NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

POSTGRADUATE STUDIES PROGRAM

MASTER THESIS

Hate Speech Detection on Twitter: A Social-Aware Approach

Georgios C. Apostolopoulos

Supervisor: **Alexios Delis**, Professor NKUA

ATHENS

MARCH 2021

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εύρεση Υβριστικού Λόγου στο Twitter: Μια Προσέγγιση με Ανάλυση Κοινωνικών Δικτύων

Γεώργιος Χ. Αποστολόπουλος

Επιβλέπων: **Αλέξιος Δελής**, Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΜΑΡΤΙΟΣ 2021

MASTER THESIS

Hate Speech Detection on Twitter: A Social-Aware Approach

Georgios C. Apostolopoulos
RN: CS2180003

SUPERVISOR:

Alexios Delis, Professor NKUA

THESIS COMMITTEE:

Alexios Delis, Professor NKUA
Alexandros Ntoulas, Assistant Professor NKUA
Panagiotis Liakos, Postdoctoral Researcher NKUA

MARCH 2021

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Εύρεση Υβριστικού Λόγου στο Twitter: Μια Προσέγγιση με Ανάλυση Κοινωνικών Δικτύων

Γεώργιος Χ. Αποστολόπουλος

AM: CS2180003

ΕΠΙΒΛΕΠΩΝ:

Αλέξιος Δελής, Καθηγητής ΕΚΠΑ

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ:

Αλέξιος Δελής, Καθηγητής ΕΚΠΑ
Αλέξανδρος Ντούλας, Επίκουρος Καθηγητής ΕΚΠΑ
Παναγιώτης Λιάκος, Μεταδιδακτορικός Ερευνητής ΕΚΠΑ

ΜΑΡΤΙΟΣ 2021

ΠΕΡΊΛΗΨΗ

Η ανάλυση συναισθημάτων αναφέρεται στη διαδικασία λήψης πληροφορίας σχετικά με την αντίληψη του χρήστη για ένα προιόν, μια υπηρεσία, μια διασημότητα, έναν πολιτικό ή ακόμα και μια γενικότερη ιδέα ή συμπεριφορά. Στην παρούσα διπλωματική εργασία, θα γίνει προσπάθεια εντοπισμού υβριστικού λόγου στο Twitter. Το μίσος, είναι ένα πολύ ισχυρό συναίσθημα, καθώς, όταν εκφράζεται χωρίς περιορισμό, δύναται να καταστρέψει την ποιότητα μιας συζήτησης. Επιπλέον, το μίσος συνήθως συνοδεύεται από ύβρεις κι απειλές. Συνεπώς, η προσπάθεια εντοπισμού του μίσους στα κοινωνικά δίκτυα όπως το Twitter, είναι μια διαδικασία που πρέπει να υλοποιηθεί προσεκτικά. Ωστόσο, δεν είναι εφικτό να πραγματοποιηθεί χειροκίνητα, καθώς, στις μέρες μας, η κίνηση στα κοινωνικά δίκτυα αυξάνεται κι όλο και περισσότεροι άνθρωποι χρησιμοποιούν διαδικτυακές εφαρμογές κι εργαλεία. Κατά συνέπεια, μέσω αυτοματοποιημένων μεθόδων, η προσπάθεια αυτή δύναται να απλοποιηθεί. Επιπροσθέτως, έρευνες έχουν υλοποιηθεί σχετικά με τα κατάλληλα εργαλεία για την απλούστευση του συγκεκριμένου έργου, με την πλειοψηφία να χρησιμοποιεί μηχανική μάθηση. Στην παρούσα εργασία, προσπαθούμε να εντοπίσουμε το μίσος στο Twitter μέσω υφιστάμενων μεθόδων και τεχνικών. Παράλληλα, θα ακολουθήσουμε μια τεχνική βασισμένη στην ανάλυση κοινωνικών δικτύων, αξιοποιώντας τα χαρακτηριστικά του χρήστη (αριθμός ακολούθων, αριθμός tweets κλπ.) και λαμβάνοντας υπόψη όλες τις πιθανές μετρικές που θεωρούνται σημαντικές. Κλείνοντας, επιχειρούμε συνδυασμό των ανωτέρω τεχνικών, με σκοπό να διαπιστωθεί κατά πόσο είναι εφικτή μια σημαντική βελτίωση στη διαδικασία εύρεσης υβριστικού κειμένου.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Αυτόματος εντοπισμός υβριστικού λόγου

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: υβριστικός λόγος, twitter, κατηγοριοποίηση με χαρακτηριστικά χρήστη

ABSTRACT

Sentiment analysis refers to the process of retrieving information about a user's perception of a product, service, celebrity, politician or even a general idea or behavior. In the current thesis, we will examine Twitter's tweets and attempt to identify hate speech in them. This specific sentiment is very powerful, as when used without measure, it can severely destroy the quality of a conversation. Furthermore, hate is most often combined with insults, abuse and threats. Thus, the effort to identify hate in social media, like Twitter, is a task that needs to be done carefully. However, it is not feasible for humans to do this process manually, as nowadays, the traffic in social media augments and more people use online applications and tools. With an automated approach, this effort can become significantly easier. Additionally, research has been conducted on what tools can be used to accomplish this task and the majority uses machine learning. In this research, we investigate hate-speech detection on Twitter using methods that already exist. In addition, we follow an approach, based on social networks analysis, making use of user's profile (number of followers, number of tweets etc.) and any useful metrics we can think of. Finally, we combine those approaches to determine whether we can achieve a significant improvement in the task of hate speech detection.

SUBJECT AREA: Automated hate speech detection

KEYWORDS: hate speech, twitter, user features classification

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Alexios Delis and my tutor, Dr. Panagiotis Liakos, for their undivided attention and their invaluable guidance in formulating the research, providing me with the tools I needed to successfully complete my dissertation. Lastly, I am deeply indebted to my family for their endless support.

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

The massive use of social media has led to a significant increase in hateful activities, as technology offers anyone the ability to express their opinion publicly, using online tools and applications. This infrastructure is often exploited by users who benefit from posting unfiltered feed of messages on social media. Therefore, it would be beneficial to analyze general hateful behaviors of large groups or individuals, in order to be able to discourage them or eliminate them before they occur. However, this phenomenon can't be manually pursued, as it is not a scalable task and it would require a lot of resources. In this effort, to avoid vast manual undertaking, a lot of investigation has been launched in the field of automatic hate speech detection.

On Twitter, specifically, hatred is more common when controversial events take place in real life and is expressed with insults, abuse or even death wishes. In our work, we create a dataset containing tweets that refer to the Presidential Elections in the United States of America in November 2020 and that specifically contain the word "Trump". The two candidates were Donald Trump for the Republicans and Joe Biden for the Democrats. During the election campaign, both parties signified the importance of them being elected. Due to the fact that the polls were unable to predict the final winner, the situation in the U.S.A remained highly uneasy and the public was divided. It was a very turbulent period for the U.S.A and the election outcome was so unpredictable that it caused a massive use of hateful language in the social media. We focus on the problem of classifying the tweets as either hateful or not hateful. We define hateful tweets as those that contain abusive language targeting individuals or groups of people. Similar inspections [2], have already been conducted, using deep learning and neural networks to classify tweets based on their textual context.

This thesis makes the following contributions: a) We manually distinguish the hateful tweets and obtain an analogy of 21% hate and 79% non-hate, in a total of 1312 tweets. We experiment with both text classification of the tweets and a combination of text and user features classification. The latter constitutes the innovation of the thesis and it refers to a complex model specifically designed for the task of automatic hate speech identification. This model is a neural network with two inputs. On the first input, we feed the network with the tweets in form of word vectors. On the second input, we feed the network with user-specific data. b) We investigate whether the baseline methods of pure text classification can be enhanced adding user features. We supplement the model with various attributes, in an effort to evaluate their influence on hate detection. Among the attributes, we examine users' total tweets, the number of followers they have, the number of users they follow and the volume of tweets they have posted. In addition, we examine the number of tweet's retweets and likes and, finally, if the tweet is a response to another tweet or user.

At the same time, we present appropriate statistics to evaluate the quality of our data. c) Consequently, we combine all those features, in an effort to determine which of those can improve the classification outcome. We repeat the process with equally weighted classes, maintaining 50% of hateful tweets. For more details, the github repository[1] provides the code that supports our approach.

1.1 Thesis structure

In Chapter 2, we elaborate the dataset's production process from Twitter, using `tweepy` library, collecting tweets for one month before the U.S elections. In Chapter 3, we use textual classification on our dataset, using both a Convolutional Neural Network (CNN) [7] and a Recurrent Neural Network (RNN) [6]. We achieve an accuracy of approximately 84% for the CNN and 81% for the RNN. We experiment with Global Vectors (GloVe) [11] and we notice better results when using GloVe instead of random embeddings for our models. In Chapter 4, we investigate user features and present statistics from our dataset. We inspect which attributes can have significant impact on the classification outcome. Finally, in Chapter 5, we combine textual classification and user features classification in order to determine whether we can achieve better results. We achieve an increase in classification accuracy of approximately 3% for both CNN and RNN models. We observe that the best metric for both the CNN and RNN is a combination of the number of user's followers and a flag that indicates if the tweet is a reply. Using those features we achieve an accuracy of approximately 87% for the CNN and 85% for the RNN. Therefore, we implement a complex network that uses as input both the tweet's text and the attributes we desire. Lastly, we conclude in Chapter 6.

[1] https://github.com/giorgos-apo/hate-speech-detection-using-user-attributes

2. DATASET

In machine learning, a crucial task in order to achieve high classification accuracy, is data preprocessing. Before deciding to create a custom dataset, we experimented with existing datasets from previous works [16], [12]. Prior work is basically conducted in an effort to distinguish hateful comments using the tweet's text. In our case though, we are interested in the user's attributes that may be valuable for the classification process. Thus, we explain in Section 2.1, why the existing datasets were not helpful and why it was necessary to develop a new one.

To begin with, we clarify that our purpose after the classification, is to be able to answer questions like "is the number of followers a possible indicator of someone's abusive language?". Thus, our dataset consists of 1312 tweets, extracted a month before the elections in the United States in November 2020. Each tweet includes the term "Trump" and the purpose of our research is to automatically classify it as either hate or non-hate.

2.1 Problems with existing datasets

While experimenting with existing datasets, we face two major issues, that create two corresponding necessities. Before analyzing the difficulties, we must point out that the percentage of hateful tweets in the datasets was 20%.

2.1.1 Multiple tweets from the same author

The first issue, is that the dataset we have examined [16] consists of more than one tweet per user. This is, in general, a source of major shortcomings. Experimenting with text classification, a researcher shall obtain tweets from multiple users, to make sure their implementation is adjustable to different speaking habits, as every person expresses his sentiments differently in their speech. Simultaneously, the same drawback applies in our research, in which we establish a social-aware approach and, therefore, we examine user's characteristics. More specifically, while evaluating the original dataset, we found approximately 12K tweets. However, there were only 1.5K distinct tweet authors. There was a specific author that had written 3.8K tweets (approx. 30% of the dataset). It is obvious that this author introduces significant bias to the dataset, since his attributes appear in 30% of the tweets. To comprehend the issue consider Table 2.1, in which we only keep the tweet's id, the number of author's followers and the tweet's label, which is 1 (hate). Next, notice that all the tweets are labeled as hateful. Furthermore, the number of author's followers is 289 and is the same for all the tweets in the table, since they share the same author. The ML model reads an input with 5 tweets. In every input there are 289 user followers and the output label is 1 (hate). Therefore, the model will be trained to respond

that if a user has 289 followers he is more likely to use abusive language. However, this is not true since it has been decided using only one single user for the training process. It is obvious that a user who has authored 30% of the tweets, has high impact on the dataset. The reason behind that, is that if a user constantly uses abusive language, it can lead to a biased result. Therefore, we can safely state that when a dataset includes multiple tweets from one user with the same label, the classification outcome is most likely biased.

Table 2.1: Hateful tweets from the same author.

tweet_id	user_followers	is_hate
120291112	289	1
231192998	289	1
231192998	289	1
298787102	289	1
293700112	289	1
176251992	289	1

2.1.2 Hateful tweets from a few authors only

After removing duplicate users from the dataset, the second problem that occured was the amount of people that use hateful language in their tweets. As mentioned earlier, if we keep one tweet per author from the dataset, we shall retain approximately 1.5K tweets. From those, we expect to retain an analogy similar to the one we had before removing the duplicate users, thus, approximately 20% hate. However, from the 1.5K tweets, only 15 were labeled as hateful (1% of the tweets). This clearly states that we don't have sufficient data, in order to identify hateful tweets. People express themselves with a unique way when writing tweets. Therefore, it is not safe to collect hateful tweets by only a few users. Neural networks are trained by the inputs that we feed them. Thus, we must provide input data that cover different styles of writing. Otherwise, the models will be trained to detect hate speech in very few occasions.

2.2 Dataset information

In the following subsections, we explain in detail the information extraction process, in order to obtain more representative samples for our dataset in its final form.

2.2.1 Step 1: first scraping

Using `tweepy` [14] python library and utilizing the Twitter API, we follow the following process:

1. we use the term “Trump” to collect a number of tweets, keeping the tweet’s id and text from Twitter.
2. we make sure we keep only one tweet per author, dumping tweets if they share the same `user_id` with a previous one.
3. we clear out possible duplicate tweets.
4. we read the tweets and label them as hateful or not (binary 1/0).
5. we repeat the previous steps, running the scrapper day and night, because there is an important time difference between Europe and the USA.

2.2.2 Step 2: information extraction

After keeping the id and the text for each tweet, we use the Twitter API again, to collect information for all our tweets. This process doesn’t occur simultaneously with the first scraping, because we want to offer a proper amount of time, so as our tweets get likes and retweets. Otherwise, those values would be very low. Thus, our dataset consists of one row for each tweet and it includes, in each row, the following information:

- the tweet’s id (`tweet_id`).
- the tweet’s text (`tweet_text`).
- the tweet’s label (`is_hate`).
- tweet’s total retweets (`tweet_retweets`).
- whether the tweet is a reply to another user. (`tweet_is_reply`).
- the user’s id (`user_id`).
- user’s total tweets (`user_total_tweets`).
- user’s number of followers (`user_followers`).
- the number of users followed by the tweet’s author (`user_following`).
- the number of likes (`tweet_likes`).

Lastly, we provide an example of one row from our dataset in the Table 2.2.

Table 2.2: Sample row from the dataset.

Feature	Value
tweet_id	1312328990565175296
tweet_text	President Tsai @iingwen wishes U.S. President Donald #Trump a speedy recovery
is_hate	0
tweet_retweets	15
tweet_is_reply	0
user_id	1080742203633262592
user_total_tweets	3352
user_followers	3411
user_following	1542
tweet_likes	1

3. TEXT CLASSIFICATION

The initial benchmark for our dataset, is to check how our text classification results compare to those in [2]. We inspect our dataset's behavior using a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN). For the RNN, we specifically use a Long Short-Term Memory neural network (LSTM). At the same time, we make use of both random embeddings and GloVe embeddings [11]. In this chapter, we explain how we preprocess the text. We present Global Vectors (GloVe) and describe how we use it to convert tweets into vectors. Finally, we analyze each model and the results it produces.

3.1 Why GloVe?

GloVe stands for Global Vectors and refers to a model that converts sentences into vectors, in order to feed them to a neural network. Despite the fact that anyone can create custom embeddings, GloVe offers the ability to use pretrained data, where words are represented in vectors of 25, 50, 100 or 200 dimensions. For our research, however, the machine we use is not capable of supporting more than 50-dimensional vectors because of RAM limitation. Therefore, we experiment with the 50 dimensions file. GloVe, in general appears to behave better [8] than random embeddings because they are trained on billions of monolingual examples. Finally, GloVe is used to calculate the weights of the vectors from pretrained words, using the following code (not strict format):

```
# embedding_dim = 50
# vocab_size = number of words in our vocab
# token = our vocab
def produce_glove_vector_matrix(embedding_dim, vocab_size, token):
    glove_file = <path_to_50d_glove_file>
    glove_vectors = dict()
    # we create a dict from glove file
    # word -> vector
    for line in glove_file:
        values = line.split()
        word = values[0]
        vectors = np.asarray(values[1:])
        glove_vectors[word] = vectors
    # we compare the glove dictionary with our dictionary
    word_vector_matrix = np.zeros((vocab_size, embedding_dim))
    for word, index in token.word_index.items():
        vector = glove_vectors.get(word)
```

```
            if vector is not None:
                word_vector_matrix[index] = vector
        # we return the weights
        return word_vector_matrix
```

Thus, we provide an execution sample of the above code. We assume we are given a list of vectors, which is represented by the "token" parameter of the function. Then, we also pass the embedding dimension (equal to 50). The embedding dimension refers to the vector size of the GloVe pretrained data (i.e each word is represented by a 50 dimensional vector). Lastly, we pass the vocabulary size to the function. The function repeats the same process for every word of every sentence, in order to calculate the weight of each word. Assume the phrase "trump was the president". The previous method returns a list of weights. In our example, it returns a list of 4 arrays, one for each word. We point out that each array is 50-dimensional. Therefore, we retrieve the following information.

```
[[-1.1660e-01  6.0989e-01  4.6737e-01  3.6106e-02  9.8803e-02 -1.8909e-01
    -3.1798e-01  2.8476e-01 -2.5497e-03 -5.9989e-01  3.2075e-01  7.0234e-02
    -1.8843e+00 -1.7281e-01  4.6168e-01 -3.3900e-01 -2.3952e-01  3.9531e-01
    1.1337e+00  4.1996e-01  2.3564e-01 -1.0695e+00 -1.2222e-01  4.6364e-02
    1.3494e-02  6.5034e-01  9.5719e-01  1.7752e-01 -1.5586e-01 -7.3075e-01
    3.4446e-01 -1.9114e-02 -5.4315e-01 -1.0741e+00 -6.9212e-02  1.0205e+00
    4.8992e-02  7.9524e-02  6.5599e-01 -3.3597e-01  4.3663e-01  5.4676e-02
    -3.0074e-01 -6.4048e-01  5.2968e-01  9.3046e-02  4.4165e-01 -4.1807e-01
    -5.1880e-01  2.5806e-01]
[-3.5947e-02  9.1527e-01  8.4885e-01 -7.5355e-01 -5.0724e-01  3.1926e-01
    -4.6347e-02 -8.8508e-02 -1.1041e-01 -3.7209e-01  2.3058e-01  6.0452e-01
    -5.9729e+00 -1.9737e-01  4.5525e-02 -2.7410e-01  6.6767e-01  4.0142e-01
    -1.0232e+00 -1.1479e-01  5.2377e-01  6.5502e-01 -1.2134e-03  1.6294e-01
    8.8063e-01  6.5330e-01 -4.8531e-01 -9.1314e-01 -3.7358e-01 -5.4828e-01
    -4.6986e-01  9.1771e-01 -3.9752e-01 -1.0546e+00  3.1071e-01  2.4028e-01
    -8.8314e-01  1.1334e-02 -2.4900e-02 -4.6060e-01 -7.2345e-01 -9.2369e-02
    -4.2489e-03  6.7815e-01  2.6334e-02 -4.7434e-01  9.5158e-01 -2.8755e-01
    -6.0420e-01 -3.5384e-03]
[ 2.5320e-01 -1.4884e-02  5.9371e-01  1.5902e-01  1.2754e-01  2.2428e-01
   8.9421e-01  3.6396e-01 -3.1339e-01 -5.1857e-01  2.9637e-01 -4.1098e-02
  -6.4555e+00  3.2260e-01  3.7280e-01 -6.1690e-01  4.6744e-01  5.0600e-01
   3.1950e-02  1.0155e-01 -1.9615e-01  1.3364e-01 -2.7140e-01 -4.1728e-01
   7.7940e-03  1.3573e-01 -7.2992e-02  2.5208e-01  5.1148e-01  1.5120e-01
```

```
   8.4398e-02 -2.4791e-01 -1.5913e-01  1.5005e-01  7.7243e-01  3.6632e-01
  -9.8310e-02 -6.4317e-02 -7.1983e-04 -1.5231e-01 -1.4604e+00 -3.1696e-01
  -4.1762e-01  7.3363e-02  3.2043e-01  3.4324e-01  1.0895e-02 -2.8932e-01
   4.5493e-01  1.8659e-01]
[ 1.0231e+00  6.2470e-01  8.6370e-01  3.6068e-01 -4.7532e-01 -2.1307e-01
  -9.7116e-01  9.1805e-01  1.6307e-01 -7.9805e-01 -4.1119e-01 -1.1976e+00
  -3.6883e+00 -1.3654e-01  9.5745e-01  3.2482e-01 -1.0018e+00  5.6574e-01
   5.9419e-01 -1.3655e-01 -2.9172e-01  1.1794e-02 -5.8037e-01 -5.6299e-01
   1.0961e+00  5.6508e-01  8.2377e-01 -1.6840e-01 -1.8427e-02  3.6756e-03
   5.2218e-01 -2.9420e-02 -3.5490e-01 -3.8193e-01  1.3838e+00  1.3739e+00
   2.3569e-02  7.7102e-01  3.8753e-01 -9.7413e-01 -2.0959e-01 -4.6311e-01
   3.8450e-01  3.7239e-01  2.8644e-01 -7.9481e-02  1.8964e-01 -2.8996e-01
  -4.0469e-01 -4.9608e-01]]
```

Having the above information, allows us to use weights in our models' embedding layers. This constitutes a feature provided by `Keras` embedding layer. In our work, we experiment with using weights in the training process and compare the results with those we get without using weights.

3.2 Text preprocessing

For text preprocessing, we mainly use the same concepts with the prior work [2] and we add some new features. Specifically for each sentence we maintain the following approach. The first step is to use a script from GloVe[1], in order to clean the text and avoid emojis, hashtags and capital letters. Afterwards, we divide the sentence into a list of words, avoiding punctuation and stopwords[2]. Furthermore, a vocabulary is produced using the `keras.Tokenizer()` method. This function stores every distinct word in our tweets using a key-value pattern. Thus, each word is represented by an integer. Next, we use `keras.Tokenizer.fit_on_texts()` method to convert lists of words into lists of integers, or equivalently, to vectors. Finally, we find the maximum length between the lists and we pad zeros to those that contain less integers, in order to have fixed length for all vectors. This step is mandatory for neural networks, in order to operate on the data. At this point, to better explain our process, we obtain a tweet from the dataset and we will use it to elaborate the previous steps. After that, imagine that we repeat the following process for every tweet. Therefore, assume a tweet with the following text message:

[1] https://nlp.stanford.edu/projects/glove/preprocess-twitter.rb
[2] List of stopwords from `gensim.parsing.preprocessing` library

“@TheLeoTerrell: Good News! Leo 2.0 just received message from Team Trump. Going to be used in Campaign. Cannot wait to help”

After applying the script from GloVe website, it becomes as follows:

<user>: good news! leo <number> just received message from team trump. going to be used in campaign. cannot not wait to help

Then, we split it into a list of words without punctuation:

[’user’, ’good’, ’news’, ’leo’, ’number’, ’received’, ’message’, ’team’, ’trump’, ’going’, ’campaign’, ’wait’, ’help’]

Now, we can use the vocabulary created by `keras.Tokenizer`. This vocabulary is a key-value store where the key is a unique integer and the value is the corresponding word. Thus we shall produce a vector that looks like this:

[2, 19, 205, 22, 53, 36, 34, 44, 9, 7, 107, 23, 66]

The final step is to pad zeros to all the lists that have a length which is minor to the max list length. We retrieve the vector that has the maximum length and keep its length as the `max_length`. Then, we pad zeros to all vectors, in order to make all of them have equal length of `max_length`. Therefore, if we assume that the longest list includes 18 integers, then, we need to append 5 zeros. Thus, the vector becomes:

[2, 19, 205, 22, 53, 36, 34, 44, 9, 7, 107, 23, 66, 0, 0, 0, 0, 0]

Once done with preprocessing and before we feed our neural networks with data, we need to create the train and test datasets using:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
        random_state=42, test_size=clf_config.TEST_SIZE, stratify=y)
```

We point out that the `train_test_split`[3] function is used to split arrays or matrices into random train and test subsets. We note that `X` is the list of the padded sequences we created before and `y` is the list of labels, which refers to a list of `1`s and `0`s. Furthermore, we define the split ratio by declaring the `test_size` variable.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

3.3 CNN model

The first model we use to train our data is the CNN in Figure **??**, which is specifically designed for text classification [7]. CNNs are the preferred method for image classification [13], because convolutional layers serve as feature extractors, and thus they "learn" the feature representations of their input images. However, CNNs behave equally well in cases of sentence classification, because they can learn an internal representation of the time series data and achieve a good performance. The model has an Input layer, to which we feed the vectors list. The number 87 indicates the size of each vector and in fact, it is the max vector length. We remind, that all the other vectors have appending zeros, in order to match the max length. In the next step, we have added the Embedding layer. As we have already mentioned, we use 50-dimensional vectors for our work, so as to be able to benefit from GloVe's pretrained vectors.
At this point, the model remains the same even if we don't use GloVe. The only difference is that with GloVe, we use weights in the embedding layer, as explained in Section 3.1. The next layer is a 1D Convolution layer with 64 output filters of size 10. During our experiments, we tried various combinations, but those numbers appear to behave the best. We decide to use the previous values, according to related reasearch [1]. Next, we add the Pooling layer that downsamples the incoming vectors. Pooling layers provide an approach to down sample feature maps by summarizing the presence of features in patches of the feature map. The main pooling categories are average pooling and maximum pooling. Here, we use maximum pooling, which is an operation that calculates the maximum, or largest, value in each patch of each feature map. The Pooling layer, is presented in Figure 3.2, according to [9]. Additionally, we add a Flatten layer that converts the output to a 1D array, in order to input it to the next Dense layer that consists of 32 units. The flattening process is mandatory, since classification models need to get inputs in the form of 1-dimensional linear vectors. Finally, we use a Dense layer with 1 neuron only, because we want it to determine a binary class of either hate or not.

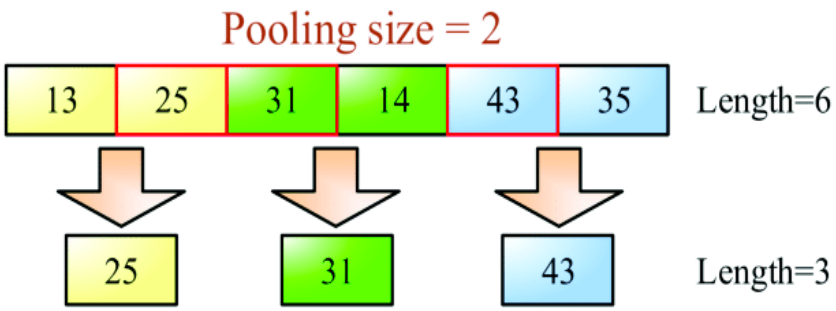


Figure 3.1: MaxPooling1D layer.

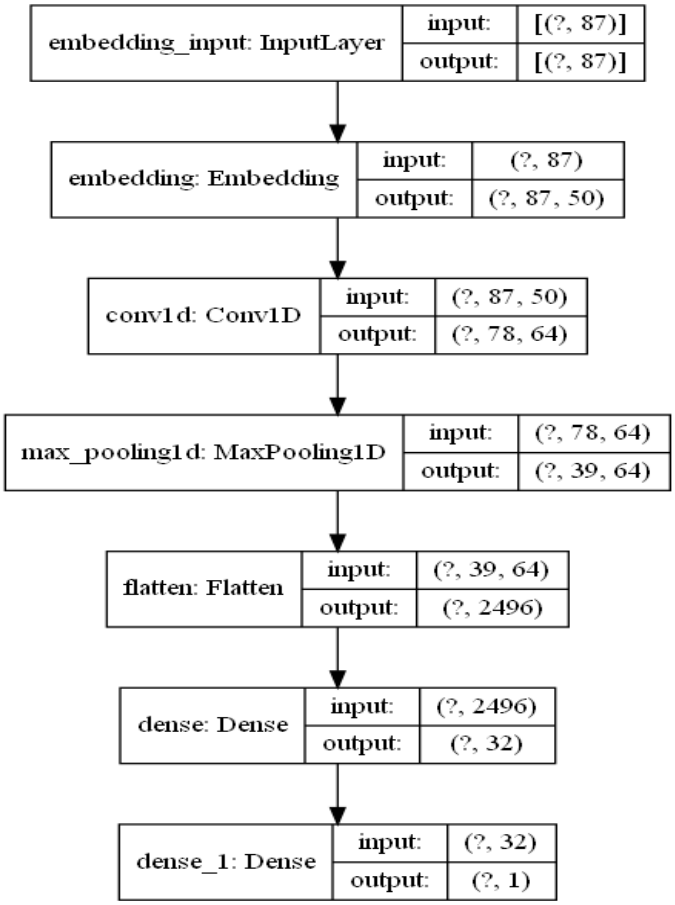


Figure 3.2: MaxPooling1D layer.

We notice that the input consists of vectors that have length equal to 87 integers. The embedding layer transforms the input into 2-dimensional vectors with shape (87, 50). Next, we use a 1-dimensional convolution layer that transforms vector's shape into (78, 64) because we use 64 filters. The pooling layer is used with the default pool size. Thus, it keeps half of the data and the output shape is (39, 64), where 39 is half of the 78 which is given as input. Lastly, we observe that the Flatten layer gets an 2-dimensional input vector with size (39, 64). Then, it produces an output of 1-dimensional vector of size 2496.

3.4 LSTM model

The second model we use to train our data is the LSTM network in the Figure 3.3. The LSTM network is an RNN and its main idea is to utilize sequential information processing. By this, we mean that each layer is capable to recognize patterns on the data given as input and predict the result on the output. Firstly, we add an Input layer of size 87 (max vector size in our dataset) and an Embedding layer, like in the case of the CNN. In contrast, an RNN is repetitive because it performs the same task for each successive element, with

output depending on the previous calculation. Therefore, we use an LSTM layer, to which we add a dropout to avoid overfitting [15]. Dropout's task is to randomly ignore some layer outputs from the network. By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections. For our purpose, we use a dropout rate of 0.5. Specifically, in each repetition, the LSTM network removes half of the inputs in order to avoid overfitting. Using dropout, the model Additionally, the LSTM layer consists of 100 units, inside the `keras.Bidirectional` layer. We use the Bidirectional layer, because it allows us to run the inputs both ways between the past and the future. Therefore, it causes better accuracy since the network is aware of both the past and future context of the phrases. Specifically in the case of text classification, the Bidirectional LSTM is one of the best apporaches [17]. The number 100 for the units and 0.5 for the dropout rate has been decided after various experiments and prior research [15]. For our work, we use 0.5 dropout rate, since it provides the best classification results, ensuring reliability and avoiding model overfitting. Lastly, we add a Dense layer with 1 neuron, that determines the final classification outcome.

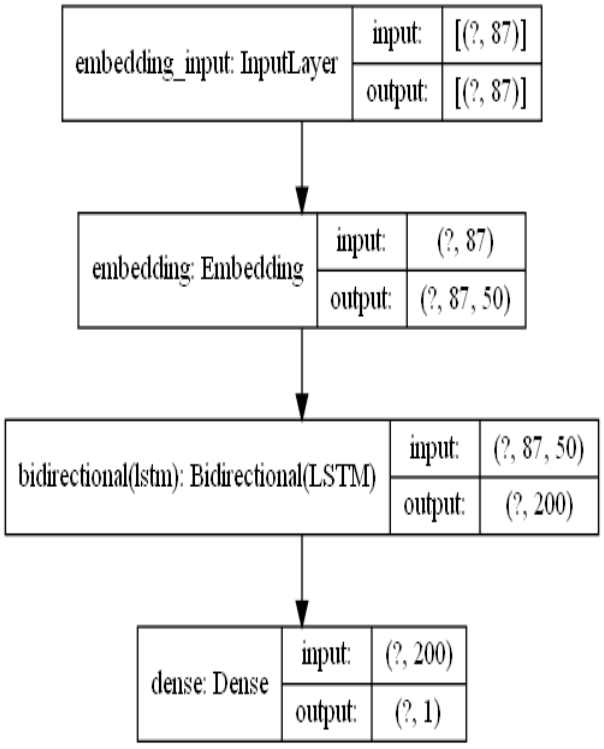


Figure 3.3: LSTM model for tweet classification.

The input layer and the embedding layer are similar to those used for the CNN model. The Bidirectional LSTM transforms the input into a 1-dimensional vector of size 200. Lastly, it provides the output to the final Dense layer which has only one neuron and determines

whether the tweet is hateful or not.

3.5 Classification process

The classification process is the same for the two neural networks. In both cases, the model is compiled using `keras.Model` compile function[4]. Specifically, we use the following line of code:

```
model.compile(optimizer=Adam(learning_rate=0.001),
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

The loss function that best suits our problem is the `binary_crossentropy`, because we undertake a binary classification project. The optimizer we choose is `Adam`. We have tried various optimizers[5], but, even though we have not observed significant difference in neither accuracy nor loss with any of them, `Adam` consistently works well across the a range of cases [5]. In addition, we fit the model and train it with the following code:

```
model.fit(X_train, y_train,
          epochs=<number of epochs>,
          validation_split=<validation size>)
```

In this line of code, we observe the parameters `EPOCHS` and `VALIDATION_SIZE`. The validation size is the portion of the training dataset that our network is going to keep to validate its results. Apart from those, we specify one more attribute, the `TEST_SIZE`, which indicates the percentage of the whole dataset that will be used for testing the model. The number of epochs indicates how many times the weights of the network will change. Increasing the epochs, we reach a point where validation accuracy remains the same, or, at least, it doesn't increase significantly. At the same time, validation loss starts increasing again and when we reach this point, we shall stop training the data. Simultaneously, there is not a general rule for the splitting ratio of the dataset in training, testing and validation subsets. However, it is a common approach to use some of the training data for validation purposes. This amount of data varies between datasets, but it ensures minimum overfitting, making sure that the validation data is not the same as the test data. The strategy we follow, is that we check the validation loss. When the validation error starts increasing, it may be an indication of overfitting. Therefore, we investigate a lot of possible combinations for those parameters. For the CNN, we get best results with the follwing configuration:

[4]https://github.com/tensorflow/tensorflow/blob/v2.4.1/tensorflow/python/keras/engine/training.py#L449-L549

[5]https://keras.io/api/optimizers/

```
VALIDATION_SIZE = 0.25
EPOCHS = 5
TEST_SIZE = 0.15
```

However, for the LSTM model, we observe that more epochs are needed. We inspect model's behavior with multiple experiments concerning the epochs value. The best results are achieved with the following configuration:

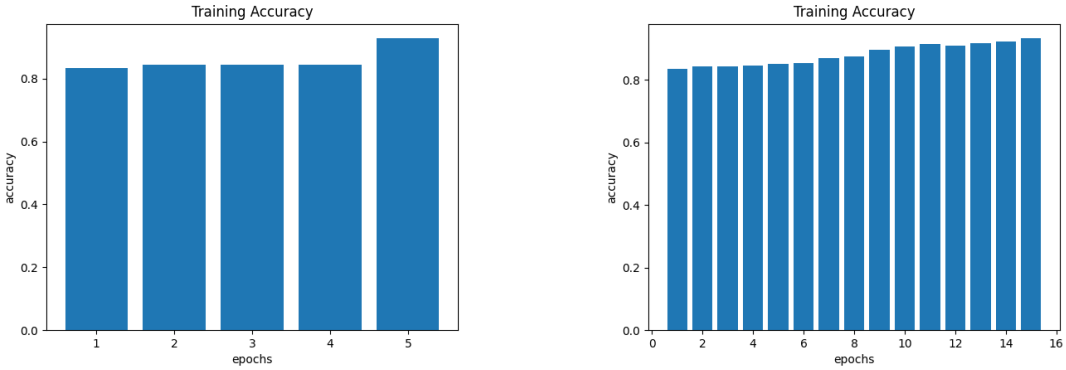
```
VALIDATION_SIZE = 0.25
EPOCHS = 15
TEST_SIZE = 0.15
```

3.6 Results

In this section, we present the classification results on Table 3.1 for both CNN and LSTM models, with or without GloVe. Using the CNN model, we achieve an accuracy of 81.71%, which, in case of GloVe embeddings, reaches 83.79%. With the LSTM model, the results we get are slightly worse. Thus, we reach 78.19% accuracy and, using GloVe, we improve it by 3 percentage points and, therefore, we achieve a final accuracy of 81.24%. Furthermore, an interesting observation we make, is that using CNN, we need less epochs to reach higher training accuracy. Specifically, in Figure 3.4, we indicate that for the CNN, we approach 100% training accuracy in 5 epochs, whereas for the LSTM we need approximately 15 epochs to reach the same value.

Table 3.1: Text classification results.

Model	Accuracy
CNN	81.71%
CNN + GloVe	83.79%
LSTM	78.19%
LSTM + GloVe	81.24%



(a) CNN accuracy plot

(b) LSTM accuracy plot

Figure 3.4: Text classification accuracy plots.

4. SOCIAL ANALYSIS APPROACH

In this chapter, we focus on the social analysis approach of our work. We present the user attributes that we use to classify our data, along with some useful statistics. The attributes we examine can be found in Subsection 2.2.2. Doing data investigation, we observe some values for each attribute, which we demonstrate in Table 4.1.

Table 4.1: Collection of important statistics.

Attribute	Mean	Min	Max	Std
tweet_retweets	230.88	0	19912	1275.72
tweet_likes	3.14	0	232	9.74
user_following	2700.5	0	104K	6269.08
user_followers	3633.1	0	179K	11884.76
user_total_tweets	56259.89	3	3.6M	144067.8

Thinking over the statistics that we have mentioned above, the first thing to consider is the vast existence of zero values in tweet likes and retweets. Specifically, we mention that we have a relatively small amount of likes for our tweets. Despite we waited some days before we extracted the final information, tweet likes didn't augment as we wanted. More precisely, we have 1017 tweets (approx. 78% of the total dataset) that have 0 likes. At the same time, we have 621 (approx. 47%) tweets with zero retweets. Those characteristics seem rational, considering that we have avoided collecting too many tweets from popular accounts, such as news channels etc., in order to produce a result that is not biased and that appears to be more realistic. However, those values can have an impact on the classification process and we determine that in the next Chapter. Apart from those characteristics, we have collected 217 tweets (approximately 17%) that are replies. From the scatter plots, we can see that the tweets are well dispersed. In each feature, we observe that both the orange dots (hate) and the blue dots (non hate) are placed everywhere on the two axes which is the tweet's id. Thus, there is for example, a tweet that is hateful and its author has very few followers, but there is also another tweet that is not hateful and its author also has very few followers. This is a sign that the amount of user followers doesn't appear to be an important factor that can determine whether a post is hateful or not. The same applies also for the amount of people the user follows. However, examining the tweet likes, it seems that hateful tweets have a small number of likes. Observing the equivalent scatter plot, the tweets with a higher amount of likes, appear to be non hateful.

To better understand the quality of our data, we depict scatter plots for each feature in Figure 4.1. For all the graphs, the x-axis represents each single tweet and the y-axis

represents the examined feature. Using those plots, it becomes easier to observe the average value and the standard deviation for each attribute, as well as the min and max values.

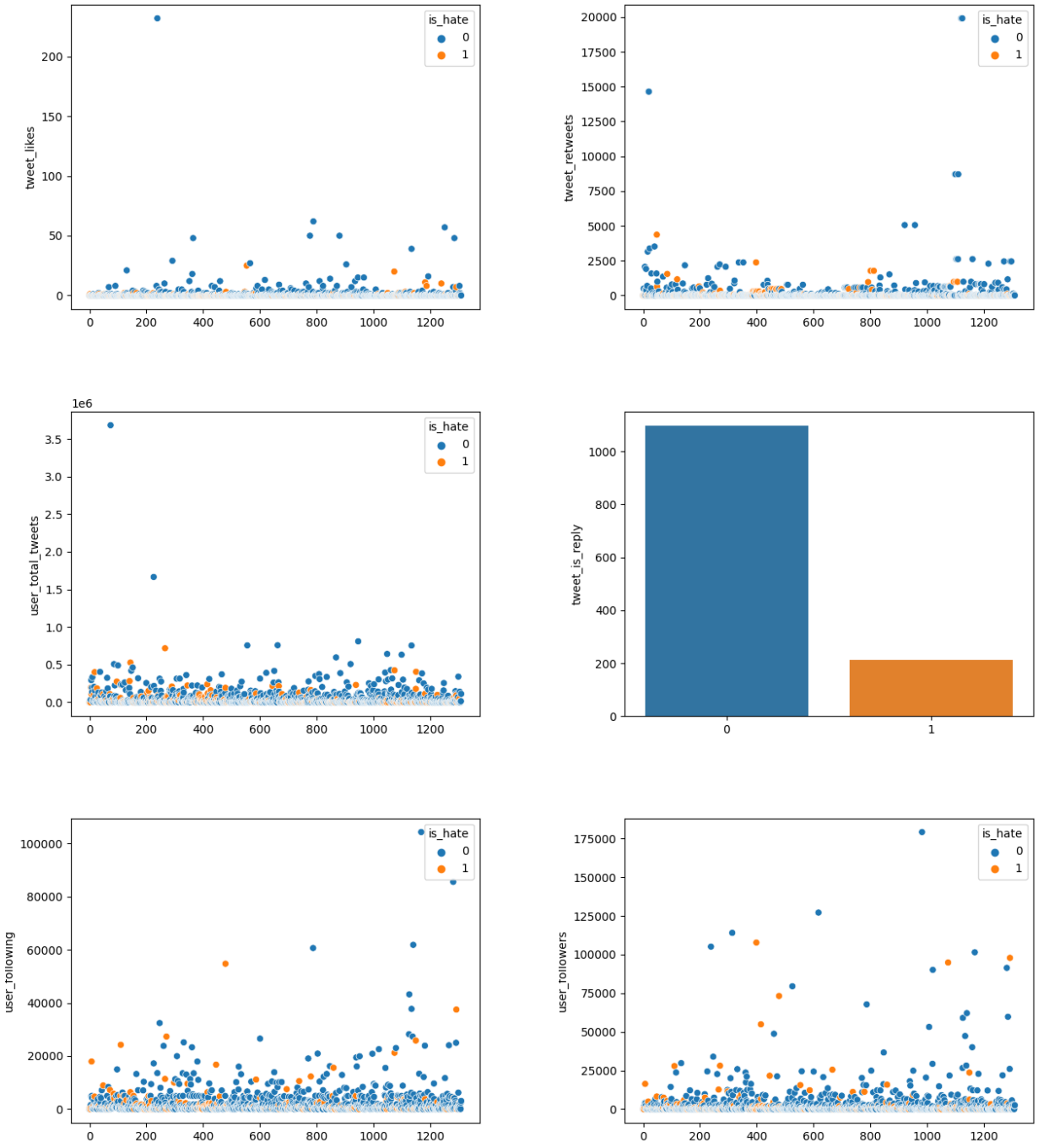


Figure 4.1: Features scatter plots.

Lastly, we indicate the amount of zero values per feature on Table 4.2. In order to avoid zero likes or retweets, we waited some days before extracting the final form of the data from Twitter. However, on the following table, we observe that we don't have enough likes nor retweets for the tweets on the dataset.

Table 4.2: Zero values per feature.

Attribute	Zero values	Percentage
tweet_retweets	621	47%
tweet_likes	1017	77%
user_following	11	0.8%
user_followers	3	0.2%

In Figure 4.2 we show the class distribution of our data. Despite this distribution is imbalanced, we have followed the examples from previous research where hateful tweets hold approximately 20% of the whole dataset. Because of this fact, however, we repeated the same experiments with equally balanced classes and the results are introduced in Section 5.5.

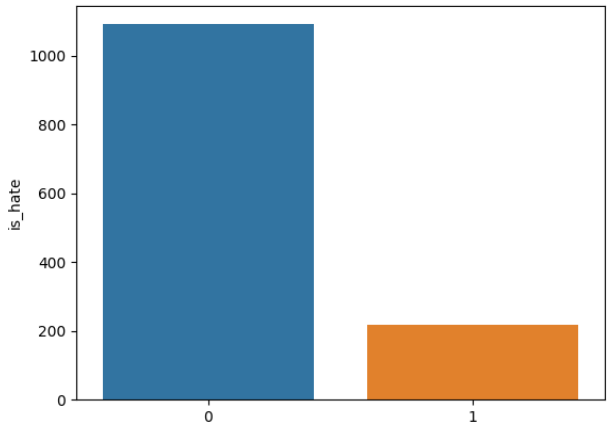


Figure 4.2: Class distribution graph.

Additionally, we present graphs in Figure 4.3 that help us identify outliers in each feature. The impact of outliers can be severe in datasets [10] and, if there is a large number of outliers, it can be detrimental. According to prior work [4], however, it is not always necessary to trim outliers off of the dataset. When outliers don't occupy a large portion of the dataset, they tend to introduce more realistic data.

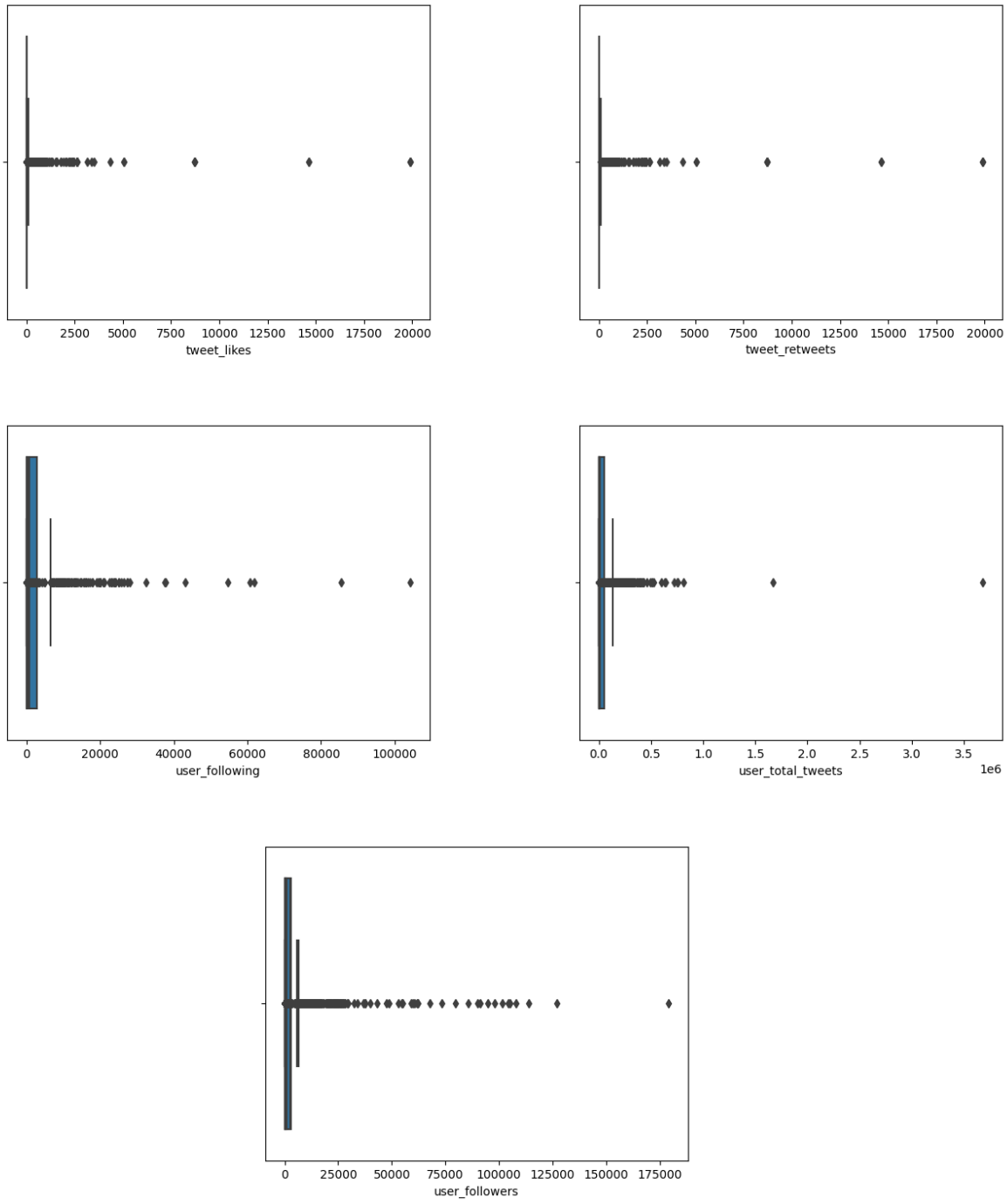


Figure 4.3: Outliers plots.

We introduce Table 4.3, in which we present the percentage of outliers in each feature. We use 95% as `max_threshold` and 5% as `min_threshold`. We identify outliers as the values that are above the `max_threshold` or below the `min_threshold` of each feature. The thresholds refer to percentiles. Therefore, the 95% defined as `max_threshold` represents

one feature that has higher value than 95% of values. Respectively, the 5% defined as `min_threshold` represents one feature that has higher value than 5% of values.

Table 4.3: Percentage of outliers per feature.

Feature	Outliers (%)
`tweet_retweets`	2.7%
`tweet_likes`	2.8%
`user_following`	5.6%
`user_followers`	6.8%
`user_total_tweets`	3.8%

Finally, in our dataset, outliers haven't been removed, since they don't provoke issues. Simultaneously, based on the plots and the table, the outliers are not so many. The only exception, in which there seems to be a number of outliers that is higher, is in the case of users' followers. However, we have mentioned earlier that we don't expect those features to have a clear impact on the classification process. Therefore, for those two reasons, we didn't proceed with outliers' elimination.

5. COMBINED CLASSIFICATION

In this chapter, we present the major contribution of our work, which is to combine text classification along with user data [3]. In this effort, we are primarily interested in finding out whether social network features, such as the amount of likes or the number of followers, can improve the classification results. We use again both CNN and RNN for the textual input, but, we also experiment with multiple mixtures of user attributes, that we feed to a secondary neural network, that is structured with `Dense` layers. After that, we concatenate the outcomes using `keras.Concatenate`[1] layer and we feed the combined result to some additional `Dense` layers, before the final neural network produces the final classification outcome.

We collect data concerning tweets' authors using `Tweepy` library, which extracts tweets from the Twitter API. A sample output calling the API looks like the json below. To improve readability we remove some data from the json response and keep only the necessary features.

```
{
    "id": 136601234342,
    "full_text": "Not about the policy ... #Trump \n@ThisWeekABC",
    "in_reply_to_user_id": null,
    "user": {
        "id": 1181322,
        "followers_count": 720,
        "friends_count": 205,
        "statuses_count": 8387
    },
    "retweet_count": 34,
    "favorite_count": 331
}
```

We notice, thus, that the user follows 205 users and is followed by 720 users. In addition, the user has authored 8387 tweets in total. Concerning the specific tweet, we notice that it is not a reply using the `"in_reply_to_user_id"` variable. Furthermore, we can see that the tweet has 34 likes and 9 retweets.

[1] https://keras.io/api/layers/merging_layers/concatenate/

5.1 CNN plus user features

The neural network we use is a concatenation of two smaller networks. The first one is a CNN, that is identical to the one we explain in Section 3.3. Specifically, we use an Embedding layer that takes an input of vectors of length 87. Then there is a `Conv1D` layer with 64 output filters of size 10. Once more, we use a `MaxPooling1D` layer that downsamples the data. Because we use the default pool size, which is 2, the layer splits the data in half. Thus, it transforms the input from 78 vectors into an output of 39 vectors. Finally, the data is transformed into a one dimensional vector using a Flatten layer. Next, we use a Dense layer of 32 units with which we pass the output into the Concatenation layer.
The second network, consists of an Input layer and four Dense layers with 200 units each. The input shape, in this case, is equal to the amount of user features that we use for the classification. Therefore, it varies between one and six. Thus, in Figure 5.1, where we bring up an example of our network, we observe that six features (all of them) are given as input to the neural network. The size of the 4 Dense layers is 128. Both the number of Dense layers and their sizes is decided after trial and error, as there is not a general rule to decide neither of them.
The Concatenation layer takes two inputs. The first has a length of 32 and is the output of the CNN that we use for text classification. The second consists of 128 units and is the output of the neural network that we use for the user features. Next, the layer transforms the dual input into a single output that consists of 160 units. After the Concatenation layer, there are 2 Dense layers that consist of 128 units. The number 128 is decided after trials. Finally, we use a Dense layer with 1 neuron only, because we want it to determine a binary class of either hate or not.
The model receives a mixed input. We present a sample input, but we clarify that we only hand over one repetition and not the whole input stream. Therefore, we show one vector for the CNN model and one combination of features for the second model. The first input is given to the CNN model and is a 87 length vector as follows.

```
[129, 128, 65, 4, 59, 2301, 22, 4581, 2, 4, ..., 0, 0]
```

We notice that the vector terminates with zeros, because of the padded zeros at the end, in order to achieve a fixed length for all vectors. The second input refers to user features. Thus, assuming we decide to train the model using the attributes: `user_total_tweets` and `tweet_retweets`, it receives an input that looks as follows.

```
[1291, 72]
```

The first input is a vector and each integer represents the position of the respective word in the vocabulary. The second input implies that the user has posted 1291 tweets and that the specific tweet has 72 retweets.

5.2 LSTM plus user features

In this case, again, the neural network we use is a concatenation of two networks. The first one is an LSTM, that is exactly the same with the one we explain in Section 3.4. The Embedding layer takes as input the vectors from the vocabulary. The LSTM layer that we use is bidirectional. Using a bidirectional layer, we are able to ascertain that each input is aware of the previous output. This is very important for text classification, as the model is aware of both the past and future of the process. For text classification, this approach significantly improves accuracy [17]. Lastly, we pass the output to a Dense layer that consists of 40 units, in order to feed it to the Concatenation layer.
The other network, is similar to the one we used for the CNN, and it consists of an Input layer and four Dense layers. The input shape is equal to the amount of user features that we use for the classification and we present the model in Figure 5.2. There are 4 Dense layers that consist of 200 units each. The number of units is decided after trial and error. The Concatenation layer takes two inputs. The first input comes from the LSTM neural network and has a size of 40 units. The other input has a size of 200 units and it derives from the neural network used for the user features. The Concatenation layer produces an output that consists of 240 units. After the Concatenation layer, there are also two Dense layers that consist of 200 units each. The number of units is decided after trial and error. Lastly, we use one more Dense layer that produces the final outcome, in which the network determines if the tweet is hateful or not.
In this case, the model works like the one used for the CNN. Therefore, we present a sample input for the model, and we clarify again that we only hand over one repetition and not the whole input stream. The first input is given to the CNN model and is a 87 length vector as follows.

```
[321, 556, 2, 910, 77, 5, 3, 3111, ..., 0, 0]
```

Once again, we notice that the vector terminates with zeros. The second input refers to user features and for the example we use 4 of them. Thus, assuming we decide to train the model using the attributes: `user_total_tweets`, `user_followers`, `tweet_likes` and `tweet_retweets`, it receives an input that looks as follows.

```
[999, 1231, 5, 9]
```

The first input is a vector and each integer represents the position of the respective word in the vocabulary. The second input implies that the user has posted 999 tweets. It also implies that the user has 1231 followers and that the specific tweet has 5 likes and 9 retweets.

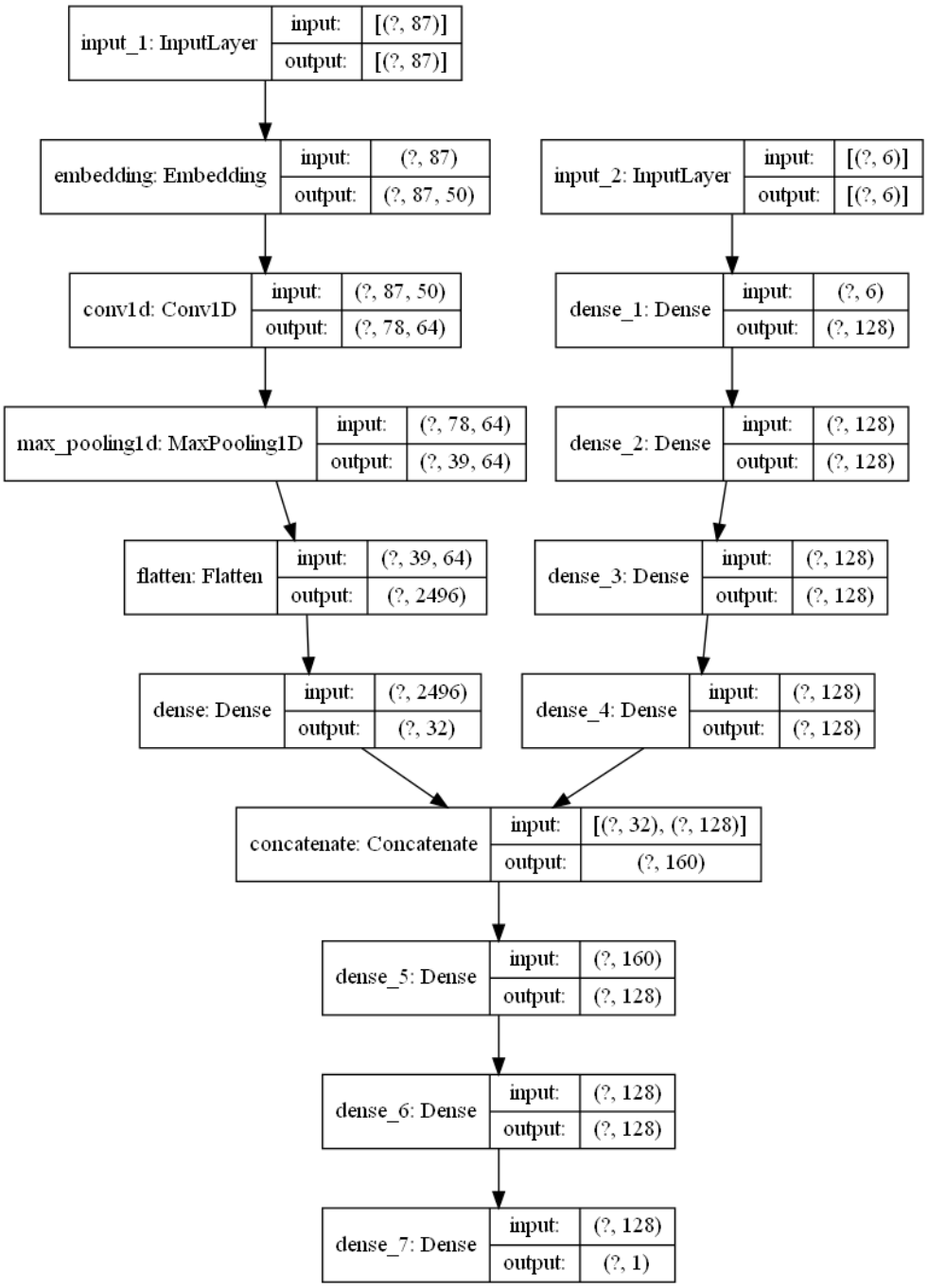


Figure 5.1: CNN plus user features network plot.

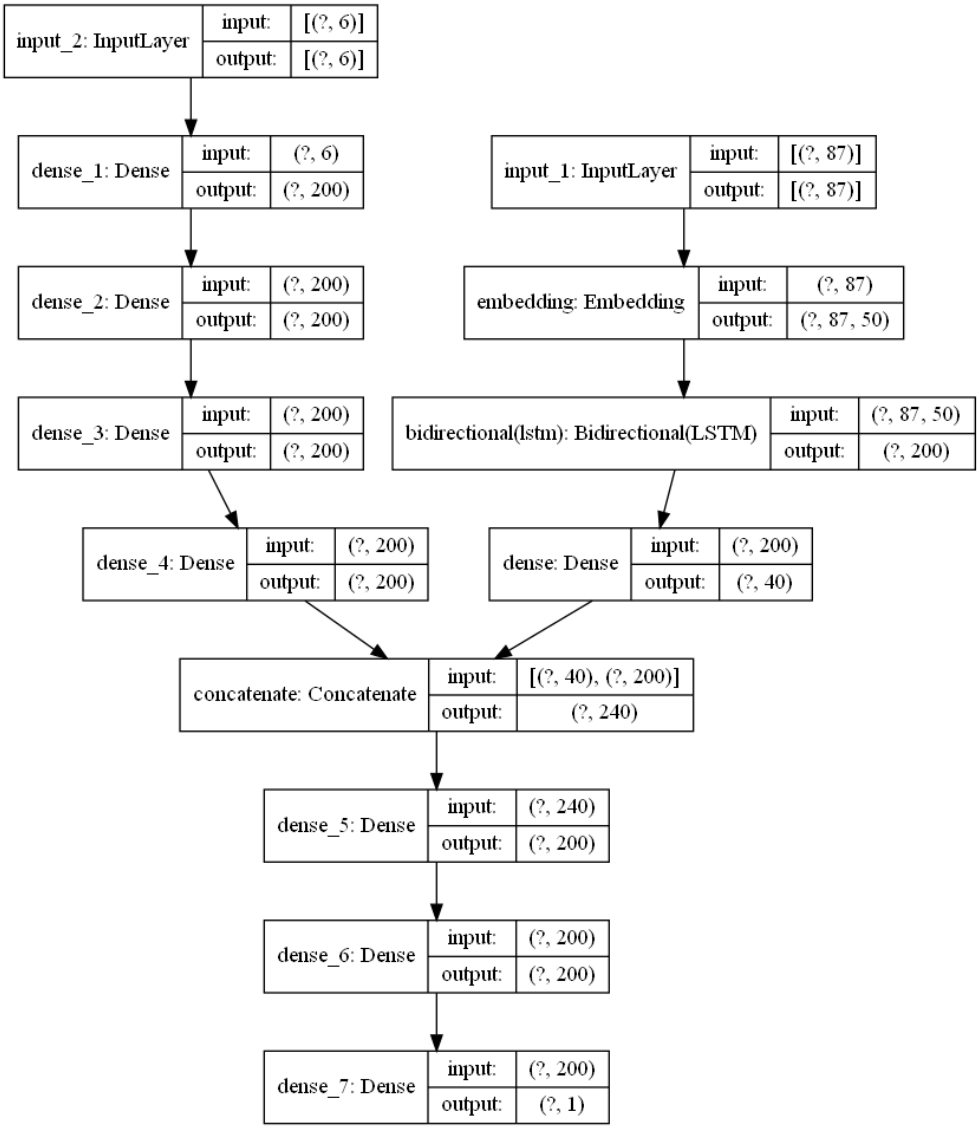


Figure 5.2: LSTM plus user features network plot.

5.3 Classification process

For both neural networks, the classification process is similar. In both cases, the model is compiled using `keras.Model` compile function[2].

```
model.compile(optimizer=Adam(learning_rate=0.001),
              loss="binary_crossentropy",
              metrics=["accuracy"])
```

The main difference with simple text classification is that, in this case, we need more epochs until we reach an accuracy close to 100%. Therefore, for the newly created CNN we use 15 epochs instead of 5 that we use for the pure CNN and for the new LSTM we need 25 instead of 15. However, to avoid overfitting, we don't increase the amount of epochs any more, as the validation accuracy would start dropping. More specifically, the classification configuration is as follows:

```
VALIDATION_SIZE = 0.25
EPOCHS = 15
# EPOCHS = 25 for LSTM
TEST_SIZE = 0.15
```

A big difference that is introduced in the case of combined classification is that we use user attributes. In order to facilitate our work, we put the attributes we desire to use in a configuration file that looks exactly like below.

```
# choose classification attributes below
# and add them to the 'chosen_attributes_list'
f1 = "tweet_retweets"
f2 = "tweet_likes"
f3 = "user_following"
f4 = "user_followers"
f5 = "user_total_tweets"
f6 = "tweet_is_reply"
chosen_attributes_list = [f1, f2, f3, f5, f5, f6]
```

Thus, if we want to change the features, we can go to the `chosen_attributes_list` and select the ones we need. After that, we feed the model and fit it with the following line of code, where `x` is an array of 2 inputs:

[2]https://github.com/tensorflow/tensorflow/blob/v2.4.1/tensorflow/python/keras/engine/training.py#L449-L549

```
model.fit(x=[data.X_train_text, data.X_train_user],
        y=data.y_train,
        epochs=<EPOCHS>,
        validation_split=<VALIDATION_SIZE>)
```

We can pass input parameters to the variables `EPOCHS` and `VALIDATION_SIZE`, which refer to the number of epochs and the validation split ratio respectively.

5.4 Results and evaluation

In this section, we discuss the results from the classification process. The amount of possible inputs is large. We have 6 features in total, that create 63 combinations. Specifically, we have 6 single features, 15 possible doubles, 20 triples, 15 quadruples, 6 quintuples and 1 sextet. We have tried all of the possible combinations. We hand over the Tables 5.1 for the CNN and 5.2 for the LSTM. In these tables, we present the five better mixtures considering their accuracy. The total list size is 63 and is equal to the number of possible features combinations.

Table 5.1: TOP 5: CNN plus user features classification results.

	User Features	Accuracy
1	`user_followers, tweet_is_reply`	86.96%
2	`user_followers, user_total_tweets`	86.54%
3	`tweet_retweets, tweet_likes, tweet_is_reply`	85.42%
4	`tweet_likes, user_followers`	85.42%
5	`user_total_tweets`	84.78%

If we don't provide any social features at all, the model produces an accuracy of 83.76%. Therefore, we realize that the two user features that produce the best accuracy for our dataset are `user_followers` and `tweet_is_reply`. In this case, we must mention that there have been two times when the model has reached almost 88% and in an average of five efforts, 86.96% was achieved.

Table 5.2: TOP 5: LSTM plus user features classification results.

	User Features	Accuracy
1	`user_followers, tweet_is_reply`	84.12%
2	`user_followers, user_following`	83.64%
3	`tweet_retweets, user_following`	83.62%

4	`user_total_tweets`	83.12%
5	`tweet_likes, user_followers, user_total_tweets`	82.42%

If we don't provide any social features at all, the model produces an accuracy of 81.22%. Those results indicate that in the case of LSTM neural network the user features that produce the best accuracy are the same as in the case of the CNN (`user_followers` and `tweet_is_reply`). This is an interesting outcome and it we establish 84.12% accuracy as a five runs average.
Concluding our effort, we compare our results with the text-only classification method that we introduced in Chapter 3. Specifically, if we look in the Table 5.3, we run the processes five times and we collect the average of those runs. We observe that the combined classification offers a better classification outcome than the text-only method on average.

Table 5.3: Text classification vs. Combined classification.

Model	Text Accuracy	Combined Accuracy
`CNN`	84.52%	86.31%
`LSTM`	81.47%	83.58%

This improvement in the classification outcome is established combining the classic textual classification with the user's features. Our model is able to utilize the sentences and the numbers from the user features to determine whether a tweet is hateful or not. The attributes we supply to our model appear to be beneficial and the accuracy increases slightly. However, the increase is not huge because of the factors we have discussed in the previous chapter. We elaborate that the user features have acted beneficially and have not added complexity to our model. The amount of time needed for fitting the model didn't increase when we added the secondary neural network.

5.5 Results with balanced classes

There is a question that we need to answer, to make sure that our work is sound. Specifically, we have taken into account that a proper dataset is normally consisted of approx. 20% hateful tweets, based on previous research that we have introduced earlier in this thesis. However, what would have happened if we had equally distributed classes? To answer to this question, we present the Table 5.4, in which we indicate the accuracy results for all the methods we have already mentioned. For this purpose, we have eliminated some non-hate tweets from the dataset.

Table 5.4: All models classification with a 50-50 dataset.

Model	Accuracy
LSTM	73.32%
CNN	77.19%
LSTM plus User Features	76.41%
CNN plus User Features	81.28%

Therefore, we observe that indeed, the classification accuracy is not as good as when the classes are not equally distributed. However, our new model defeats again the simple text classification model. For the CNN, we achieve almost 3% higher accuracy, whereas for the LSTM model, we achieve 4 percentage points increase in accuracy results. The increase is not massive, but it is still significant and can be beneficial for the classification process.

6. CONCLUSION

In this thesis, we propose to empower text classification models with user features as derived from social networks. The models we investigate outperform the existing models and we conclude that user features can lead to more accurate detection of hatred in social networks. Despite the fact that the increase is not massive, it is still significant and beneficial for the classification outcome. Additionally, the whole process of fitting the data to the model doesn't provoke efficiency issues, neither in time needed to train the model, nor in system requirements. Simultaneously, we experiment with equally balanced classes. In this approach, the general accuracy decreases 5% points on average, but the proposed models still outperform the existing ones.

In the future, it would be interesting to try and experiment with more complex networks, including more CNN, LSTM and hidden layers, to indicate if accuracy can increase further. Furthermore, we could inspect our models' accuracy experimenting with new models that don't necesserily include CNN or LSTM neural networks. We could implement a neural network specifically for this task. Additionally, we could inspect more user attributes apart from those we examined in this thesis. It would also be appealing to repeat the same experiments with larger datasets and observe how our models behave when we feed them with more training data. Lastly, we believe our models could operate well on data from social networks, other than Twitter. We believe that it would be interesting to try and detect hatespeech in other social media like Facebook or Instagram. The difficulty in this, would be to acquire the datasets, since Twitter provides researchers with all the necessary tools in order to exploit their API and obtain data. Concluding, having a dataset similar to the one we used in the current work, we believe our models will behave equally well regardless the social media.

ACRONYMS

Abbreviation	Full Name
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GloVe	Global Vectors

REFERENCES

[1] W. S. Ahmed and A. a. A. Karim. The impact of filter size and number of filters on classification accuracy in cnn. In *2020 International Conference on Computer Science and Software Engineering (CSASE)*, pages 88–93, 2020.

[2] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. *CoRR*, abs/1706.00188, 2017.

[3] L. S. P. Busagala, W. Ohyama, T. Wakabayashi, and F. Kimura. Multiple feature-classifier combination in automated text classification. In *2012 10th IAPR International Workshop on Document Analysis Systems*, pages 43–47, 2012.

[4] Shuxiao Chen and Jacob Bien. Valid inference corrected for outlier removal. *Journal of Computational and Graphical Statistics*, 29(2):323–334, 2020.

[5] Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On empirical comparisons of optimizers for deep learning. 2019. cite arxiv:1910.05446.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[8] Tom Kocmi and Ondrej Bojar. An exploration of word embedding initialization in deep-learning tasks. In *Proceedings of the 14th International Conference on Natural Language Processing (ICON-2017)*, pages 56–64, Kolkata, India, December 2017. NLP Association of India.

[9] Ping-Huan Kuo and Chiou-Jye Huang. A green energy application in energy management systems by an artificial intelligence-based solar radiation forecasting model. *Energies*, 11:819, 04 2018.

[10] Jason W. Osborne and Amy Overbay. The power of outliers (and why researchers should always check for them). *Practical Assessment, Research & Evaluation*, 9(6):1–12, 2004.

[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[12] Georgios Pitsilis, Heri Ramampiaro, and Helge Langseth. Effective hate-speech detection in twitter data using recurrent neural networks. *Applied Intelligence*, 48:in press., 12 2018.

[13] Waseem Rawat and Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation*, 29:1–98, 06 2017.

[14] Joshua Roesslein. Tweepy: Twitter for python! *URL: https://github.com/tweepy/tweepy*, 2020.

[15] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[16] Zeerak Waseem and Dirk Hovy. Hateful symbols or hateful people? predictive features for hate speech detection on twitter. In *Proceedings of the NAACL Student Research Workshop*, pages 88–93, San Diego, California, June 2016. Association for Computational Linguistics.

[17] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3485–3495, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.