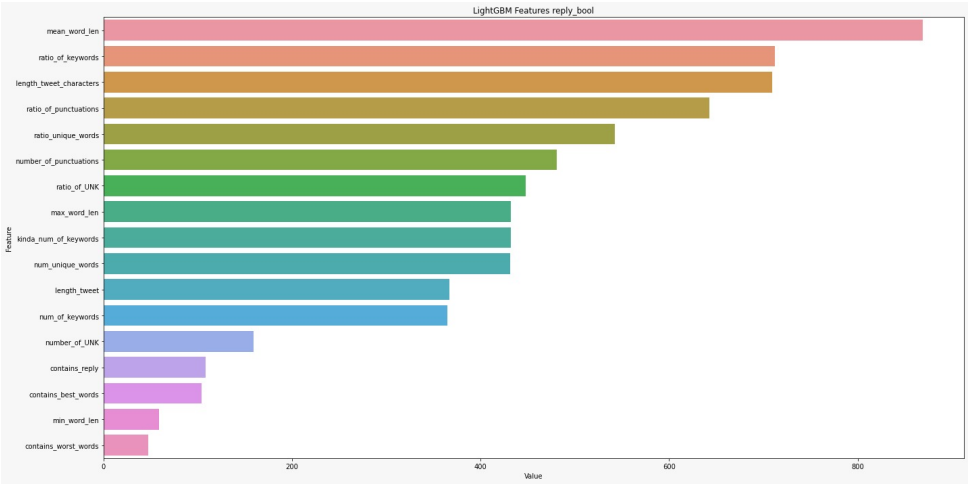


FIGURE B.1: Feature importance for the Reply engagement in the Text Based Model.

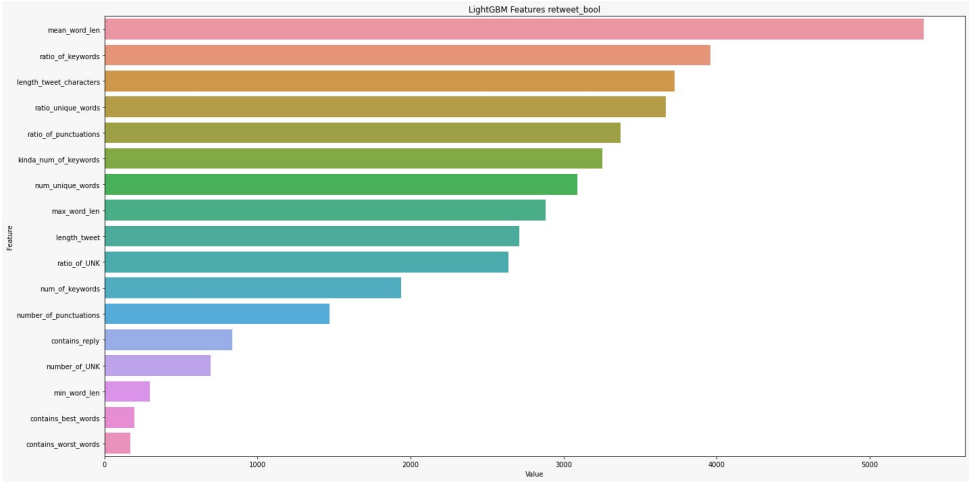


FIGURE B.2: Feature importance for the Retweet engagement in the Text Based Model.

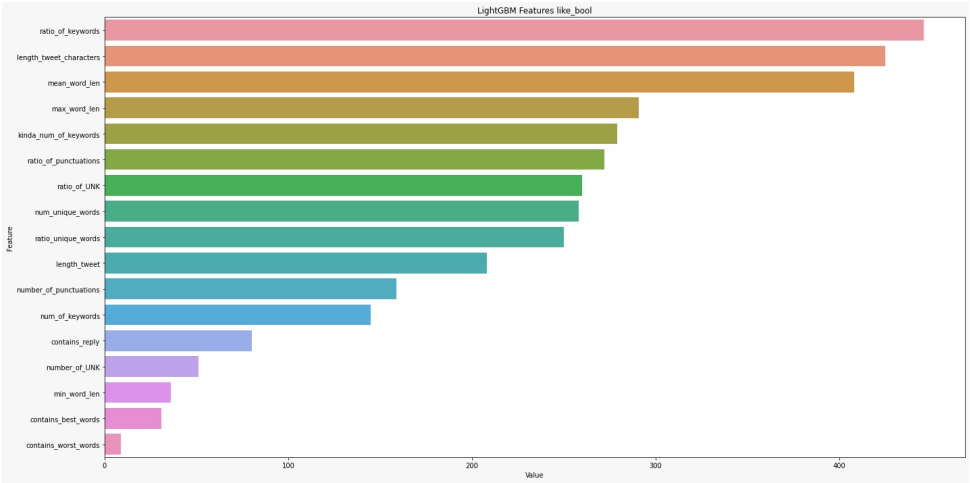


FIGURE B.3: Feature importance for the Like engagement in the Text Based Model.

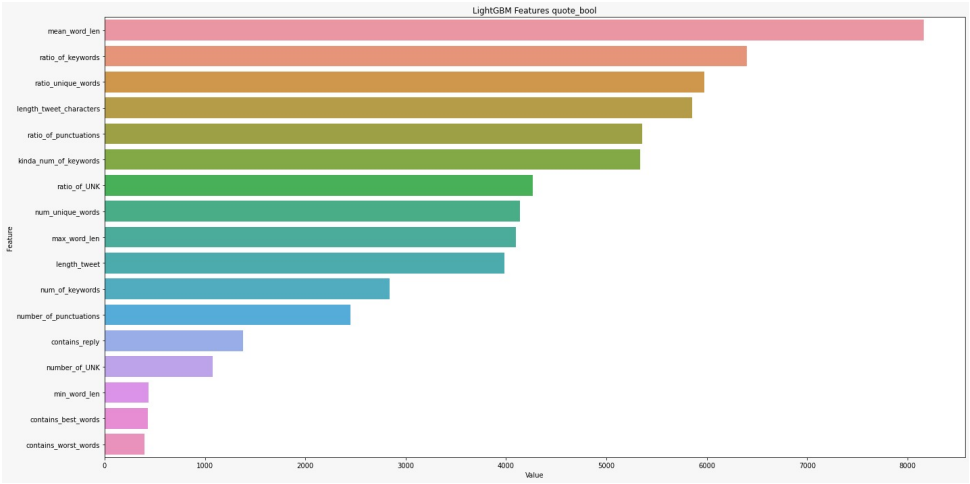


FIGURE B.4: Feature importance for the Quote engagement in the Text Based Model.

B.2 Feature importance of the Non-Text Based Model

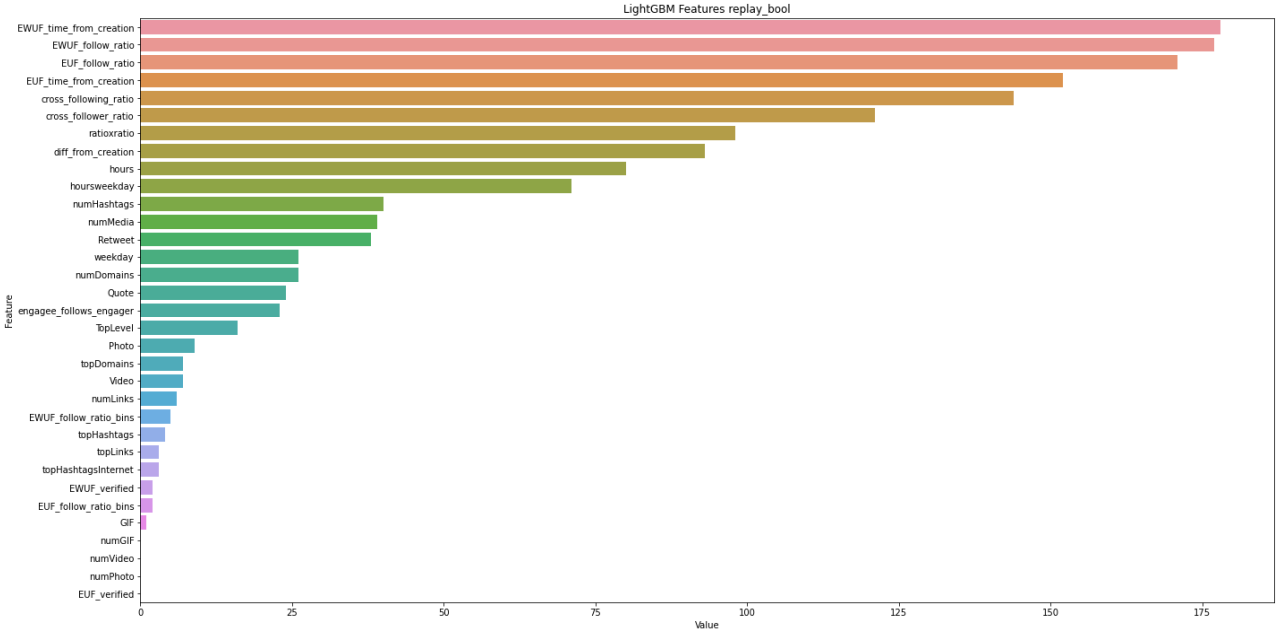


FIGURE B.5: Feature importance for the Reply engagement in the Non-Text Based Model.

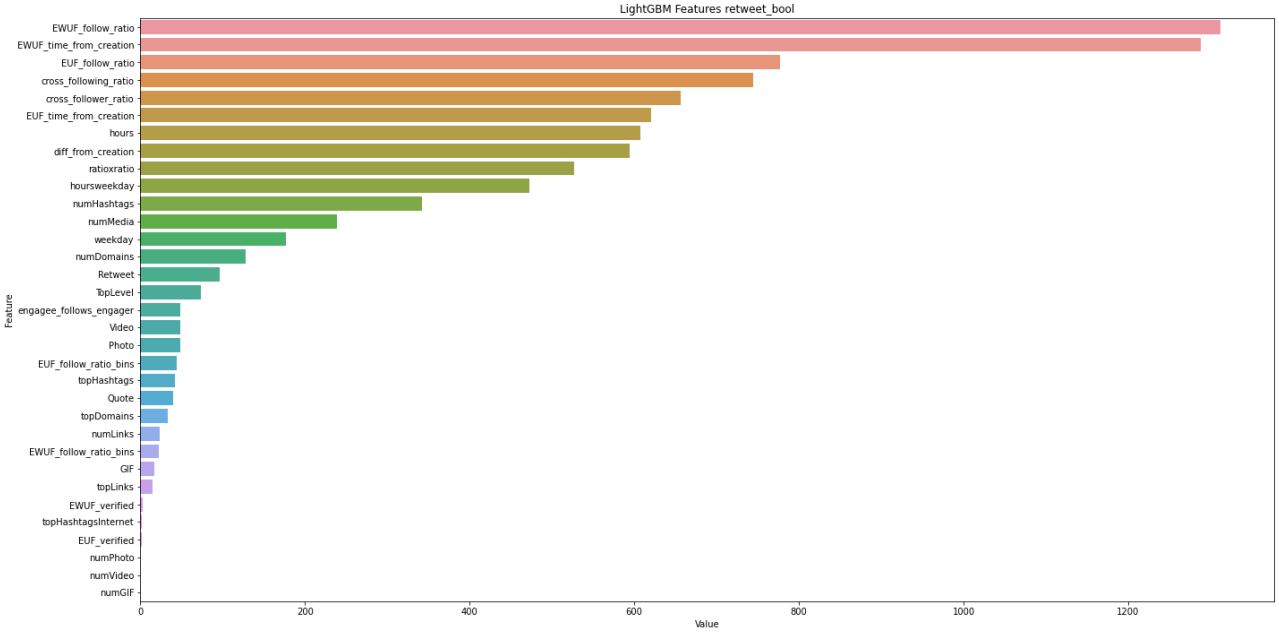


FIGURE B.6: Feature importance for the Retweet engagement in the Non-Text Based Model.

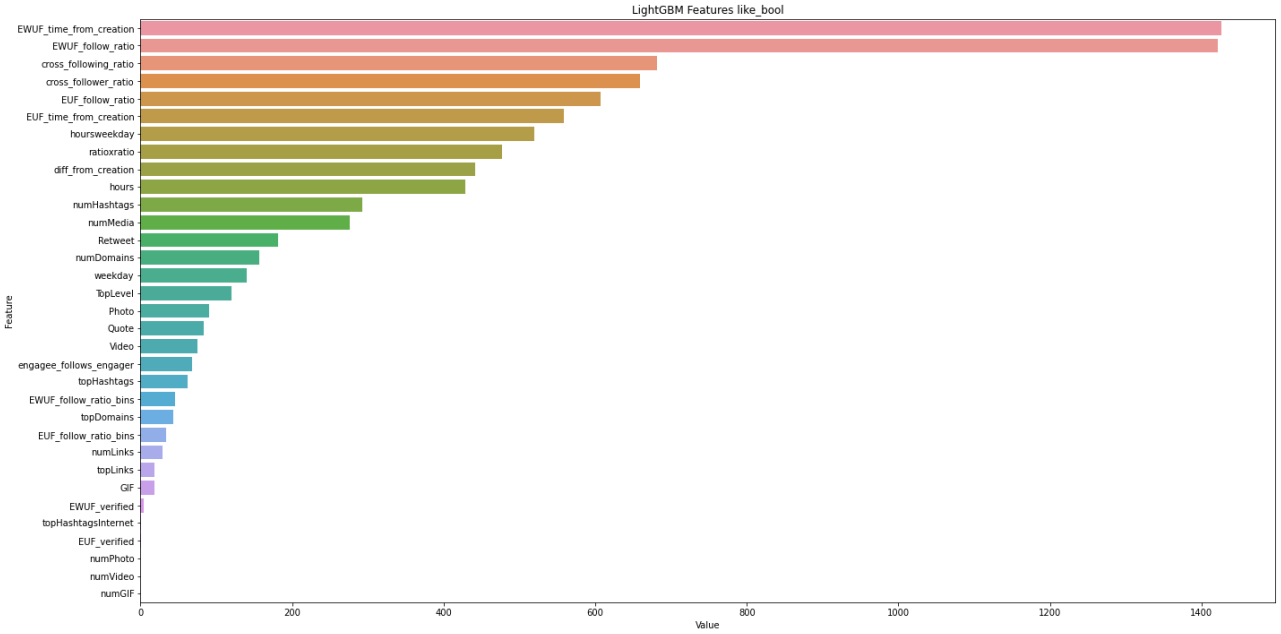


FIGURE B.7: Feature importance for the Like engagement in the Non-Text Based Model.

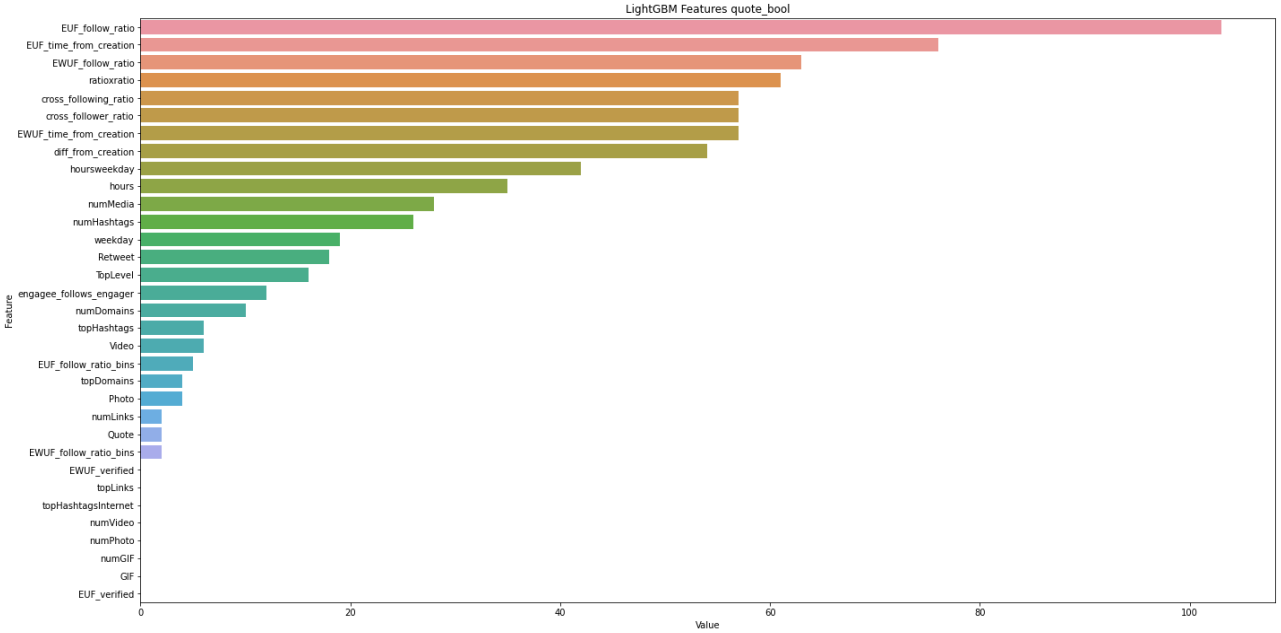


FIGURE B.8: Feature importance for the Quote engagement in the Non-Text Based Model.

B.3 Feature importance of the Mixed Model (RT)

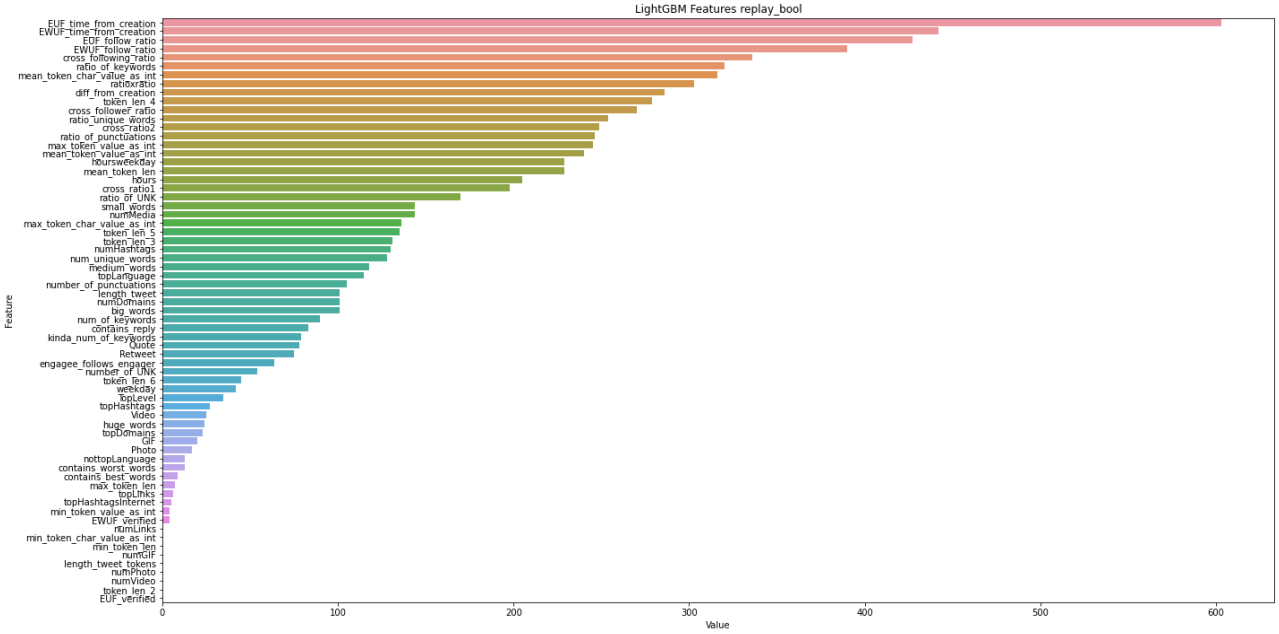


FIGURE B.9: Feature importance for the Reply engagement in Mixed Model (RT).

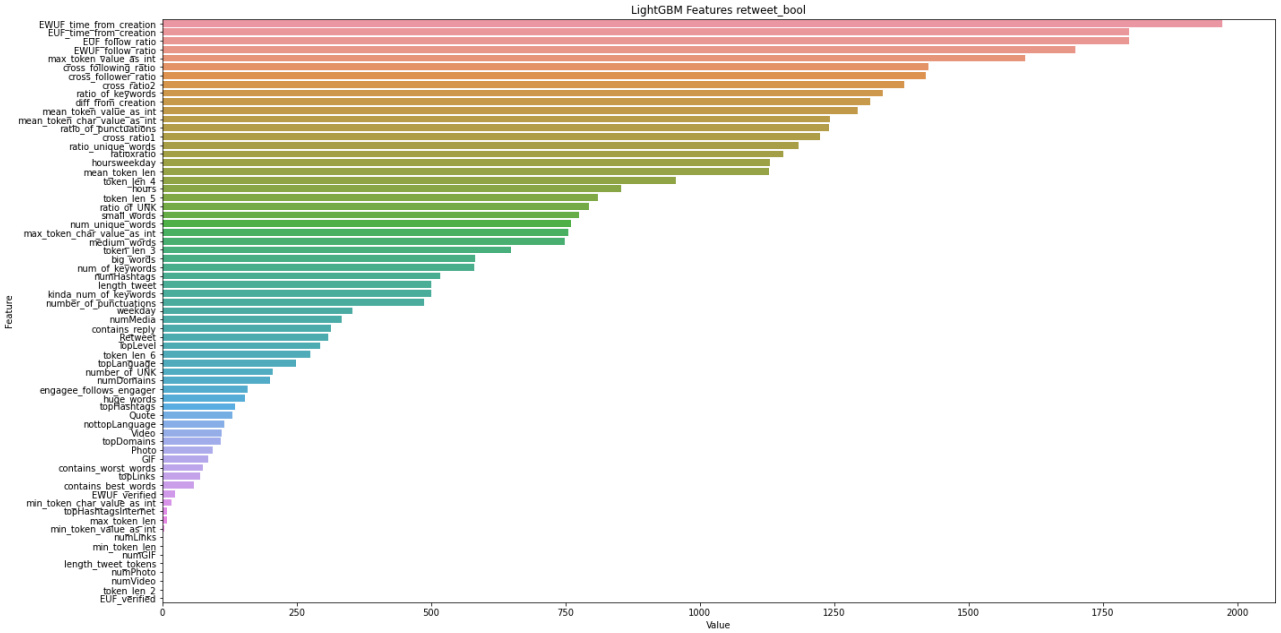


FIGURE B.10: Feature importance for the Retweet engagement in Mixed Model (RT).

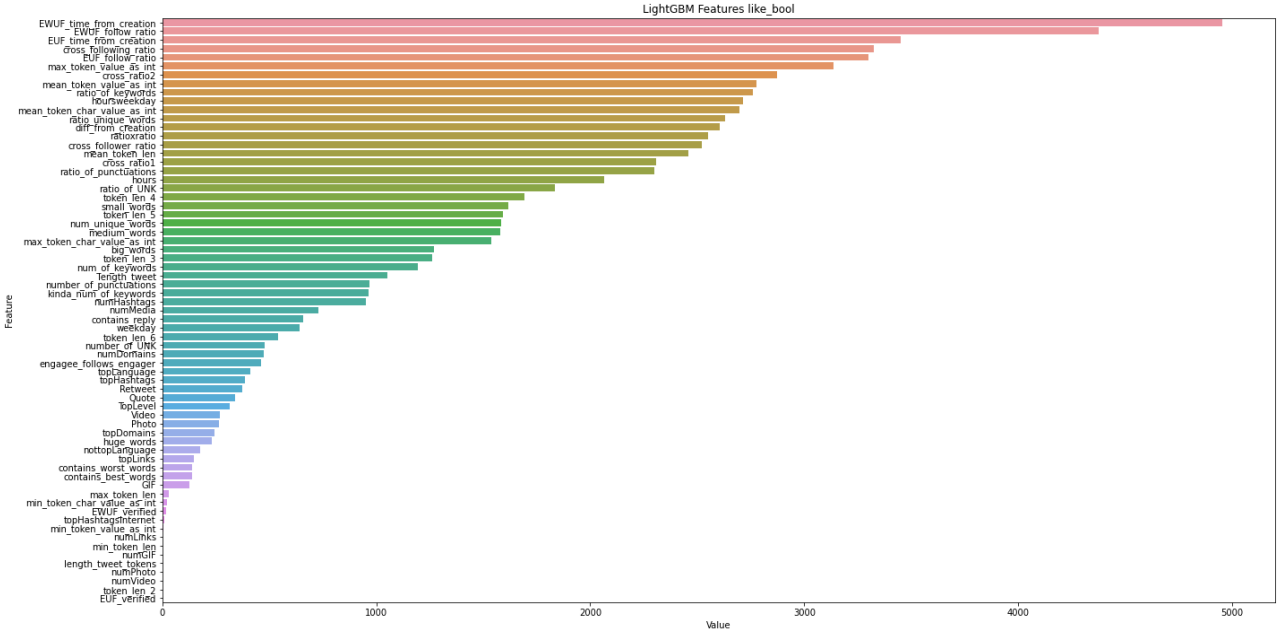


FIGURE B.11: Feature importance for the Like engagement in Mixed Model (RT).

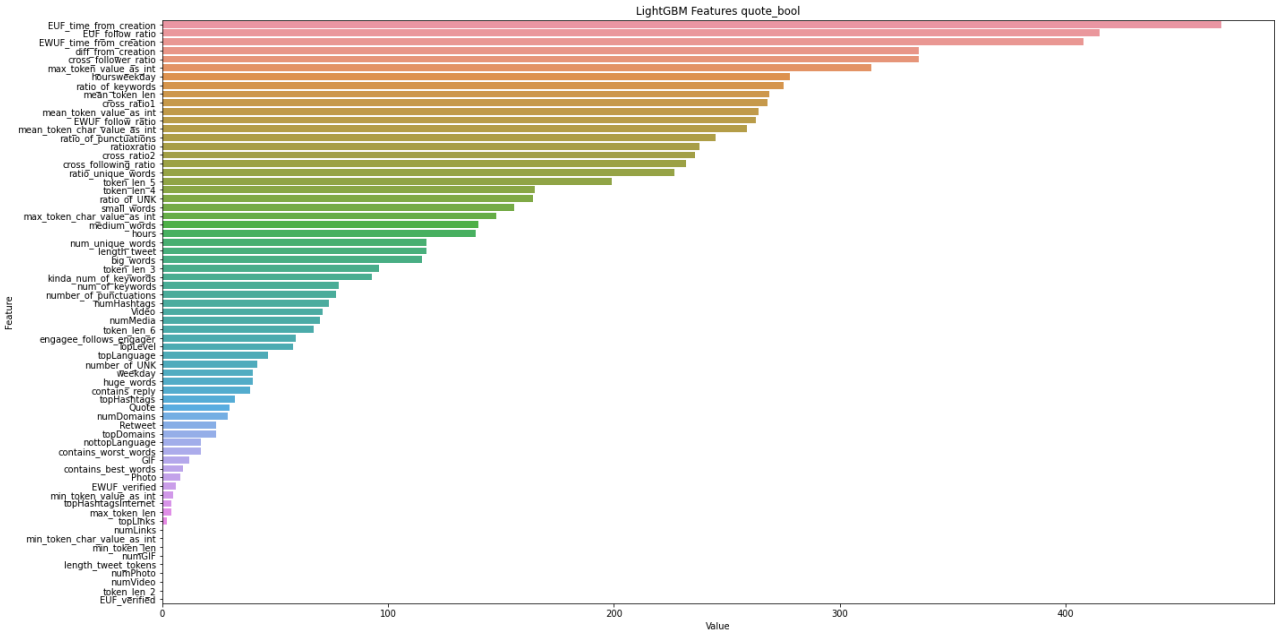


FIGURE B.12: Feature importance for the Quote engagement in Mixed Model (RT).

B.4 Feature importance of the Mixed Model (NRT)

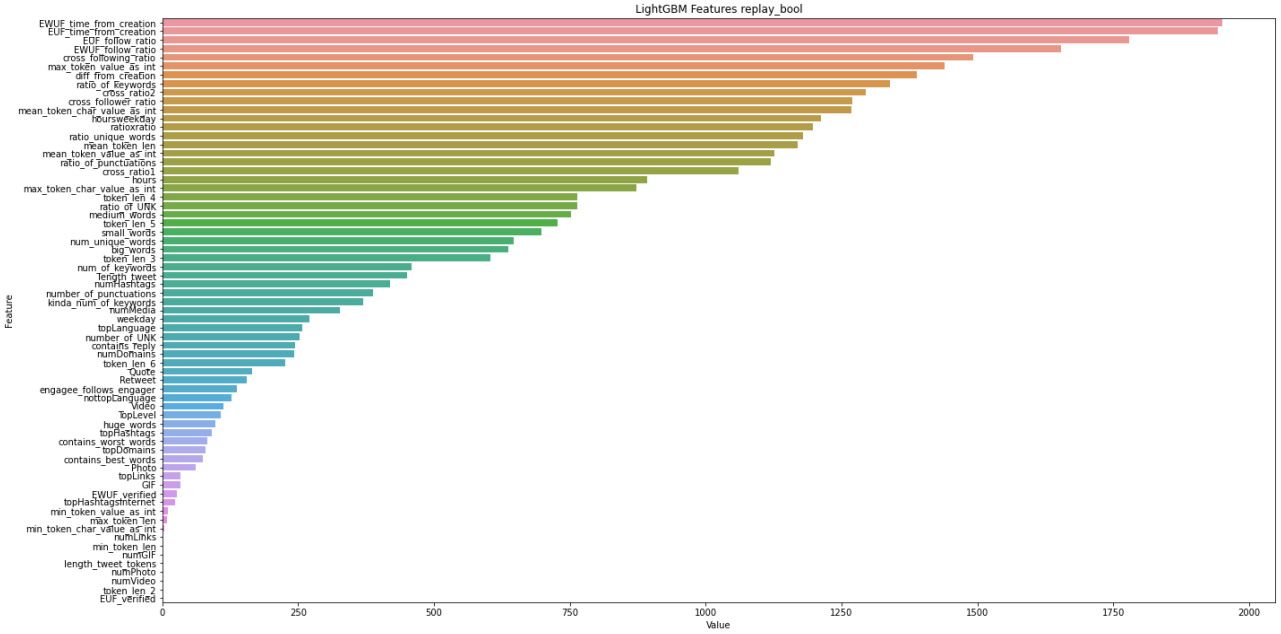


FIGURE B.13: Feature importance for the Reply engagement in Mixed Model (NRT).

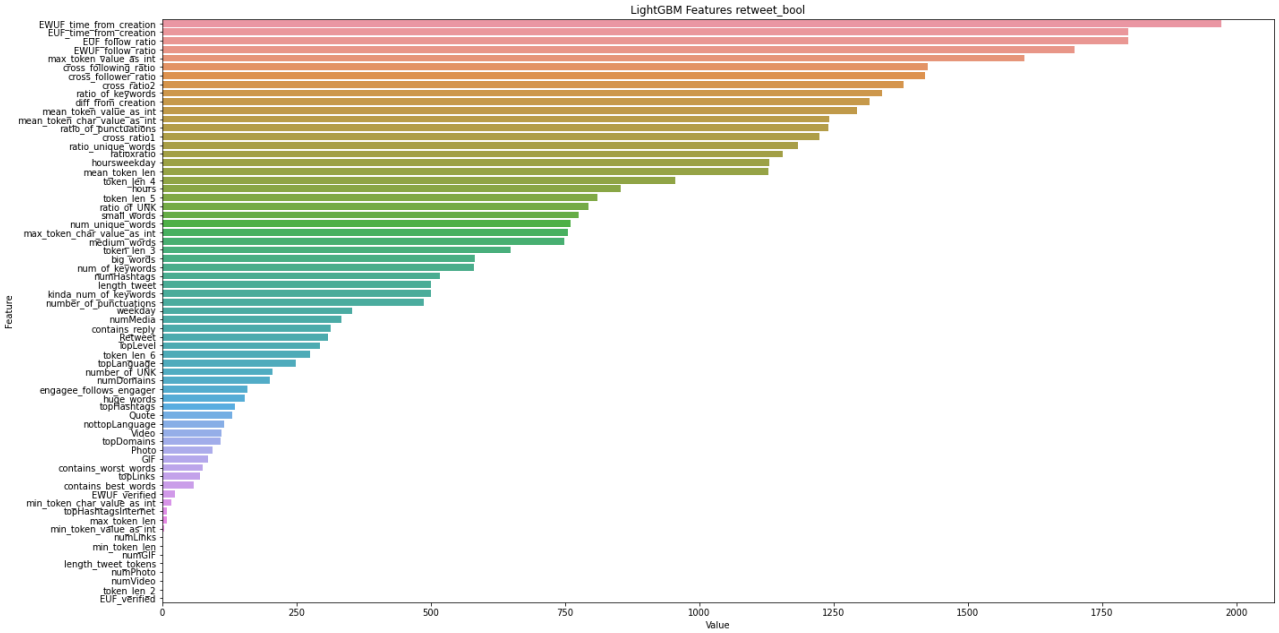


FIGURE B.14: Feature importance for the Retweet engagement in Mixed Model (NRT).

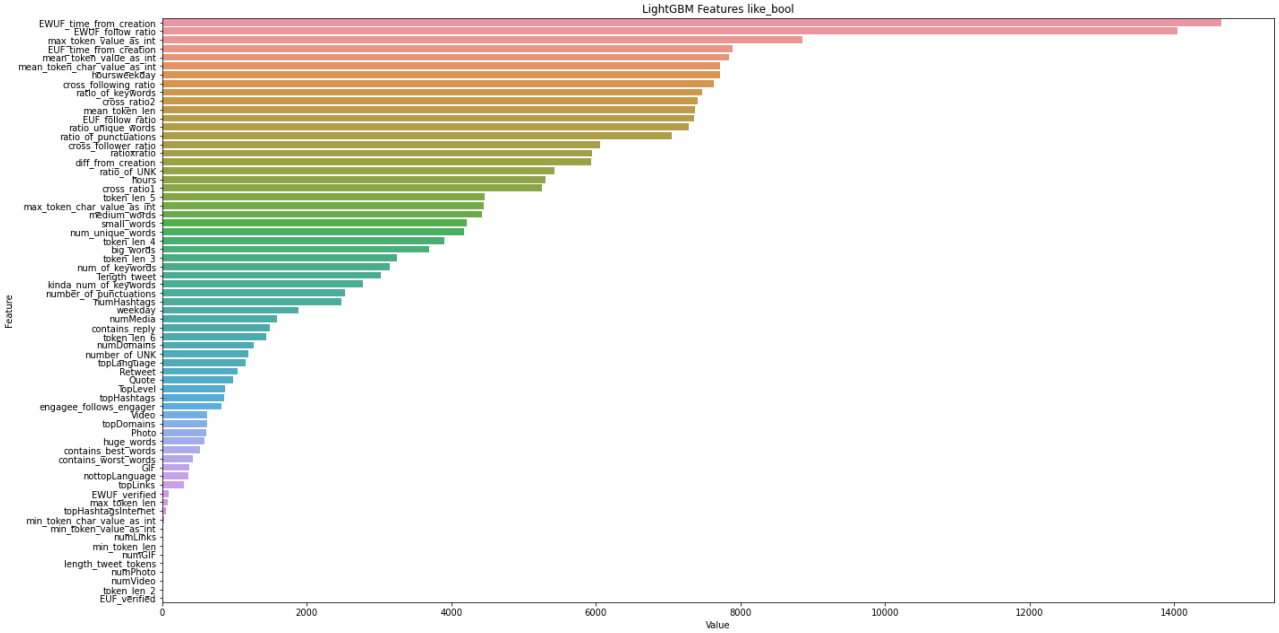


FIGURE B.15: Feature importance for the Like engagement in Mixed Model (NRT).

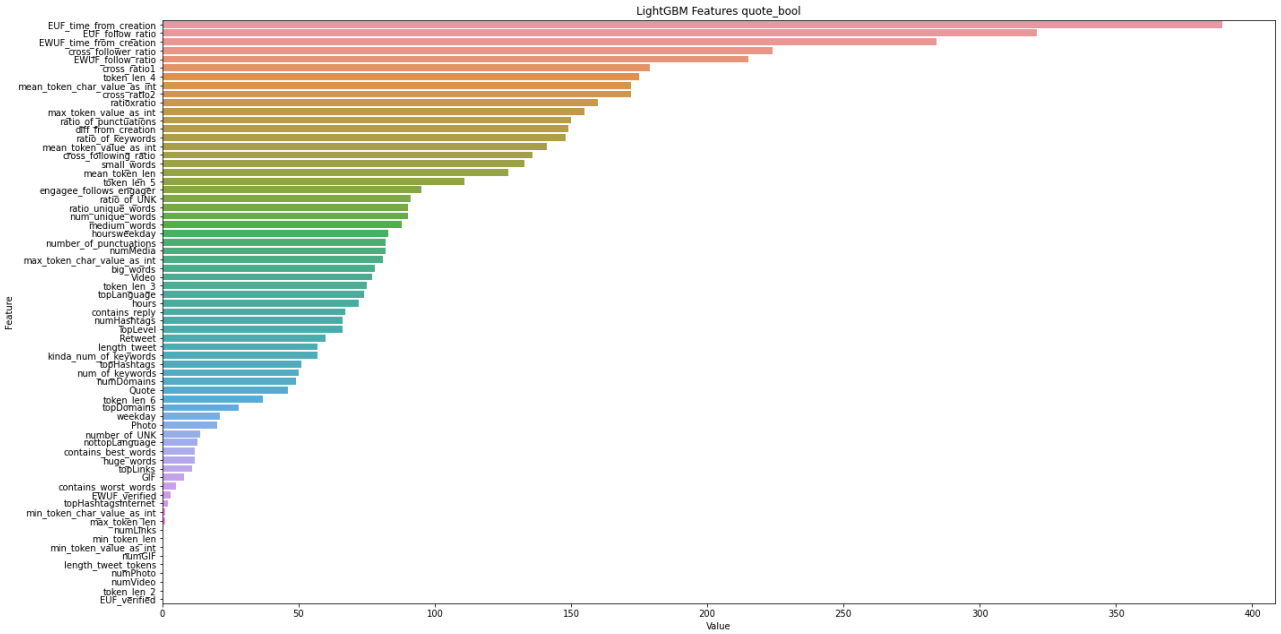


FIGURE B.16: Feature importance for the Quote engagement in Mixed Model (NRT).

Appendix C

Distribution of work

The reader can find in this Appendix the distribution of work between each member of the project group.

- **Marcos:** He has implemented and developed non-text related features as well as trained different models using different classifiers and hyper-parameters. In addition to this, he has implemented memory optimisations to the text model. He has written Chapter 6 of this document.
- **Adrià:** He has implemented and developed almost all text-related features of the Text Model as well as trained different models using different classifiers and hyper-parameters. He has written Chapter 5 of this document.
- **Marcos and Adrià:** They have implemented and arranged the mixed final model used in the final submission.

The rest of the Chapters of this document have been written collaboratively by both authors of this document.

Bibliography

[1] Jussi Karlgren. An algebra for recommendations. *Syslab Working Paper No 179*, 1990. URL https://jussikarlgren.files.wordpress.com/1990/09/algebrawp.pdf.

[2] Will Hill, Larry Stead, Mark Rosenstein, and George Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, page 194–201, USA, 1995. ACM Press/Addison-Wesley Publishing Co. ISBN 0201847051. doi: 10.1145/223904.223929. URL https://doi.org/10.1145/223904.223929.

[3] Jussi Karlgren. Newsgroup clustering based on user behavior - a recommendation algebra. Technical Report T94:04, SICS, 1994.

[4] Acm recommender systems. URL https://recsys.acm.org/recsys20/.

[5] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[6] Alireza Rezazadeh. A generalized flow for b2b sales predictive modeling: An azure machine-learning approach. *Forecasting*, 2:267–283, 08 2020. doi: 10.3390/forecast2030015.

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

[8] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.

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

[10] Machine types | compute engine documentation | google cloud. URL https://cloud.google.com/compute/docs/machine-types#e2_high-memory_machine_types.

[11] Shayna. The 100 most common words in english, Mar 2012. URL https://www.espressoenglish.net/the-100-most-common-words-in-english/.

[12] Jerry LowJerry Low is the a professional blogger who is passionate about SEO and web development. He writes helpful web development guide. 10 easy ways to get more retweets, Jul 2017. URL https://www.crazyegg.com/blog/get-more-retweets/.

[13] Tf-idf, Dec 2019. URL https://es.wikipedia.org/wiki/Tf-idf.

[14] sklearn.preprocessing.minmaxscaler¶. URL https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.