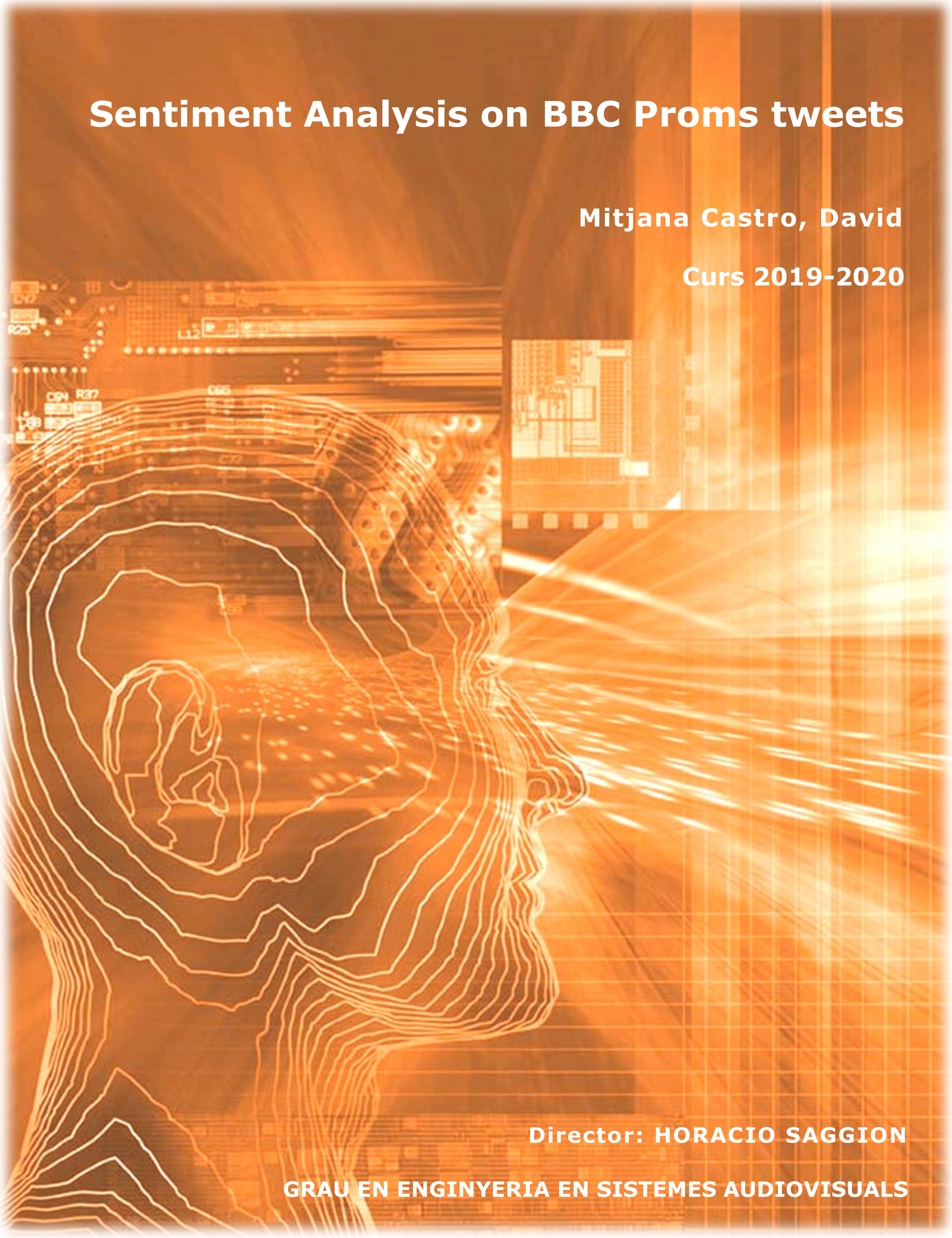


Sentiment Analysis on BBC Proms tweets

Mitjana Castro, David

Curs 2019-2020

Director: HORACIO SAGGION

GRAU EN ENGINYERIA EN SISTEMES AUDIOVISUALS

Universitat Pompeu Fabra Barcelona
Escola Superior Politècnica

Treball de Fi de Grau

SENTIMENT ANALYSIS ON BBC PROMS TWEETS

TREBALL FI DE GRAU DE

David Mitjana Castro

Director: Horacio Saggion

Bachelor's degree in Audiovisual Systems Engineering

Curs 2019-2020

Universitat Pompeu Fabra Barcelona

Escola d'Enginyeria

Acknowledgments

First, I wish to thank Horacio Saggion, my project supervisor, for his expert advice and guidance through the process.

I also want to mention all the people that have helped me to some extent in the development of this work: my friends Lucía Hernández, Lorena Pavón, and Adrián Salvador.

This project also means the end of my journey in UPF university. I would like to thank Taurons Rugby, the UPF rugby team, for introducing me to rugby five years ago and for making these years a whole different experience.

Last but not least, I would like to thank my parents for always supporting me in all my decisions and helping me personally, academically and professionally.

Abstract

Nowadays, with the increasing amount of user-generated data available online in micro-blogging services like Twitter, the human behavior of wondering what other people think has become an important field of study. Sentiment Analysis is a sub-field of Natural Language Processing (NLP) that tackles the issue of detecting the sentiment polarity of a piece of text, and is being used by a broad type of different businesses to extract insights of what people think about a product. The goal of this work is to perform a three-point scale {*negative*, *neutral*, positive} Sentiment Analysis on tweets from several editions of the BBC Proms, one of the biggest classical music festivals. We present a comparison of machine learning approaches and a state-of-the-art deep learning approach to classify Twitter data. In the experiments carried out, the deep learning approach, which is based on a CNN, outperforms the machine learning approaches achieving comparable results to the state-of-the-art.

Resum

Avui en dia, davant l'increment del contingut generat per usuaris disponible online en serveis de micro-blogging com Twitter, la necessitat humana innata de saber què és el que pensen els alters ha esdevingut un important camp d'estudi. *Sentiment Analysis* (anàlisi del sentiment) és un sub-camp del Processament del Llenguatge Natural que adreça la tasca de detectar el sentiment o opinió que expressa un text. Aquest tipus d'estudi està essent usat per una gran varietat d'empreses per extreure informació sobre el que la gent opina del seu producte. L'objectiu d'aquest treball és aplicar *Sentiment Analysis* en una escala de tres punts {negatiu, neutral, positiu} sobre un conjunt de tweets de diverses edicions del BBC Proms, un dels festivals de música clàssica més importants del món. En el treball es presenta una comparació de mètodes basats en machine learning i un mètode més vanguardista de deep learning. Els experiments duts a terme mostren que el mètode basat en deep learning supera els mètodes de machine learning, obtenint results similars als mètodes més actuals.

Resumen

El actual incremento de contenido generado por usuarios que está disponible online en servicios de micro-blogging como Twitter, ha provocado que la necesidad innata de los humanos de saber qué es lo que piensan los demás se haya convertido en un importante campo de estudio. *Sentiment Analysis* (análisis del sentimiento) es un sub-campo de Procesamiento del Lenguaje Natural (PLN) que estudia la tarea de detectar la polaridad del sentimiento que expresa un texto. Este tipo de estudio está siendo usado por una gran variedad de empresas para extraer información útil sobre lo que la gente opina de su producto. El objetivo de este trabajo es aplicar *Sentiment Analysis* en una escala de tres puntos {*negativo, neutral, positivo*} en un conjunto de tweets de varias ediciones del BBC Proms, uno de los festivales de música clásica más importantes del mundo. En el trabajo se presenta una comparación de métodos basados en machine learning y un método vanguardista de deep learning. Los experimentos realizados muestran que el método basado en deep learning supera los métodos de machine learning, obteniendo resultados similares a los del estado del arte.

Contents

List of figures

List of tables

1. INTRODUCTION

1.1 Motivation

In the last few years the field of Natural Language Processing (NLP) has experienced a "boom" in popularity thanks to the vast amount of accessible data and advances in deep learning. Many industries and research fields have benefitted from NLP, becoming a bridge between data science and human language.

One widely used application of NLP is Sentiment Analysis (SA), which consists in understanding the opinion expressed by a text. First research on document classification by sentiment appeared in a work by Pang et al. [1], where they explored the performance of different machine learning methods on the task of SA. Since then, and with the fast growing amount of user generated data, SA quickly caught the attention and interest of varied industries and research fields [2]. It provides businesses the ability to apply automatic text analysis on any type of unstructured text source: survey answers, Facebook or Twitter comments, user feedback, emails, etc. This provides companies with very useful insights about their product and enables them to make data-driven decisions.

Social media monitoring is one of the most exploited use cases of SA, since it is impossible for companies to stay updated with all the incoming data. In platforms like Twitter, around 6000 tweets are generated every second, which leads to around 500 million tweets per day[1]. SA allows to keep track of all this data and analyze it to, for example, get real time insights about customers.

Besides all this, this project has meant to me a starting point into the world of data analysis, which I have not had the opportunity to deeply explore during these years.

[1] https://www.internetlivestats.com/twitter-statistics/

1.2 Proposal

The aim of this work is to perform a three-point scale {*negative*, *neutral*, *positive*} SA on a group of tweets from the *BBC Proms*, the world's greatest classical music festival. There are three main approaches which have been followed to carry out SA: rule-based approaches, machine learning approaches and deep learning approaches. All of them have been deeply explored in the literature, but not a lot of efforts have been put to explore how they perform on a three point scale polarity score with Twitter data. Hence, this work focuses on the following points:

- Make a comparison of machine learning and deep learning based Twitter classifiers, by implementing state-of-the-art methods.
- Compare classifiers results with state-of-the-art systems presented in *SemEval 2017 task4: Sentiment Analysis in Twitter, subtask - A* [3].
- Use the best classifier to classify *BBC Proms* tweets and perform Sentiment Analysis on them. For this task I will use Kibana[2] to implement an interface to visualize and analyze the classified data.

The set of events to be analyzed are the *BBC Proms* editions from 2014, 2016 and 2017. Also known as the *Henry Wood Promenade Concerts* or *The Proms*, *BBC Proms* is an eight week classical music festival held every summer in London since 1895, with the goal of bringing the best in classical music to the widest possible audience[3]. It is one of the biggest and best-known music festivals in the world, hosting around 90 concerts and with the potential of reaching over fifteen million people[4].

But before diving into the implementation of the proposal and the analysis of the *BBC Proms* tweets, we first need to have a good understanding of what Sentiment Analysis is and the different approaches that have been proposed over the last years in the literature.

[2] https://www.elastic.co/kibana
[3] https://www.bbc.co.uk/programmes/articles/2kSNxH9Cj9PT62ZzTnvWpYZ/the-bbc-proms-whats-it-all-about
[4] https://evanevanstours.com/blog/all-about-the-proms/

1.3 Sentiment Analysis

Sentiment Analysis (SA), also known as opinion mining, can be described as the process of computationally identifying and categorizing opinions expressed in a piece of text, especially in order to determine whether the writer's attitude towards a particular topic or product is positive, negative or neutral[5].

Precedents of attempting to get public opinion can be traced back to the first decades of the XXth century when the task was focused on quantifying and measuring the opinion from questionnaires. The first academic studies that tackled the task of identifying public opinions appeared after WWII, with highly political motivations. Later in the mid-1990s, the computational linguistics community started to study text subjectivity analysis and the first computer-based sentiment analysis systems appeared.

But it was not until the burst of subjective texts on the Web in the early 2000s that computer-based SA took off. A clear fact about this major outbreak is that 99% of the papers of sentiment analysis have been published after 2004 [4]. The increase of the number of papers in SA since 2000 can be seen in figure 1.1. Although the term sentiment analysis first appeared in [5], the research on sentiments appeared earlier [1].

Nowadays the task of SA has a wide range of practical applications. Thanks to the recent advances of deep learning techniques and the large and increasing amount of user-generated text in social media platforms, it has become a hot topic of research not only in the field of Natural Language Processing (NLP) but also in political and social sciences and many other research fields. It also provides useful business insights for market predictions, helping companies to develop their strategies, to understand customer's opinions, the company's reputation, etc.

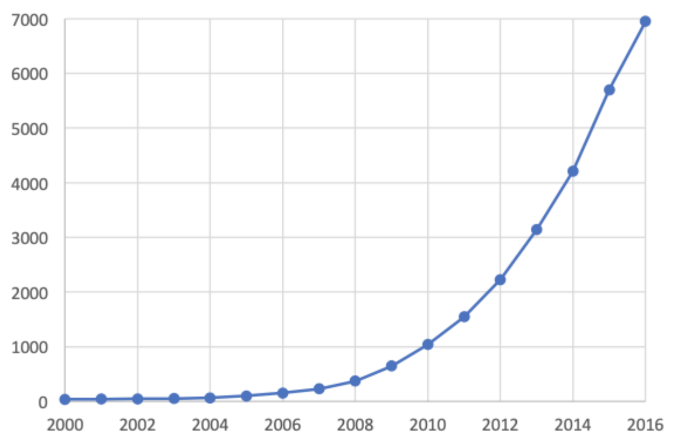


Figure 1.1: Number of papers published per year related to Sentiment Analysis. Picture is taken from [4].

[5] https://www.lexico.com/definition/sentiment_analysis

1.3.1 Sentiment Analysis on music

The relationship between music and human emotion is unquestionable, and a lot of research has been done to study this close connection. Technology advances have changed the way we listen and interact with music in recent years. This has boosted the interest in research areas like Music Information Retrieval (MIR) or Music Emotion Recognition (MER), both experimenting with text and audio features.

Regarding text-based sentiment analysis on music, MIR researchers have taken advantage of NLP techniques such as Named Entity Recognition (NER) or Information Extraction (IE) for multiple tasks such as mood classification of lyrics, genre classification, recommender systems or artist similarity. However, the majority of sentiment analysis tasks in the music domain have tackled formal texts but not noisy user-generated content, i.e. social media and microblogging services like Twitter.

First approaches on musical knowledge extraction from user-generated content were focused on reviews [6] and online forums [7]. Initial approaches for detecting musical named entities are presented in [8] and [9], and a first Musical Entity Linking (MEL) has been presented in [10].
Furthermore, a method to recognize classical musical entities in Twitter content generated by users is presented in [11], which focuses on detecting two kinds of musical entities: *contributor* and *musical work.*

Nonetheless, the task of IE and NER from user-generated content of a specific domain like music is still challenging due to the highly noisy and informal nature of the texts generated by users.

1.4 State of the art

There are three main sentiment classification levels to be considered:

- *document-level*: classifies a whole document talking about one topic (e.g. product review)
- *sentence-level*: detects whether a sentence expresses a positive or negative sentiment
- *aspect-level*: classifies the sentiment with respect to the specific aspects of entities

The scope of this work is SA of Twitter texts, therefore only *sentence-level* sentiment polarity classification methods are being considered for the task at hand.

On the topic of *sentence-level* classification, and more specifically Twitter sentiment analysis, the International Workshop on Semantic Evaluation (SemEval) has dedicated part of its efforts to the task since 2013 edition. The most recent edition where they explored three point scale {*negative, neutral, positive*} SA of Twitter data was in *SemEval-2017 Task 4: Sentiment Analysis in Twitter* [3]. The competition results for *subtask A* are used to analyze the state-of-the-art methods of SA in Twitter data and as a benchmark for the methods presented later in this work.

Since the beginning of 2000 different approaches to SA were explored by researchers. These approaches can be grouped into three main groups: *rule-based approaches, machine learning approaches,* and *deep learning approaches.*

1.4.1 Rule-based approaches

These methods basically compute an overall sentiment based on a set of manually created rules. These methods typically include word polarity computation based on lexicons, which are dictionaries of positive and negative scores for a list of words and can be used to compute the average sentiment of a tweet. A complete description of a rule-based method for sentiment analysis and comparison with previous similar methods is presented in [12]

The advantage of using rule-based methods is that they do not need prior training to classify a document, therefore they require minimal data preparation, and can be easily done with libraries like *TextBlob*[6] and *VADER*[7].

[6] https://textblob.readthedocs.io/en/dev/quickstart.html
[7] https://github.com/cjhutto/vaderSentiment

A drawback of using this type of methods is that they do not have any understanding of the meaning of the sentence, hence are limited to basic sentiment analysis, and cannot perform other tasks such as sarcasm detection.

1.4.2 Machine learning approaches

Machine learning methods use huge amount of data to train a model that will be used to predict the polarity of a text. The most common classification algorithms used in machine learning based methods are Linear Regression, Support Vector Machines (SVM) and Naïve Bayes. Initial findings of the benefits of some of these algorithms for text classification tasks were introduced in [13].

In *SemEval 2017 Task 4 – subtask A*, three of the top-10 teams used machine learning approaches. They all used SVM classifiers based on different type of features: lexical features, semantic features, dense word embeddings, etc.

1.4.3 Deep Learning approaches

In recent years, deep learning techniques have transformed the field of NLP, considerably outperforming non-deep learning methods in a number of tasks, and improving the performance with less need of engineered features [14]. However, this is thanks to their ability to generalize well when trained with large amounts of data. A good example of the proved effectiveness of deep learning methods is that, in *SemEval 2017 task 4 - subtask A*, the top-4 ranked teams all used deep learning or deep learning ensembles. The most popular deep learning techniques for Sentiment Analysis are Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs).

<u>RNNs</u>

RNNs are a type of networks that perform especially well with sequential data, it is, data that presents continuity in time. That’s why they have become popular in text classification tasks because their architecture allows the neural network to capture temporal behavior and sequential data, being a more “natural” approach for textual data since text is naturally sequential. The most popular and used version of RNNs are Long Short-Term Memory (LSTM) networks.

<u>CNNs</u>

As for CNNs, they are faster and simpler than RNNs and are well-known for their applications in computer vision and for the way they simulate the functioning of our brain. Nonetheless, they have shown remarkable good results in several NLP tasks [15], outperforming previous approaches with simpler and faster methods. One of the first works that showed impressive results of CNNs for sentiment analysis was introduced in 2014 by Yoon Kim [16]. The method achieved state-of-the-art results in sentence-level classification problems, with a basic CNN architecture with one-layer convolution layer, little computational costs and simple classification features using pre-trained word vectors. Figure 1.2 shows the basic architecture of the CNN presented by Kim.

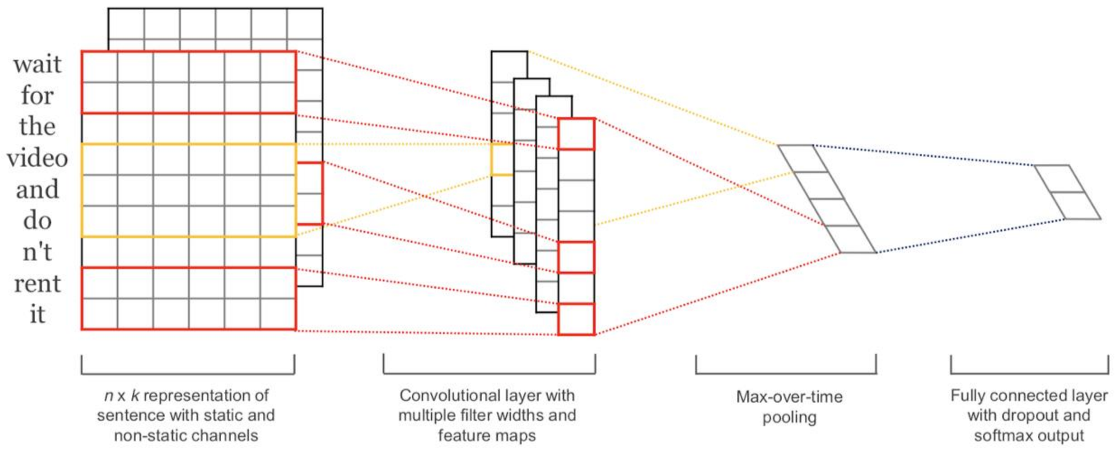

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1.2: Model architecture of the CNN presented by Kim. Picture taken from *[16]*

Since the introduction of CNN systems for text classification, many approaches of CNNs combining other techniques have been explored. In [17] they present a benchmark of CNN baseline configurations for text-classification tasks following Kim's model.

The two top ranked systems in SemEval-2017 Task 4 presented ensemble methods of CNNs and LSTMs networks [18][19]. The fourth ranked system [20] introduced an ensemble method of ten CNNs to classify a tweet, by using the same word embeddings as inputs for all the networks but varying the initial weights. The proposed network architecture is the same as the one presented by Kim in [16], with the addition of a fully connected layer, and the results of the different networks are combined by selecting the sentiment given by the majority of them. Lastly, a different approach is presented in [21], where deep learning techniques are combined with rule-based approaches, introducing a Lexicon integrated CNN method with attention.

2. METHODOLOGY

In this section I will explain the process followed to build the final classifier that will be used to analyze the tweets from BBC Proms. The framework in which it is built is very similar to the one used in any Sentiment Analysis task, and can be seen in figure 2.1.

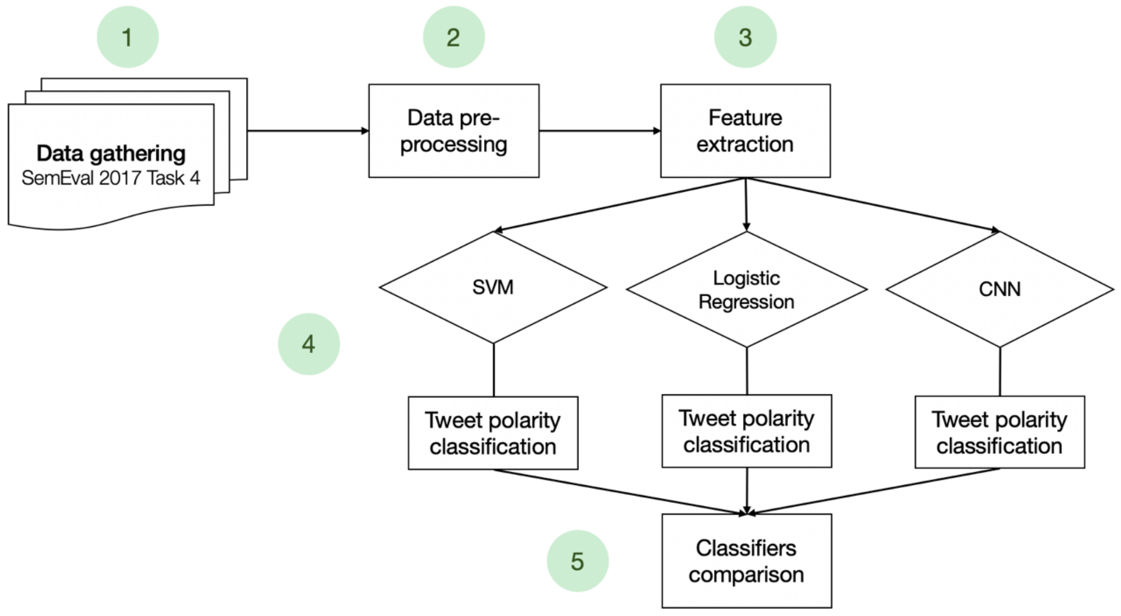


Figure 2.1: Block diagram of the implemented framework. (1) data is gathered from a reliable source; (2) data has to be pre-processed in order to reduce vocabulary size and prepare it for feature extraction; (3) representative features have to be extracted from the data; (4) different classifiers are built; (5) the classifier with the best performance is chosen to classify the BBC Proms tweets.

2.1 Implementation

All the implementation has been done in the Python programming language. In the text pre-processing I have created custom scripts to process the texts; the machine learning methods have been developed with the scikit-learn library[8], and the CNN model has been implemented with the open source library PyTorch[9].

More details on the implementation are explained in each corresponding section. All the code can be found in GitHub[10].

[8] https://scikit-learn.org/stable/
[9] https://pytorch.org/
[10] https://github.com/mitji/TFG_Twitter-Sentiment-Analysis-on-BBC-Proms

2.2 Training data

The task at hand is a supervised learning task, and the first common issue in this type of problems is to find enough labeled data to feed the classifier. If the system is provided with a decent amount of data, then it will be able to learn specific behaviors and relationships from the input data and it will classify correctly any given unlabeled input.

2.2.1 Data gathering

The aim of this work is to classify a group of tweets in a three-point scale: {*negative, neutral, positive*}, hence a reliable dataset of annotated tweets is needed and it has to be big enough to train a model. Thankfully, there is a renowned NLP competition called SemEval where they have explored the topic of three-point scale Twitter Sentiment Analysis in a number of their recent editions, in which they have provided the participants with annotated data to train, test and evaluate their models.

SemEval is an ongoing series of computational semantic analysis systems[11], where several teams compete on different NLP tasks. The most recent edition where they have explored Tweet polarity classification was in 2017, more specifically in *Task 4 - Subtask A* [3], which focuses on the overall sentiment of a tweet {negative, neutral, positive}. The dataset they provided for the task is available and can be downloaded in[12]. It includes annotated training and test data from the 2013 to the 2016 edition. I have combined all this data and I have used it as the training set, with a total of 53484 tweets. All the data is structured in .txt files with the following format: *tweetId<TAB>label<TAB>text*, pretty simple to read and process in the Python scripts. In the website of SemEval 2017 Task 4[13] they also provide the test data that all the teams used to evaluate their systems, which makes it suitable to compare my models performance with state-of-the-art methods presented in *SemEval 2017 Task 4 – subtask A* by evaluating it with the same metrics. More details about the training and test dataset are shown in table 1.1.

	negative	**neutral**	**positive**	**all**
training	8498 (15.89%)	24197 (45.24%)	20789 (38.87%)	53484
test	3972 (32.34%)	5937 (48.33%)	2375 (19.33%)	12284

Table 1.1: Distribution of training and test data from SemEval 2017 task 4 - subtask A

[11] https://en.wikipedia.org/wiki/SemEval

[12] https://www.dropbox.com/s/byzr8yoda6bua1b/2017_English_final.zip?file_subpath=%2F2017_English_final%2FGOLD%2FSubtask_A

[13] http://alt.qcri.org/semeval2017/task4/index.php?id=data-and-tools

2.2.2 Train and validation split

The split of training data into train and validation sets is 85% for the train set and 15% for the validation set. The validation dataset is used to evaluate the model fit on run-time and make decisions to fine tune the hyper-parameters and apply early stopping if the model performance does not improve during training.

	negative	**neutral**	**positive**	**all**
train	7229 (15.90%)	20554 (45.21%)	17678 (38.89%)	45461
validation	1269 (15.82%)	3643 (45.40%)	3111 (38.78%)	8023

Table 1.2: Train and validation datasets split for training and fine-tuning the models

2.2.3 Text pre-processing pipeline

To get the data in a clean, standard format for future analysis and to obtain the best possible results in the classification of the tweets, the data must be in an understandable format that is easy to read by a machine. Working with Twitter data this can become a very challenging task due to the amount of slang language, emojis, hashtags, contractions, misspelled words and noisy data that has little value for sentiment analysis and can have a negative impact on the classification.

Next I will explain the different techniques I have used to prepare the data for both the machine learning and the deep learning approaches presented in this work.

I have implemented a custom text pre-processing pipeline in Python inspired by the pipeline presented in [22] with the following structure:

- **Step 1: Pre-processing**. Initial data cleaning and processing of the text before passing it through the Twitter POS tagger and tokenizer.
- **Step 2: Tokenization & POS Tagging**. Step to identify domain specific tokens like hashtags, urls or mentios with the POS tagger and tokenize the tweets.
- **Step 3: Post-processing**. Extra processing step that takes advantage of the tokenization and POS tagging to correctly remove unwanted data and refine the POS tags.

Figure 2.2 shows the structure of the text pre-processing pipeline described above.

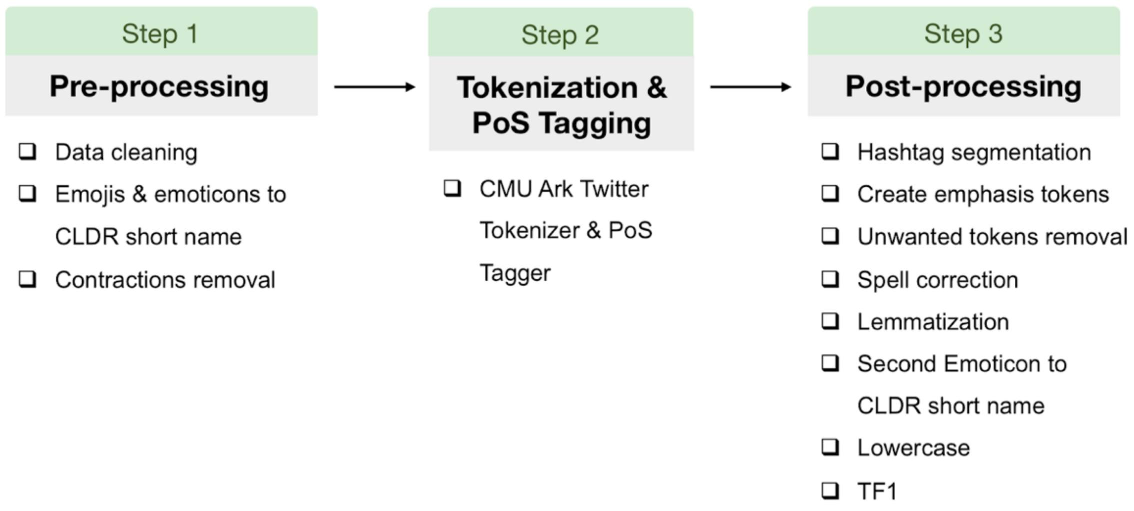


Figure 2.2: Custom text-preprocessing pipeline

Step 1: Pre-processing

Data cleaning

First of all, the raw data downloaded from SemEval has to be read and prepared to be pre-processed. As mentioned in section 2.1.1, all the training data is structured in .txt files with the following format: *tweetId<TAB>label<TAB>text*. Then, for the text pre-processing, all the training files are combined into one file with the format *sentiment<SPACE>text*, ignoring the tweet id since it has no relevance for the sentiment detection. In addition to this, multiple white spaces are converted into single white spaces to make it easier to split tweets and generate tokens, double quotes (") are removed from beginning and end of tweets, and all remaining double quotes are replaced by single quotes ('). All the data is encoded in unicode format, so it must be read and processed accordingly in the following steps.

Emojis and emoticons to CLDR short name

Emojis and emoticons have to be preserved because they can carry a lot of information about the sentiment of a short text like a tweet.

Emojis

Taking into account the Unicode encoding and before any further processing, we need to replace the emojis by words to keep their meaning. For this I have used the python library emoji[14] that does exactly this, converting emojis into a short descriptive name. For instance, the sentence `“Confinement is finally over 😊 ”` will become `“Confinement is finally over :smiling_face_with_smiling_eyes:”`.

[14] https://pypi.org/project/emoji/

Emoticons

As for emoticons, I have created a large dictionary of 126 emoticons with their corresponding short name using the same “source of truth” that uses the python emoji library, using the associated CLDR Short Name[15]. However, it is difficult to define an exact equivalence between emojis and emoticons, so a general approach for the emoticons short names have been followed, trying to find at least one word that can be found in the short text of similar emoticons or emojis. For example, in my custom dictionary the emoticons *“:), (:, 8), =), xD”* are represented as *“:grinning_face:”*.

After this step, all emojis and emoticons have been replaced by their CLDR short name with textual meaning. It may happen that some characters of an emoticon are encoded in a different encoding system, or for some reason a character has not been decoded in the first step. This is solved in the *Emoticon Parser* of *Step 3*.

Contractions removal

Before applying the tokenizer and POS tagger I replace contractions by their full extended form. The task of extending a contraction can be very hard because it requires contextual knowledge to replace the contraction with the correct words. As an example, the contracted form *“I’d”* can have two expanded forms: *“I would”* and *“I had”*.

For this step I use two methods. First, I use the python library *pycontractions*[16] that does a decent job on this. In the case of the previous example, *pycontractions* would take context into account and successfully convert the sentence `“I’d love to go”` into `“I would love to go”`.

Secondly, I use a custom contraction dictionary extracted from[17] as a backup in case the *pycontractions* library doesn’t work properly.

Step 2: Tokenization & POS Tagging

Tokenization of Twitter data can be tricky because it is difficult to avoid splitting expressions that should be treated as a single token, and is also challenging to detect Twitter specific entities like URLs, hashtags, dates or at-mentions.

There are several tokenizer tools from common NLP libraries like nltk or spacy, but they do not detect specific Twitter entities, and as a result they do not perform well in the tokenization of specific domain tokens like hashtags, mentions or emoticons. For this task I have used the Twitter tokenizer and Part-Of-Speech Tagger (POS Tagger) tool from

[15] http://www.unicode.org/emoji/charts/full-emoji-list.html

[16] https://pypi.org/project/pycontractions/

[17] https://github.com/tthustla/yet_another_tiwtter_sentiment_analysis_part1/blob/master/Yet_Another_Twitter_Sentiment_Analysis_part1-Copy1.ipynb

CMU ARK research group [23]. I also considered using TwitIE, a tokenizer and POS Tagger plugin from GATE, but I ended up choosing the tool from CMU because it was easier to adapt into my pre-processing pipeline by adapting an available python wrapper[18] that allows to run the java library from CMU in python.

At the end of this step, every tweet is represented as an object (or dictionary) with the fields *sentiment*, *raw_text*, *cleaned_text* and *tokens*, which is an array of objects where every object is a token with two fields: *token*, which is the word itself, and *tag*, which is the POS tag.

An example of a tweet structure after the Tokenizer and POS tagger can be seen In the following code snippet:

```
{
     "sentiment": "positive",
     "raw_text": "I just realized that Go Set a Watchman comes out tomorrow!!!",
     "cleaned_text": "I just realized that Go Set a Watchman comes out tomorrow!!!",
     "tokens": [
        {
           "token": "just",
           "tag": "ADVERB"
        },
        {
           "token": "realize",
           "tag": "VERB"
        },
         ...
     ]
}
```

Step 3: Post-processing

This final step after tokenization consists in the main part of text processing that prepares the data for Sentiment Analysis. Taking advantage of the tokens and POS Tags generated in step 2, I decide which tokens I need to modify, keep and omit. Here is where I perform hashtag segmentation, spell correction, word normalization with lemmatization, removal of unwanted tokens like punctuation marks, numbers or mentions, lowercase all the data and at the end apply vocabulary reduction.

Hashtag segmentation

As explained in the article *Working with Twitter Data in Python*[19], hashtags are usually created by merging several words together, and splitting them can become handy in

[18] https://github.com/ianozsvald/ark-tweet-nlp-python
[19] https://medium.com/analytics-vidhya/working-with-twitter-data-b0aa5419532

Sentiment Analysis. Apart from this, hashtags are usually used to express some sort of opinion with regard a specific topic. Thankfully, the POS tagger detects all hashtags entities. To split the hashtags into actual words, I use the python library *Ekphrasis*[20] [22], a library developed by the winning team at *SemEval 2017 Task 4 - subtask A*. The library implements a word segmentation method that uses the Viterbi algorithm to split the words based on the statistics provided by a Twitter corpus, and the word segmentation is suitable for hashtag segmentation. Each new string generated from the hashtag segmentation is mapped into a new 'HASHTAG' token, and the original complete hashtag token is removed.

Emphasis tokens

Emphasis expressions are usually connected to strong sentiment, positive or negative, so they are worth detecting. Emphasis expressions like *"!"*, capitalized words, or words with more than 2 consecutive vowels are taken into account. No matter the tag a token has been given, if it is detected as an emphasis token, it will be converted into an emphasis token.

Remove unwanted tokens

After the POS tagging is easy to remove those tokens that have no relevance to the sentiment analysis task. In this sub-step I have removed punctuations (those on emoticons and emphasis tokens have already been converted into different POS tags), mentions, URL, emails, numbers, discourse markers and unknown tokens classified as 'OTHERS' by the POS tagger.

Spell correction

After testing several libraries (TextBlob[21], SymSpell[22]) that detect and correct misspelled words, I concluded that they are very time-consuming and do not improve my model's performance, so I have decided to apply a basic spell checker. First, I check that each character is not repeated more than twice in a word. In addition to this, if a word has a vowel consecutively repeated more than twice, the word is converted into an 'EMPHASIS' token. For example, the word "sooooo" that can be found in a sentence like "sooooo good", will be converted first into a token with the tag 'EMPHASIS' and then the token itself will be corrected to "soo".

Lemmatization

In order to normalize the text and reduce the vocabulary size I apply lemmatization, a technique to find the root form of inflected words, which can be simply described as words that have been derived from another word to express a different grammatical category by using a prefix, suffix, infix, or another internal modification such as a vowel

[20] https://github.com/cbaziotis/ekphrasis
[21] https://textblob.readthedocs.io/en/dev/
[22] https://github.com/wolfgarbe/SymSpell

change[23]. For example, the words *"are", "is", "am"* and *"were"* have the same root word, which is *"be"*.

There are two ways to normalize a word to its root form: lemmatization and stemming. They differ in the way they generate the root forms of words and the root word itself. Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the language. Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization the root word is called Lemma. A lemma is the canonical form, dictionary form, or citation form of a set of words.

I have chosen lemmatization because it converts each word to a word that belongs to the language, which can be useful in a later stage of analysis or processing. The only drawback of using lemmatization is that if what matters is speed, it is slower because it has to scan a whole corpus to find the lemma. The choice between stemming and lemmatization really depends on the application you are working on

NLTK library[24] provides a lemmatizer that uses the WordNet Database to find the lemma. In order to find the correct lemma, the lemmatizer needs the context in which you want to lemmatize the word, that is, it needs the part-of-speech (POS) tag, and this is achieved by providing the POS tag given by the CMU ARK POS Tagger and Tokenizer.

Second Emoticon to CLDR short name

This is the second emoticon parser of the pre-processing pipeline. It is included because the POS tagger also detects emoticons, especially those written in strange characters or with some encoded characters. For those emoticons that for any reason have not been detected in Step 1, I add this extra emoticon-to-text parser. An example of what I mean by "strange characters" is the following sentence: `"c'mon now &gt;_&;gt"`. The POS tagger decodes `";gt"` into `">"`, creating the token `">_>"` with the tag `"EMOTICON"`. Then I check if this emoticon is in my custom dictionary, and I replace ">_>" by `"face with raise eyebrow"`. If the emoticon is not found in my emoticons dictionary, the token is removed.

Lowercase

At the end all tokens are lowercased as another stage of vocabulary reduction.

TF1: Stop-word removal technique

Stop-words, by definition, are meaningless words that have low discrimination power [24]. In NLP those words are removed from texts, helping to reduce vocabulary size, data sparsity and the feature space.

[23] https://www.datacamp.com/community/tutorials/stemming-lemmatization-python
[24] https://www.nltk.org/

In [25] they study the effect of six different stop-word removal methods on sentiment polarity classification of Twitter data. They use six different datasets, including the one from SemEval 2013 edition, which means that the conclusions of this work can be extrapolated to my scenario with similar training data. In the paper they show that the classical approach on stop-word removal, which consists in using a pre-compiled list of stop-words, have a negative impact on the performance of Twitter sentiment classification problems because it can remove relevant information or modify the context of a sentence. They show that the most effective technique is TF1, which basically removes those words that occur only once in the whole dataset. Compared to the other best-ranked method - Mutual Information (MI) - TF1 gives the best trade-off between good performance and processing time, hence is the recommended method for Twitter sentiment analysis tasks. As shown in figure 2.3, TF1 reduces around 65% of feature space on average.

The effect of TF1 method on my training dataset is noticeable, reducing the vocabulary size by 56.1%.

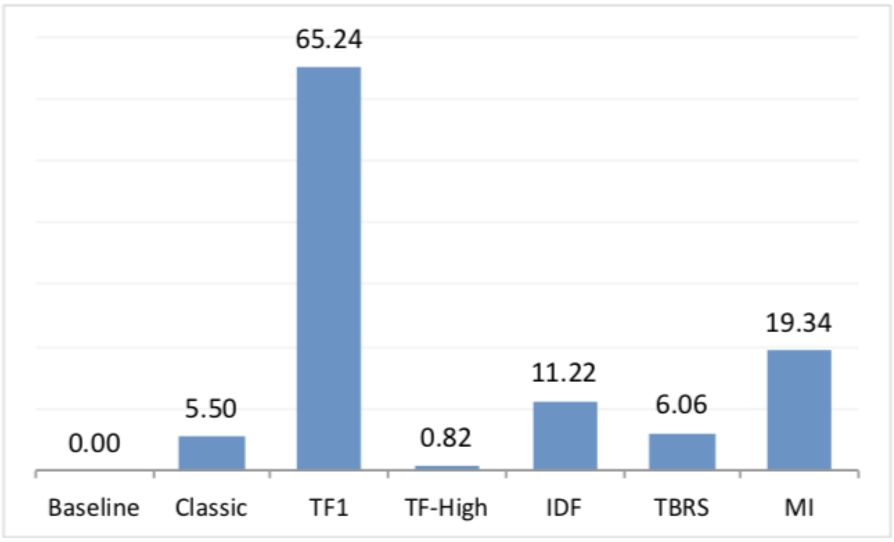
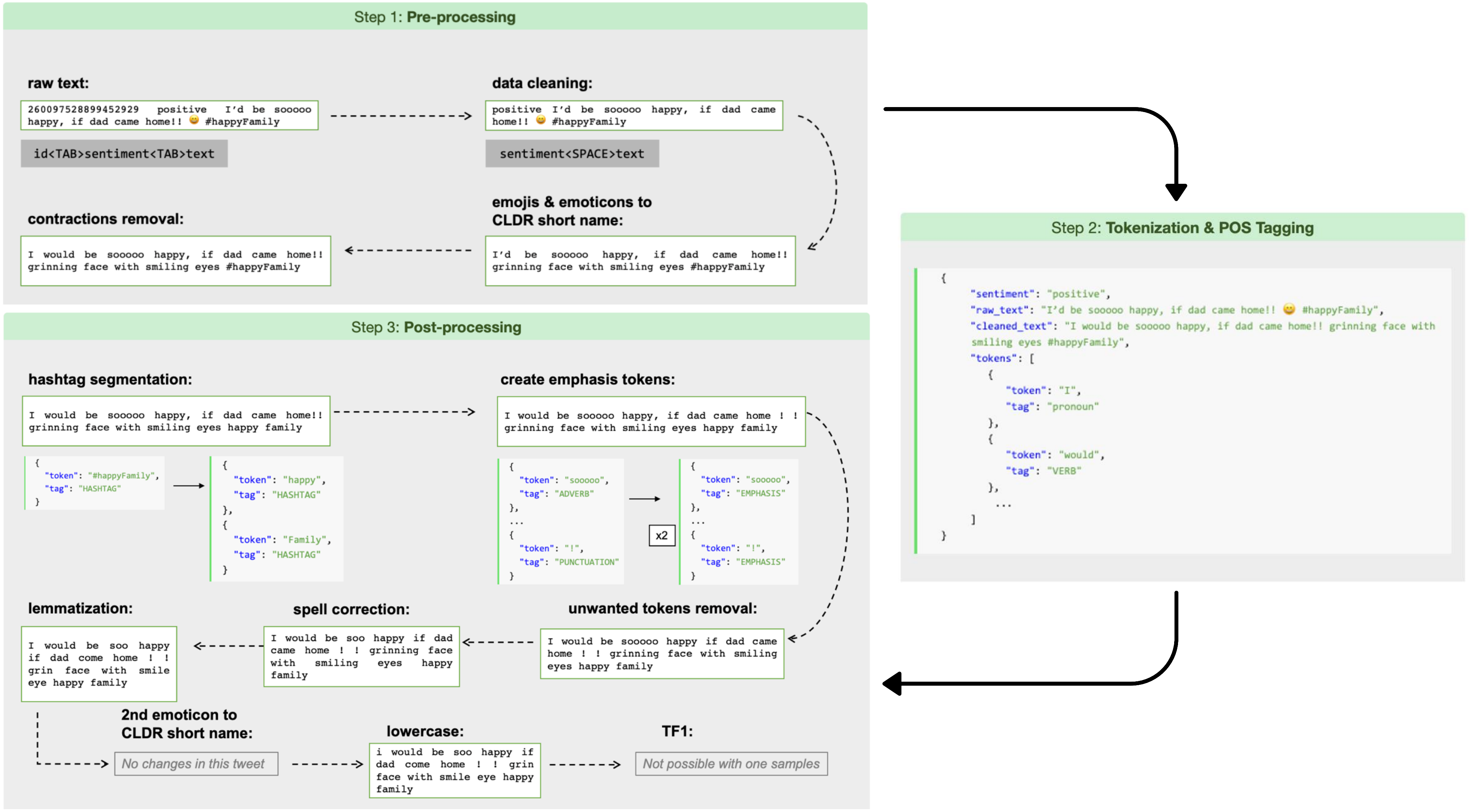


Figure 2.3: Reduction rate on the feature space of different stop-lists. Picture taken from *[25]*

At the end, all the processed tweets are put together into a *.json* file where each tweet is an object with the final structure shown in figure 2.4 (see next page). The figure shows a detailed example of the evolution of a tweet in every step of the pre-processing pipeline.


Step 1: Pre-processing
raw text:
data cleaning:
emojis & emoticons to CLDR short name:
contractions removal:

```
2600975288899452929   positive   I'd be sooooo happy, if dad came home!! 😄 #happyFamily
```

`id<TAB>sentiment<TAB>text`

```
positive I'd be sooooo happy, if dad came home!! 😄 #happyFamily
```

`sentiment<SPACE>text`

```
I'd be sooooo happy, if dad came home!! grinning face with smiling eyes #happyFamily
```

```
I would be sooooo happy, if dad came home!! grinning face with smiling eyes #happyFamily
```

Step 2: Tokenization & POS Tagging

```
{
    "sentiment": "positive",
    "raw_text": "I'd be sooooo happy, if dad came home!! 😄 #happyFamily",
    "cleaned_text": "I would be sooooo happy, if dad came home!! grinning face with
    smiling eyes #happyFamily",
    "tokens": [
        {
            "token": "I",
            "tag": "pronoun"
        },
        {
            "token": "would",
            "tag": "VERB"
        },
         ...
    ]
}
```

```
I would be sooooo happy, if dad came home!! grinning face with smiling eyes happy family
```

```
{
  "token": "#happyFamily",
  "tag": "HASHTAG"
}
```

```
{
  "token": "happy",
  "tag": "HASHTAG"
},
{
  "token": "Family",
  "tag": "HASHTAG"
}
```

```
I would be sooooo happy, if dad came home ! ! grinning face with smiling eyes happy family
```

```
{
  "token": "sooooo",
  "tag": "ADVERB"
},
...
{
  "token": "!",
  "tag": "PUNCTUATION"
}
```

```
{
  "token": "sooooo",
  "tag": "EMPHASIS"
},
...
{
  "token": "!",
  "tag": "EMPHASIS"
}
```

```
I would be sooooo happy if dad came home ! ! grinning face with smiling eyes happy family
```

```
I would be soo happy if dad came home ! ! grinning face with smiling eyes happy family
```

```
I would be soo happy if dad come home ! ! grin face with smile eye happy family
```

No changes in this tweet

```
i would be soo happy if dad come home ! ! grin face with smile eye happy family
```

Not possible with one samples

Figure 2.4: Example of a tweet passed through all the steps of the pre-processing pipeline

2.3 Handling imbalanced data

Although the training data is fairly reliable and big, it is highly unbalanced. There is a clear problem of class imbalance because there are many more instances of neutral and positive tweets than negative. As it can be seen in *table 1.1*, from the total of 53484 tweets, 15.89% are negative, 45.24% neutral (three times the amount of negative) and the remaining 38.87% are positive tweets.

Class imbalance is a common problem in ML and DL, especially in classification problems, and if it is not handled properly it can lead to a not accurate model performance. However, there is not a “go to” solution to solve the problem of class imbalance to improve the performance of a model, but different techniques have to be tried to find the best-suited method for the dataset at hand.

Next I will present different techniques to solve the class imbalance problem and I will explain which ones I have chosen, given my training dataset.

Gather more data
First, you can look for more data, but in my case it was difficult to find another source of representative data, and also makes sense to find a a dataset with a polarity distribution of Twitter data with more neutral and positive tweets than negative, so this solution of class imbalance was dismissed.

Change performance metrics
Secondly, you can change the performance metrics and use metrics that give better insights, but since I am using the same metrics as the ones used in *SemEval 2017 task4 - subtask A*, this option was also not considered. More details on the used performance metrics used can be found in section 2.3.

Use different machine learning algorithms
Thirdly, different machine learning algorithms can be tried, and then see which of them perform better with the dataset at hand and with imbalanced data. This part is explored in section section 2.5 and 2.6, where different classifiers are explored.

Resampling
There are also a number of resampling techniques that can be explored. These methods consist in adding samples to the minority class to match the number of samples of the majority class, or the opposite, which is removing samples of the majority class or classes to match the minority class. I have used random undersampling, random upsampling and data augmentation.

- ***Random downsampling:***

This technique consists in randomly removing samples of the majority classes, in my case the neutral and positive classes.
I use the resample module from scikit-learn[25] to remove samples from the neutral class to get the same number of samples as the negative class (8498).

- ***Random upsampling:***

This technique consists in randomly replicating samples of the minority class. I use again the resample module from scikit-learn[25] to randomly replicate tweets from the negative class.

- ***Data augmentation: Back Translation***

Data augmentation is a technique typically used in computer vision which consists in increasing the diversity of the training set by applying random (but realistic) transformations[26] by rotating, translating, scaling or adding noise to the image. This can also be applied to texts, where the approach can be shuffling the words of a sentence or replacing some words by their synonyms to create new texts with the same meaning. An extended list of different approaches to data augmentation for text are introduced in this article [27], and some implementations of those methods are implemented in this GitHub repository: https://github.com/kothiyayogesh/medium-article-code.

From the methods proposed in the aforementioned article, I have used Back Translation. This technique has given top results in several works. In [28] they used this technique for the *Toxic Comment Classification Challenge* on *Kaggle*, in which they ranked 1st. In [26] they used this method to augment unlabeled text and learn a semi-supervised model with only 20 labeled samples on the IMDB dataset, outperforming a previous state-of-the-art model trained with 25000 labeled samples.

The main idea of back translation is simple, it consists in the following steps:

- Translate text from English to another language (e.g. Catalan)
- Translate the translated text back to English
- Check if the new English sentence is different from the original. If it is different, then we keep the new sentence as the augmented version of the original text.

Since the *SemEval* dataset has very few negative tweets, each tweet is back translated twice, and from the resulting list of augmented tweets I randomly take the number of tweets I want to add to the dataset. That is, if the dataset had 8000 negative samples and 20000 samples would be needed to solve the class imbalance problem, we would need

[25] https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html
[26] https://www.tensorflow.org/tutorials/images/data_augmentation
[27] https://amitness.com/2020/05/data-augmentation-for-nlp/
[28] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52557

12000 new samples. This would mean that 16000 augmented samples would be generated (two for each original sample) and 12000 would be randomly taken.

In the article[27] they propose the use of the library TextBlob to do the translations, but I soon reached the limit of api calls that TextBlob makes "under the hood". I used *googletrans*[29] library instead, which is a free and unlimited library for Python that implements the *Google Translate API*. One positive thing about back translation is that misspelled words would be fixed or replaced by a word with a different meaning, which is perfect for the purpose of back translation. For instance, the text `its not that I'm a GSP fan (...)` would be back translated into `It's not that I'm a GSP fan (...)`.

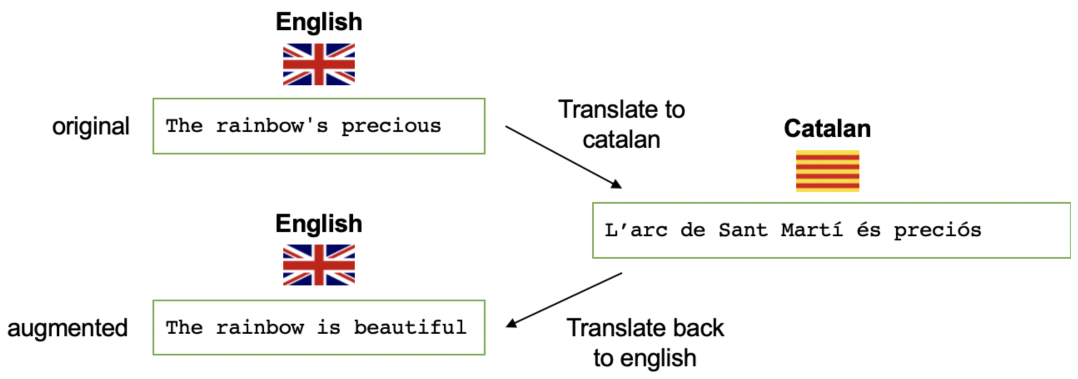


Figure 2.5: Back translation example. Inspired in a figure by Amit Chaudhary in [27]

Binary classification

Lastly, you can ignore the majority class or merge minority classes and try binary classification. In my case I discard the majority class, that is, all the neutral tweets that represent 45.21% of the total data, simplifying the classification problem to 2 classes with 29287 tweets, where 70.98% are positive tweets.

[29] https://py-googletrans.readthedocs.io/en/latest/

2.4 Performance metrics

To assess the performance of the methods proposed in this work, I have applied the same evaluation measures used in *SemEval 2017 task 4 - subtask A* [1] to be able to compare my classifiers performances with the methods presented in *SemEval*.

The primary metric used to rank the participants' results is macro-average recall (*AvgRec*, see next paragraph for a description of *AvgRec*). Besides *AvgRec*, they also show the results for two secondary measures, macro average F_1^{PN} and *accuracy*, which will also be explained later in this section. But before describing each measure, we must know what *macro-average* means. The *macro-average* evaluation consists in computing the measure individually for each class, and then taking the average of all classes giving equal weight to each class.

Macro average recall *(AvgRec)*

Briefly, recall is the ratio $\frac{tp}{tp+fn}$, where tp is the number of true positives and fn the number of false negatives, and it provides insights of the ability of a classifier to find all the positive samples. That is, given our dataset, it provides an idea of the ability of the classifier to classify correctly all the negative, neutral and positive samples. With the single label multi-class task at hand, recall for the positive class is computed as:

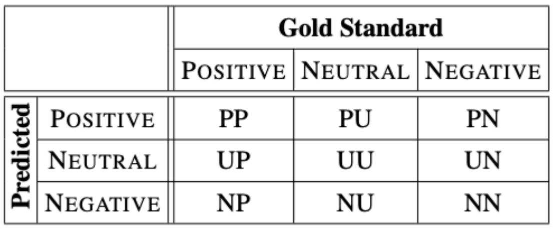
$$R^P = \frac{PP}{PP + UP + NP}$$

Where PP, UP, PN and NP are the cells of the confusion matrix of figure 2.6.

Then, *AvgRec* is the average recall across the positive (P), negative (N) and neutral (U) classes, and it is calculated as follows:

$$AvgRec = \frac{1}{3}(R^P + R^N + R^U)$$

where R^P, R^N and R^U are the recall of the positive, negative and neutral classes, respectively, and are calculated analogously to R^P. As mentioned in [1], *AvgRec* is more robust to class imbalance compared to F_1^{PN} and *accuracy*. For more details in the advantages of *AvgRec* see [27].



		Gold Standard		
		POSITIVE	NEUTRAL	NEGATIVE
Predicted	POSITIVE	PP	PU	PN
	NEUTRAL	UP	UU	UN
	NEGATIVE	NP	NU	NN

Figure 2.6: Confusion matrix of the classification of the tweets. Cell XY stands for "the number of tweets that the classifier labelled X and the gold standard labels as Y". P, U and N stand for positive, neutral and negative. Picture taken from table in [28].

Macro average F1

By definition, F_1 score is the harmonic mean of precision and recall[30]. With the task at hand, the F_1 score for the positive class, F_1^P, is calculated as follows:

$$F_1^P = 2 * \frac{precission^P * \ recall^P}{precission^P + \ recall^P}$$

Where $recall^P$ is the recall of the positive class explained before, and $precission^P$ is computed as follows:

$$precission^P = \frac{PP}{PP + PU + PN}$$

Where PP, PU, PN and NP are the cells of the confusion matrix of figure 2.6.

In [3] this measure is calculated over the positive and negative classes, excluding the neutral class. It is calculated as follows:

$$F_1^{PN} = \frac{1}{2} * (F_1^P + F_1^N)$$

where F_1^P and F_1^N refer to the F_1 of the positive and negative classes, respectively, and F_1^Ncomputation is analogous to F_1^P.

Accuracy

This metric indicates the overall percentage of correctly classified negative, neutral and positive tweets. That is, in our case this is computed as follows:

$$Acc = \frac{PP + UU + NN}{PP + PU + PN + UP + UU + UN + NP + NU + NN}$$

[30] https://en.wikipedia.org/wiki/F1_score

where $PP + PU + PN + UP + UU + UN + NP + NU + NN$ is basically the total amount of classified tweets.

2.5 CNN classifier

In this section is presented the design and implementation of the tweet polarity classifier based on the Convolution Neural Network (CNN) presented by Yoon Kim in [16]. The idea is to build a simple classifier and from there, try to improve its performance by fine-tuning the hyperparameters following a similar experimental approach as the one described in [17] to find the optimal configuration for my dataset. Once the optimal configuration is found, I try to improve the model by adding lexicon features to the feature vector from the CNN. The implementation of the CNN classifier can be found in GitHub[31].

2.5.1 CNN architecture

The CNN presented by Kim in [16] is a simple CNN with only one layer of convolution on top of word vectors. Figure 2.7 shows that architecture for a 2-class classification problem, which is analogous for the classification task at hand. We will note go into many details of the architecture and the CNN functioning, for more information see [16] and [17]. Nonetheless, some key aspects and elements of the architecture must be explained to understand how text is treated by the CNN.

Word embeddings
Word embeddings are vector representations of words that allows words with similar meaning to have a similar representation [29]. This representation is a vector, hence similar words would have very similar vectors, and if visualized in the vector space, they would be very close. Word embeddings have been widely used in NLP because they have the ability of capturing context of a word within a document, semantic and syntactic similarity with other words, etc. There are several available pre-trained word vectors like GloVe[32] and word2vec[33], and in [16] Kim states pre-trained word vectors perform very well and can be used as 'universal' feature extractors for text classification tasks.

Input of the CNN
The CNN gets as an input a tokenized sentence which is converted into a *sentence matrix*, where each row is a word vector representation of each token. This word representations are the outputs of the aforementioned pre-trained word embeddings. The word vectors are initialized with the vectors obtained from the pre-trained word vectors. Those words that are not found in the pre-trained word vectors are randomly initialized.
If we denote the dimensionality of the word vectors by *d*, and the length of a tweet by *s*, then the dimensionality of the *sentence matrix* is *s x d.* At this point we have a matrix of

[31]https://github.com/mitji/TFG_Twitter-Sentiment-Analysis-on-BBC-Proms/tree/master/classifiers
[32] https://nlp.stanford.edu/projects/glove/
[33] https://code.google.com/archive/p/word2vec/

s x d filled with numbers, which can be treated as an image [30], and therefore we can perform convolutions on it with linear filters.

Linear filter sizes

The linear filters slide over full rows of the matrix, that is, over the words. Thus, filters width has to be the same as the length of the word vectors, and then different filter heights (referred as filter size) are explored to detect different n-grams, since in each convolution the filters will be sliding over combination of *n* words, where *n* is the size of the filter. Hence, the idea of this filters is to detect representative n-grams patterns, and therefore different combinations of filter sizes must be explored to find the most representative n-grams for the dataset at hand.

Feature map

Each convolution will generate a feature map, so this will generate a number of feature maps of different dimensionality, which is given by the sentence length and the filter size. That is why a max-pooling operation is applied to each feature map so that all have the same size while they keep the most important information. Here we will be using 1-max pooling, which keeps the most representative feature from each feature map. Then all the feature maps are concatenated and passed through a softmax function to generate the final classification.

Regularization

Two common type of regularization techniques may be applied to prevent overfitting: *dropout* and *l2 norm constraint*.

The values for all these hyperparameters of the CNN are explained in the following section.

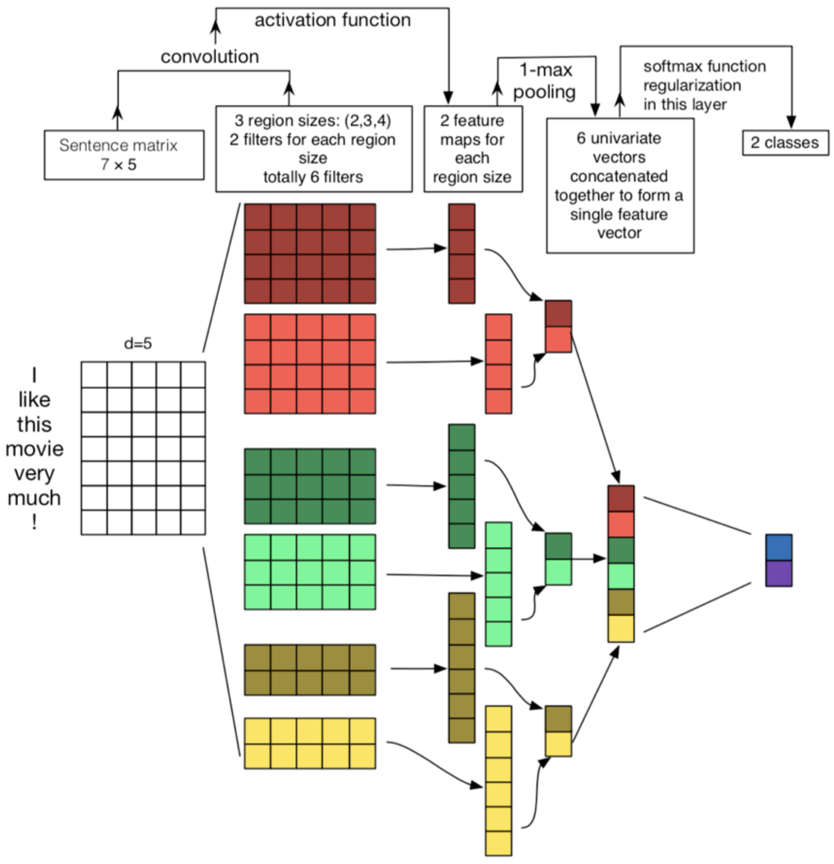

activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Figure 2.7: *Illustration of the architecture of a two-class classification CNN, analogous to the three-class classification task at hand. Picture taken from [17] .*

2.5.2 Baseline model

As explained at the beginning of this section, the CNN structure is the same as the one from Kim [16]. The baseline configuration for that CNN is described in table 2.1. This baseline model configuration is defined based on the conclusions from [17], and will be taken as a point of reference to compare the results of the hyperparameters tuning.

The parameters highlighted in grey, *activation function* and *polling strategy*, will be kept static during all experiments, since the chosen values are the values that have shown to be the best for text-classification tasks [17].

Hyperparameter	**Value**
Input word vectors	GloVe(200d)
Filter region size	(1, 1, 1)
Feature maps	100
Learning rate	0.001
Activation function	ReLu
Pooling strategy	1-max pooling
Regularization	
Dropout rate	0.5
L2 norm constraint	3

Table 2.1: CNN baseline configuration

The baseline model presented above achieves the following results:

	AvgRec	F_1^{PN}	***Acc***
Baseline model	61.90%	58.40%	62.40%

Table 2.2: Baseline model performance

In the next section we will see the effect of each of the hyperparameters on the performance of the model, and how they should be set to enhance the performance of the model.

2.5.3 Model optimization

First, we will try to improve the model performance by changing the learning rate and the filter sizes. Then we will find which is the best resampling technique to apply to our dataset in order to solve the class imbalance problem, which can lead to a bad model performance if it is not addressed properly. Once the optimal dataset is found, we will carry out several experiments with other different hyperparameter configurations to improve the baseline model configuration.

The idea is to experiment with each hyperparameter individually while keeping the other hyperparameters static, and then see how they can improve the baseline model. At the end, different classifiers with different fine-tuned combinations are built in order to find the best one.

Effect of learning rate

To find the best learning rate we will experiment with different values and see how the model learns with each value. Results are shown in table 2.3.

Learning rate	***AvgRec***	F_1^{PN}	***Acc***
0.001 - *baseline*	61.90%	58.40%	62.40%
0.0005	62.55%	60.60%	62.91%
0.0001	**62.89%**	62.66%	62.99%

Table 2.3: Results of different learning rates

In figure 2.8 we can see how the initial learning of *0.001* rate makes the model learning inconsistent, achieving the lowest validation loss in the first epoch. On the other hand, setting the learning rate to the half, to *0.0005* improves the model performance by a 0.65%, whereas an even smaller learning rate of *0.0001* improves the performance by a 0.99% if we increase the number of epochs to 10.

With the smallest learning rate, the model learns much slower and then captures more important information, but with the drawback of taking more time to train the model.

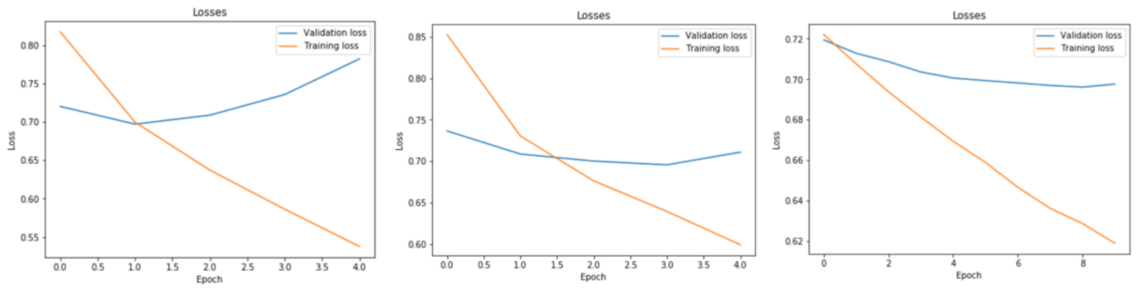


Figure 2.8: Evolution of the validation and training loss during epochs. From left to right: lr 0.001, lr 0.0005, lr 0.0001

Effect of filter region size

As mentioned in [17], the filter region size can have a large effect on performance, therefore it must be tuned, and each dataset has its own optimal filter region size. Since we are dealing with Twitter data, where we generally find short sentences, we will explore a range of region sizes from 1 to 4. The same approach as in [17] is followed to find the best filter region size. We first find the optimal single filter region size, and then explore combinations of several filter region sizes close to the optimal single size found before. The results obtained following this approach are shown in table 2.4.

Filter region size	***AvgRec***	$\boldsymbol{F_1^{PN}}$	***Acc***
(1, 1, 1) - *baseline*	61.90%	58.40%	62.40%
(2, 2, 2)	62.20%	59.57%	62.34%
(3, 3, 3)	61.01%	57.10%	61.56%
(4, 4, 4)	62.94%	60.93%	62.92%
(2, 3, 4)	62.93%	60.36%	62.80%
(1, 2, 3, 4)	61.67%	59.45%	63.80%
(3, 4, 5)	62.48%	60.98%	63.48%
(2, 2, 4, 4)	**63.28%**	61.08%	62.41%
(2, 2, 4)	61.32%	57.48%	62.03%
(2, 4, 4)	61.51%	59.74%	63.81%

Table 2.4: Results for different filter region sizes

For the *SemEval 2017 task 4* training dataset, the best single region size is 4, which suggests that the n-gram that best describes our data is 4-gram. Then we perform several experiments of combinations of different filter region sizes close to this single region, and, to the contrary of what is shown in [17], we do not achieve better results. We perform some more combination of filter region sizes based on the two best single region sizes, which are 2 and 4, getting an increase of 1.38% in *AvgRec* over the baseline model for the filter region sizes (2, 2, 4, 4).

Effect of resampling techniques

Here are presented the results for different resampled versions of the training dataset using the methods presented in section 2.3. The results are reported in table 2.5.

It can be seen that the two best resampling techniques are downsampling by 50% and upsampling by 244.54%, hence both will be considered in the final experiments.

Another thing we can also notice from the results shown in table 2.5 is the excellent performance of the model in the binary classification when dealing only with positive and negative samples. This shows the complexity added by the multi-class classification problem we are tackling in this work.

Resampling technique	***AvgRec***	F_1^{PN}	***Acc***
Original dataset – baseline	61.90%	58.40%	62.40%
Random downsampling			
Downsampling 65% (to fit number of negative samples)	62.38%	53.30%	54.38%
Downsampling 50%	**63.43%**	60.29%	59.51%
Downsampling 20%	62.60%	63.85%	63.08%
Random upsampling			
Upsampling 284.33% (to fit number of neutral samples)	61.92%	60.07%	61.95%
Upsampling 244.54% (to fit number of positive samples)	**62.99%**	61.93%	61.80%
Upsampling 200%	61.90%	60.29%	62.32%
Data augmentation			
10499 new negative tweets (to fit number of positive samples)	62.56%	62.29%	60.55%
5250 new tweets	62.38%	60.64%	62.89%
Binary classification	**84.86%**	85.88%	83.72%

Table 2.5: Results for the different resampling techniques applied to the train dataset

Effect of input word vectors

In [16] it is shown that pre-trained word vectors can be used across datasets, and using non-static vectors to fine-tune the word vectors for each specific task gives better results than static vectors.

Although GloVe vectors have been shown to perform poorly than Word2Vec with Twitter data [18], I have experimented with both of them to see which one performs better with my dataset and model architecture. In table 2.6 we can see that in this case using 200 dimensions GloVe vectors perform better than Word2Vec.

Input word vector	***AvgRec***	F_1^{PN}	***Acc***
GloVe (200d) - *baseline*	**61.90%**	58.40%	62.40%
Word2vec	59.85%	56.70%	60.32%

Table 2.6: Results for different input word vectors

Besides using pre-trained word vectors such as GloVe or Word2Vec, two other approaches can be followed to create the word vectors: 1) use word embeddings trained on our own dataset; 2) use character embeddings instead of word embeddings, i.e. FastText.

Effect of number of feature maps

The optimal number of feature maps for each filter region size depends on the dataset, but the recommended range is between 100 and 600 taking into account that increasing the number of feature maps will increase the training time of the model [17].

Num. of feature maps	***AvgRec***	F_1^{PN}	***Acc***
100 - baseline	61.90%	58.40%	62.40%
200	62.85%	60.48%	62.93%
300	**63.31%**	61.63%	62.76%
400	62.42%	60.96%	63.80%
500	61.56%	59.47%	63.12%
600	60.94%	59.36%	63.85%

Table 2.7: Results for different number of feature maps

From the table above we can see that the optimal number of feature maps is 300, with an increase of 1.41% over the baseline model.

Effect of regularization

Regularization must be taken into account specially when increasing the number of feature maps, which reduces the model performance due to overfitting. For these cases, we will see if increasing the dropout rate improves the performance.

As seen in table 2.7, the increase of the number of feature maps improves the model performance up to a size of 300. From there, the model performance starts dropping. In table 2.8 we show the results of avoiding that drop of the performance by increasing the dropout rate to 0.7. The result is an increase of a 1.57% over the baseline model.

Num. of feature maps	***AvgRec***	F_1^{PN}	***Acc***
400	**63.47%**	61.93%	62.59%
500	62.23%	60.26%	62.39%
600	62.30%	60.18%	63.23%

Table 2.8: Results for different number of feature maps with dropout rate set to 0.7

With all the information about the weight of each hyperparameter on the classifier performance, we can say that the hyperparameters that the model is more sensitive to are the *filter region size* and the *number of feature maps*. This makes a lot of sense since they determine the type and the number of features that are being extracted.

We also have seen how manually increasing the size of the train dataset by using different resampling techniques can improve the model performance.

In table 2.9 are presented the results for the different configurations I have experimented with combining different values of hyperparameters. It can be seen that the two best classifiers are achieved with the configuration presented in table 2.10 and with downsampling and upsampling the dataset. Although the best *AvgRec* result is achieved with the downsampling technique, the chosen model is the one with the upsampled dataset. The reason why this model has been chosen is that although it has a slightly lower performance on *AvgRec*, the other two measures obtained are a 3% higher than with the downsampled dataset. This drop in F_1^{PN} and *accuracy (Acc)* can be understood by looking to the confusion matrix of this model in figure 2.9. As it can be seen, the model trained with the downsampled dataset performs remarkably well in classifying positive and negative samples but fails to classify the neutral samples, classifying 37.11% of them as negative samples. Thus, the decision on the final CNN classifier is made with a trade-off on the correctly classification of negative samples, in order to have a more balanced classifier.

Model configuration	***AvgRec***	$\boldsymbol{F_1^{PN}}$	***Acc***
400 filters, 2244, dropout 0.7, lr 0.001	63.00%	62.49%	63.12%
300 filters, 2244, dropout 0.5, lr 0.001	63.36%	61.90%	62.27%
Downsampling 50%	64.16%	59.39%	59.42%
Upsampling 244.54%	**64.02%**	62.25%	62.58%
300 filters, 2244, no dropout, lr 0.001	63.71%	62.88%	63.29%
300 filters, 222, no dropout, lr 0.0005	60.51%	61.11%	62.81%
300 filters, 222, dropout 0.5, lr 0.0005	62.68%	62.34%	63.18%

Table 2.9: Results for different combinations of hyperparameters

Table following table shows the hyperparameter configuration for the final model:

Hyperparameter	**Value**
Input word vectors	GloVe(200d)
Filter region size	(2, 2, 4, 4)
Feature maps	300
Learning rate	0.001
Activation function	ReLu
Pooling strategy	1-max pooling
Dropout rate	0.5
L2 norm constraint	3

Table 2.10: Final hyperparameter configuration

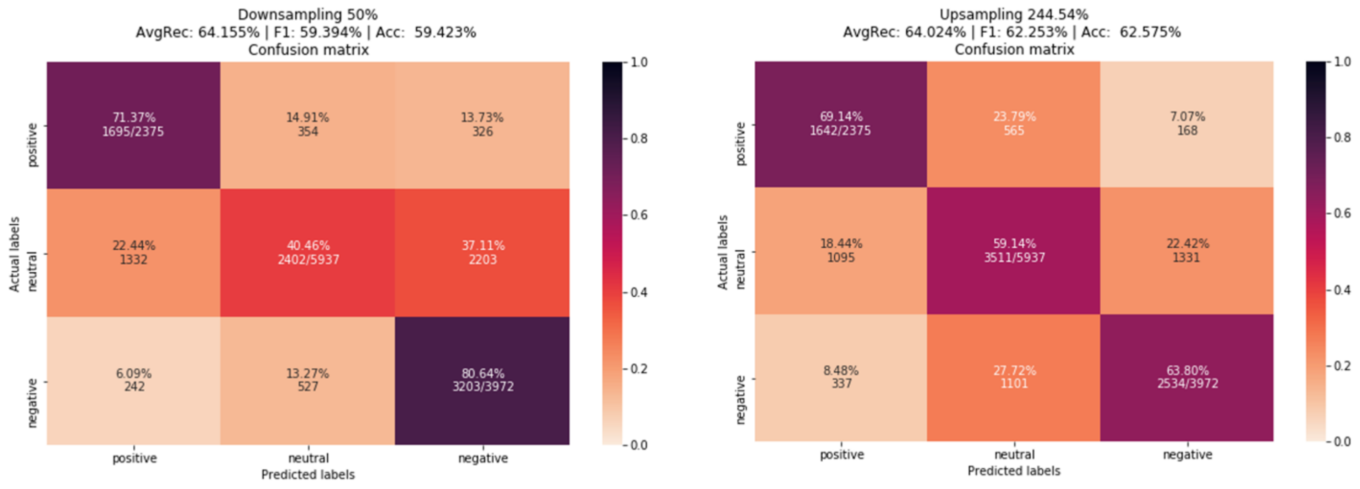


Figure 2.9: Confusion matrices for the two best models

2.5.4 Adding lexicon features

Now that we have the best configuration for the architecture of the network, we will see if adding lexicon features to the feature vector of the CNN can improve the model performance.

Lexicons are used in rule-based systems to compute the overall sentiment of a tweet, and a major part of sentiment analysis systems rely on them. Besides this, it has been shown that sentiment lexicon based features are the most influential features in the improvement of a logistic regression classifier performance [31]. We will see if this is also true for a CNN based classifier.

Next I will explain the different lexicons I have used, and which features are extracted from them.

In this work I have used four automatic generated lexicons, namely: ML-SentiCon [32], accessible in http://www.lsi.us.es/~fermin/index.php?title=Datasets; SCL-NMA [33]; SCL-OPP [34], [35]; SemEval-2015 English Twitter Lexicon [36]. For more information about the lexicons check the related papers, and for the former three also see http://saifmohammad.com/WebPages/lexicons.html.

For each of the automatic constructed lexicon, the same features described in [37] are added to the feature space of a tweet, which are the following: (1) *tweet polarity,* (2) *average polarity of the positive terms,* (3) *the average polarity of the negative terms,* (4) *the score of the last positive term,* (5) *the score of the last negative term,* (6) *the maximum positive score* and (7) the *minimum negative score.*

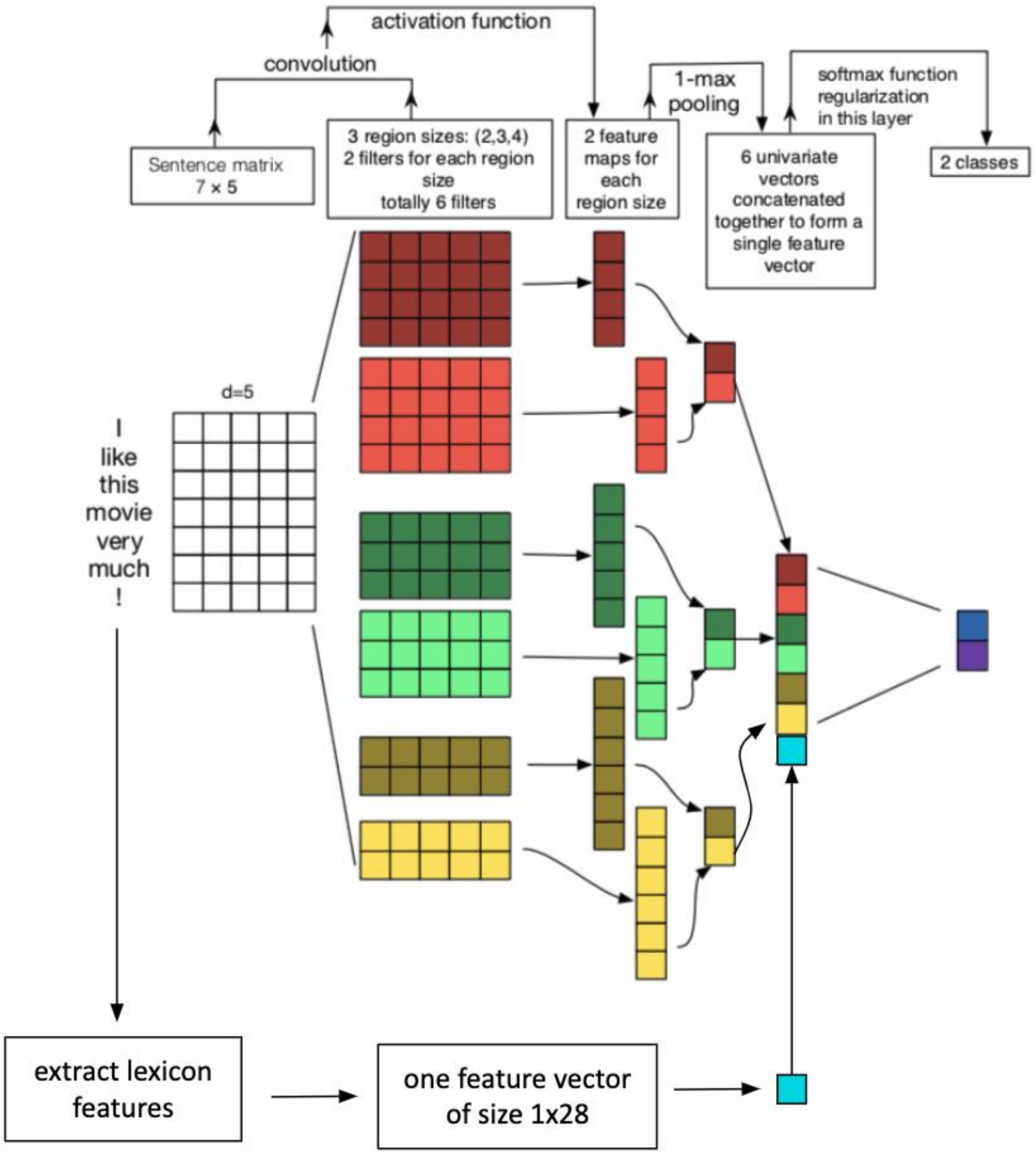
The *polarity* of a tweet *T* given a lexicon *L* is computed as follows:

$$polarity = \begin{cases} 1 - \frac{N}{P} & ; if P > N \\ 0 & ; if P = N \\ \frac{P}{N} - 1 & ; if P < N \end{cases}$$

Figure 2.10: Tweet polarity computation. Picture taken from [37]

Where *P* and *N* are the number of positive and negative tokens found in the lexicon, respectively. If *N* or *P* are zero, the divisions are not possible, and then the tweet polarity is equal to the average polarity of the nonzero polarity tokens. That is, if *P* > *N* but *N* is *0*, then *polarity = mean of polarity of list of positive tokens*

The lexicon features are extracted for each input tweet and added to the final feature vector, as it can be seen in the figure below. For each tweet 28 different features are extracted, as a result of the seven features extracted with each of the four lexicons.



activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I like this movie very much !
extract lexicon features
one feature vector of size 1x28

Figure 2.11: Architecture of the CNN with the addition of the lexicon features (in cyan color) to the feature vector. The picture is a modification of a picture from [17].

The improvements of the addition of 28 features to the feature vector are reported in table 2.11. The table shows the results of the baseline configuration with both systems, and the best models achieved with them. It can be seen that the CNN + lexicon method performs better than the basic CNN. In the case of the CNN + lexicon system the best models is obtained with the configuration seen in table 2.10 and trained with the dataset with the neutral class downsampled to the 50%, with a gain of 2.48% in *AvgRec* in relation to the baseline model of the initial CNN architecture. The improvements in relation to the best model achieved with the initial CNN architecture are of a 0.22%.

Model configuration	***AvgRec***	F_1^{PN}	***Acc***
CNN			
Baseline	61.90%	58.40%	62.40%
Downsampling 50%	64.16%	59.39%	59.42%
Upsampling 244.54%	64.02%	62.25%	62.58%
CNN + lexicon			
Baseline	62.66%	60.69%	64.51%
Downsampling 50%	**64.38%**	61.27%	60.45%
Upsampling 244.54%	64.10%	63.02%	61.78%

Table 2.11: Results of experiments with the CNN + lexicon features

2.6 Machine learning classifiers

2.6.1 Feature extraction

In order to use text in machine learning algorithms, it has to be converted into a numerical representation so that it can be passed to the algorithm, similarly to how the CNN extracts features from vector representations of each word.

I have followed a similar approach to the one proposed in [37], which ranked 8th in *SemEval 2017 task 4 - subtask A*. They present a representation of tweets using a rich set of features using a Support Vector Machine to classify the tweets.

In this work I have used three different types of features: *basic features*, *syntactic features* and *lexicon* features. Next I will describe each group of features.

Basic features
We will compute the presence or absence of contiguous sequences of n-grams of 1, 2, 3 or 4. An n-gram is a contiguous sequence of n words in a text, which can be used in text classification to extract contextual information from a text, and is one of the most widely used features.

Syntactical features
These types of features are very useful to distinguish between neutral and non-neutral tweets. It has been shown that subjective and objective texts have different POS tags [38], and as stated in [39], non neutral words in Twitter data are more likely to present the following POS tags: nouns, adjectives, adverbs, abbreviations and interjections.
Hence, the number of occurrences of nouns, adjectives and adverbs are added to the feature vector.

In addition to this, in the pre-processing I have manually added the POS tag 'EMPHASIS' to exclamation marks, capitalized words and words with repeated vowels (see section 2.2.3 - Step 3). Those tokens generally express some sort of subjectivity, therefore the number of emphasis tokens present in the tweet is added to the feature space.

This result in a total of four syntactical features added to the feature space.

Lexicon features
I use the same lexicon features explained in section 2.5.4.

2.6.2 Algorithms

For the implementation of the machine learning classifiers I have used the SGDClassifier module from scikit learn. This module implements several regularized linear models with stochastic gradient descent (SGD) learning, and allows a high level hyperparameter optimization. For more details on the SGDClassifier module, check the documentation on scikit-learn[34].

Two widely used algorithms for classification problems have been used for this task: Logistic Regression and Support Vector Machine (SVM).

Logistic Regression
Logistic regression is one of the most common classification algorithms for supervised learning. It is a linear model trained on labelled data, and typically used for binary classification. It explains the relationship between variables. Given an input variable *x*, logistic regression is applied, and the output variable *y* is a discrete value from 0 to 1.
Still, it can also be used for multi-class classification problems using the One-vs-all or One-vs-one approaches[35]. In this work I use the one-vs-all approach, which consists in training a single classifier for each class that is able to predict whether a sample is from that class or not.

Support Vector Machine (SVM)
SVM is very similar to logistic regression, and is used for classification and regression problems, although is mainly used for classification problems. It creates a decision boundary which separates non-linear data distribution into classes.

2.6.3 Results

N-gram optimization
Some experiments are done to find the combination of n-grams that provide the best representation of a tweet sentiment. Using 1 to 4 n-grams shows the best results when tested individually. However, when combined with the other two types of features, the best results are obtained with representations of 1 to 3 n-grams, so the last is the final configuration used.

- *Number of features:*

The n-gram extraction generates hundreds of thousands of features, which can become difficult to handle for the classifier. In [36] a comparison of n-gram combinations from 1 to

[34] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[35] https://en.wikipedia.org/wiki/Multiclass_classification#One-vs.-rest

[36] https://towardsdatascience.com/another-twitter-sentiment-analysis-with-python-part-5-50b4e87d9bdd

3 n-grams with different number of features is presented, and a reasonable number of features seems to be 100000, as shown in figure 2.12.

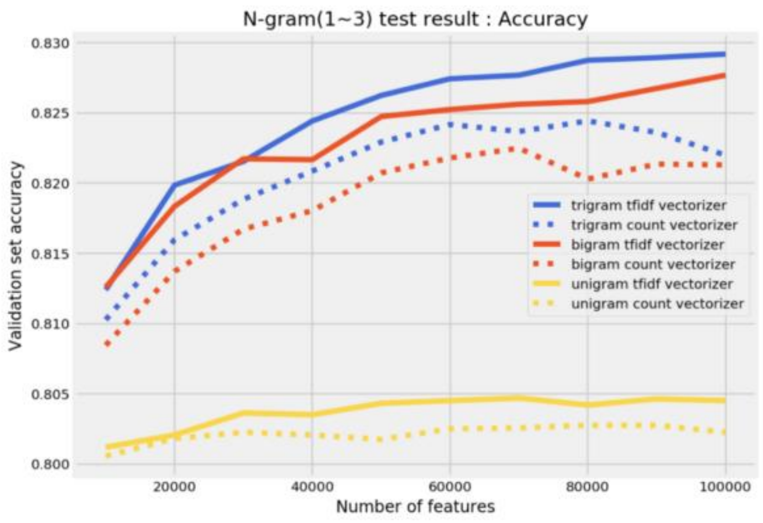


Figure 2.12: Comparison of different combination of n-grams with different number of features. Picture taken from [36]

Hyperparameter optimization

Four hyperparameters have been tuned to obtain the best possible results. The other hyperparameters are kept with their default configuration as they show the best results. The values of *learning*_rate and *validation_fraction* are obtained experimentally, being the combination shown in table 2.12 the best both for the SVM and the Linear Regression classifiers.

If *early_stopping* is set to true, we need to provide the classifier with a validation set to evaluate the model in each iteration and stop training when validation loss has not improved after five iterations. This reduces the time of model building and ensures that the model performs correctly with unseen data. As mentioned at the beginning of section 2, the chosen percentage for the train and validation split is a 15%, which shows to be the optimal percentage.

Hyperparameter	**value**
learning_rate	0.05
max_iter	300
early_stopping	True
validation_fraction	0.15

Table 2.12: Hyperparameter configuration of the SGDClassifier both for the SVM and the Linear Regression classifiers

Experiments for each individual group of features have been done to see which features are the most representative.

	Syntactical			Ngrams			Lexicon			Combined		
	AvgRec	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*
SVM	*37.48%*	*18.48%*	*48.21%*	*56.72%*	*52.22%*	*59.72%*	*46.54%*	*31.37%*	*50.93%*	*58.86%*	*54.87%*	*60.43%*
LR	35.19%	8.47%	47.75%	57.12%	52.84%	59.61%	48.78%	37.60%	52.58%	**59.34%**	54.98%	59.82%

Table 2.13: Results with test set on each individual group of features

Comparing the results of table 2.13 with the final CNN results (see table 2.9), the numbers presented here are very poor. In section 2.3 we have seen that the dataset is highly imbalanced, and different techniques to solve this problem are presented. Next we will show the results of the two machine learning classifiers trained with the different resampling techniques presented in section 2.3.

- Downsampling:

	65%			50%			20%		
	AvgRec	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*
SVM	55.85%	53.37%	46.19%	60.24%	58.39%	57.49%	59.27%	56.31%	59.15%
LR	57.82%	55.54%	49.06%	60.58%	58.50%	55.10%	60.43%	57.43%	58.44%

Table 2.14:Results for the classifiers trained with downsampled data

- Upsampling:

If we upsample the dataset, the number of samples needed for validation have to be similar to the number of samples used in the previous methods, therefore a lower *validation_fraction* split is being used in this case, 0.05, because the validation dataset used for the CNN is not suitable for the SGDClassifier.

	284.33%			244.54%			200%		
	AvgRec	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*
SVM	59.95%	56.43%	60.11%	60.85%	58.45%	60.97%	59.04%	54.95%	60.05%
LR	61.03%	58.71%	59.70%	60.34%	57.93%	61.30%	60.04%	57.33%	61.50%

Table 2.15: Results for the classifiers trained with upsampled data

- Data augmentation:

Working with this resampling technique, the best results are obtained with a value of *validation_fraction* of 0.10, since the algorithms are also being trained with a bigger dataset.

	10499 new neg tweets			**5250 new neg tweets**		
	AvgRec	F_1^{PN}	*Acc*	*AvgRec*	F_1^{PN}	*Acc*
SVM	59.54%	58.42%	59.30%	59.40%	58.17%	59.05%
LR	61.13%	59.82%	59.79%	60.09%	58.94%	59.52%

Table 2.16: Results for the classifiers trained with augmented data

- Binary Classification:

	Binary classification		
	AvgRec	F_1^{PN}	*Acc*
SVM	80.43%	78.60%	79.08%
LR	81.43%	79.69%	80.16%

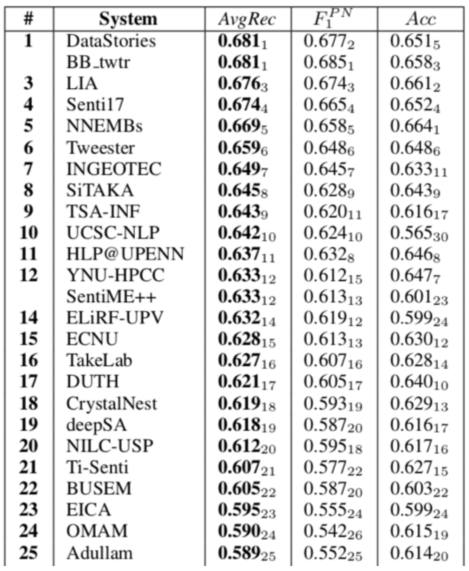
Table 2.17: Results for the binary classifiers

From these results we can conclude that the best classifier is the Logistic Regression classifier with augmented data, with an *AvgRec* of 61.13%, what represents an improvement of a 1.79% from the Logistic Regression model trained with the original imbalanced dataset.

2.7 Final Classifier

2.7.1 Benchmark

As mentioned before in this work, the results of *SemEval 2017 task 4 – subtask A* are the benchmark against which I measure my classifiers' performances, since the same data is being used for the training and the evaluation of the systems. Figure 2.13 shows the top 25 results of *SemEval 2017 task 4 – subtask A* competition with their respective performance metrics. For more details on the results and the task see [3].



#	System	*AvgRec*	F_1^{PN}	*Acc*
1	DataStories	$\mathbf{0.681}_1$	0.677_2	0.651_5
	BB_twtr	$\mathbf{0.681}_1$	0.685_1	0.658_3
3	LIA	$\mathbf{0.676}_3$	0.674_3	0.661_2
4	Senti17	$\mathbf{0.674}_4$	0.665_4	0.652_4
5	NNEMBs	$\mathbf{0.669}_5$	0.658_5	0.664_1
6	Tweester	$\mathbf{0.659}_6$	0.648_6	0.648_6
7	INGEOTEC	$\mathbf{0.649}_7$	0.645_7	0.633_{11}
8	SiTAKA	$\mathbf{0.645}_8$	0.628_9	0.643_9
9	TSA-INF	$\mathbf{0.643}_9$	0.620_{11}	0.616_{17}
10	UCSC-NLP	$\mathbf{0.642}_{10}$	0.624_{10}	0.565_{30}
11	HLP@UPENN	$\mathbf{0.637}_{11}$	0.632_8	0.646_8
12	YNU-HPCC	$\mathbf{0.633}_{12}$	0.612_{15}	0.647_7
	SentiME++	$\mathbf{0.633}_{12}$	0.613_{13}	0.601_{23}
14	ELiRF-UPV	$\mathbf{0.632}_{14}$	0.619_{12}	0.599_{24}
15	ECNU	$\mathbf{0.628}_{15}$	0.613_{13}	0.630_{12}
16	TakeLab	$\mathbf{0.627}_{16}$	0.607_{16}	0.628_{14}
17	DUTH	$\mathbf{0.621}_{17}$	0.605_{17}	0.640_{10}
18	CrystalNest	$\mathbf{0.619}_{18}$	0.593_{19}	0.629_{13}
19	deepSA	$\mathbf{0.618}_{19}$	0.587_{20}	0.616_{17}
20	NILC-USP	$\mathbf{0.612}_{20}$	0.595_{18}	0.617_{16}
21	Ti-Senti	$\mathbf{0.607}_{21}$	0.577_{22}	0.627_{15}
22	BUSEM	$\mathbf{0.605}_{22}$	0.587_{20}	0.603_{22}
23	EICA	$\mathbf{0.595}_{23}$	0.555_{24}	0.599_{24}
24	OMAM	$\mathbf{0.590}_{24}$	0.542_{26}	0.615_{19}
25	Adullam	$\mathbf{0.589}_{25}$	0.552_{25}	0.614_{20}

Figure 2.13: Top 25 results from SemEval 2017 task 4 – subtask A. A total of 38 teams participated in this subtask. Picture is taken from [3]

2.7.2 Machine learning and deep learning classifiers comparison

As a final step of the implementation of the classifier that we are going to use to classify the *BBC Proms* tweets, we will compare the best classifiers achieved with the Machine Learning and deep learning techniques, and see how good they are comparing it to the results from *SemEval 2017 task 4 – subtask A.*

In table 2.18 we can clearly see that the best model is the final CNN + lexicon model, which obtains an *AvgRec* of 64.38%. Comparing it with the benchmark results shown in figure 2.13, we can see it performs very well, being between the top 8 methods. The Logistic Regression classifier achieves similar results to the 20th ranked classifier.

Therefore, the classifier that we will be using to classify the *BBC Proms* in next section is the CNN + lexicon classifier from the CNN implementation.

	AvgRec	F_1^{PN}	***Acc***
Best CNN (CNN + lexicon)	**64.38%**	61.27%	60.45%
Best Logistic Regression	61.13%	59.82%	59.79%

Table 2.18: Top performances achieved with each of the explored classifiers

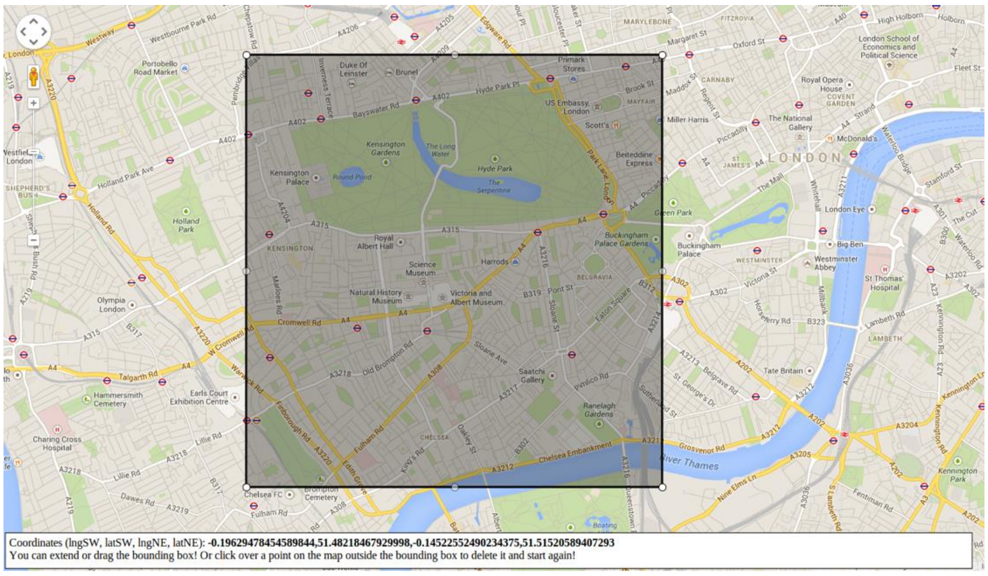
3. ANALYSIS OF BBC PROMS TWEETS

Once we have the optimal Twitter classifier, we are ready to classify the tweets from BBC Proms. The tweets to be analyzed are from the *BBC Proms* editions from 2014, 2016 and 2017. This data is provided by Horacio Saggion, so if you are interested in accessing this data, please contact him at horacio.saggion@upf.edu.

Once the tweets are classified, I will build a dashboard with different visualizations of the classified data with Kibana[37]. Kibana is a powerful visualization tool that allows you to visualize data in different ways, and apply filters to analyze specific entities of your data, filter by date, by topic, keywords, etc. The visualizations will make the analysis on the classification results much easier and understandable.

3.1 Data summary

For each of the *BBC Proms* edition, two group of tweets are provided. One group is of tweets filtered by the location where the set of concerts are held. This location is shown in figure 3.1. The other group is a set of tweets filtered by hashtags, where the hashtags used are *"bbcproms"*, *"bsproms"* and *"bbcprom"*, and filtered by the users @bbcproms[38] and @RoyalAlbertHall[39]. Table 3.1 shows the number of tweets for each group and each edition. The tweets from both groups of each edition are added together and the repeated tweets are removed, then all of the tweets from the three editions are put together into a single file which will be used for the classification and analysis.



Coordinates (lngSW, latSW, lngNE, latNE): **-0.19629478454589844,51.48218467929998,-0.14522552490234375,51.51520589407293**
You can extend or drag the bounding box! Or click over a point on the map outside the bounding box to delete it and start again!

Figure 3.1: Screenshot of the search area of the group of tweets filtered by location

[37] https://www.elastic.co/kibana
[38] https://twitter.com/bbcproms
[39] https://twitter.com/royalalberthall

	Location filter	**Hashtag filter**	**all**
2014	181602	51435	149962 (47.21%)
2016	82947	81736	109672 (34.53%)
2017	19999	80434	57985 (18.26%)
		Total	*317619*

Table 3.1: Number of tweets for each group of tweets of each edition. The last column shows the number of unique tweets

Having duplicated samples can make the analysis very tough because it adds a lot of repeated data that can lead to a not accurate analysis. As seen in table 3.1, the last column shows the total number of tweets once the duplicates are removed, which is much lower than the sum of location and hashtag without removing duplicated samples. At the end we obtain a total of 317619 tweets to analyse, from which the 47.21% are from the 2014 edition, 34.53% from 2016, and the remaining 18.26% from 2017 edition.

The differences in the size of the available data for each year is due to two things. First it may be that less data has been scrapped from each edition. Secondly, when loading the files, some files had "corrupted" lines that were not able to be read in Python, and these lines were omitted for the final group of tweets.

3.1 Data classification

Before the actual classification of all the data, we will see how the CNN + lexicon classifier performs with manually annotated data from the *BBC Proms*. To do this, I have manually annotated 110 tweets from the 2014 edition, where there are 73 positive samples, 16 neutral, and 21 negative samples. The classifier achieves an *AvgRec* of 73.27%. The results are shown in table 3.2. It is also shown the confusion matrix of the classification results (see figure 3.2), where we can see that the CNN + lexicon model is very robust predicting positive and neutral samples, and very decent for the negative class.

AvgRec	$\boldsymbol{F_1^{PN}}$	***Acc***
73.27%	63.16%	78.84%

Table 3.2: Results of the CNN + lexicon model evaluated on BBC Proms manually annotated data

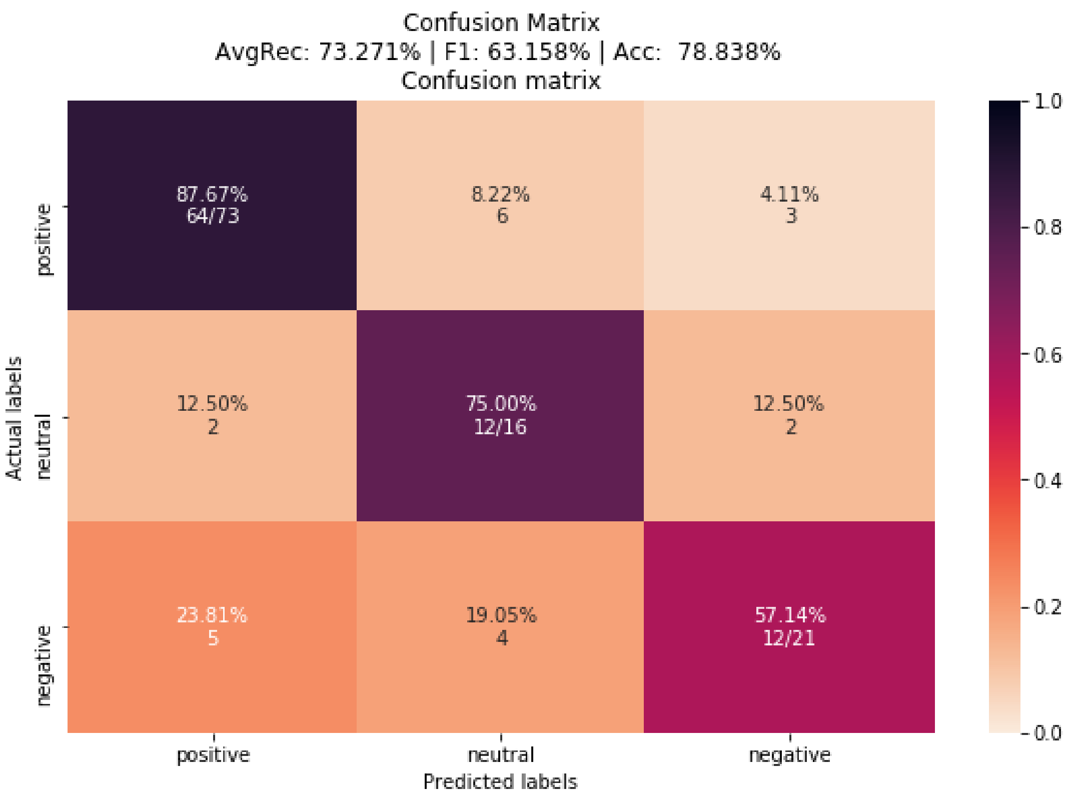


Figure 3.2: Confusion matrix of the evaluation of the CNN + lexicon model with manually annotated BBC Proms

Finally, the classification results on the *BBC Proms* tweets are shown in the table below.

	negative	**neutral**	**positive**
2014	25083	49887	66495
2016	6349	47228	47248
2017	4541	21700	31744
Total	35973	118815	145487

Table 3.3: Results of the classification of the tweets from BBC Proms

3.2 Data visualizations

After passing the tweets through the pre-processing pipeline and classifying them, all the tweets are stored in *.jsonl* format, one of the formats of input data accepted in Kibana. Each tweet is represented by an object with the following fields: *created_at*, *user*, *raw_text*, *is_retweet*, *hashtags*, *user_mentions*, *tokens*, *sentiment*, *proper_nouns*, *adjectives*. This fields carry a lot of information about a tweet, apart from the polarity sentiment, and will become useful to perform different visualizations and apply different filters on Kibana.

For the visualizations, I have built a dashboard where all the visualizations can be seen and modified by applying filters to them. If you want to take a look to the dashboard, you can access it in this link*: https://1b29a17d68924f4c8434d9ce1af8e164.eu-west-3.aws.elastic-cloud.com:9243/app/dashboards#/view/49a719f0-ee25-11ea-b2f9-2b1fb5a19ff3?_g=(filters%3A!()%2CrefreshInterval%3A(pause%3A!t%2Cvalue%3A0)%2Ctime%3A(from%3A'2014-07-08T22%3A00%3A00.000Z'%2Cto%3A'2017-09-24T18%3A19%3A33.090Z')).

The link will ask for a username and password, which are given below:

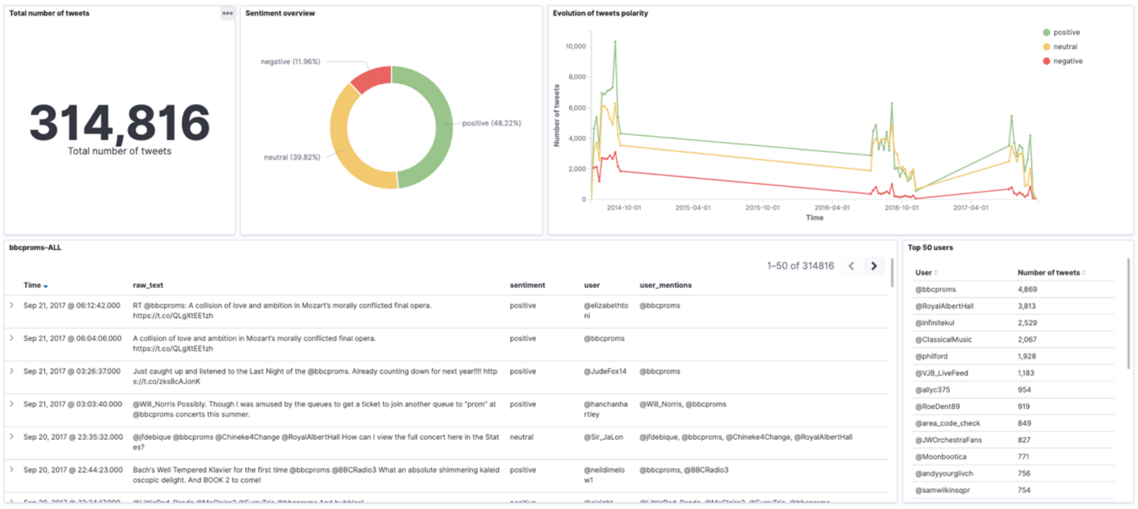
- username: *demo*
- password: *demopass*

**NOTE*: *I do not guarantee that the links to the dashboards will be available after 20/09/2020. If you are interested in consulting the visualizations and the link does not work, please contact me at d.mitjana@gmail.com.*

Next I will introduce the dashboard built with Kibana and I will describe the visualizations that have been implemented and which type of filtering can be applied.

The first set of visualizations of the dashboard aims to summarize the polarity distribution of the tweets, see the evolution of the overall sentiment during all of the three editions, and a list of all the classified tweets. Those visualizations can be seen in figure 3.3.

After this, several visualizations on specific entities of the tweets are made. Those include the top 50 hashtags, the top 50 mentions of users, the top 100 used keywords, and the top 100 proper nouns and adjectives. These specific entities are extracted thanks to the data provided by Twitter when scrapping tweets and thanks to the Twitter POS tagger from CMU Ark. In those visualizations, represented as word clouds, it is easy to find the most used words in each of them, and we can easily filter by these keywords to see all the tweets where these words appear. For more detail on those word clouds, see next section.



	Time	raw_text	sentiment	user	user_mentions
›	Sep 21, 2017 @ 06:12:42.000	RT @bbcproms: A collision of love and ambition in Mozart's morally conflicted final opera. https://t.co/QLgXtEE1zh	positive	@elizabethtoni	@bbcproms
›	Sep 21, 2017 @ 06:04:06.000	A collision of love and ambition in Mozart's morally conflicted final opera. https://t.co/QLgXtEE1zh	positive	@bbcproms	
›	Sep 21, 2017 @ 03:26:37.000	Just caught up and listened to the Last Night of the @bbcproms. Already counting down for next year!!!! https://t.co/zks8cAJonK	positive	@JudeFox14	@bbcproms
›	Sep 21, 2017 @ 03:03:40.000	@Will_Norris Possibly. Though I was amused by the queues to get a ticket to join another queue to "prom" at @bbcproms concerts this summer.	positive	@hanchanhartley	@Will_Norris, @bbcproms
›	Sep 20, 2017 @ 23:35:32.000	@jfdebique @bbcproms @Chineke4Change @RoyalAlbertHall How can I view the full concert here in the States?	neutral	@Sir_JaLon	@jfdebique, @bbcproms, @Chineke4Change, @RoyalAlbertHall
›	Sep 20, 2017 @ 22:44:23.000	Bach's Well Tempered Klavier for the first time @bbcproms @BBCRadio3 What an absolute shimmering kaleidoscopic delight. And BOOK 2 to come!	positive	@neildimelow1	@bbcproms, @BBCRadio3

User	Number of tweets
@bbcproms	4,869
@RoyalAlbertHall	3,813
@infinitekul	2,529
@ClassicalMusic	2,067
@philford	1,928
@VJB_LiveFeed	1,183
@allyc375	954
@RoeDent89	919
@area_code_check	849
@JWOrchestraFans	827
@Moonbootica	771
@andyyourglivch	756
@samwilkinsqpr	754

Figure 3.3: Overview of the dashboard

3.3 Analysis

The first thing we can notice in the evolution of the sentiment polarity through time, which can be seen in more detail in figure 3.4, is that the first noticeable spike of positive tweets is found in the first week. This looks logical, since at the beginning everyone might be excited for *The Proms*. Another thing we can notice is that the number of negative, neutral and positive tweets generally follow a similar evolution, which does not give very useful insights of the overall perception of the *BBC Proms*.

However, in the same figure we can see that the number of positive tweets is much higher than the neutral or negative ones. One may think that this is caused by tweets generated by non-personal accounts. We then remove all the tweets created by official and informative accounts that can be find in the top of the table shown in figure 3.5, and the results are still the same. So, in general terms, we can conclude that *BBC Proms* are very well perceived by the audience.

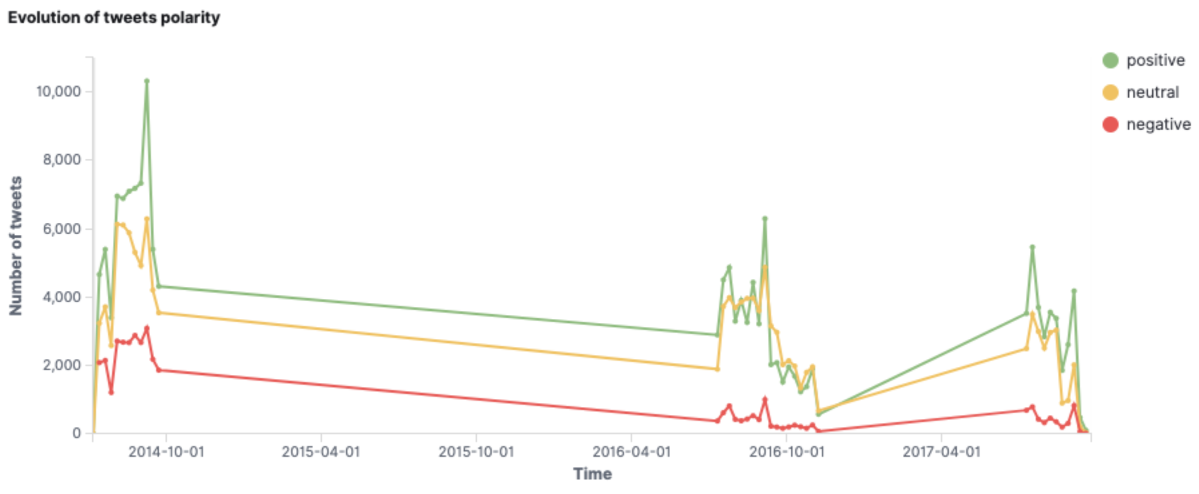
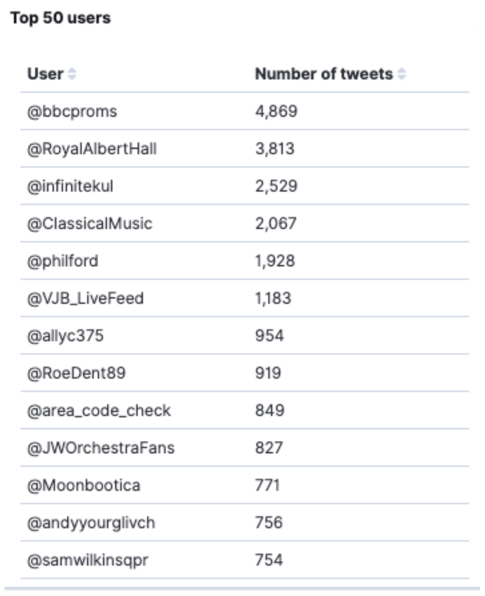


Figure 3.4: Evolution of tweet polarity during each edition



Top 50 users

User	Number of tweets
@bbcproms	4,869
@RoyalAlbertHall	3,813
@infinitekul	2,529
@ClassicalMusic	2,067
@philford	1,928
@VJB_LiveFeed	1,183
@allyc375	954
@RoeDent89	919
@area_code_check	849
@JWOrchestraFans	827
@Moonbootica	771
@andyyourglivch	756
@samwilkinsqpr	754

Figure 3.5: List of the top users. The first 10 are removed to see if the polarity distribution varies, but it remains the same

One of the best things of Kibana dashboard is that by clicking in one of the words of a word cloud it automatically changes all the visualizations and filters them to show only the results for tweets containing that specific word. This is the closer we can get to domain specific entity recognition, by clicking and filtering manually on entities of interest.

Looking at the word cloud of the top 100 proper nouns shown in figure 3.6, we can detect some classical music specific entities. *“baremboin”*, a very famous musician, *“beethoven”*, *“mozart”* and many more entities can be seen.

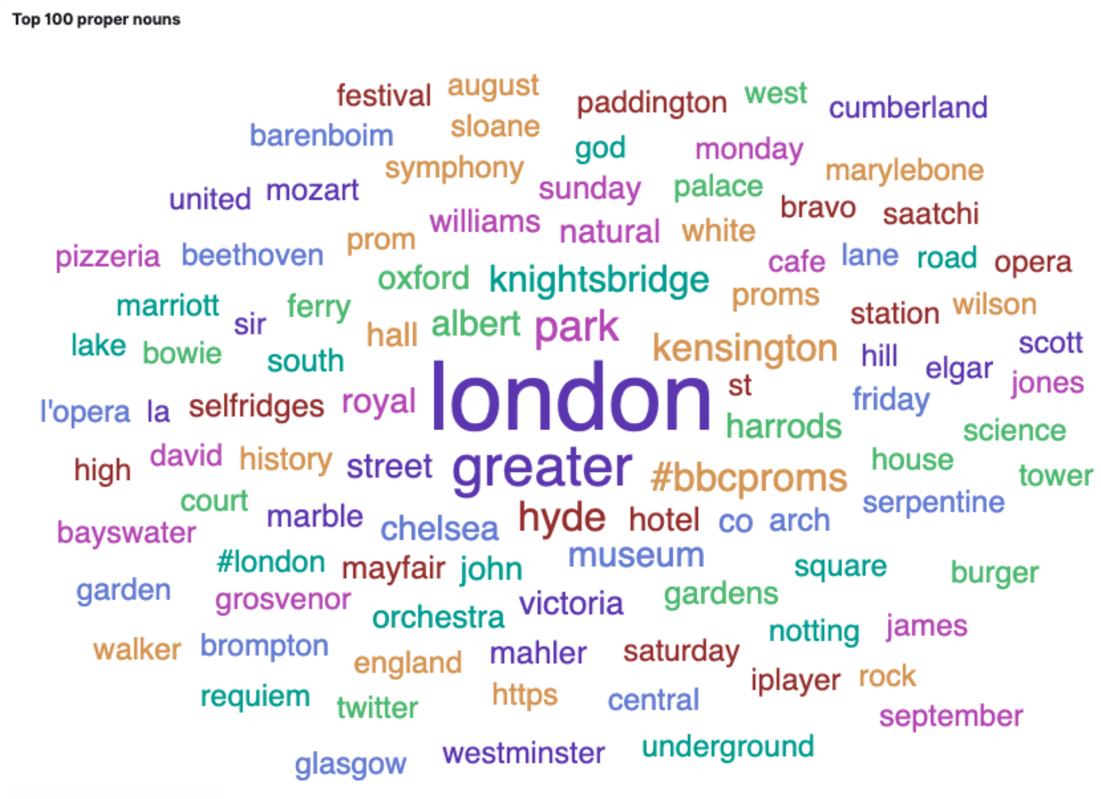
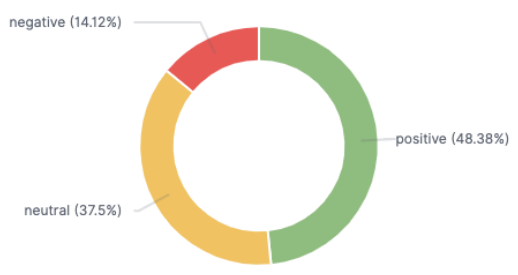
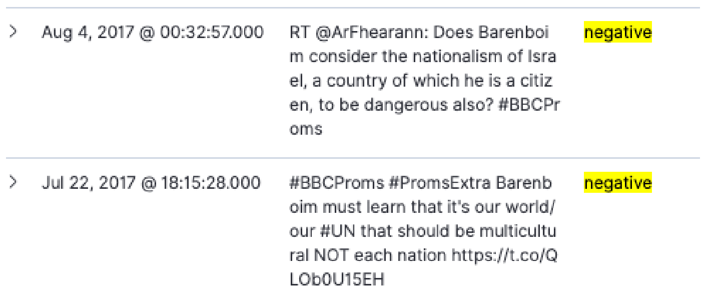


Figure 3.6: List of the top 100 proper nouns. Some classical musical entities can be detected, like "beethoven", "symphony", "mozart", "barenboim", "mahler", "orchestra"

In the word cloud above, we can click, for example, in *"Barenboim"* and see the sentiment distribution of the tweets mentioning Barenboim. Surprisingly, we find a not very good polarity distribution. This might make us think that he did not perform well in *The Proms*, so we go to look into specific tweets, and we find out that in 2017 Barenboim made some statements on his speech in the Proms that did not like to some people, but his performance was brilliant, as many tweets mention it. Details on the polarity distribution and some negative tweets mentioning Barenboim can be seen in figure 3.7.





>	Aug 4, 2017 @ 00:32:57.000	RT @ArFhearann: Does Barenboim consider the nationalism of Israel, a country of which he is a citizen, to be dangerous also? #BBCProms	negative
>	Jul 22, 2017 @ 18:15:28.000	#BBCProms #PromsExtra Barenboim must learn that it's our world/ our #UN that should be multicultural NOT each nation https://t.co/QLOb0U15EH	negative

Figure 3.7: Results of analyzing the figure of Daniel Barenboim in the BBC Proms

Another visualization that really shows the perception of Barenboim's speech and performances can be seen in figure 3.8. There we can see the evolution of the opinion on Daniel Barenboim during the three editions of the *BBC Proms*. As it can be seen, in the 2014 and 2016 editions the number of tweets were low, with a majority of positive tweets. However, in 2017 appears a big spike, which is associated to his speech and performance that year. Looking at this visualization, the previous observations become much more understandable.

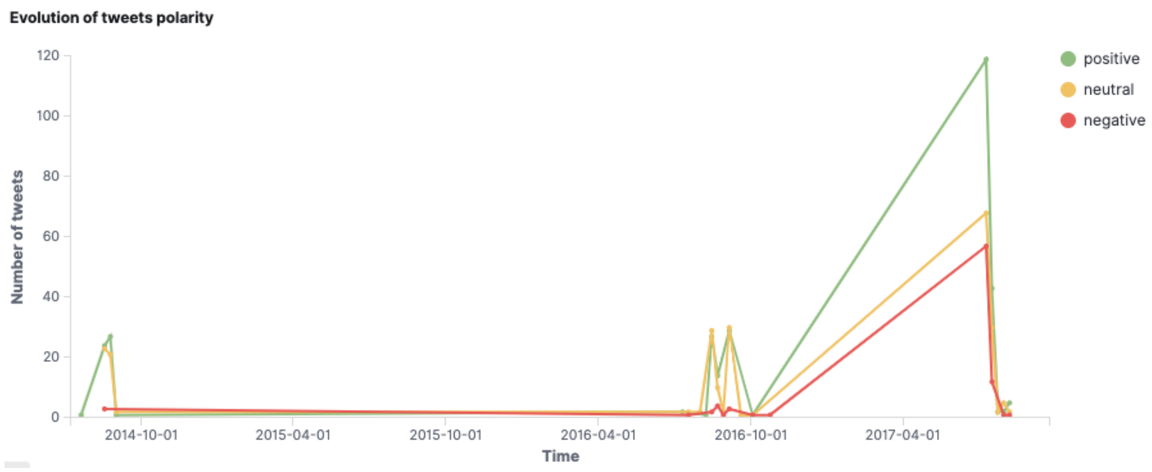


Figure 3.8: Evolution of the sentiment towards Daniel Barenboim between 2014 and 2017 editions

4. CONCLUSIONS

4.1 Conclusions

In this work it has been presented the implementation of a complete journey to perform Twitter Sentiment Analysis, from data gathering and data pre-processing to the final classification and analysis of the tweets.

The results of the classifiers show that a simple CNN model with few hyperparameter optimization outperforms machine learning algorithms, obtaining comparable results to the state-of-the-art. It has also been shown that adding lexicon features to the feature vectors in the CNN improves the performance. This reinforces the well-known ability of deep learning methods to outperform widely used machine learning algorithms in a number of tasks.

The low results obtained with the machine learning approaches can be due to a not accurate feature selection. Those methods rely heavily on the features they use, and in this work this has not been deeply explored.

Regarding the visualizations and the analysis of the BBC Proms tweets, I have built a useful visualization tool in Kibana where the data can be explored, filtered and analysed. The analysis performed on the group of tweets is superficial due to the lack of a domain specific entity recognition, which would have allowed a much deeper analysis and powerful visualizations of the classified tweets. However, thanks to the POS tagger used, I have been able to detect some tokens which have been tokenized as proper nouns, that later in the analysis have been detected as musical entities and I have been able to perform some analysis on them.

4.3 Future work

There are several ways that this work can be improved. These can be divided in two type of enhancements: implementation of the classifier and final analysis.

- **Implementation:**
    - First, more annotated data would possibly improve the model performance.
    - The proposed CNN based method has a very simple design. Hence, future work can be focused on improving the network architecture, where several things can be explored: (1) different type of embeddings can be used; for

instance FastText[40] character embeddings, and would also be interesting to see how our model performs with word embeddings trained with our dataset. More complex systems can be explored. (2) The top results presented in *SemEval 2017 task 4 – subtask A* presented complex systems where different combinations of deep learning methods with different set of features where explored. Therefore, it is obvious that a better classifier would be obtained with a more complex model, such as a deep learning ensemble method.

  - Finally, as for the machine learning methods, a richer set of features can be explored. This includes exploiting word embeddings to extract embedding features or using clusters to map each tweet to a set of clusters.

- **Analysis:**
  - For the final analysis, a domain specific named entity recognition (NER) would be needed. It will be suitable to use the method presented by Lorenzo Porcaro and Horacio Saggion in [11]. This would allow deeper analysis on specific musical entities like composers, specific pieces of music, conductors, performers, etc.
  - Finally, with the built-in Twitter classifier, one can perform Sentiment Analysis on any domain. Hence, the method proposed in this work can be exploited to apply Sentiment Analysis in other applications which involve user-generated text.

[40] https://fasttext.cc/

BIBLIOGRAPHY

[1] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up?," 2002, doi: 10.3115/1118693.1118704.

[2] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Found. Trends Inf. Retr.*, 2008, doi: 10.1561/1500000011.

[3] S. Rosenthal, N. Farra, and P. Nakov, "SemEval-2017 Task 4: Sentiment Analysis in Twitter," pp. 502–518, 2018, doi: 10.18653/v1/s17-2088.

[4] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysis—A review of research topics, venues, and top cited papers," *Computer Science Review*. 2018, doi: 10.1016/j.cosrev.2017.10.002.

[5] T. Nasukawa and J. Yi, "Sentiment analysis: Capturing favorability using natural language processing," 2003, doi: 10.1145/945645.945658.

[6] B. Whitman and S. Lawrence, "Inferring descriptions and similarity for music from community metadata," *Proc. 2002 Int. Comput. Music Conf.*, 2002.

[7] M. Sordo, J. Serrà, and X. Serra, "A Method for Extracting Semantic Information from on-line Art Music Discussion Forums," 2012.

[8] X. Zhang, Z. Liu, H. Qiu, and Y. Fu, "A hybrid approach for chinese named entity recognition in music domain," 2009, doi: 10.1109/DASC.2009.27.

[9] S. Oramas, L. Espinosa-Anke, M. Sordo, H. Saggion, and X. Serra, "ELMD: An automatically generated Entity Linking gold standard dataset in the Music Domain," 2016.

[10] S. Oramas, A. Ferraro, A. Correya, and X. Serra, "Mel: a Music Entity Linking System," 2017.

[11] L. Porcaro and H. Saggion, "Recognizing Musical Entities in User-generated Content," *Comput. y Sist.*, 2019, doi: 10.13053/CyS-23-3-3280.

[12] C. S. Yang and H. P. Shih, "A rule-based approach for effective sentiment analysis," 2012.

[13] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," 1998, doi: 10.1007/s13928716.

[14] W. Yin, K. Kann, M. Yu, and H. Schütze, "Comparative Study of CNN and RNN for Natural Language Processing," 2017, [Online]. Available: http://arxiv.org/abs/1702.01923.

[15] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, 2011.

[16] Y. Kim, "Convolutional neural networks for sentence classification," *EMNLP 2014 - 2014 Conf. Empir. Methods Nat. Lang. Process. Proc. Conf.*, pp. 1746–1751, 2014, doi: 10.3115/v1/d14-1181.

[17] Y. Zhang and B. Wallace, "A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification," 2015, [Online]. Available: http://arxiv.org/abs/1510.03820.

[18] M. Cliche, "BB_twtr at SemEval-2017 Task 4: Twitter Sentiment Analysis with CNNs and LSTMs," no. 2014, pp. 573–580, 2018, doi: 10.18653/v1/s17-2094.

[19] M. Rouvier, "LIA at SemEval-2017 Task 4: An Ensemble of Neural Networks for

Sentiment Classification," 2018, doi: 10.18653/v1/s17-2128.

[20] H. Hamdan, "Senti17 at SemEval-2017 Task 4: Ten Convolutional Neural Network Voters for Tweet Polarity Classification," vol. 6, no. 2014, pp. 700–703, 2018, doi: 10.18653/v1/s17-2116.

[21] B. Shin, T. Lee, and J. D. Choi, "Lexicon Integrated CNN Models with Attention for Sentiment Analysis," pp. 149–158, 2018, doi: 10.18653/v1/w17-5220.

[22] C. Baziotis, N. Pelekis, and C. Doulkeridis, "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis," no. January, pp. 747–754, 2018, doi: 10.18653/v1/s17-2126.

[23] O. Owoputi, B. O'Connor, C. Dyer, K. Gimpel, N. Schneider, and N. A. Smith, "Improved part-of-speech tagging for online conversational text with word clusters," 2013.

[24] R. Tsz-Wai Lo, B. He, and I. Ounis, "Automatically Building a Stopword List for an Information Retrieval System."

[25] H. Saif, M. Fernandez, Y. He, and H. Alani, "On stopwords, filtering and data sparsity for sentiment analysis of twitter," *Proc. 9th Int. Conf. Lang. Resour. Eval. Lr. 2014*, no. i, pp. 810–817, 2014.

[26] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, and Q. V. Le, "Unsupervised Data Augmentation," *arXiv*, 2019.

[27] F. Sebastiani, "An axiomatically derived measure for the evaluation of classification algorithms," 2015, doi: 10.1145/2808194.2809449.

[28] P. Nakov, A. Ritter, S. Rosenthal, F. Sebastiani, and V. Stoyanov, "SemEval-2016 task 4: Sentiment analysis in twitter," 2016, doi: 10.18653/v1/s16-1001.

[29] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations ofwords and phrases and their compositionality," 2013.

[30] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," 2008.

[31] H. Hamdan, P. Bellot, and F. Bechet, "Lsislif: Feature Extraction and Label Weighting for Sentiment Analysis in Twitter," no. SemEval, pp. 568–573, 2015, doi: 10.18653/v1/s15-2095.

[32] F. L. Cruz, J. A. Troyano, B. Pontes, and F. J. Ortega, "Building layered, multilingual sentiment lexicons at synset and lemma levels," *Expert Syst. Appl.*, 2014, doi: 10.1016/j.eswa.2014.04.005.

[33] S. Kiritchenko and S. Mohammad, "The Effect of Negators, Modals, and Degree Adverbs on Sentiment Composition," 2016, doi: 10.18653/v1/w16-0410.

[34] S. Kiritchenko and S. M. Mohammad, "Happy Accident: A sentiment composition lexicon for opposing polarity phrases," 2016.

[35] S. Kiritchenko and S. M. Mohammad, "Sentiment composition of words with opposing polarities," 2016, doi: 10.18653/v1/n16-1128.

[36] S. Kiritchenko, X. Zhu, and S. M. Mohammad, "Sentiment analysis of short informal texts," *J. Artif. Intell. Res.*, 2014, doi: 10.1613/jair.4272.

[37] M. Jabreel and A. Moreno, "SiTAKA at SemEval-2017 Task 4: Sentiment Analysis in Twitter Based on a Rich Set of Features," pp. 694–699, 2018, doi: 10.18653/v1/s17-2115.

[38] A. Pak and P. Paroubek, “Twitter as a corpus for sentiment analysis and opinion mining,” 2010, doi: 10.17148/ijarcce.2016.51274.

[39] Z. Zhou, X. Zhang, and M. Sanderson, “Sentiment analysis on twitter through topic-based lexicon expansion,” 2014, doi: 10.1007/978-3-319-08608-8_9.