UNIVERSITAT DE BARCELONA

FUNDAMENTAL PRINCIPLES OF DATA SCIENCE MASTER'S THESIS

Twitter engagement model for the RecSys 2020 Challenge

Author:
Pere GILABERT

Supervisor:
Dr. Santi SEGUÍ

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

June 30, 2020

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Twitter engagement model for the RecSys 2020 Challenge

by Pere GILABERT

Recomendation systems is a wide field of research and it is present in many area of our daily life. The RecSys ACM conference is the most important conference in the recommendation area and each year it holds a competition, the RecSys Challenge. The work here presented aims to solve the RecSys 2020 Challenge which consists of giving a certain probability of two Twitter users to interact. We have developed a model which uses the power of Gradient Boosting Trees to combine multiple features we created to represent each interaction between users. Features such as popularity or engagement were combined with and embedding of the tweet text to create an interdisciplinary model that is able to reach 0.75 on the Precision-Recall area under the curve metric and 17.64 on the Relative Cross Entropy. The popularity feature and previous reactions to the same language were discovered as the most relevant features for our model. Regarding the competition, our team reached the ninth place of the challenge.

Acknowledgements

To my project manager Santi for his high dedication in this work and his interest in the competition. At no time I have felt alone and it has been a pleasure to work with him.

To my family and friends for the constant support and confidence in all the projects I do.

Contents

Chapter 1

Introduction

1.1 Introduction

We live surrounded by information. At all times, fresh news appear while others become obsolete. In addition, because of the large volume of information provided to us every day, users have become very selective about spending time on a particular application, social network or game. They need relevant and engaging content, aligned with their interests, so they don't get bored and stop using it.

Machine learning is one of the most powerful fields of research in the technology sector, which seeks to provide solutions to specific problems using, generally, a large volume of historical data. From these data, patterns of systematical behavior can be predicted. Currently, these techniques are applied in fields as diverse as marketing, medicine or information technology. A subfield of Machine Learning, although born first, is the world of recommendation systems.

Within the recommendation systems field there are several widely known applications. This is the case of song recommendation systems, ads for products to be purchased or content that is shown to users on social networks.

This project focuses on the study of the social network Twitter. This is possible thanks to a huge set of data made public by this Twitter platform (Belli et al., 2020) and that tries to solve the problem of recommending tweets.

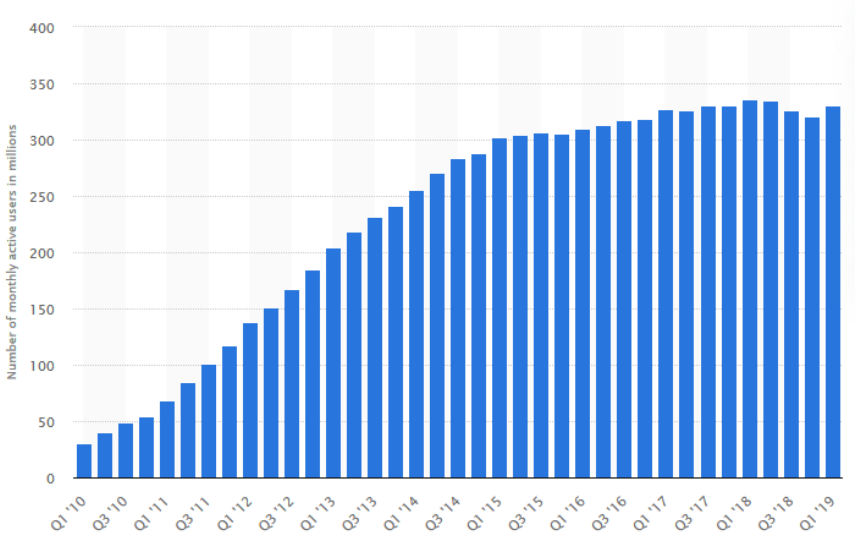


FIGURE 1.1: Active Twitter (2010-2019). Image font: *Statista*

Twitter has, nowadays, more than 330 million as we can see in Figure 1.1 active users so its recommendation algorithm is one of the most relevant factors for its continuous growth. Its users will spend more time within the platform if the content

shown to them is in line with their interests, so it is an interesting challenge how to decide the information to show to each user. This is the problem we tackle in this project. The final goal is to be able to design a tweet recommendation system to show more relevant information to the user and thus improve their browsing experience. Concretely, the aim is to increase the positive reactions of users to the tweets shown, that is, we want to maximize the number of likes, replys, retweets and retweet with comment. These are the four actions a user can react to a tweet with.

With this approach in mind, we face different challenges and some difficulties:

- First and foremost, the large volume of data. The examples provided in the dataset mentioned above contains about 200 million examples of user interactions and 100 million examples of user pairs that have not interacted with each other (negative sampling).
- Secondly, the need to develop a multidisciplinary model that combines classic characteristics of users directly (such as the number of followers, the frequency of tweets...) with more complex characteristics that need the understanding of the environment (such as the topics of interest of a user from the text of his/her tweets, the popularity of a user, his influence on the network...).

1.1.1 Motivation of this project

This project is challenging for several reasons. Firstly, on a personal level, because it is the first Challenge with worldwide recognition in which I participate and furthermore this challenge is part of a high level congress in the data science sector. The possibility of presenting a paper at the ACM RecSys 2020 conference is also a good incentive to try to do my best since it would be my first presented paper.

Secondly, it is a unique opportunity to apply all the knowledge learned during this last year in which I have completed the Masters in Fundamental Principles of Data Science. The concepts learned in controlled environments like small problems, should serve to think big and develop a complex solution to the problem at hand.

1.1.2 Goals

The main objective of this project is to develop a sophisticated recommendation system. The model must combine different disciplines of the Machine/Deep-Learning world such as:

- To extract features from the given dataset.
- To be able to generate new features by combining temporal information of past tweets.
- To create a Language Model using some State-of-the-Art model that works in the multilingual case and to generate a text embedding of the text.
- To create a robust model where all these features are combined.
- To submit several solutions to the Recsys platform and compete with the top teams.

1.2 Document structure

- Chapter 1: Introduction to the thesis topic.
- Chapter 2: The Recsys Challenge. An overview of the Recsys conference and the definition of the current challenge.
- Chapter 3: Background. All the theoretical concepts necessary to understand the procedure followed in this project. Here we will devote some time on the classification algorithm, the language model, and the baseline method proposed by the organizers.
- Chapter 4: Implementation details. Here we will explain all the process to construct the model, including the analysis of the dataset and the construction of the features.
- Chapter 5: Results. The outcomes of the previous chapter work.
- Chapter 6: In the last Chapter, a summary of all the work done, conclusions and further steps.

Chapter 2

The RecSys Challenge 2020

2.1 RecSys

The ACM Recommender Systems conference (RecSys) is one of the most important conferences in the world. It is entirely dedicated to recommendation systems, a very extensive field in many areas, and which is very specific to each problem faced. This is like this because there is not a perfect recommendation system for all problems since it depends a lot on the type of data available, the type of objective to be achieved, the users it is aimed at, among other factors.

The 2020 edition is the 14th edition of the conference and the 10th edition of the challenge. In this contest, a problem is presented to be solved and the participants have approximately three months to develop a recommendation algorithm to be able to claim the prize.

The last four competitions organized by RecSys were:

- 2019 Trivago: Their products are accommodations that they offer to its clients. The participants had to predict the accommodations each user clicked while surfing its web.
- 2018 Spotify: Spotify is a well consolidated platform for music recommendation. The goal of the challenge was to complete playlist given only the first few songs.
- 2017 XING: The problem presented by XING was the job recommendation but focusing on the problem of cold start, that is, when we do not have information about a new user that starts to use the platform or a new job that does not have information of the users who would like it.
- 2016 XING: A job recommendation system.

2.2 The problem

The problem presented in the 2020 RecSys Challenge is the recommendation of tweets but not in a direct way. Given a pair of users and a tweet generated by one of them the goal is to assign a probability from 0 to 1 that the second user will react to the tweet as it is shown in Figure 2.1. There are 4 classes of reactions a user can give which are:

- Like: The user can mark the message with a heart to indicate she/he likes it.
- Reply: The user can replay to the creator of the tweet.

- Retweet: The user can share a tweet from another person in its own personal page.
- Retweet with comment: The user can share a tweet from another person in its own personal page, adding an extra message.

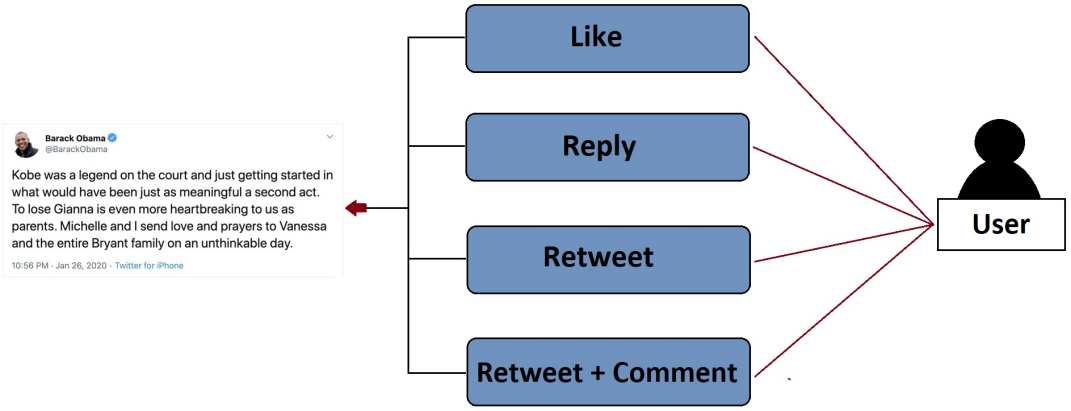


FIGURE 2.1: Problem statement

Moreover, this is not a classical classification/regression problem with 4 classes since a user can give multiple reactions to the same tweet for example, and a typical combination, is Like & Retweet. Thus, we have 4 classification/regression problems to deal with.

2.3 Evaluation

In order to evaluate the performance of a submission, 2 metrics for the 4 targets will be used to compute the final score of a submission, so each submission has 8 values. Then, the average for the first metric is computed and for the second metric too. Finally, all the users will be ranked according to these two values in the following way. All users are ranked with respect to the first metric but and with respect to the second metric separately. This two positions are added together to obtain the final classification score. So, for example, a participant that is the first on the first metric and second on the second one, will have a total score of $1 + 2 = 3$.

In order to understand the two metrics used to evaluate each submission, some concepts need to be explained.

- Precision: In a binary classification problem we have elements of one class (positive) and elements of the other class (negative). If we count the number of correctly classified elements in the positive (TP: True Positive) and the negative elements that are missclassified as postive (FP: False Positive), we can define the precision as

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.1}$$

- Recall: Using the same notation as before, the Recall computes formula is

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.2}$$

- Cross-Entropy / Log-Loss: In a binary classification problem, we can define $\hat{y}_i$ as the real label and y_i as the predicted target. Then, the Cross-Entropy for a set of N examples is defined as

$$\mathrm{CE}_{\mathrm{pred}} = -\frac{1}{N}\sum_{i=1}^{N}\Big[\hat{y}_i \log(y_i) + (1-\hat{y}_i)\log(1-y_i)\Big] \tag{2.3}$$

2.3.1 Precision-Recall Area Under the Curve (PR-AUC)

Precision-Recall is a well know metric to evaluate the performance of a regression algorithm. The plotted curve summarizes the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds. Then the PR-AUC is the value of the area under the Precision-Recall curve.

2.3.2 Relative Cross-Entropy (RCE)

To compute the Relative Cross Entropy first we need to define:

$$\mathrm{CTR} = \frac{\text{Positive examples}}{\text{Total number of examples}} \tag{2.4}$$

This value is specially designed for unbalanced datasets (like this one). From here, we can compute the Naive Cross-Entropy that is the Cross-Entropy between the real labels and a vector with the value CTR repeated to match the labels length. That is:

$$\mathrm{CE}_{\mathrm{naive}} = -\frac{1}{N}\sum_{i=1}^{N}\Big[\hat{y}_i \log(\mathrm{CTR}) + (1-\hat{y}_i)\log(1-\mathrm{CTR})\Big] \tag{2.5}$$

Finally, the Relative Cross Entropy can be defined as the quotient

$$\mathrm{RCE} = 100 \times \frac{\mathrm{CE}_{\mathrm{naive}} - \mathrm{CE}_{\mathrm{pred}}}{\mathrm{CE}_{\mathrm{naive}}} \tag{2.6}$$

As a final note, one can see that minimizing the Cross-Entropy, the Relative Cross Entropy is maximized.

2.4 The data

The data made public by Twitter is about 200 million interactions between users during two weeks. Specifically, the data provided is from 2020-02-06 to 2020-02-19. In addition, during this period, 100 million negative interactions have also been generated, that is, pairs of users and tweets in which there has been no interaction.

The data is divided into three different sets. The first one is the training set. This dataset contains interactions of the users during the first week available, i.e. from 2020-02-6 to 2020-02-12 with the information of whether or not an interaction took place and the type of interaction. The two other datasets contain the information for the following week, i.e. 2020-02-13 to 2020-02-19 but without the information on whether or not there was an interaction between the users. During the whole contest, only one of these two files was provided and it was used to upload partial solutions to a web platform that evaluates this file. During the last two weeks of the

competition, the third file was provided with the same format as the second one, to generate the final solution.

Each example consists of multiple fields:

2.4.1 Tweet Features

- Text tokens: Ordered list of Bert ids corresponding to Bert tokenization of Tweet text
- Hashtags: Tab separated list of hastags (identifiers) present in the tweet
- Tweet id: Tweet identifier
- Present media: Tab separated list of media types. Media type can be in (Photo, Video, Gif)
- Present links: Tab separated list of links (identifiers) included in the Tweet
- Present domains: Tab separated list of domains included in the Tweet (twitter.com, dogs.com)
- Tweet type: Tweet type, can be either Retweet, Quote, Reply, or Toplevel
- Language: Identifier corresponding to the inferred language of the Tweet
- Timestamp: Unix timestamp, in sec of the creation time of the Tweet

2.4.2 Engaged With User Features (Tweet creator)

- User id: User identifier
- Follower count: Number of followers of the user
- Following count: Number of accounts the user is following
- Is verified: Is the account verified?
- Account creation time: Unix timestamp, in seconds, of the creation time of the account

2.4.3 Engaging User Features (Tweet receiver)

- User id: User identifier
- Follower count: Number of followers of the user
- Following count: Number of accounts the user is following
- Is verified: Is the account verified?
- Account creation time: Unix timestamp, in seconds, of the creation time of the account

2.4.4 Engagement Features

- Engagee follows engager?: Does the account of the engaged tweet author follow the account that has made the engagement?
- Reply engagement timestamp: If there is at least one, unix timestamp, in seconds, of one of the replies
- Retweet engagement timestamp: If there is one, unix timestamp, in seconds, of the retweet of the tweet by the engaging user
- Retweet with comment engagement timestamp: If there is at least one, unix timestamp, in seconds, of one of the retweet with comment of the tweet by the engaging user
- Like engagement timestamp: If there is one, Unix timestamp, in seconds, of the like

2.4.5 Additional concepts

There are some concepts that may need a further explanation:

Firstly, the Bert identifiers are a direct conversion of the tweet text into a list of numerical values. We'll talk about them in more detail later.

Secondly we have the UNIX timestamp. This is a common format for representing dates in digital format that counts the seconds that have passed since January 1, 1970 (UTC).

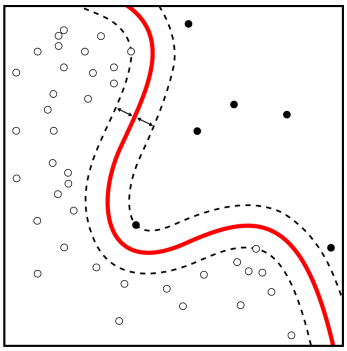
Lastly, we say that a Twitter account is verified if a blue mark appears next to the user's name. This mark is a distinction given by Twitter to the accounts of important people and serves to distinguish that account from possible fake accounts created by other users.

Chapter 3

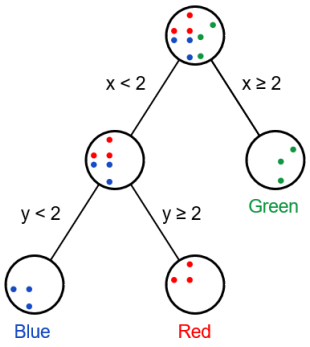
Background

3.1 Classifiers and Neural Networks

Neural networks have meant a gigantic advance in the field of artificial intelligence as they are capable of facing very different problems with excellent results. The basis of this project is based on the development of two different types of algorithms, capable of learning some specific characteristics and predicting a value. The first one is a classifier. A classifier is an algorithm capable of distinguishing samples from different classes as can be seen in Figure 3.1.



(A) Decision boundary for a binary problem. Image font: *Wikipedia*



(B) Decision tree. Image font: *Victor Zhou*

FIGURE 3.1: Two types of classifiers

For this specific problem we have used LightGBM (Ke et al., 2017), a well known algorithm based on the Gradient Boosting technique. In order to be able to understand this concept, first we need to talk about what a Decision Tree is.

A decision tree is a structure used in Machine Learning as a classifier. As we can see in Figure 3.1b, from a series of chained conditions, we can classify a sample into a specific category. In each node of the tree, the dataset is separated into two categories depending on a condition. In each terminal node of the tree, there is a label, that is, a category for the data set that satisfies the previous conditions.

One of the major advantages of these methods is their interpretability. Decisions made at each decision node can be represented by a combination of the initial actual characteristics of the data. In contrast, most machine learning algorithms (and deep learning algorithms such as neural networks) do not have this property. The learned model can be thought of as a black box that receives some input elements and from which a prediction comes out.

There are many algorithms that use this method to classify. RandomForest, XGBoost, Adaboost or LightGMB, are some of the most known methods.

Now we add a new concept, Gradient Boosting Decision Trees. Here we do not use just one decision tree but combine multiple trees to make the prediction. The trees are trained iteratively, that is, one at a time, to minimize a target we call loss function. The way to train is relatively simple, you separate the data that is in a node if that minimizes the loss function. A good way to prevent the algorithm from overfitting, is to limit the number of splits that can make each tree.

There are two different strategies when it comes to building a tree. The first, level-wise, expands by levels. This method keeps the tree balanced so the search time is as short as possible. The second, leaf-wise, expands the node that optimizes the most. This second method is more likely to overfit the dataset but is more scalable when we have a very large dataset. Figure 3.2 shows these two methods.

(A) Level-wise. XGBoost (Chen and Guestrin, 2016) uses this method

(B) Leaf-wise. LightGBM uses this method

FIGURE 3.2: Two types of tree growth

One of the most relevant points to consider when expanding the tree, is to decide how to split the data in each node. The optimal solution would be to go through all the examples and all the characteristics of our dataset but this would take an enormous amount of time.

Instead of doing this, LightGBM uses a grouping of the data features and splits them over these groups instead of the original features. This is a great improvement in the speed of the algorithm.

3.2 Natural Language Processing

Natural Language Processing is a field of linguistics that, combined with artificial intelligence, creates algorithms for processing human language. There are several related problems in this field such as

- **Named Entity Recognition**: Where the goal is to classify words based on whether they refer to locations, dates, organizations... In Li et al., 2012 an example of this task is shown, using Twitter data.
- **Translation**: Converting an original phrase in one language into a the same meaning sentence in a different language. The latest (and the State-of-the-Art in translation) model available is T5 (Raffel et al., 2019) created by Google, it is able to translate in real time among other things.
- **Text reconstruction**: Given a text with missing words, fill it with words that make sense in the global context of the sentence. As we will see later on, the BERT model is trained using this task.
- **Sentiment Analysis**: Given a sentence, to be able to classify its sentiment, so to differentiate whether it is expressing a positive, negative or neutral emotion, for example. This is a very common task of te previous years and similar analysis can be found in Agarwal et al., 2011; Pak and Paroubek, 2010; Kouloumpis, Wilson, and Moore, 2011 also using the data gathered from Twitter.

3.2.1 BERT model

The BERT Model (Pre-training of Deep Bidirectional Transformers for Language Understanding) (Devlin et al., 2018) is a text model developed by Google in 2018 that was a revolution in the NLP sector. This model is trained using two giant datasets (BookCorpus (Zhu et al., 2015) and Wikipedia) and the authors released several pre-trained models. In particular, the one of our interest (Bert-Multiligual-Base-Cased) is trained using Wikipedia in 104 languages.

The structure of the model is quite simple. It uses 12 fully-connected layers of 786 neurons each. In addition, it uses 12 special structures called heads that represent patterns of interaction between different word types. These patterns do not share parameters with each other so that one pattern does not influence another. An example of these 12 patterns can be found:

1. Interaction of a word with the next word in the sentence.
2. Interaction of a word with its previous word.
3. Interaction of a word with its other appearances in the sentence.
4. Interaction of a word with punctuation marks.

With 12 patterns per layer and 12 different layers, there are 144 interaction patterns (attention patterns) in each BERT model. In total, all the BERT architecture has over 110 million parameters.

The input format of phrases to the model is also very particular. Phrases always begin with a specific token *[CLS]* and end with another key token, *[SEP]*. In the case that the input does not consist of a single sentence as we will see in Question Answering tasks for example, the format is *[CLS]* sentence 1 *[SEP]* sentence 2 *[SEP]*.

To encode the words in tokens, a conversion using three different embeddings is used. The first one is where transformations are made in order to find the root of the verbs and other transformations to reduce the word complexity. The second one is the segment embedding. We will have only one segment in case the input is only one sentence and two segments if the task requires two input sentences. The third and last embedding, consists of the position of the word within the sentence. We can see how two consecutive phrases are coded in the example of the original paper in Figure 3.3.

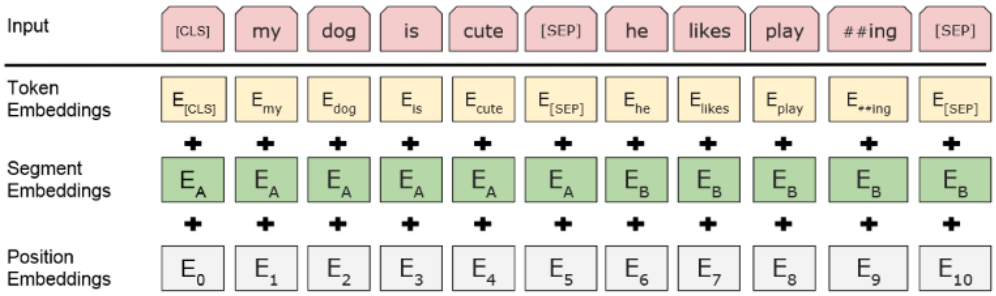


FIGURE 3.3: Triple embedding for BERT tokenization

To create the BERT pre-trained model, the authors trained it in two different tasks. Firstly the *Masked LM* task where a few words in the sentence are hidden and the model has to reconstruct them.

Second, the *Next Sentence Prediction* (NSP) task. Given two sentences the model has to identify whether the second sentence is next (in context) to the first one. This task is used in *Question Answering* (QA) where the problem faced is to try to predict if a sentence is an answer to a given question. Another problem, *Natural Language Inference* (NLI) consists on, given two sentences, predicting if they convey the same meaning, for example, the description of a situation or a product review made by two different people.

The token *[CLS]* has a special function within the classification tasks. The final hidden state of that token is the input used by the classifier to determine the output of the model. As we can see in Figure 3.4, the lasts hidden states are discarded and only the first one is kept. In this case, the hidden state corresponding to the token *[CLS]* has 784 values that enter the classifier, a fully-connected network that reduces the dimension from 784 to 2 and applies a softmax at the end to predict for example, if the sentiment of the sentence is positive (1) or negative (0).

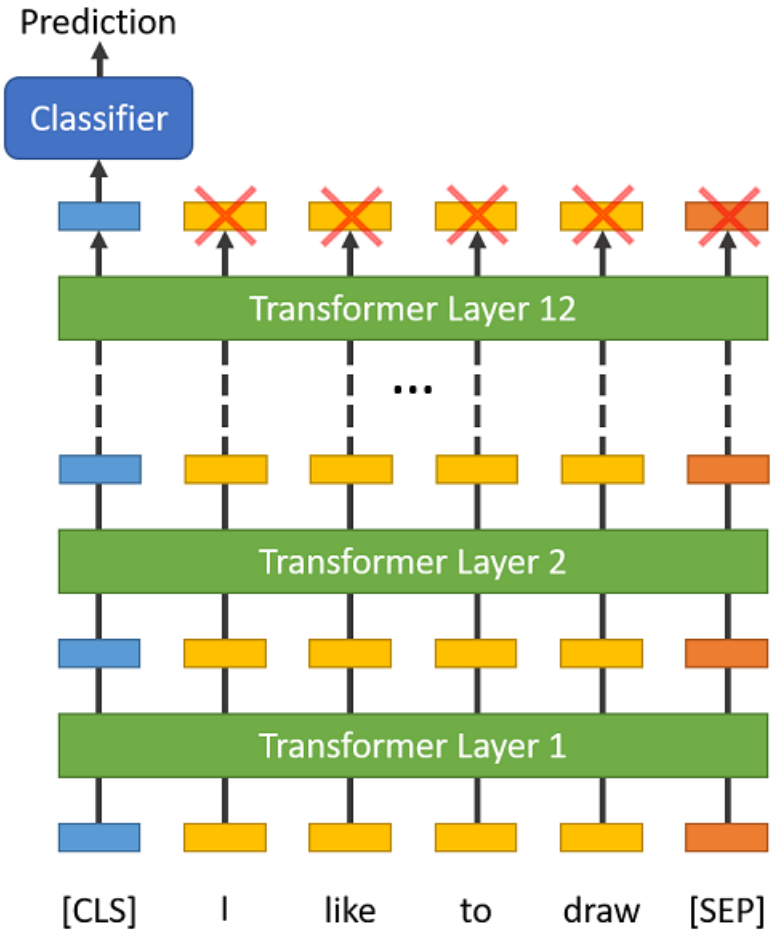


FIGURE 3.4: CLS token usage. Image font: *Chris McCormick*

3.3 Baseline model

In the original twitter paper (Belli et al., 2020), a baseline model is presented. A distinction is made between features by the type of data these features represent and not by the intrinsic properties of what they represent in reality.

- *Numerical features*: In order not to introduce a numerical value as a feature, numerical features are separated into different groups (buckets) depending on the quantile they are placed on the distribution function. In the paper it is proposed to work with 50 buckets. Each bucket is represented with a vector of zeros and a one according to the bucket it represents (*One Hot Encoding*).
- *Category features*: They are converted directly into buckets using their value. For example, the "tweet type" characteristic that can take values: *Retweet*, *Replay*, *Quote* or *TopLevel* can be encoded with one of the vectors $[0,0,0,1]$, $[0,0,1,0]$, $[0,1,0,0]$ and $[1,0,0,0]$.
- *ID features*: In this case the authors decided to make a hash of the value and split it also into 50 buckets to make the One Hot Encoding. For example, the language *22C448FF81263D4BAF2A176145EE9EAD* would be converted to a number from 0 to 49 and transformed with One Hot Encoding to a vector of 50 values.

The model they present has the following architecture (Figure 3.5):

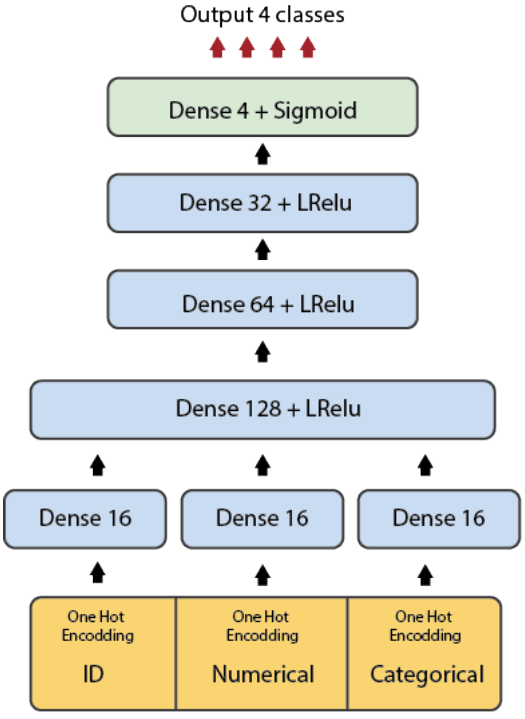


FIGURE 3.5: Baseline model presented

The three types of features, transformed into the One Hot Encoding format, are passed into a layer of 16 neurons each. This makes a total of 48 neurons. Their output is combined after the activation Leaky ReLU is applied. These are all connected with a layer of 128 neurons followed by a layer of 64 neurons followed by a 32 neurons one and finally a layer of 4 neurons to obtain the final output of the four classes

that we want to predict. Between the layers a Leaky ReLU is applied except for the output of the network where a sigmoid is applied instead of a softmax in order to have a multiclass output.

Chapter 4

Implementation details

In this Chapter we will explain the different features created from the original dataset we mentioned in section 2.4.1. In addition, we will also explain the implemented text model which is able to classify how popular a tweet will be by looking only at the text. Finally, we will go over how we have unified these two concepts to create the final model.

4.1 Organizing the data

One of the first problems that arose when we started the challenge was the large volume of data available to us. The original dataset with about 167 million samples was too big to fit in the RAM memory of our computer or the server we had dedicated to the challenge.

So, the first thing we did was to split the data into smaller files we called *train_batches*. Each of them contained 1 million samples from the original file. Next, we noticed that the samples in the original file did not follow any particular order. We decided that, if we wanted to take advantage of the past to predict the future, the most important thing was to sort those samples. That is how we generated a set of files that we called *train_batches_timestamp* where each of them refers to a specific time of day.

File	001	002	003	···	166	167	168
% positive	48.42	48.40	49.61	···	38.38	35.92	27.31
Unique tweets	422475	425174	430127	···	369857	365057	337080
Unique writers	341187	341175	344058	···	298881	296113	275728
Unique engaged	821514	843079	823413	···	690067	679229	592966
Day	02-06	02-06	02-06	···	02-12	02-12	02-12
Hour	00	01	02	···	21	22	23

TABLE 4.1: Files sorted by timestamp with some statistics.

In Table 4.1 we can see some statistics of the firsts and lasts batches. We can see how this temporal order is important in Figure 4.1.

It shows the number of users (tweet makers, tweet receivers and number of unique tweets) by hour of the day. There is a clear pattern that repeats every 24h.

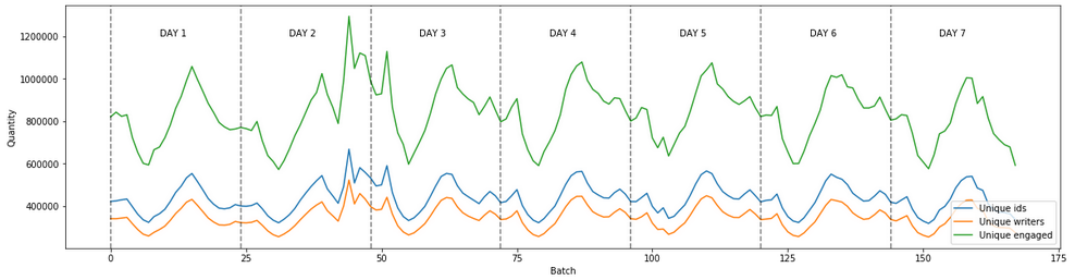


FIGURE 4.1: Number of users and tweets by hour

4.2 Analysing the dataset

From the training dataset with 167 million examples, 76.5 million are positive (with some interaction between users) and the rest are negative (without interaction). We can see in the Figure 4.2 that the number of tweets per user is very unbalanced. Here we show the top 100 users regarding the number of tweets present in the dataset. The maximum number of times a user appears is 172 thousand times and the dis-

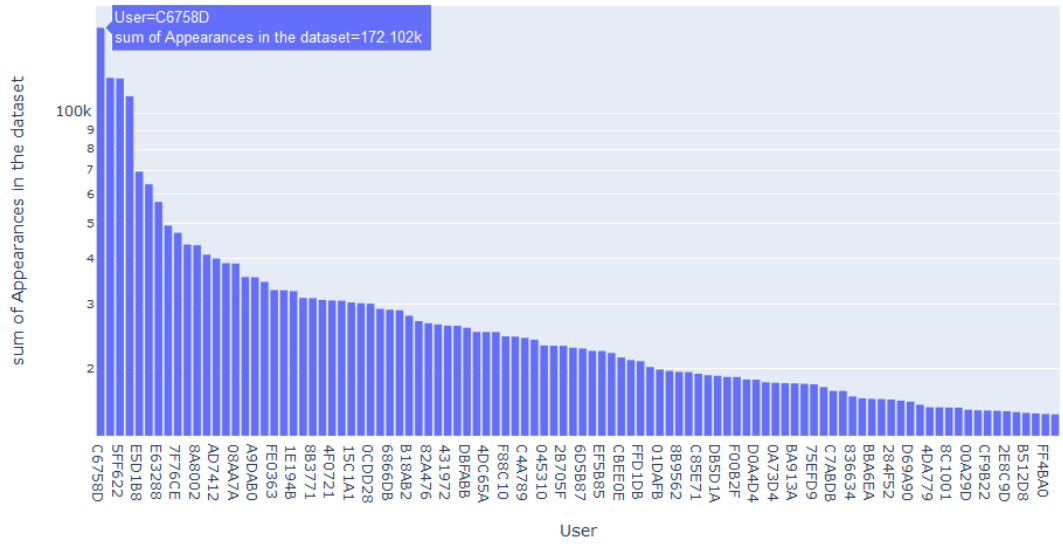


FIGURE 4.2: Presence of users in the dataset (Top 100), logarithmic scale

tribution has a very long tail with many users showing up only once. This is not the number of tweets a user made but the number of tweets his/her tweet is shown with an interaction attached.

When analysing the dataset, looking deep into the features, we have noticed some patterns we would like to explore. In the next section, features that came from that analysis are explained in detail.

4.3 Features creation

Machine learning algorithms, specifically the LightGBM classifier that we have used, needs numerical features to be trained. As we have mentioned, the classifier chooses features and distributes the data in a classification tree, trying to make the best possible splits. Thus, on the one hand we have converted the original features to numerical variables using different strategies such as One Hot Encoding, and on the other hand, we have generated new features by combining information from different samples of the training data, which has been one of the most important part of this project. All these features can be split in three different groups.

4.3.1 Tweet features

The features of the tweet refer to the text of the tweet, as well as the time in which it was done or the language in which it was written, that is, everything that refers to the context of the message and the specific text. The text, being a larger section to which we will dedicate a more detailed explanation, is found in a later section.

1. Present media: This field is formed by the GIF, Photo and Video values. So we've created 4 features:

   (a) Media: contains the total number of media elements in the tweet.

   (b) Photo: contains the number of photos in the tweet.

   (c) Gif: contains the number of GIFS in the tweet.

   (d) Video: contains the number of videos in the tweet.

   A concrete example, the tweet that contains the string *Photo\tPhoto\tVideo* is represented by the vector [3, 2, 0, 1] since it has 3 media elements 2 of which are photos and 1 video.

2. Hashtags: A hashtag is a keyword that accompanies a tweet and serves as a category or tag. In the original dataset, the hashtags were hashed so that we could not reconstruct the original word. We introduce a new feature with the number of hashtags present in the tweet.

3. Present links: This field is formed by all the links present in the tweet. They are hashed so there is no information of what link it is. The feature created is a numerical one containing how many links are in the tweet.

4. Present domains: As the previous one, this field contains the domains this tweet links to. They are also hashed so the feature created is how many domains the tweet contains.

5. Tweet type: There are four types of tweet: Retweet, Quote, Reply, and Toplevel. We encoded each of them in a categorical 0-1 variable so we obtained a 4-dimensional vector for each tweet with a single one and three zeros. For example the vector [0, 1, 0, 0] represents a Quote tweet.

6. Languages: As we can see in Figure 4.3 (in log scale), there are 66 different languages present in the original dataset. Although they're all hashed and we can't tell which one is which at first, from the text of the tweet it's easy to tell. We can see how the language with the most presence is English with some 67 million samples. Secondly, and by far the most important, we have Japanese

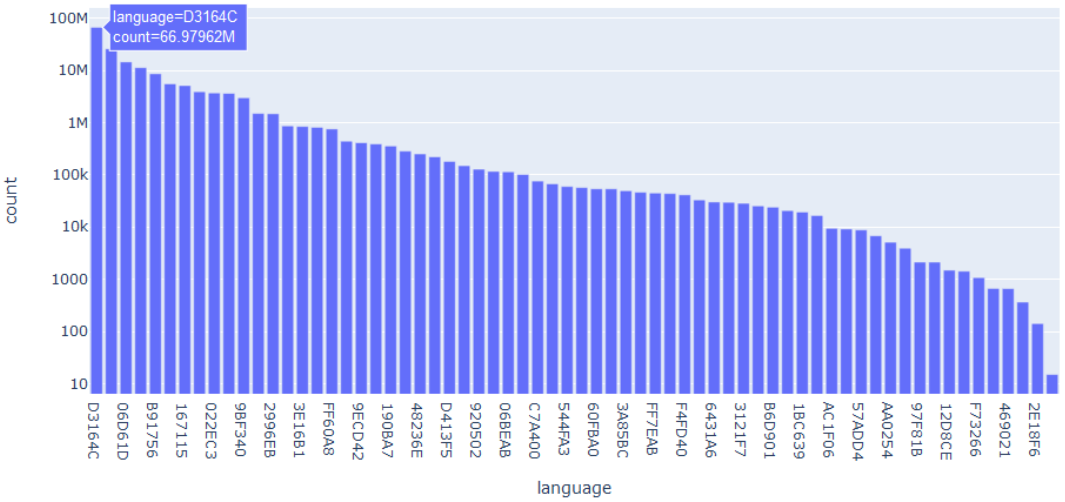


FIGURE 4.3: Languages present in the original dataset.

with about 26 million samples. In last positions we have languages with only 142 and 15 examples that are impossible to decode since the decoder was not trained using them (We will devote more time in the Bert encoder-decoder later in this Chapter). We used a One Hot Encoding of 66 positions to encode the language.

7. Time: Time is a major factor when making a publication in social networks. It is well known by the "Instagramers" and "Twitters" the hours when they have to publish their photos or messages to have the greatest impact and to reach more people. In our case, we have dealt with this problem in two different ways. First of all, we have introduced a feature with the time of the tweet. A tweet is more likely to reach more people if it's done in the afternoon rather than in the morning. On the other hand, we have also introduced a new feature with the number of weeks since the creation of the tweet to know if the tweet is no longer current.

8. Relevant words: As suggested in Suh et al., 2010 and *10 Ways To Get More Retweets*, there are words that are very influential in the capture of Likes and Retweets. This is a topic that we would have liked to explore more in depth but, due to lack of time, we could not fully analyze. What we did was an initial analysis and we incorporated the following two features: If the tweet contains the characters RT or the word "Retweet".

4.3.2 Tweet creator features

1. Verified account: The account of this user may or may not be verified. We encoded this information in a single binary feature where 1 represents that the user's account is verified and 0 represents that it is not.

2. Ratio feature: As it is suggested in Teutle, 2010, the ratio between following users and followers is a good metric to determine the importance of a node in the network. In this case, this ratio is representative of the importance of a user. We added the quotient between number of followers and number of following users as a new feature.

3. Engagement features: A key factor in knowing if two users will interact with each other is whether or not they have interacted before. We've built 4 features (one for each class we want to predict) that contain the ratio between the tweets made by one user and how many of them contain an interaction in some way by the second user.

4. Popularity: One of the best features we have implemented is popularity. Here we look at how people react to a user's tweets not because of the content of the tweet but because of the person who made it. As an example, we have many tweets from Barack Obama where the probability of Like or Retweet is very high simply because he is the one who writes it. So we created 4 new features we called them popularity with the number of reactions a user had. It is important to mention that we can not include future information, otherwise our classifier would overfit the data. In order to do so, we only considered past tweets to the one we were currently evaluating.

4.3.3 Tweet receiver features

1. Engagee follows engager: This is a simple binary feature where a 1 represents that the creator of the tweet is followed by the other user.

2. Verified account: We incorporated the same feature as we did in the tweet creator to capture if the account of the tweet receiver is verified.

3. Ratio feature: In the same way as we did for the tweet creator ratio, we created the same ratio for the tweet receiver user.

4. React to language: Let's assume that the person who tweets is A, the person who receives the tweet is B. We've created a feature that analyzes how likely it is that person B will react to person A's tweet based on the language in which the tweet is written. In the case that the tweet is written in a "foreign" language for person B (i.e. languages in which B does not react), we want to assign a very low probability to that example. This helps the classifier to be more sure which examples are negative, i.e. that they have not had any interaction. To build this feature, we have used the temporal order of the dataset that we have commented in 4.1. So, to fill this feature in the train dataset, we only used the past tweets, otherwise we would incorporate future information, which would cause an overfitting. Then, for the validation and test sets we used all the information of all the training batches.

5. Popularity: In a Similar way as we did for the tweet creator user popularity, there are users who react very strongly to almost anything they are shown. With that, we incorporated 4 new features. Like the previous features, it is important not to use future information to build popularity, so we only use information from tweets created prior to the tweet we analyze.

4.4 Language model

As discussed in Chapter 2, BERT is a very capable model that can solve various language-related tasks with few modifications to the original model. If the task is relatively simple, simply adding a layer at the end of the model to serve as a classifier is sufficient. Our problem here was not to classify the tweet texts but to learn a

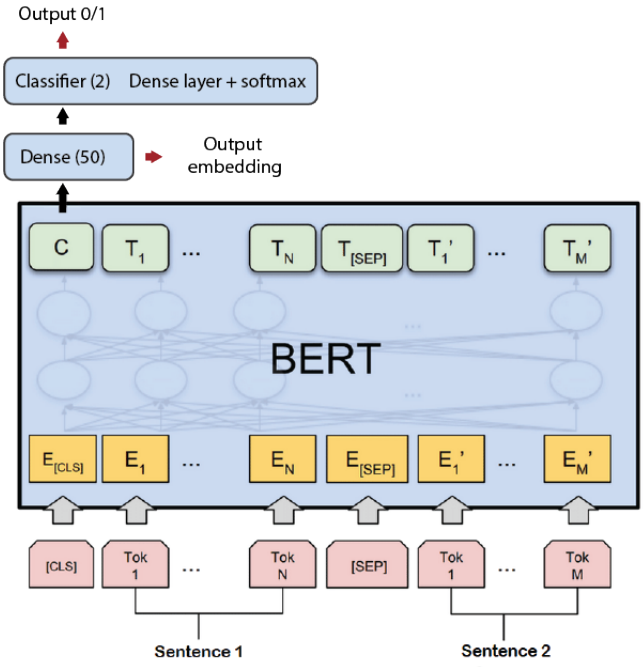


FIGURE 4.4: Fine-Tunning the BERT model. Approach 1

vector representation (embedding) in order to represent the users' preferences. Ideally, we would pass two tweets to our model and, by learning from pairs of tweets that users have liked, the model would return whether both tweets are usually liked by one same user and so if they belong to the same field of interest. So, the first approach we made was the model that we can see in Figure 4.4 where, as input it receives a couple of sentences and as output it returns if these two sentences can be of interest or not for the same user. Apart from this, as we have mentioned that we need an embedding of the text, we introduced a dense layer of 50 neurons before the classifier. Thus, once the model is trained, by inputting the same sentence twice, we can obtain its embedding of size 50. This is the vector of features we can input in the global model.

This approach did not give the expected results so we switched to a simpler but more effective approach that we can see in Figure 4.5. Instead of focusing on sentence pairs, we focused only on the content of the text. Thus, we moved from relying on users to just worrying about the text of the tweet. The model we used is very similar to the previous one, where now the input is only one sentence and for the output we use a layer of 50 neurons for embedding and another layer of 4 neurons with a sigmoid at the end to predict the probabilities of Like, Retweet, Reply and Retweet with comment. To train the model we first calculated, for each tweet of the train dataset, the ratio of positive interactions to the total number of interactions.

From this model we can extract more information since, the embedding of 50 represents the text and the output of 4 values represents the probabilities of the four

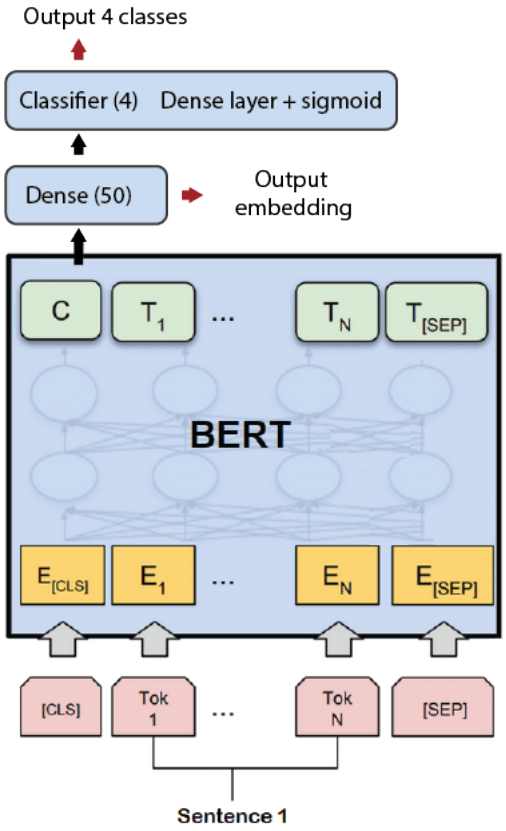


FIGURE 4.5: Fine-Tunning the BERT model. Approach 2

classes we have to predict. So in the final model, we have introduced those two value vectors, in total, 54.

4.5 Train-Val split

To do a good training of the classifier, we need a good training and validation sets. We could think that, mixing the data in a random way and make a separation of, for example, 10% for the validation, is a good idea. But a serious problem arises. What about the tweets that appear many times in the dataset? Those tweets will be in both sets which will produce a worse generalization of the classifier. Our approach, and not the only one as we will discuss in Future Work section, is to split the tweets into the two sets so that neither tweets nor users are shared. This way we can be sure that the final result in the test set is faithful to the result of our training.

4.6 Model ensemble

As we have already mentioned, the final model is a LightGBM with all the features we explained in detail, with the incorporation of text in the form of embedding. We created four copies of the model to train the four targets separately. To train each model we have used the binary Cross-Entropy as a loss function. In addition, so that the classifier does not overfit, we have limited the number of tree nodes that are built to 70 and with a maximum depth of 10 levels. We also introduced two regularization factors L_1 and L_2 to the loss with parameter $\lambda = 1$ in both cases. All

the models are trained for 1000 epochs with early-stopping if the Cross-Entropy on the validation set stops decreasing.

Additionally, we also created three versions of the same model for each class and trained them using a different set each. Then we averaged the output of these three versions to generate the final submission.

In Figure 4.6, we show the final architecture of our model. We can see the three groups of features we explained and the text features that are inputted to the four LightGBM classifiers, each one with a different output, i.e. one is returning the probability of Like and the others return the probabilities for Replay , Retweet and Retweet with Comment. We can see also that the final prediction is the average of the same model bootstraped three times to make a better prediction.

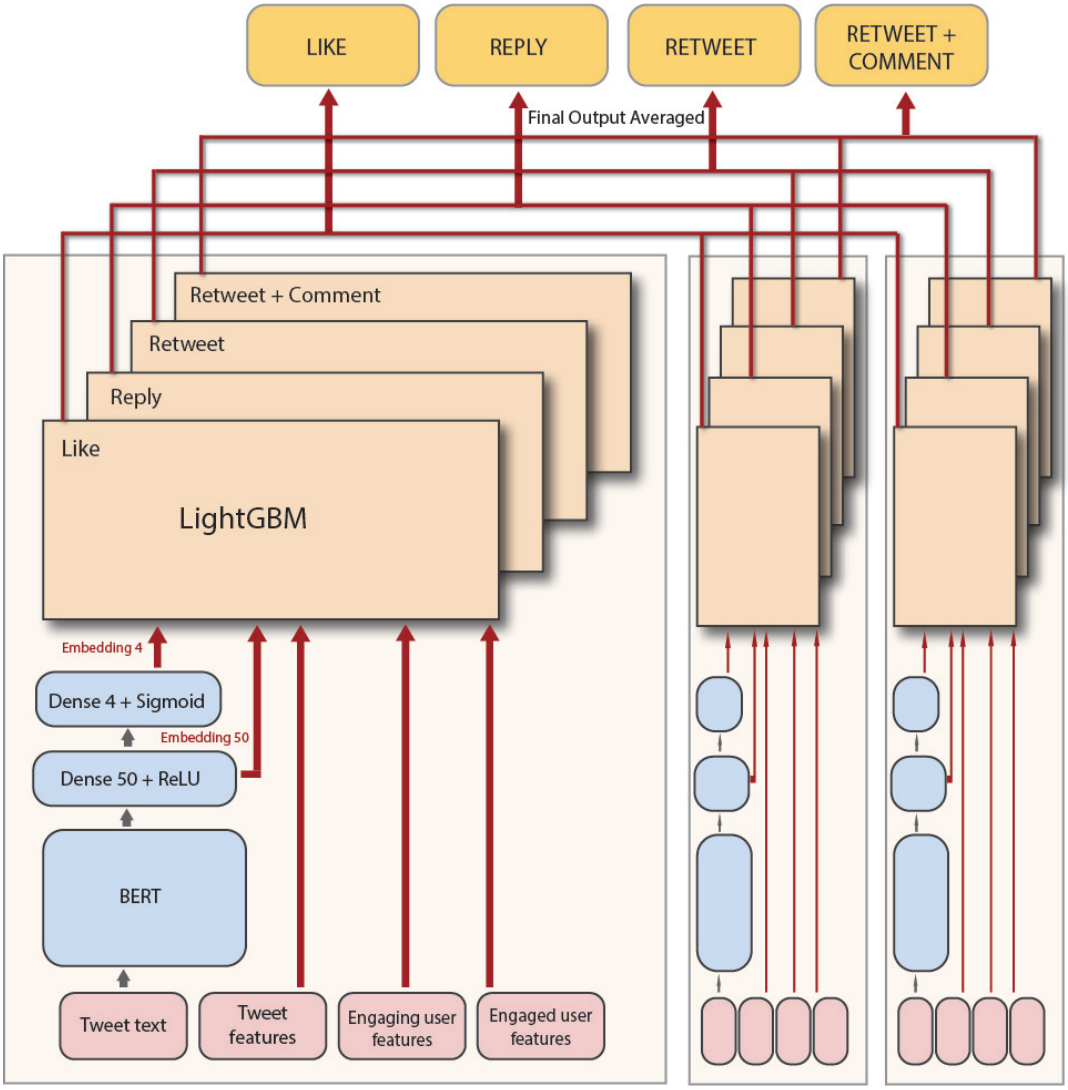


FIGURE 4.6: Architecture of the final model

Chapter 5

Results

5.1 Feature importance

Since the LightGBM is a tree algorithm, the importance of each feature can be retrieved from the model. There are two types of importances provided inside the lightgbm package, *split* and *gain*. The first one, *split*, returns the number of times the feature was used to split data when constructing the tree. The second one, *gain*, contains the amount of loss that is reduced when using this feature. In Figures 5.1 and 5.2 we can see an example of these two metrics for the *LIKE* model implemented.

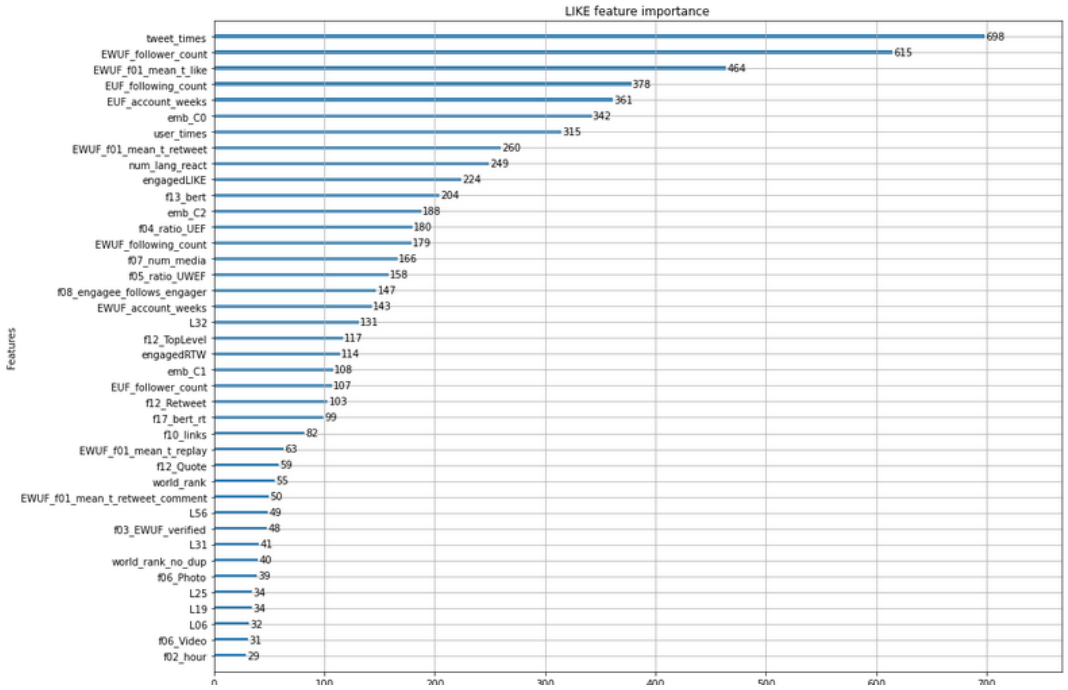


FIGURE 5.1: Split features in the LIKE model

We can see that the feature that was used the most to split is *tweet_times*. This is the feature where we count the appearances of a user in the dataset. In the next places we found the EWUF (Engaged With User Features i.e. tweet creator) and EUF (Engaging user features i.e tweet receiver). These are the popularity features we commented in Chapter 4. We can also see that between the top features it appears the *emb_C0* that is the first component of the 4-dimensional vector we used in the language embedding which corresponds to the LIKE embedding so we know that, this way, the embedding is working. In this same way we can see that the gain these features give is highly related with the order of that feature in the split plot. We can see in the *gain plot* that the feature that gave the most gain is the *emb_C0*, followed

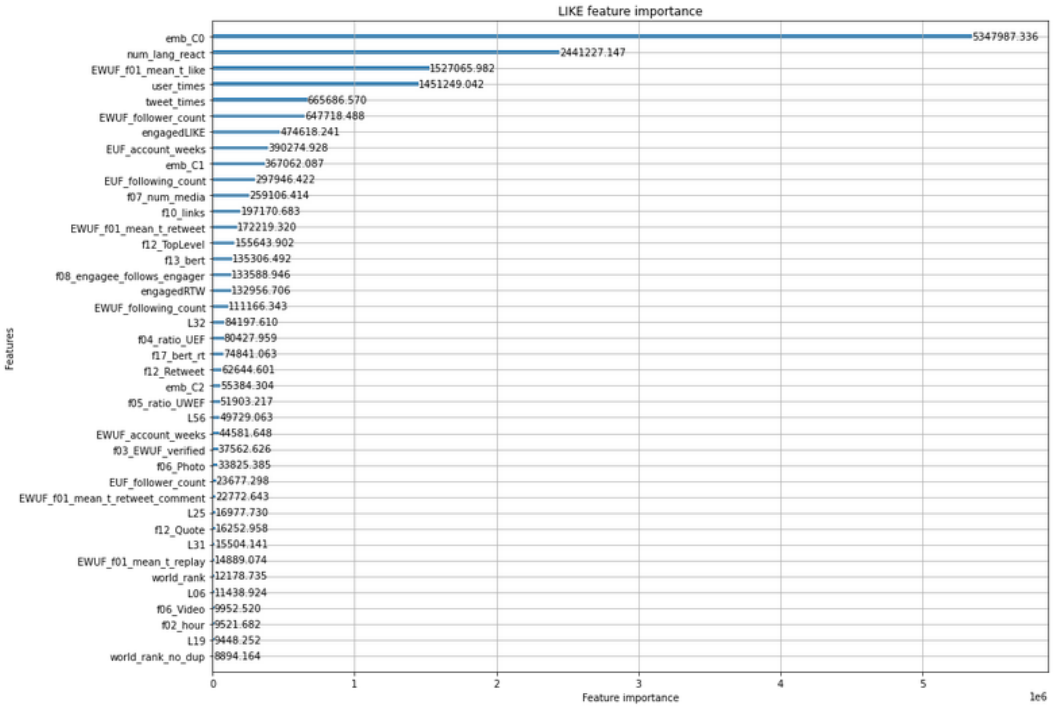


FIGURE 5.2: Gain features in the LIKE model

by *num_lang_react*. This one is controlling previous interactions of the users in the language the tweets were written, so it is very important to discriminate between liked and not liked tweets (specially relevant in the not liked ones).

5.2 Training the model

In Figure 5.3 we can see how the model is performing in the test set regarding Cross-Entropy and Precision Under the Curve metrics. We can see that, after 100 epochs the values reached are close to the minimum and maximum in the plots of both rows.

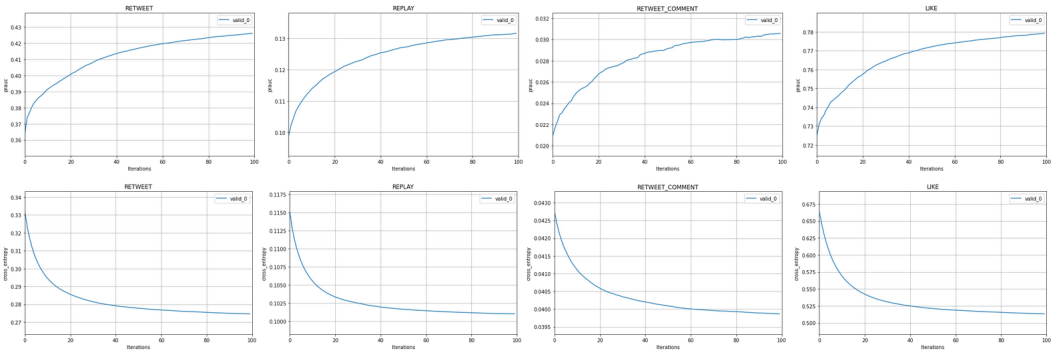


FIGURE 5.3: Evaluation metrics for the final model

5.3 Competition Results

One of the goals of this project was to generate submissions and improve the model in next iterations. We can see in Figure 5.4 how we were improving during the challenge. The left hand side plot shows the mean, for the four targets, of the Relative Cross Entropy metric. Each point in the curve represents a different submission. In the same way, the right hand side plot shows the mean evolution for the Precision-Recall Area Under the Curve metric. In Figure 5.5 we can see the final position of

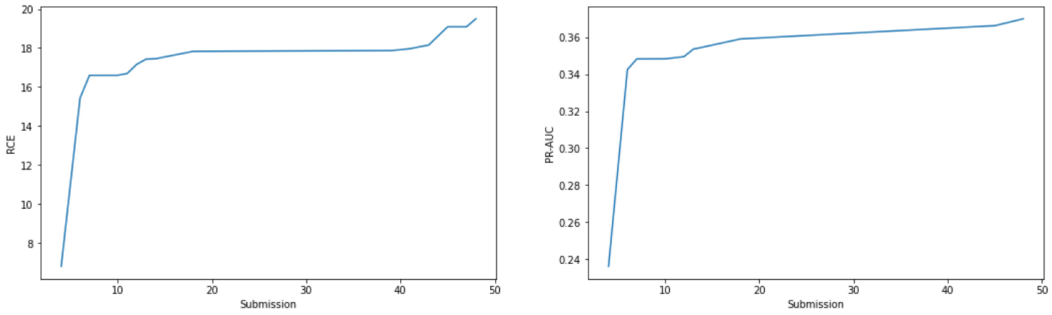


FIGURE 5.4: Evolution of the mean for the RCE and PR-AUC metrics during the competition

our team *Not_Last_Place* in the private leaderboard. From a total of 45 participants in the final phase of the challenge we ended in the 9th position.

Competition Results

Rank	Team/User Name	Overall Score	Method Name	PRAUC Retweet	RCE Retweet	PRAUC Reply	RCE Reply	PRAUC Like	RCE Like	PRAUC RT with comment	RCE RT with comment
1	NVIDIA RAPIDS.AI	9	rapids.ai v2	0.6111	37.91	0.2185	24.23	0.9108	53.01	0.0796	17.40
2	learner	14	Layer 6 AI	0.5395	30.96	0.2218	21.94	0.7761	25.42	0.0757	14.61
3	Team Wantedly	18	Team Wantedly	0.5266	30.06	0.1918	20.44	0.7716	24.76	0.0724	14.86
4	learner_recsys	20	ffm+sgd+prior+embeddings	0.4529	22.50	0.1161	10.42	0.7221	18.28	0.5037	-0.04
4	BanaNeverAlone	20	last_test_sub_2	0.5042	27.21	0.1850	18.71	0.7531	21.20	0.1237	8.34
4	AskMiso & CLIP	20	askmiso & CLIP v1	0.5135	28.57	0.2069	21.09	0.7650	23.65	0.0644	12.79
5	wsm_LLLLL	21	Method 1: Expected + Tweet	0.5516	-0.04	0.5135	-0.02	0.6666	3.52	0.5037	-0.05
6	POLINKS	22	baseline	0.5516	-0.03	0.5135	-0.05	0.7131	-0.00	0.5037	-0.00
6	Los Trinadores	22	factored.ai Golduck	0.3673	16.93	0.5135	-0.04	0.7310	20.09	0.5037	-0.00
7	Better	23	Better	0.4992	26.98	0.1879	19.35	0.7484	21.17	0.0623	12.10
8	myaunraitau	27	myaunraitau	0.4916	25.65	0.1622	16.76	0.7387	19.74	0.0523	9.82
9	Not_Last_Place	28	last-combined	0.4891	21.80	0.1761	18.60	0.7511	17.64	0.0516	10.30
10	narsil	29	quite random	0.4865	24.21	0.1570	14.98	0.7464	20.01	0.0527	9.23
11	Chilling	30	AI	0.3088	12.33	0.1137	11.82	0.6602	12.70	0.5037	-0.04
12	stathisch	34	Basic NN	0.4370	21.22	0.1293	13.83	0.7199	17.58	0.0309	6.43
13	Marysia	35	sample method	0.3232	11.81	0.1116	8.79	0.6038	6.65	0.5037	-0.00

FIGURE 5.5: Final leaderboard of the competition

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We created a model capable of performing really good in the challenge and, for that, we were in the top positions of the leaderbord until the last week where we struggled with some problems regarding the final data provided by Twitter. The dataset was reduced each week and we needed to adapt to this situation at the end.

We can see that we satisfied all the goals proposed at the beginning of this document:

- We extracted multiple features of the dataset, encoding them into categorical variables using the One Hot Encoding technique. We also applied logarithms to some of the features in order to not take into account the exact value (possible way our model may overfit the training data) but the trend of the data.
- We created multiple features such as the popularity, engagement and language reaction which past data was combined in order to predict the future.
- We developed a Language Model using BERT, a State-of-the-Art model from 2018 that was able to predict the engagement of a tweet based only in the text. BERT is one of the latest models that supports multilingual text.
- We created a classifier (LightGMB) were all the features created were grouped. Here the predictions of the language model were used in order to create two text embeddings.
- We submitted over 50 solutions to the public leaderboard were we were in the top 3 positions for several weeks. For the last phase of the challenge, we submitted 2 more solutions to the private leaderboard to obtain the final 9th position.

6.2 Future Work

Since the competition was only opened for 3 months, we could not do all the things we wanted to do in order to perform better. Things that would have probably improved the model are:

- Creating a better embedding with another task that involves the users and not just the text. This was one of the parts were we struggled the most because text models are very heavy and take a lot of time to train. The model we trained with pairs of sentences was a great approach but did not perform as we expected.

- More features of the users. We did not exploit in its full capacity the users and their connections. Algorithms such Personalized PageRank and Network related metrics could give a good improvement of the results.
- Since we worked results-based in to obtain the best score in the competition, we did not spend all the required time analyzing the features we were creating. Further analysis of these features could be performed in order to discard non-relevant ones or to exploit features that are under its maximum potential.
- Other different separations for the dataset could be considered, for example, separating the data according to the day the tweet was made, since the train dataset contains a week of tweets, we could keep the last day for validation and the first days for training.
- Dependencies between the targets could have been analysed in more detail. For example, the target Retweet+Comment is always associated with a Retweet and it's something we haven't taken into account. A model that uses the other interactions as input could give better results.

Appendix A

Technicalities

A.1 Github Repository

All the code necessary to run this project can be found in the following github repository

https://github.com/perecasxiru/Recsys2020_final.git

There are two main files which are the only necessary to be executed in order to reproduce the results. The execution order that has to be followed is the following.

1. **10_Final_Notebook_M4.ipynb**: It contains the main execution. This notebook does the following operations:
   (a) Create the folder *train_batches* with the data split in files of 1 million samples.
   (b) Create the folder *train_batches_timestamp* where all the data is organized in files according to the hour the tweet was written. Each file is also sorted internally.
   (c) Transforms given test files to csv formats and store them into disk.
   (d) Create *popularity* feature.
   (e) Create *hashtag popularity* features.
   (f) Create *engagement* features.
2. **10_Generate_Bert_Embedding**: It contains all the code related with the text processing and text model. It needs to be executed from top to bottom. At the end, there are some configuration cells that can be fully customized in order to generate embeddings for any other choosen file.
3. **10_Final_Notebook_M4.ipynb**: After executing all the BERT part of the process, now we can merge the embeddings with the other features and follow the final steps:
   (a) Run *Transformation* to transform all the dataset into numerical variables
   (b) Split into train and validation sets.
   (c) Train the models.
   (d) Apply the models to predict the test.
   (e) Generate the submission.

Bibliography

10 Ways To Get More Retweets. https://www.quicksprout.com/twitter-retweets/. Accessed: 2020-06-18.

Agarwal, Apoorv et al. (2011). "Sentiment analysis of twitter data". In: *Proceedings of the workshop on language in social media (LSM 2011)*, pp. 30–38.

Belli, Luca et al. (2020). "Privacy-Preserving Recommender Systems Challenge on Twitter's Home Timeline". In: arXiv: 2004.13715 [cs.SI].

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. DOI: 10.1145/2939672.2939785. URL: http://dx.doi.org/10.1145/2939672.2939785.

Devlin, Jacob et al. (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv: 1810.04805 [cs.CL].

Ke, Guolin et al. (2017). "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: ed. by I. Guyon et al., pp. 3146–3154. URL: http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

Kouloumpis, Efthymios, Theresa Wilson, and Johanna Moore (2011). "Twitter sentiment analysis: The good the bad and the omg!" In: *Fifth International AAAI conference on weblogs and social media*.

Li, Chenliang et al. (2012). "Twiner: named entity recognition in targeted twitter stream". In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pp. 721–730.

Pak, Alexander and Patrick Paroubek (2010). "Twitter as a corpus for sentiment analysis and opinion mining." In: *LREc*. Vol. 10. 2010, pp. 1320–1326.

Raffel, Colin et al. (2019). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. arXiv: 1910.10683 [cs.LG].

Suh, B. et al. (2010). "Want to be Retweeted? Large Scale Analytics on Factors Impacting Retweet in Twitter Network". In: *2010 IEEE Second International Conference on Social Computing*, pp. 177–184.

Teutle, A. R. M. (2010). "Twitter: Network properties analysis". In: *2010 20th International Conference on Electronics Communications and Computers (CONIELECOMP)*, pp. 180–186.

Zhu, Yukun et al. (2015). "Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books". In: *arXiv preprint arXiv:1506.06724*.