Master's Thesis
Academic Year 2018

Word Level Approach for Tweets Classification based on its Content

Graduate School of Science and Technology,
Keio University

Victor Centellas Gil

Abstract of Master's Thesis of Academic Year 2018

Word Level Approach for Tweets Classification based on its Content

Category: Science / Engineering

Summary

Twitter has become the largest microblogging platform where users can interact between each other expressing opinions, thoughts and feelings related to any topic or source of news in a compressed 280 character message, called tweet. Hashtags are popular keywords used to label these tweets according to its content. This work tries to find out if the usage of hashtags to label tweets with similar content is accurate enough. To do so, tweets from different popular hashtags have been retrieved and processed in order to have a dataset with a content as close to reality as possible. Several embedding methods and learning algorithms have been studied to classify tweets from different hashtags based on the content. Results showed that the best performance is achieved when using the Tf-idf embedding method and support vectors machine. The learning algorithm obtained a precision around 90% for classification on 10 classes and above 70% when dealing with 100 classes trained on datasets of only 13680 and 143067 samples respectively. The results also indicated that BoW and Tf-idf methods outperformed other state of the art methods for other natural language processing tasks, such as GloVe or Word2Vec.

Keywords:

Text Processing, Bag of Words, Tf-idf, Twitter, Support Vector Machine, Multinomial Naives Bayes

Graduate School of Science and Technology, Keio University

Victor Centellas Gil

Acknowledgements

I would also like to thank my advisor Prof. Hiroaki Saito of the Department of Information and Computer Science at Keio University, and the members of the Hiroaki Saito Laboratory.

Without Prof. Saito acceptance as my advisor, my experience in Japan would have not been possible. Besides that, he gave me valuable insight on my research and supported all my choices, giving the freedom to decide what directions to take.

Members of the Hiroaki Saito Lab were very kind and always tried to help me with my work.

Contents

List of Figures

List of Tables

Glossary

API Application Programming Interface. iii, v, 7–10

BoW Bag of Words. 18, 20, 29, 37, 41, 43–48, 50

CBOW Continuous Bag of Words. 18

GloVe Global Vector. 20, 21, 37, 41, 43–46, 49, 50

Log Reg Logistic Regression. 37, 41, 43

LSA Latent Semantic Analysis. 5

LSTM Long Short Term Memory. 5

MAP Maximum A Posteriori. 23

MNB Multinomial Naive Bayes. 37, 41, 43

NB Naives Bayes. 23

NLP Natural Language Processing. 22

NNLM Neural Network Language Modeling. 5, 18

OAuth Open Authentication. 8

ORM Object Relational Maping. 15

SRM Structural Risk Minimization. 24

SVD Single Value Decomposition. 20

SVM Support Vector Machine. v, vii, 23–25, 28, 31, 32, 37, 41–45, 49, 51

Tf-idf Term frequency - inverse document frequency. v, vii, 18, 37, 41–48, 50, 51, 53

WOEID Where On Earth ID. vii, 9, 10

Word2Vec Word to Vector. 37, 41, 43–45, 49, 50, 52

Chapter 1
Introduction

The amount of information available on the internet has exponentially increased over the last few years and has become the universal source of information for millions of users around the globe. With a penetration rate of 54.4% of the world population by the end of 2017 [1], the world wide web has changed forever the way we communicate.

With the digitalization of the traditional news sources, such as newspapers and television, the content related to breaking news is available online a few minutes after it occurs and can be accessed from anywhere. This fact has facilitated the appearance of microblogging sites. Twitter has become the largest microblogging platform where users can interact between each other expressing opinions, thoughts and feelings related to any topic or source of news in a compressed 280 character message, called tweet. When an important event occurs somewhere on the globe, articles from the media and thousands and thousands of opinions from users appear on Twitter in a matter of minutes.

This huge amount of tweets available have some drawbacks. The main one is the possibility for the information to get lost in the servers given the enormous amount of data. To give access to the content by the interested users the hashtags were invented. Hashtags are popular keywords used to label tweets according to its content. Using this label method, tweets related to the same topic can be searched and tracked.

1 https://www.internetworldstats.com

1.1 Motivation

Hashtags are written in the tweet by the user and they have total freedom when it comes to selecting the desired one. From a machine learning perspective, the users label the content created by themselves. The correspondence between the hashtag and the content is not verified in any way and this opens the possibility to use it in an inadequate way.

Are the users good enough when selecting the hashtag? Is the hashtag a representative way to cluster the content of the tweets? It is possible to check it in an automatic way instead of manually verifying the relation between the tweet content and the hashtag?

These are some questions I asked myself when interacting with the platform and are going to be answered in this work.

1.2 Goals of the thesis

This work will try to answer the questions suggested in Section 1.1. This is equivalent to solving a multi-class classification problem. The algorithm is going to be trained using a dataset of tweets retrieved from Twitter. The tweet content is going to be used as the input for the model while the hashtag will be used to label the sample.

Transforming the tweet content into a numerical fixed length vector is a challenging task and for that reason a comparative study of several embedding methods is going to be conducted. This embedding methods are going to be implemented using different learning algorithms to identify the best combination of model and embedding method where the best performance is achieved. If the model is able to correctly classify a large portion of samples, that will mean that there are similarities between tweets of the same hashtag and therefore the hashtag was appropriate for labeling the tweet content.

Chapter 2
Related Work

In recent years, the research in the feature extraction field for text classification has been focused on two main approaches. While a part of the research has focused on the representation of words using vectors, the others have been investigating a character level approach for text representation.

Rumelhart et al. introduced in 1986 one of the earliest uses of word representations [15] and has been applied to statistical language modeling since then. The distributed representation of words in a vector space have been proven to achieve better performance by grouping similar words. These vectors can be used as the input in a wide range of applications, such as document classification [18], question answering [20] and information retrieval [10]. Neural networks to learn distributed representation of words dates back to 2003, where a feedforward neural network with a linear projection layer and a non-linear hidden layer was used [2].

In 2013, Mikolov et al. introduced an efficient method for learning high-quality vector representations of words from unstructured text data using the Skip-gram model. This method suffers from the disadvantage that it does not operate directly on the co-occurrence statistics of the corpus. For that reason, a new global log-bilinear regression model called GloVe [13] was emerged by combining local context window methods, such as the skip-gram model [11] and global matrix factorization introduced in LSA [5].

Traditional NNLM methods treat words as the basic units of language and assign an independent vector to each word type, requiring the storage of an extremely large table of vectors. It also demands for a strong preprocessing when dealing with social network domains like Twitter. In 2015, Ling et al. presented a bidirectional LSTM for composing word vectors from their constituent characters [9]. With this approach the large word lookup tables can be compacted into character lookup tables, scaling to large datasets and obtaining better results than other state of the art approaches. However, Dhingra et al. took one step further in 2016 and generated vector representations of the entire tweet from characters by learning complex, non-local dependencies in character sequences [6].

Chapter 3
Proposed Approach

In this chapter the proposed method is detailed. The first step is to create a dataset with a large amount of samples . This can be done by retrieving tweets through the Twitter API, filter and process them to finally store in a database for further usage. In order to drop this data into a model some numerical characteristics have to be extracted from the tweets. In this work several embedding vector methods have been used and are detailed in Section 3.2. In the last section of this chapter, the theoretical formulation of the different models used for the multi-class classification problem are explained.

3.1 Data acquisition

In order to train a classifier a lot of data needs to be collected. For this project a huge amount of tweets with its corresponding hashtag are needed. Twitter has developed some APIs to retrieve any kind of useful material. Information about the trending topics in a specific region can be obtained. This trending topics can be used to search for tweets with this specific content.

There is another Twitter API to retrieve real time tweets. The user can be subscribed to a channel and twitter streams all the tweets that match with a certain specifications such as keywords, language or location in near real time. If the list of matching words to be streamed is too general or involved in some trending topics the amount of tweets retrieved is huge and this can be a problem. If the user is not able to process this amount of information because some time consuming action is required such as post-filtering or pre-processing of the data, Twitter automatically disconnects the user to the streaming to prevent buffer saturation. When this occurs the user needs to wait to reconnect again due to connection limits exceeded.

This problem can be solved considering the following approach. All the unprocessed data retrieved from Twitter is stored in a queue message generator. This information is accessed through queries and the message obtained is then processed and if fulfills a set of requisites is then stored in a database. All the mentioned functionalities have been implemented using Python3 and a set of libraries explained in the following subsections. Figure 3.1 illustrates the workflow diagram used in this approach to solve the buffer saturation problem.

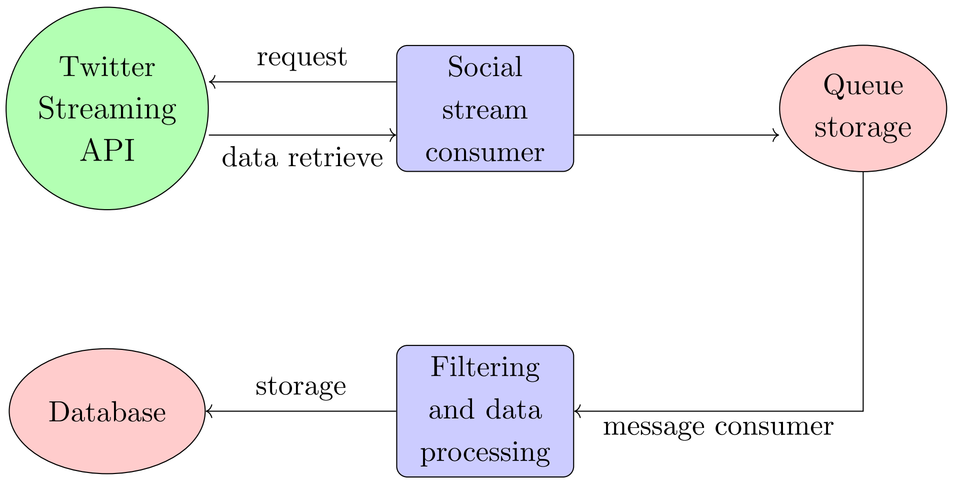


Figure 3.1: Structure for the data retrieve and storage.

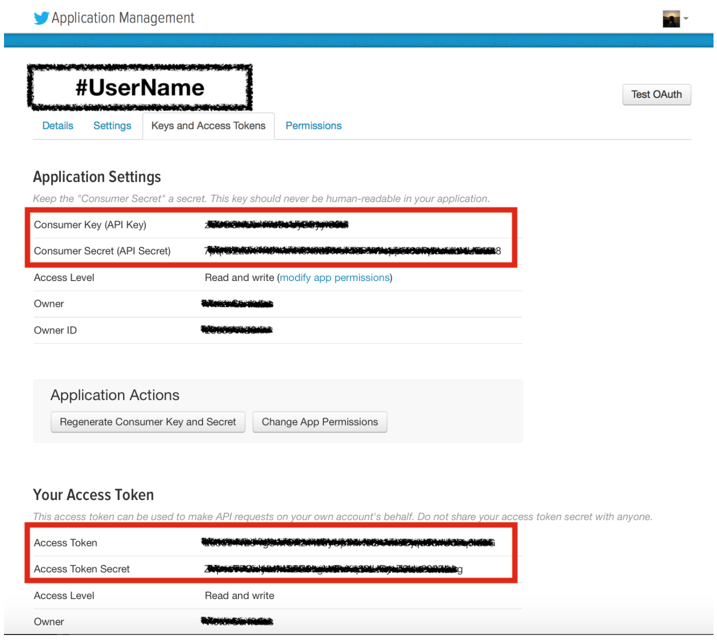
3.1.1 Retrieving tweets from Twitter Streaming API

Connecting to the Twitter stream of tweets in near real time is possible with the Twitter Streaming API[1]. This can be done using Tweepy [2], a Python library used to handle the API connection in an easy way.
The first step is to set up the authentication, in that case Twitter API uses the OAuth authentication protocol. For comunicating with the Twitter API four keys are required: Consumer Key, Consumer Secret, Access Token and Access Token Secret.

1 Twitter Streaming API Documentation: https://developer.twitter.com/en/docs

2 Tweepy Library Documentation: http://tweepy.readthedocs.io/en/v3.5.0/

In order to obtain the mentioned keys its necessary to create a new application using an existing user's account in the Twitter Applications Portal[3]. After filling the required information the four keys can be obtained in the *Keys and Access Tokens* tab as illustrated in Figure 3.2.



Application Management

#UserName

Test OAuth

Details | Settings | Keys and Access Tokens | Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	[redacted]
Consumer Secret (API Secret)	[redacted]
Access Level	Read and write (modify app permissions)
Owner	[redacted]
Owner ID	[redacted]

Application Actions

Regenerate Consumer Key and Secret | Change App Permissions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	[redacted]
Access Token Secret	[redacted]
Access Level	Read and write
Owner	[redacted]

Figure 3.2: Consumer and access token needed for the communication with the Twitter API.

Once the API connection have been set up using the *Tweepy* library it is possible to obtain the trends in a certain location using the WOEID identifier. The WOEID is a unique 32-bit reference identifier that identifies any feature on Earth [4]. In Table 3.1 the identifiers for some regions are stated.

3 https://apps.twitter.com

4 WOEID: https://en.wikipedia.org/wiki/WOEID

Country	WOEID
Global	1
Japan	23424856
Spain	23424950
United States	23424977
United Kingdom	23424975
China	23424781

Table 3.1: WOEID identifiers for some countries.

This work is focused on the english language processing and for that reason only global, United States and United Kingdom trending topics have been considered. In Twitter, the trending topics can be composed by either hashtags or trends. While hashtags are popular keywords with a # at the beginning that categorizes the accompanying text, trends are words contained in the tweets that are immediately popular at a particular time. To make this work more challenging, only trending topics containing hashtags have been taken into account.

This set of hashtags has been used to track english tweets containing those words in near real time. When a tweet containing any of those mentioned words is published by a user, all the information regarding this tweet is retrieved by Twitter Streaming API through a stream listener and published to the message queue generator.

3.1.2 RabbitMQ, a message queue generator

The main disadvantage of using the Twitter Streaming API to retrieve tweets containing popular words is the huge amount of data that needs to be processed. In this case, a filtering has to be performed before storing the information. This time consuming process can lead to not be able to process such amount of information retrieved and be disconnected of the streaming channel by Twitter due to buffer saturation.

The aforementioned situation can be avoided using RabbitMQ [5], a message-queuing software where queues can be defined and applications can be connected

5 RabbitMQ Message Broker: https://www.rabbitmq.com

to transfer messages onto it or to take off the queue a message and start processing it as illustrated in Figure 3.3. This solution also introduces the ability to have multiple consumers connected to process the messages, meeting high-scale and high-availability requirements.

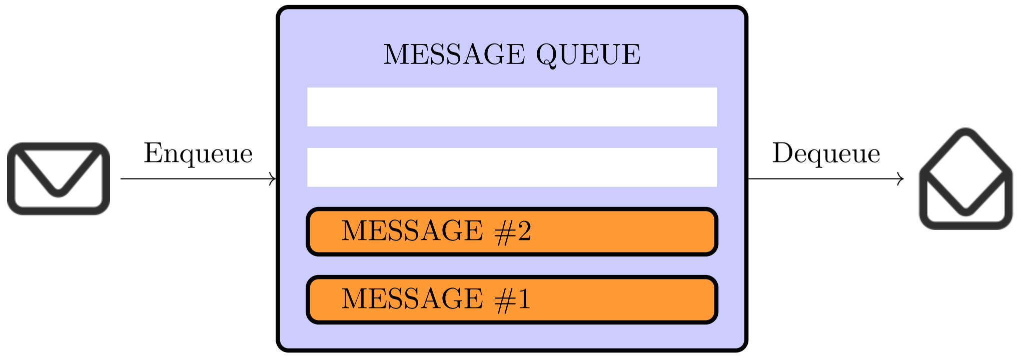


Figure 3.3: RabbitMQ structure for queueing messages.

There are two different possibilities to use the RabbitMQ software, use an existing server in the cloud or create your own server locally. In order to use the second option it is necessary to download the required software and set up the server using the terminal command line. *Pika*[6] is a python library created to manage the connections and queries between the queue generator software and the sender and consumer processes. The necessary steps to set up the server and handle the connections are:

- Server: Initialize the server
- Sender process:
    1. Create a connection to the server, in this case to *'localhost'*.
    2. Open a channel.
    3. Declare the name of the queue to the channel.
    4. Start sending messages.
    5. Close the connection at the end of the process
- Consumer process:

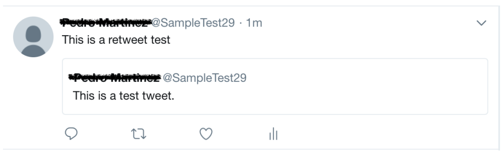
6 https://pika.readthedocs.io

1. Create a connection to the server, in this case to *'localhost'*.
2. Open a channel.
3. Declare the name of the queue to the channel.
4. Start consuming the messages and performing the filtering.
5. Stop the consumption.
6. Close the connection at the end of the process.

3.1.3 Filtering the retrieved tweets

The amount of tweets retrieved by twitter is huge but this does not mean all the information is useful. When collecting data from a giant source it is required to pre-process the information and this is not an exception. A lot of tweets have been discarded due to the strong filtering that has been carried out. This work has strongly focused in obtaining a dataset with meaningful and full of content tweets but keeping in mind that in Twitter misspellings, plain language and abbreviations are commonly used.

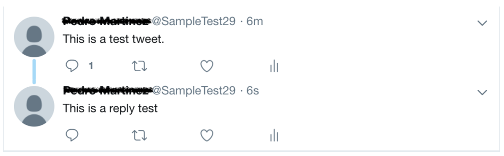
In Twitter is very common to repost or forward another user's tweet or to post a new one referring to a previous tweet as can be seen in Figure 3.4 . These examples have not been taken into account because this work is focussed in only new content creation tweets.



[redacted] @SampleTest29 · 1m
This is a retweet test

[redacted] @SampleTest29
This is a test tweet.

(a) Retweet



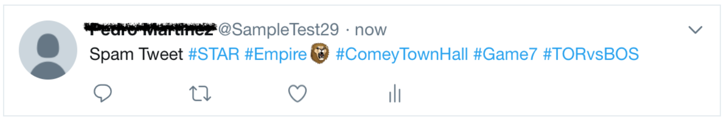
[redacted] @SampleTest29 · 6m
This is a test tweet.
1

[redacted] @SampleTest29 · 6s
This is a reply test

(b) Reply

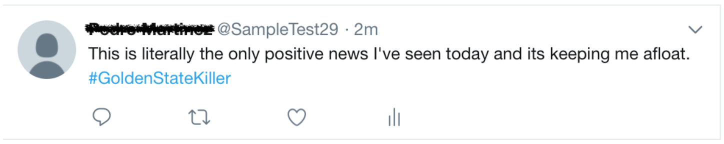
Figure 3.4: Examples of tweets not taken into account.

On the other hand, there are some tweets composed only by hashtags or containing the most populars hashtags just to appear in the flow of tweets related to that trend. For that reason, only tweets with less than 4 hashtags, 2 user mentions and more than 5 words without considering hashtags and user mentions have been taken into account. Figure 3.5 shows the difference between a useful tweet and a spam one. With this first filtering around 80% of the retrieved tweets have been discarded. Next step is to process the selected tweets and store them in a data base.


[redacted] @SampleTest29 · now

Spam Tweet #STAR #Empire #ComeyTownHall #Game7 #TORvsBOS

(a) Spam tweet


[redacted] @SampleTest29 · 2m

This is literally the only positive news I've seen today and its keeping me afloat. #GoldenStateKiller

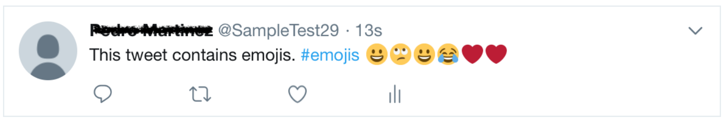
(b) Useful tweet

Figure 3.5: Difference between an useful and a spam tweet.

3.1.4 Processing the selected tweets

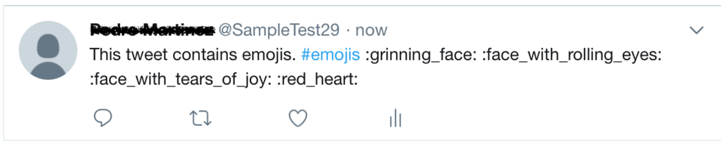
Once the tweets to be used in this work have been selected, a processing of the data is required. First of all it is necessary to delete line breaks and multiple white spaces between words. It is also necessary to add a white space after a full stop, question and exclamation marks. Another type of characters used by people when publishing tweets are Emojis. These small icons can represent a feeling, an emotional state or even to describe some wish in a compressed way. The emojis have been replaced by words describing the emoji as illustrated in Figure 3.6 using the python library *emoji*[7]. Duplicated emojis in the same tweet have been ignored.

7 https://pypi.org/project/emoji/



[redacted] @SampleTest29 · 13s
This tweet contains emojis. #emojis 😀🙄😀😂❤️❤️

(a) Tweet containing emojis

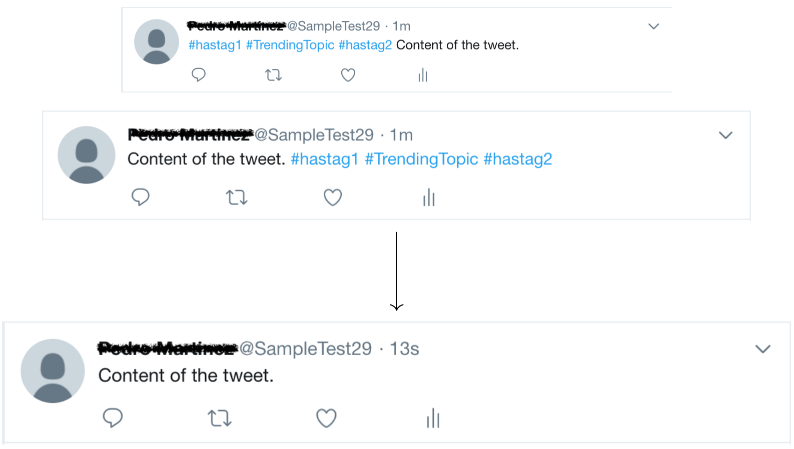


[redacted] @SampleTest29 · now
This tweet contains emojis. #emojis :grinning_face: :face_with_rolling_eyes: :face_with_tears_of_joy: :red_heart:

(b) Same tweet after the preprocessing

Figure 3.6: Difference between before and after the emojis processing.

Another required step is to extract the popular hashtag from the text to use as label for that tweet. Furthermore, the tweet can contain other hashtags that need to be deleted. There are several ways to write hashtags in a tweet. Some people write all the hashtags at the beginning of the tweet, followed by the text while others write the text first and the hashtags at the end. Figure 3.7 provides examples of the aforementioned situations.



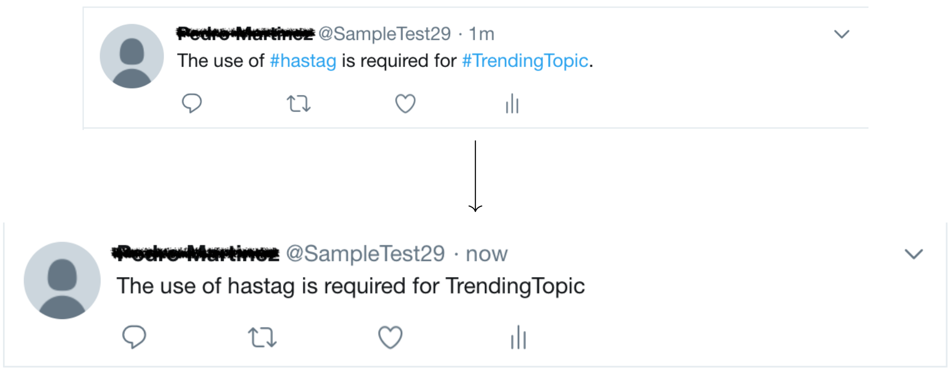
[redacted] @SampleTest29 · 1m
#hastag1 #TrendingTopic #hastag2 Content of the tweet.

[redacted] @SampleTest29 · 1m
Content of the tweet. #hastag1 #TrendingTopic #hastag2

[redacted] @SampleTest29 · 13s
Content of the tweet.

Figure 3.7: Example of tweet processing with hashtag at the beginning and at the end of the tweet.

However, there are some users that use the hashtag itself to name a place, a person or and action in the text. Deleting the hashtag in this case can lead to obtain a text with no sense. For that reason the approach conducted to solve this situation is to delete only the '#' character as can be seen in Figure 3.8 . Finally, combinations of all of the three possibilities have been also considered and processed accordingly.


[redacted] @SampleTest29 · 1m
The use of #hastag is required for #TrendingTopic.

[redacted] @SampleTest29 · now
The use of hastag is required for TrendingTopic

Figure 3.8: Example of a tweet with hashtags inside the content.

3.1.5 Storing tweets into a Data base

Last step is to store the processed tweets in an easy way to handle the access to the information again. One of the possible solutions is to lock away all this data in a '.csv' or '.txt' files. The main drawback of this choice is the lack of data organization. A more reasonable solution is to create a database and store the data using tables. The information is stored in an organized way and the access to the data can be done using *Peewee* [8], a small and simple ORM with built-in support for SQLite, MySQL and Postgresql.

To create a new database is only necessary to specify the desired name, define the tables needed and use the *create_table* command to generate them in the database. The following information related to a tweet have been stored in the database:

8 http://docs.peewee-orm.com/en/latest/

- **Tweet id:** Unique identifier for each tweet
- **Text:** The obtained text after processing it
- **Original Text:** Original text retrieved form Twitter
- **Created Time:** Time the tweet was published
- **User Name:** Name of the user who created the tweet
- **User Name Id:** Unique identifier used for twitter for each user
- **User Location:** Geographical location of the user
- **Label:** Hashtag to identify the content of the tweet

3.2 Feature extraction

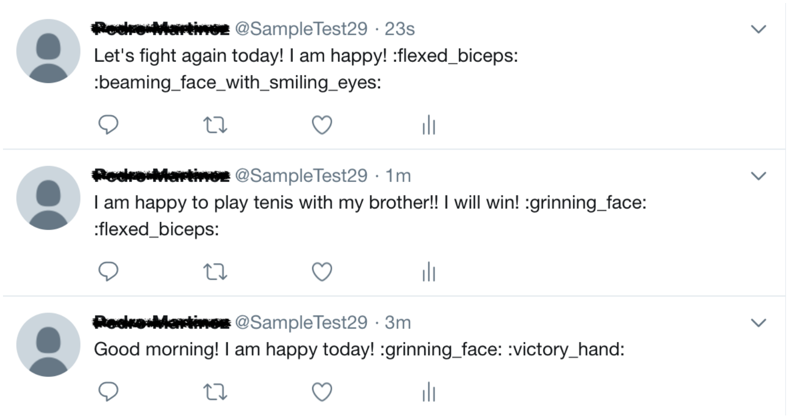
It is not possible for a classifier or learning algorithm to process tweets in its original form, only numerical inputs can be used instead. It is necessary to transform the distinctive characteristics of this original tweet into numerical vectors of fixed length. In this work only word level approaches have been taken into account. Despite character level approaches outperform state of the art results in hashtag prediction with little text preprocessing they require a huge dataset to be trained. On the other hand, word level approaches required more text processing before extracting the features but can be trained with smaller datasets and achieve good results in the multi-class classification problem.

3.2.1 Bag of Words

Bag of Words [16] is the most common way for representing texts as a fixed length vector. In this approach the words are tokenized for each tweet and then the frequency of each word in the dataset is taken into account. Before the counting takes place it is necessary to have a dictionary containing all words that appeared in the dataset. Despite the most common english words such as prepositions and pronouns are omitted, the obtained featured vector has as much columns as non-discarded words in the dataset.

When the words or tokens are considered individually, the obtained model representation is called "unigrams". However it is possible to calculate the frequency in which n consecutive words appear in the dataset. This model representation is called n-gram.

An example is provided to illustrate the concept in a easier way. Figure 3.9 shows all the tweets included in this dataset.


[redacted] @SampleTest29 · 23s
Let's fight again today! I am happy! :flexed_biceps:
:beaming_face_with_smiling_eyes:

[redacted] @SampleTest29 · 1m
I am happy to play tenis with my brother!! I will win! :grinning_face:
:flexed_biceps:

[redacted] @SampleTest29 · 3m
Good morning! I am happy today! :grinning_face: :victory_hand:

Figure 3.9: Tweets used as dataset for the example.

The list of all words from the tweets illustrated in Figure 3.9 after removing the stop words is

$$\text{“}fight\text{”}, \text{“}again\text{”}, \text{“}today\text{”}, \text{“}happy\text{”}, \text{“}flexed\_biceps\text{”}, \text{“}beaming\_face\_with\_smilling\_eyes\text{”},$$
$$\text{“}play\text{”}, \text{“}tenis\text{”}, \text{“}brother\text{”}, \text{“}win\text{”}, \text{“}grinning\_face\text{”}, \text{“}good\text{”}, \text{“}morning\text{”}, \text{“}victory\_hand\text{”}$$

And the corresponding feature vectors for the tweets are

$$[1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$
$$[0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0]$$
$$[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]$$

Despite its simplicity, this approach can only be used when dealing with small datasets because they suffer for its high dimensionality. Furthermore, the obtained representations don't take into account the distances between words neither the semantic of the words represented.

3.2.2 Term Frequency - Inverse Document Frequency

In the BoW approach all words included in the vocabulary are considered equally important leading that certain terms have no discriminating power in determining relevance. The Tf-idfintroduces a mechanism for attenuating the effect of terms that occur too often in the collection to be meaningful for relevance determination [17], composed by two terms.
The term frequency component measures the local importance of a word. This is achieved by looking at how frequently a word appears in a tweet in the same way as in the BoW approach but instead of counting the number of occurrences, computing the frequency.

$$\mathrm{tf}(W) = \frac{\#\ \text{times W in a tweet}}{\#\ \text{words in a tweet}} \tag{3.1}$$

On the other hand, the inverse document frequency is related to the number of times a word appears in different documents. For a word to be considered a keyword of a tweet it shouldn't appear that often in the other tweets. If that frequency has to be low, the inverse has to be high and can be computed as

$$\mathrm{idf}(W) = \log \frac{\#\ \text{tweets}}{\#\ \text{tweets containing word W}} \tag{3.2}$$

where W is a word in the corpus.

3.2.3 Word2Vec

Mikolov et al. introduced in 2013 an efficient method for learning high quality vector of words from large amounts of unstructured text data [11]. Two new model architectures were presented. While Skip-gram architecture predicts surrounding words given a word, the CBOW can predict a word using its context. Both model architectures are illustrated in Figure 3.10.

Since this work is focused in tweets classification based on its content, the BoW approach has been considered. This architecture is similar to the feedforward NNLM proposed in [2]. The main difference is that while in the NNLM the neural network is composed by input, projection, hidden and output layers, in the CBOW architecture the hidden layer has been removed reducing the model

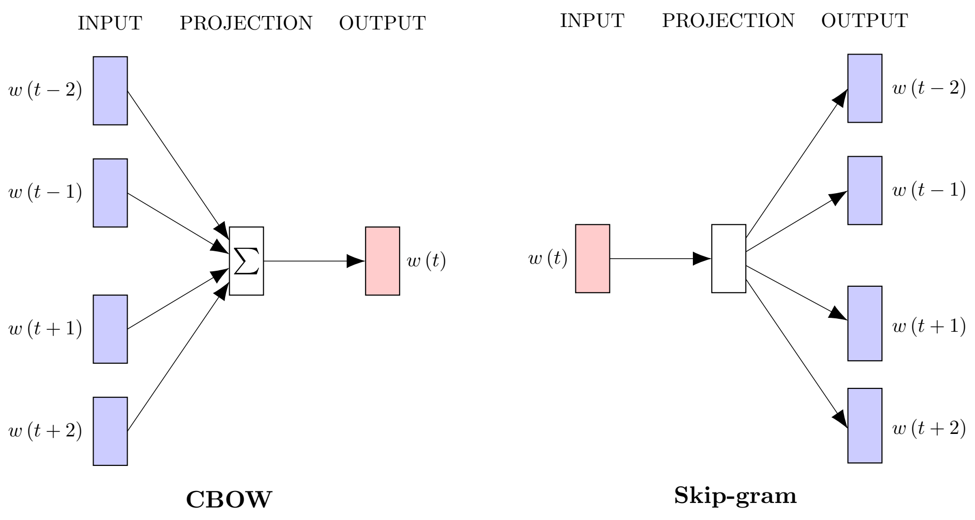


Figure 3.10: Model architectures for the Word2Vec implementation.

complexity. Despite the words representations will not be as accurate as with the neural network approach, this opens the possibility to be trained with more data in an efficient way.

Another difference is that the projection layer is shared for all words, meaning that the vectors of the projected words are averaged. This new architecture for learning word vectors does not involve dense matrix multiplications, making training extremely efficient. Once the vector representations for the words contained in a tweet are computed, the tweet embedding is obtained by averaging the vector representations of its words, as can be shown in Figure 3.11, where w_n^i represents the embedding vector for word n in tweet i.

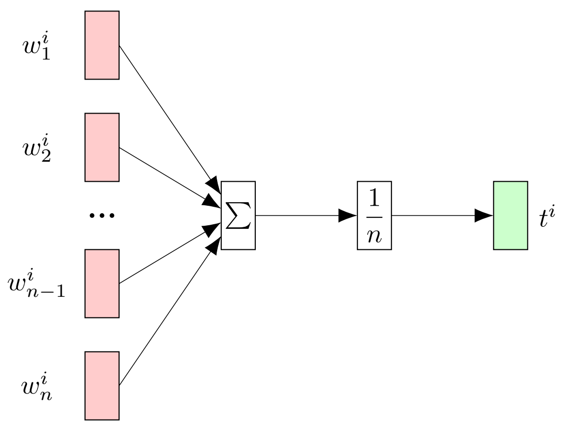


Figure 3.11: Tweet embedding using the Word2Vec architecture.

3.2.4 Global Vectors

The BoW approach introduced in Section 3.2.1 does not take into account the semantics neither the syntactics of the sentences. The neighboring words can be very useful in order to analyze the context a word is used. Setting a window size of n means that a word will be defined by its n neighboring words to the left and to the right. Using this technique to define all the words in the corpus the co-occurrence matrix can be obtained.

The results obtained with this method are very powerful. However, since all the words from the corpus are taken into account, the obtained vectors are placed in a high-dimensional space. A dimensionality reduction can be performed, using the SVD technique as stated in [7]. The main idea is to keep as much as possible the same information as before but with a lower number of dimensions. The computational cost of this approach scales quadratically since the $n \times m$ matrix, leading to a computational complexity runtime issue. Pennington et al. introduced Global Vector in which the global corpus statistics are captured directly by the model [13], solving this complexity runtime issue. Instead of using the entire corpus to scan context windows, GloVe predicts the surrounding words of every word by maximizing the probability of a context word occurring given a center word.

GloVe uses as starting point for word vector learning the ratios of co-occurrence probabilities rather than the probabilities themselves. Let the matrix of co-occurrence probabilities be denoted by X. X_{ij} is the number of times word j occurs in the context of word i and $X_i = \sum_k X_{ik}$ the number of times any word appears in the context of word i. Finally, $P_{ij} = P\left(i|j\right) = X_{ij}/X_i$ is the probability that word j appear in the context of word i. With all the notation mentioned above and keeping in mind that the ratio P_{ik}/P_{jk} depends on the words i, j and k, the most general model takes the form

$$F\left(w_i, w_j, \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \tag{3.3}$$

where $w \in \mathbb{R}^d$ are word vectors and $\tilde{w} \in \mathbb{R}^d$ are separate context word vectors.

After some transformations and manipulations, equation 3.3 can be expressed in terms of a least squares regression model, giving the following cost function

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2 \tag{3.4}$$

where V is the vocabulary size, b_i is a bias term that encompasses $\log(X_i)$ in order to achieve exchange symmetry, $\tilde{b}_k$ is another bias for $\tilde{w}_k$ and $f()$ is a weighting function that can be parametrized as

$$f\left(x\right) = \begin{cases} \left(x/x_{max}\right)^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{3.5}$$

with $\alpha = 3/4$ and $x_{max} = 100$.

Equation (3.5) can be seen as ill-defined since $\log(X_{ij})$ diverges when $X_{ij} = 0$. One possible solution is to include an additive shift in the logarithm $\log(1 + X_{ij})$ but this model weighs all co-occurrences equally, even those that rarely happen. For that reason the weighting function $f\left(\right)$ was selected assuring $x \to 0$ would vanish fast enough that the $\lim_{x\to 0} f\left(x\right)\log^2 x$ is finite.

3.3 Learning Algorithm

In this section the three different learning algorithms used in this work are explained. Despite there are a wide range of different methods for classification tasks, when it comes to the NLP domain some special characteristics are required. The algorithms used need to deal well with high dimensional input vector since a lot of features are extracted from the data in word-level approaches. The models presented are fast, have simple designs and are accurate in a number of applications of NLP while well suited for high dimensional input vectors.

3.3.1 Multinomial Naive Bayes

Naive Bayes methods are a set of supervised learning algorithm commonly used in text classification problems due to its computationally efficiency and easy implementation. The idea is to apply Bayes theorem with the *naive* assumption of independence between every pair of features as in [19].
Let C denote the class of a tweet $\boldsymbol{T}$. In order to predict the class of the tweet $\boldsymbol{T}$ using the Bayes rule, the following probability should be found

$$P\left(C|\mathbf{T}\right) = \frac{P\left(C\right)P\left(\mathbf{T}|\,C\right)}{P\left(\mathbf{T}\right)} \tag{3.6}$$

Using the naive independence assumption, equation 3.6 is simplified to

$$P\left(C|\mathbf{T}\right) = \frac{P\left(C\right)\prod_{i=1}^{n}P\left(x_i|\,C\right)}{P\left(\mathbf{T}\right)} \tag{3.7}$$

where x_i are the different terms of the feature vector for tweet $\boldsymbol{T}$.
Since $P\left(\mathbf{T}\right)$ is constant given the input, the following classification rule can be used

$$P\left(C|\mathbf{T}\right) \propto P\left(C\right)\prod_{i=1}^{n}P\left(x_i|C\right) \tag{3.8}$$

$$\Downarrow$$

$$\hat{C} = \arg\max_{C}\prod_{i=1}^{n}P\left(x_i|C\right) \tag{3.9}$$

and the MAP estimation can be used to estimate $P(C)$ and $P(x_i|C)$. While the former is the relative frequency of class C in the training set, the latter depends on the naive classifier used.

Two models are mainly used, the multivariate Bernoulli classifier and the multinominal one. The difference between them is that while the former cares about counts for a single feature that do occur and for the same feature that do not occur, the latter cares about counts for multiple features that do occur. Since the focus of this work is in multi-class classifications problem, the Multinomial NB classifier has been implemented. In the Multinomial NB classifier, the distribution is parametrized by vectors $\theta_C = (\theta_{C1}, ..., \theta_{Cn})$ for each class C, where n is the vocabulary size and θ_{Ci} is the probability $P(x_i|C)$ of feature i appearing in a sample belonging to class C.
The parameters θ_{Ci} are estimated by a smoother version of maximum likelihood like the relative frequency counting

$$\hat{\theta}_{Ci} = \frac{N_{Ci} + \alpha}{N_C + \alpha n} \tag{3.10}$$

where $N_{Ci} = \sum_{x \in S} x_i$ is the number of times feature i appears in a sample of class C in the training set S and $N_C = \sum_{i=1}^{|S|} N_{Ci}$ is the total count of all features for class C.

3.3.2 Support Vector Machines

Text data have some particular properties when referring to automatic classification purposes. Text classifiers have to deal with high dimensional input space due to the fact that a lot of features are usually extracted from the data. In the classification of other data types it can be assumed that most of this features are irrelevant but when dealing with text data only a few of them are. Finally, most of the text categorization problems are linearly separable. Support Vector Machines classifiers, introduced by Vapnik et al. [4], acknowledge the mentioned particularities and are a well suited solution for the text classification problem.

The idea behind SVM is to find the best separation between hyperplanes defined by different classes of data. If the data is no linearly separable, the input vectors are non-linearly mapped into a very high dimensional space where a linear decision surface can be constructed. This surface is obtained based on the SRM principal from computational learning theory [21] trying to find a hypothesis h for which the lowest true error can be guaranteed.

Let x_i be a training example and the target values can be $y_i \in \{-1, 1\}$. SVM searches for the separating hyperplane which separates positive and negative examples from each other with maximal margin [14] as can be seen in Figure 3.12.

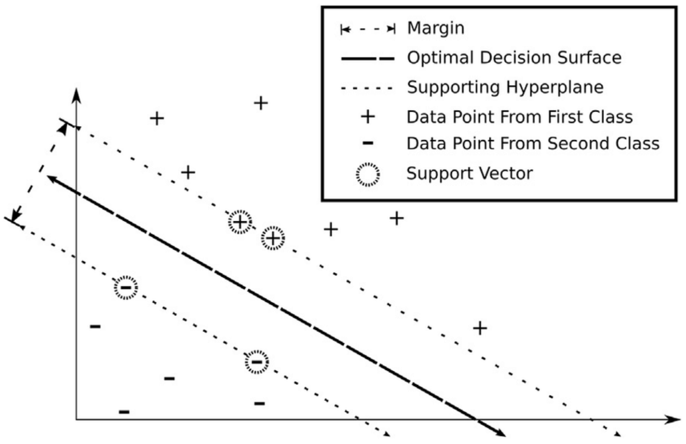


Figure 3.12: Concept of SVM and the terms of decision surface, hyperplane and margin [1].

The equation of a hyperplane is

$$\mathbf{w}^T\mathbf{x} + b = 0 \tag{3.11}$$

then, the classification of a new test sample x_i is based on the hyperplane sign. This can be formalized as

$$w^T x_i + b \geq 1 \text{ iff } y_i = +1 \tag{3.12}$$

$$w^T x_i + b \leq 1 \text{ iff } y_i = -1 \tag{3.13}$$

The optimization problem of SVM to handle non-separable cases is

$$\frac{1}{2} \cdot w_T w + C \cdot \sum_{i=1}^{n} \xi_i \tag{3.14}$$

subject to

$$\forall_{i=1}^{n} : y_i \left[w^T x_i + b \right] \geq 1 - \xi_i \tag{3.15}$$

where C is a constant to trade off between margin and training error.

SVM are inherently two-class classifiers. However is it possible to use SVM for multi-class classification problems using one of the following two approaches. In the *One-vs.-rest* strategy it is required to train a single classifier per class. Each classifier will decide if the sample belongs to one class or not and the sample will be classified as the class with the highest decision score. The main drawbacks of using this technique are that the binary classification learners see unbalanced distributions because the negative samples are larger than the positives ones, even if the class distribution is balanced. Second, the scale of the confidence score may differ between the binary classifiers.

On the other hand, in the *One-vs.-One* approach $K\left(K-2\right)/2$ binary classifiers need to be trained, where K is the number of classes. Each receives the samples of a pair of classes from the original training set and must learn to distinguish between these two classes. Then a vote is applied and the class that got the highest number of positive predictions gets predicted by the combined classifier. The main disadvantage appears when a sample receives the same number of votes [3].

3.3.3 Logistic Regression

The goal of an analysis using the Logistic Regression method is to find the best fitting model to describe the relationship between an outcome variable and a set of independent variables [8]. The difference between this method and the others described in Section 3.3.1 and Section 3.3.2 is the procedure used for training the optimal coefficients and the way the score is implemented.
The logistic regression model can be defined as

$$\pi\left(x_i\right) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}} = E\left(Y|X = x_i\right) \tag{3.16}$$

where β is a vector composed by the different weights, x_i is the input vector for the sample i with $x_{i0} = 1$ and $E\left(Y|X = x_i\right)$ is the expected value of Y given x_i.

To express the probability of the observed data as a function of the weights β, the likelihood of Equation 3.16 for a sample i can be stated as

$$l\left(\beta_i\right) = E\left(Y|x_i\right)^{y_i}\left[1 - E\left(Y|x_i\right)\right]^{1-y_i} \tag{3.17}$$

Since the expected values of Y are assumed to be independent, the likelihood function can be expressed in terms of logarithm as

$$L\left(\beta\right) = \ln\left(l\left(\beta\right)\right) = \sum_{i=1}^{n}\left[y_i \ln\left(\hat{y}_i\right) + \left(1 - y_i\right)\ln\left(1 - \hat{y}_i\right)\right] \tag{3.18}$$

where $\hat{y}_i = E\left(Y|X = x_i\right)$ is the prediction. In order to maximize the expected value, the maximum of the log-likelihood function can be found using the gradient descent algorithm.

In the Logistic Regression model the output variable is binary but for the multi-class classification problem two approaches can be used. In the *One-vs.-rest* approach k different binary classifiers are build, where each classifier tries to identify only the samples of one class.

$$h_\theta^{(i)}\left(x\right) = P\left(y = i|x;\theta\right) \qquad \left(i = 1,..,k\right) \tag{3.19}$$

Otherwise, the *multinomial* method is a modification of the binary model to handle an outcome with more than two levels. The number of logit functions needed for the multi-class problem is $K-1$, where K is the number of classes. Let assume $K = 0$ as the baseline outcome to be compared to. Then the logit function for each k category can be stated as

$$g_k\left(x\right) = \ln\left[\frac{P\left(Y = k|x\right)}{P\left(Y = 0|x\right)}\right] = \beta_{k0} + \beta_{k1}x_1 + ... + \beta_k x_j = \beta_k^T \mathbf{x} \tag{3.20}$$

where $\mathbf{x}$ is the input vector with j features and β_k is the weight vector for class k. Then, a general expression for the conditional probability is

$$P\left(Y = k|x\right) = \frac{e^{g_k(x)}}{\sum_{k=0}^{K} e^{g_k(x)}} \tag{3.21}$$

where $\beta_0 = 0$ and $g_0\left(x\right) = 0$.
In order to build the likelihood function, K binary variables are needed to indicate

the group membership of an observation. It has to be noted that the sum of this binary variables will be always $\sum_{k=0}^{K} Y_k = 1$. The conditional likelihood function for an observation i is

$$l\left(\beta_i\right) = \prod_{k=0}^{K} E\left(Y = k|x_i\right)^{y_{ki}} \tag{3.22}$$

The log-likelihood function can be expressed in terms of

$$\ln\left(l\left(\beta\right)\right) = \sum_{i=1}^{n}\sum_{k=0}^{K} y_{ki}\ln\left[E\left(Y = k|x_i\right)\right] \tag{3.23}$$

The maximum likelihood estimator, $\hat{\beta}$, is obtained by deriving the likelihood equations, setting these obtained equations to zero and solving for β using the descent gradient method.

Chapter 4
Model Evaluation

In this chapter, different aspects regarding the evaluation of the model are introduced. In the first section, the environment where the models have been trained is described. In Section 4.2, the learning algorithms are tuned in order to find the combination of parameters to obtain a better performance in the results. Section 4.3 explains the input data for the different experiments. In Section 4.4 the different metrics used to evaluate the models are introduced. Finally, Section 4.5 details the different experiments conducted.

4.1 Experiment environment

The experiments have been carried out in a server with **CPU**: 2 x Intel(R) Xeon(R) E5-3643, 3.30GHz and **GPU**: Tesla K20c, 706MHz with 64 GB RAM while the input files containing the tweets and the hashtags and other required files have been generated in a MacBook Pro (late 2013) with 2,4GHz Intel Core i5 and 4 GB 1600 MHz DDR3.

The learning algorithms used in this work have been implemented using the free machine learning library for Python *Scikit-learn* [12].

4.2 Tuning the model parameters

In machine learning algorithms most of the parameters such as, the support vectors in the SVMs or the weights in the logistic regression model, are learned while looking at the training data. However, there are some parameters that must be manually tuned by the person using these algorithms.

In the *Scikit-learn* library these model parameters are set to a value by default when initializing one of the models. However, the learning algorithms are used in a wide range of applications with notable differences in the input data introduced. These hyper parameters affect significantly the performance of the model and finding the right combination of values for the required needs is one of the most important problems that need to be faced.

The tuning of the parameters can be automatically done using the GridSearchCV method from the *Scikit-learn* library. This method trains the same learning method with all the possible combinations to find the optimal values using a cross validation to evaluate the performance of the model such as, the precision or the recall.

All the learning models have been tuned using the BoW approach and taking into account the same dataset, containing tweets from 10 different hashtags. The hashtags used and the number of samples from each hashtag are detailed in Table 4.1.

Hashtag	**Training Set**	**Test Set**
#HappyBirthdaySachin	1462	486
#heatwave	1441	506
#LateLateShawn	1474	473
#TuesdayThoughts	1470	477
#ITrySoHardBut	1490	457
#SoldAtDevilsYardSale	1443	486
#GreysAnatomy	1415	455
#TheBachelerotte	1168	419
#DebateINE	963	328
#TrumpKimSummit	930	333
Total	13256	4419

Table 4.1: Hashtags used for tuning the parameters.

4.2.1 Multinomial Naives Bayes

One of the main advantages of using Multinomial Naives Bayes for classification is its easy implementation. The only parameter to be tuned is the *alpha* parameter, related to the Laplace/Lidstone additive smoothing. The values used for the automatic tuning of the alpha parameter are

$$\alpha = [0, 0.01, 0.02, 0.03, ..., 1.98, 1.98, 1.99, 2]$$

The best performance is achieved using and alpha $\alpha = 0.05$ and the obtained score is 0.7805 as can be seen in Figure 4.1.

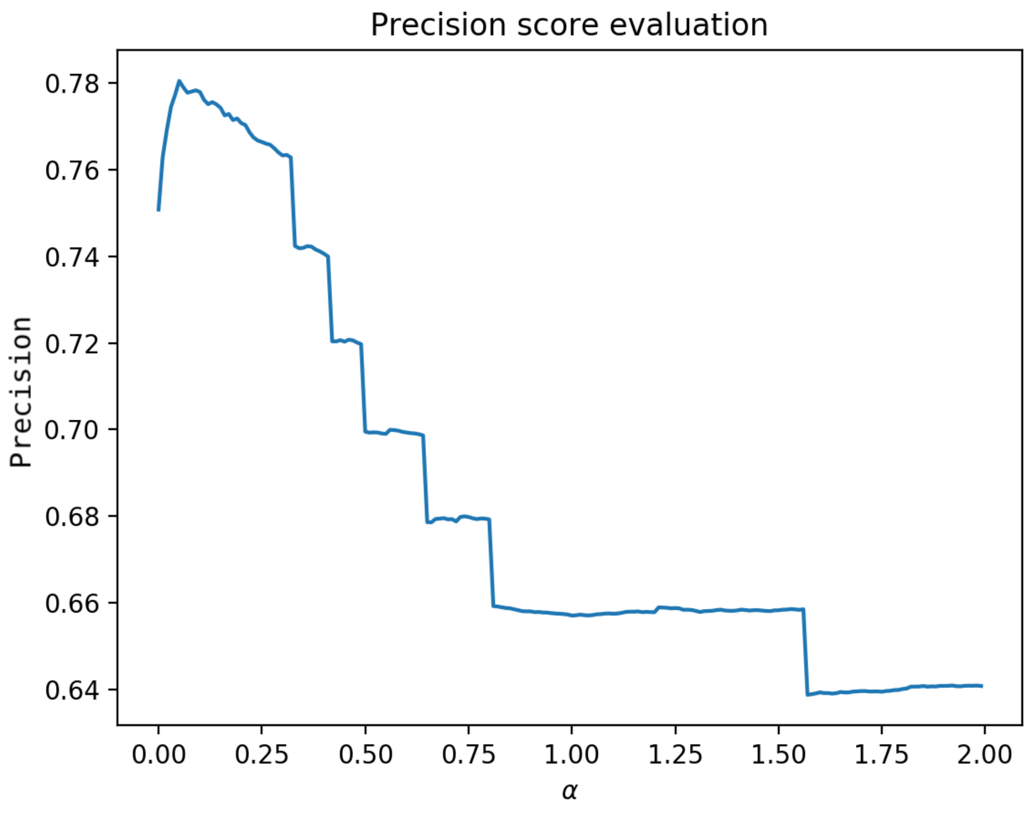


Figure 4.1: Obtained scores for α.

4.2.2 Support Vector Machines

In the SVM algorithm there are more parameters to be tuned. In this work the ones tuned are the regularization parameter of the error term C, the penalty that specifies the norm used in the penalization, the loss function and the tolerance for stopping criterion. The values used for each parameter are

$$\begin{aligned} C &= [0.1, 0.6, 1.1, ..., 9.1, 9.6, 10.1.] \\ \text{penalty} &= [\text{'l2'}] \\ \text{loss} &= [\text{'hinge'},\ \text{'squared\_hinge'}] \\ \text{tol} &= \left[10^{-5}, 8.07 \cdot 10^{-5}, 0.00015, ..., 0.00086, 0.00093, 0.001\right] \end{aligned}$$

The results after evaluating 600 model combinations are presented in Figure 4.2.

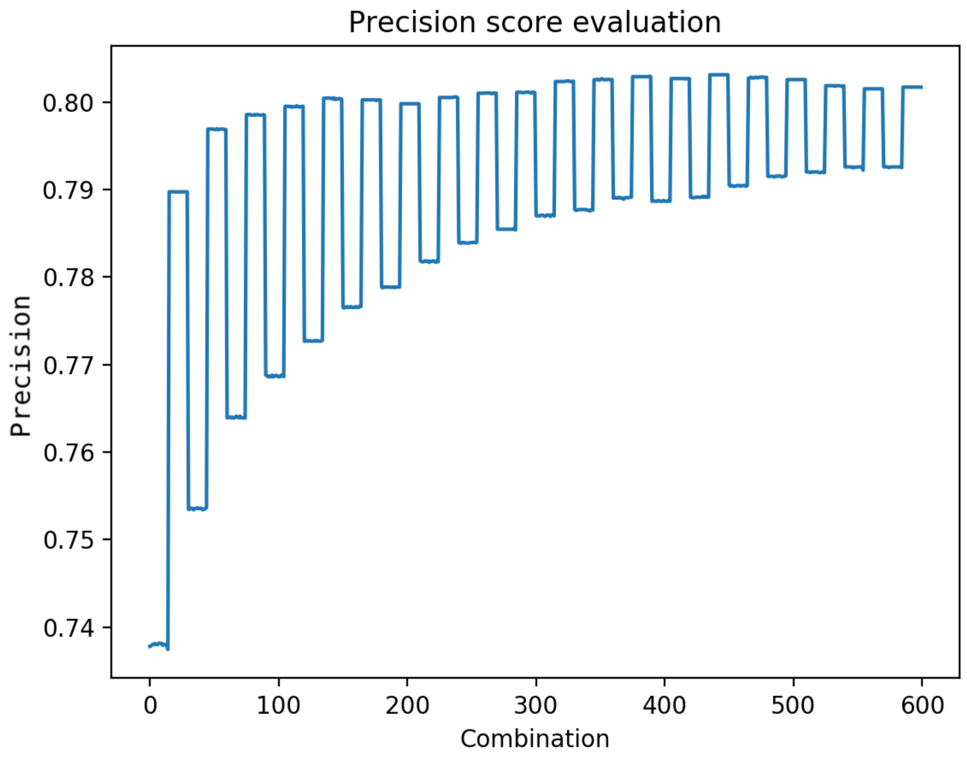


Figure 4.2: Obtained scores for the SVM tuning parameters.

The highest score achieved is 0.8031 and seems to be obtained in a range of combinations. In this combinations the only parameter not remaining constant is the tolerance *tol*. A detailed analysis is presented in Figure 4.3.

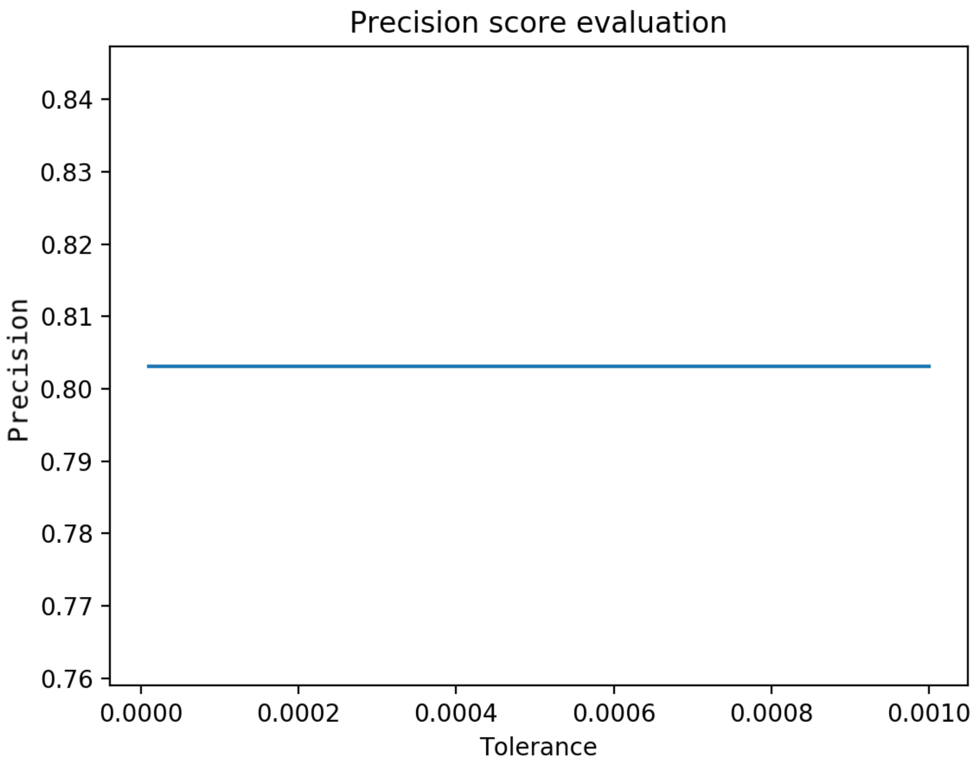


Figure 4.3: Obtained scores for the SVM modifying the tolerance.

As can be seen in Figure 4.3 the score is independent of the tolerance used, for this reason the tolerance has been set by default. The maximum score is obtained when the parameters are set to $C = 7.1$, *'loss'*='squared_hinge' and *'penalty'*='l2'.

4.2.3 Logistic Regression

One of the main advantages of the logistic regression model is its simplicity for the parameter tuning. The parameter C is used as regularization parameter $C = 1/\lambda$. λ controls the trade-off between keeping the model simple while trying to increase its complexity. If λ is very low, the model will have the power to increase its complexity and reach overfitting. On the other hand, with an increase of the value of λ, the model will become too simple and will tend to underfit.

The other parameters tuned are the *penalty* to specify the norm used in the penalization, the *tolerance* for stopping criteria, the *solver* used and the *multi_class* to specify the multi class approach used. The values used for each parameter are

$$\begin{aligned}
C &= [0.1, 0.6, 1.1, ..., 9.1, 9.6, 10.1] \\
\text{penalty} &= [\text{'l2'}] \\
\text{tol} &= \left[10^{-5}, 8.07 \cdot 10^{-5}, 0.00015, ..., 0.00086, 0.00093, 0.001\right] \\
\text{solver} &= [\text{'newton-cg'}, \text{'liblinear'}] \\
\text{multi\_class} &= [\text{'ovr'}, \text{'multinomial'}]
\end{aligned}$$

The multinomial multi classification approach can only be used when using the *newton-cg* solver. The tolerance for stopping criteria is set to default when this solver is used due to problems with the algorithm convergence. The results obtained using the *liblinear* solver are presented in Figure 4.4.

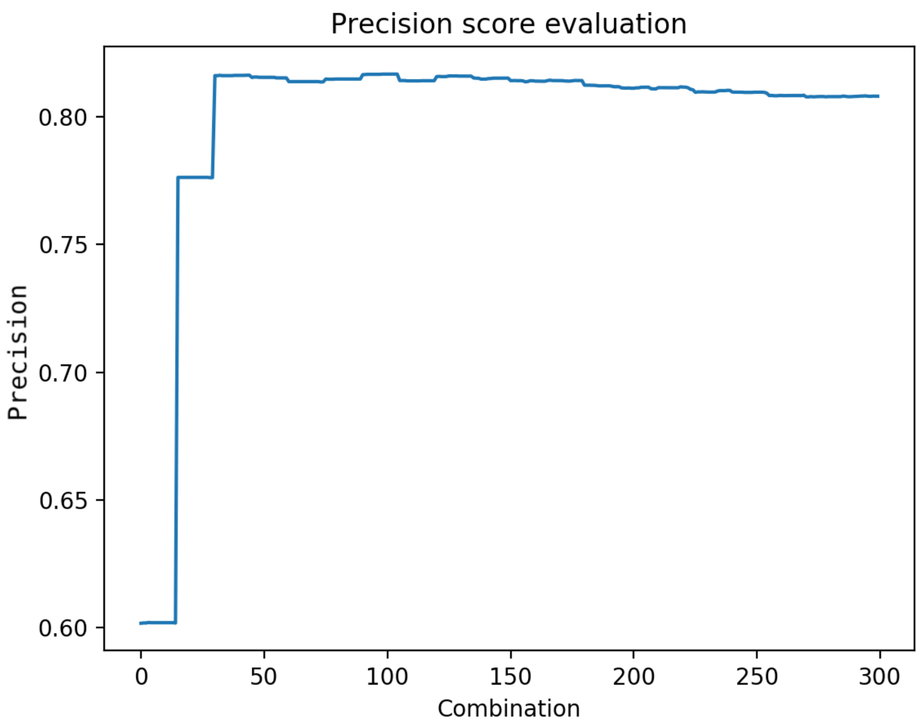


Figure 4.4: Obtained scores for the logistic regression using *liblinear* solver.

The highest score is 0.8168 and is obtained when $C = 3.1$, using the *ovr* approach for multi-classification and when the tolerance is set to 0.00071714. On the other hand, when using the *newton-cg* solver the highest score obtained is 0.8228 as can be seen in Figure 4.5. The score is obtained when using $C = 0.6$ and the multi-class approach is *multinomial*.

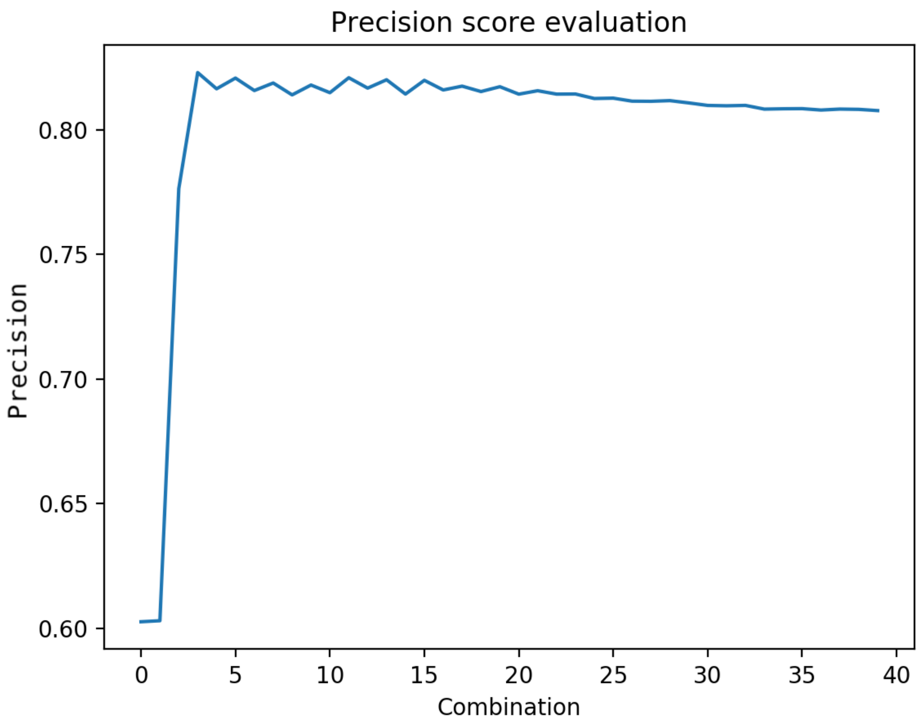


Figure 4.5: Obtained scores for the logistic regression using the *newton-cg* solver.

4.3 Dataset used

Tweets in english with the top trending topics were collected between April 20th to June 20th 2018. The number of tweets collected from each hashtag can be seen in Figure 4.6. Table A.1 in Appendix A includes all hashtags collected.

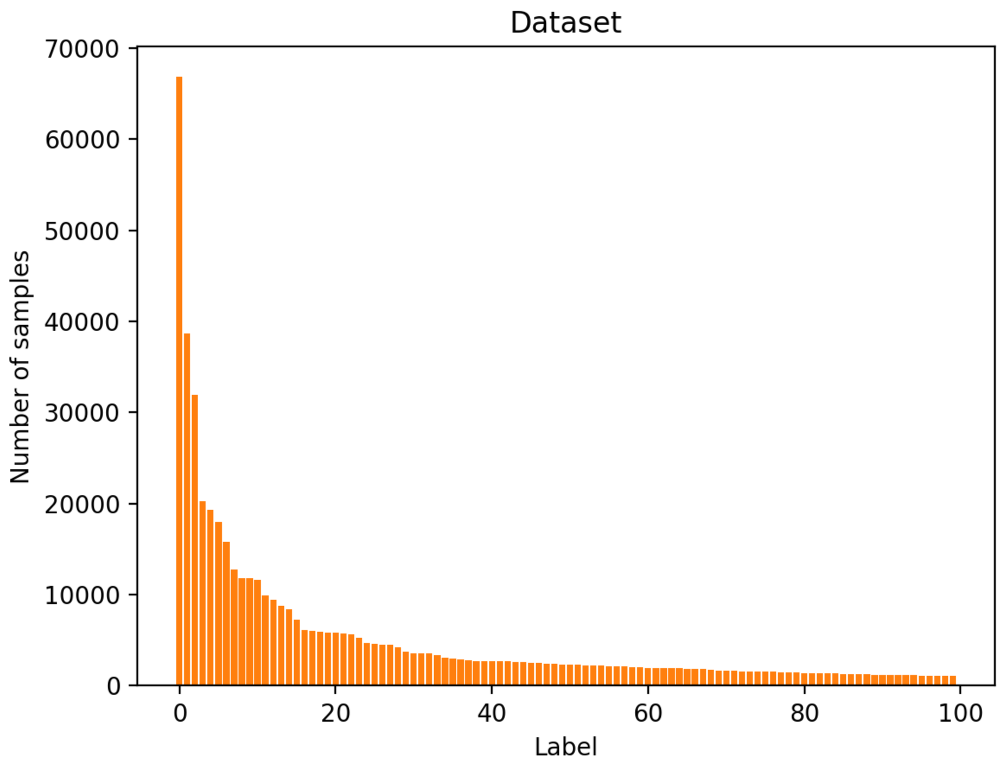


Figure 4.6: Number of samples for each hashtag in the dataset.

From Figure 4.6 can be concluded that the dataset is strongly unbalanced. It was expected due to the fact that users don't post tweets about different topics with the same frequency. In order to avoid classifying all samples to the same class an under-sampling has been performed to obtain a more balanced dataset. Figure 4.7 illustrates the dataset used in this work, containing 193926 tweets.

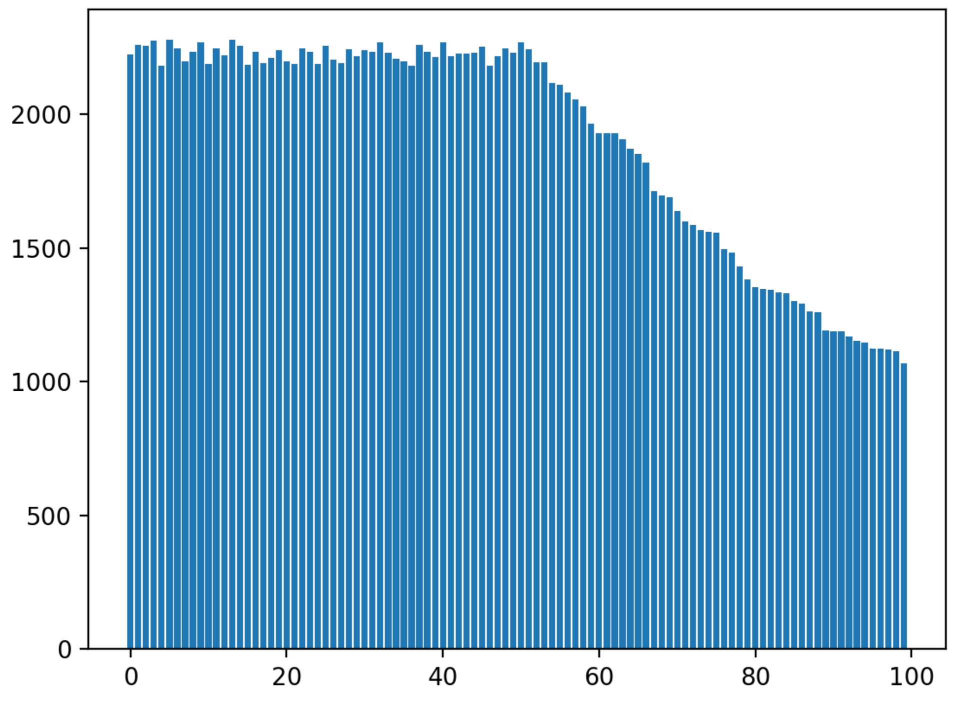


Figure 4.7: Obtained dataset after under-sampling

4.4 Metrics

In order to measure the performance of each model, the following measures have been computed
Precision

$$\text{Precision} = \frac{tp}{tp + fp}$$

where *tp* is the number of true positives and *fp* the number of false positives.
Recall

$$\text{Recall} = \frac{tp}{tp + fn}$$

where *tp* is the number of true positives and *fn* is the number of false negatives.
F1-score

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

4.5 Experiments

Detect if users label the tweet content with the proper hashtag can be achieved by evaluating the model performance in classifying tweets into different hashtags based on its content. If the classifier is able to find similarities between tweets with the same hashtag and differences with the ones with different hashtag then it can be concluded that the tweets were labelled by users with the proper hashtag. This is equivalent to evaluate the performance of the model and in this work several embedding methods and learning algorithms have been considered.

Embedding methods:

- BoW
- Tf-idf
- GloVe with a pre-trained twitter 25d vector using 27B words [1] (GloVe-pre)
- GloVe with vector created with the tweets vocabulary (GloVe)
- Word2Vec

Learning Algorithms

- MNB
- SVM
- Log Reg

1 https://nlp.stanford.edu/projects/glove/

Initially, 10 different experiments have been performed. In every experiment, the different embedding methods have been evaluated using all the learning algorithms. Tweets from 10 different hashtags have been taking into account as starting point for the experiments. In subsequent experiments, the number of hashtags used have been slightly increased, until the last experiment where the dataset was composed by tweets from 100 different hashtags. It should be noted that the hashtags have been randomly chosen in each of the experiments and no pre-selection has been conducted to select highly differentiated hashtags. The experiments are detailed in the following subsections.

4.5.1 Experiment 10#

In this experiment tweets from 10 different hashtags from the database have been randomly selected. The hashtags involved in this experiment are detailed in Table 4.2. The training set was composed by 13680 samples while the test set by 4561.

Hashtag	**Label**
#MondayMotivation	0
#ARMYHiveStreamingParty	1
#MITB	2
#StormHector	3
#IAmwayForward	4
#BMWMotorrad310	5
#Game7	6
#MLBDraft	7
#Redemption18	8
#BelowDeckMed	9

Table 4.2: Hashtags used in the 10# experiment.

4.5.2 Experiment 20#

The model has been evaluated using as input tweets from 20 different hashtags selected randomly from the database. Table 4.3 shows the hashtags used and its respective label. The models have been trained and evaluated using 29115 tweets as input and 9705 respectively.

Hashtag	Label	Hashtag	Label
#SaveShadowhunters	0	#LateLateShawn	10
#SDLive	1	#BBNaija	11
#The100	2	#WorldBookDay	12
#LHHATL	3	#BMWMotorrad310	13
#the100	4	#Origin	14
#WednesdayWisdom	5	#ComeyTownHall	15
#TeenMom2	6	#PoseFX	16
#civility	7	#ChicagoFire	17
#HappyBirthdayAriana	8	#90DayFianceHappilyEverAfter	18
#MITB	9	#BelowDeckMed	19

Table 4.3: Hashtags used in experiment 20#.

4.5.3 Experiment 30# to 100#

The hashtags and the labels used for experiments 30# to 100# are detailed in Appendix B, in Tables B.1, B.2, B.3, B.4, B.5, B.6, B.7 and B.8 respectively.

Chapter 5
Results

In this chapter the obtained results of the experiments are detailed. Section 5.1 refers to the experiments proposed in Section 4.5. Due to the obtained results, new experiments have been executed for several purposes and are detailed in Section 5.2. The aim of the experiment described in Subsection 5.2.1 was to identify the best combination of learning algorithms and embedding methods. The experiments detailed in subsections 5.2.2 and 5.2.3 have been conducted by modifying the dataset in order to improve the performance of the algorithm.

5.1 Results experiment 10# to 100#

The number of samples used for training and test sets for all the experiments are detailed in Table 5.1.

Hashtag	**Training Set**	**Test Set**
Experiment 10#	13680	4561
Experiment 20#	29115	9705
Experiment 30#	44730	14910
Experiment 40#	58347	19449
Experiment 50#	74472	24825
Experiment 60#	87895	29299
Experiment 70#	101955	33985
Experiment 80#	114802	38268
Experiment 90#	129868	43290
Experiment 100#	143067	47690

Table 5.1: Training and data sets used in each experiment.

5.1.1 Results experiment 10#

The obtained results in this experiment are summarized in Table 5.2.

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.889	0.886	0.886
	Tf-idf	0.895	0.876	0.883
	GloVe-pre	0.497	0.456	0.446
	GloVe	0.779	0.75	0.758
	Word2Vec	0.765	0.736	0.741
SVM	BoW	0.864	0.848	0.854
	Tf-idf	0.898	**0.887**	**0.892**
	GloVe-pre	0.515	0.496	0.473
	GloVe	0.802	0.789	0.792
	Word2Vec	0.837	0.82	0.827
Log Reg	BoW	0.887	0.857	0.868
	Tf-idf	**0.899**	0.851	0.868
	GloVe-pre	0.524	0.514	0.505
	GloVe	0.807	0.793	0.798
	Word2Vec	0.827	0.799	0.809

Table 5.2: Results obtained in Experiment 10#. The underlined numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Focusing in Table 5.2 on the learning algorithms, all of them achieve similar results when comparing with the same embedding method used. On the other hand, the differences between embedding methods are significant. The biggest difference can be found in the *GloVe-pre* method. The precision achieved in the best case is only 0.524 while compared to the rest of embedding methods, where the precision is above 0.75 in the worst case. The best performances have been obtained using *Log Reg* model regarding the precision (0.899) and the *SVM* when it comes to recall (0.887) and F1-Score (0.892). The best performance is achieved when using the *Tf-idf* as the embedding method.

In order to take a deeper look into the discrepancies between predicted and actual labels, the confusion matrix has been computed. In Figure 5.1, the confusion matrix for the *SVM* model using *Tf-idf* as embedding method is detailed.

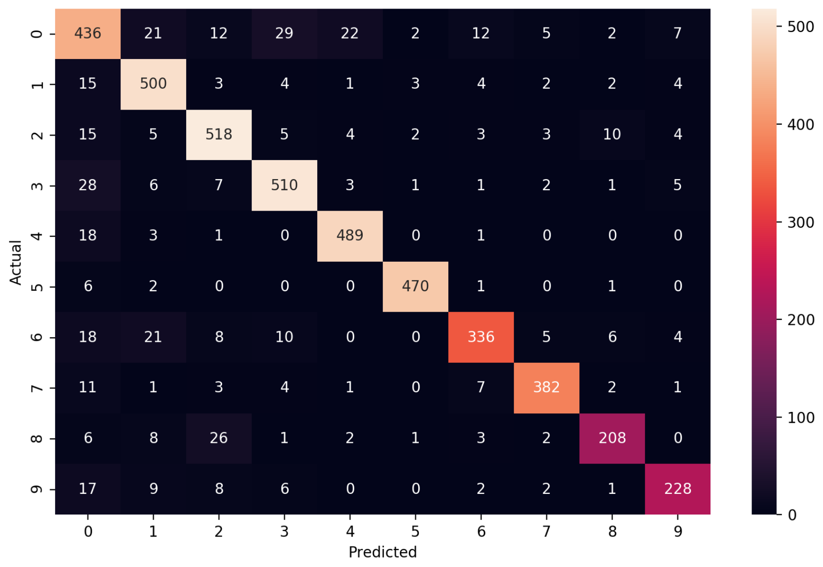


Figure 5.1: Confusion matrix for the SVM model using Tf-idf embedding method in Experiment 10#.

Most of the predictions are on the diagonal, which means that most of the predicted labels correspond to the correct ones. However, there are a number of misclassifications that have been summarized in Table 5.3.

Label	False positive	False negative	Samples	fp %	fn %
0	134	112	569	23.55	19.68
1	76	38	564	13.48	6.74
2	68	51	548	12.41	9.31
3	59	54	538	10.97	10.04
4	33	23	512	6.45	4.49
5	9	10	480	1.88	2.08
6	34	72	412	8.25	17.48
7	21	30	408	5.15	7.35
8	25	49	273	9.16	17.95
9	25	45	257	9.73	17.51

Table 5.3: Misclassifications obtained with the SVM model and the Tf-idf embedding method. **fp%** and **fn%** are the percentage of false positive and false negative over the number of samples respectivley.

The highest number of misclassifications are related to the hashtag with the label 0. According to Table 4.2 in Section 4.5 this label corresponds to the hashtag #MondayMotivation which is very general and can cover a wide range of meanings. This means that the words used in tweets labelled with this hashtag are similar to tweets with other hashtags, such as the ones with labels 3 and 4, which correspond to #StormHector and #IAmwayForward.

5.1.2 Results experiment 20#

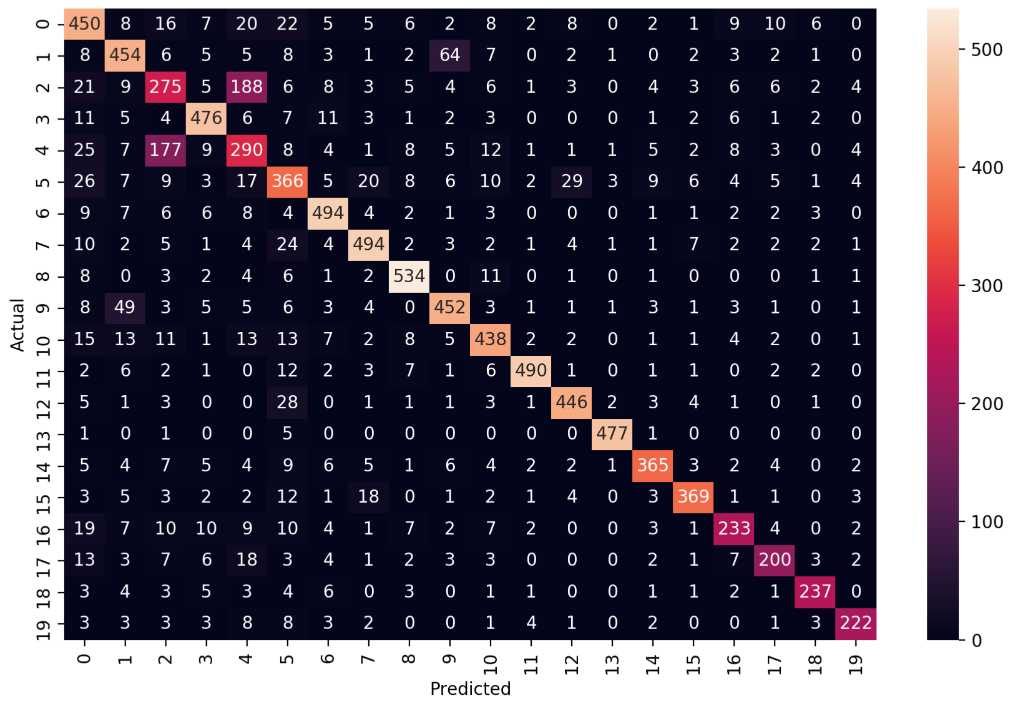
The obtained results in this second experiment are detailed in Table 4.3 and are similar to the ones obtained in Experiment #10 but with lower performance, as expected due to the increase on the number of classes.

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.801	<u>0.788</u>	<u>0.793</u>
	Tf-idf	<u>0.811</u>	0.773	0.785
	GloVe-pre	0.315	0.301	0.270
	GloVe	0.635	0.574	0.582
	Word2Vec	0.575	0.569	0.565
SVM	BoW	0.767	0.754	0.759
	Tf-idf	**<u>0.812</u>**	**<u>0.801</u>**	**<u>0.806</u>**
	GloVe-pre	0.309	0.328	0.274
	GloVe	0.644	0.631	0.625
	Word2Vec	0.666	0.649	0.65
Log Reg	BoW	<u>0.811</u>	<u>0.776</u>	<u>0.789</u>
	Tf-idf	0.810	0.761	0.775
	GloVe-pre	0.358	0.353	0.333
	GloVe	0.661	0.641	0.647
	Word2Vec	0.668	0.651	0.654

Table 5.4: Results obtained in Experiment 20#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

The number of possible classes have been increased by a factor of 2 while the obtained results for Precision, Recall and F1-Score are above 0.8 in all three cases. The best results are achieved when using SVM as learning algorithm and *Tf-idf* as embedding method. The obtained Precision, Recall and F1-Score are 0.812, 0.801 and 0.806 respectively.

On the other hand, the performance of the *GloVe-pre* has decreased significantly between 0.30 and 0.36 in all three metrics and models. The *GloVe* and Word2Vec have achieved better performance than *GloVe-pre* but are far away from the results achieved by *Tf-idf* or *BoW* methods. The obtained results by using the *BoW* embedding are over 0.75 and can also be accepted as good. The confusion matrix have been computed and is presented in Figure 5.2.



Actual \ Predicted	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	450	8	16	7	20	22	5	5	6	2	8	2	8	0	2	1	9	10	6	0
1	8	454	6	5	5	8	3	1	2	64	7	0	2	1	0	2	3	2	1	0
2	21	9	275	5	188	6	8	3	5	4	6	1	3	0	4	3	6	6	2	4
3	11	5	4	476	6	7	11	3	1	2	3	0	0	0	1	2	6	1	2	0
4	25	7	177	9	290	8	4	1	8	5	12	1	1	1	5	2	8	3	0	4
5	26	7	9	3	17	366	5	20	8	6	10	2	29	3	9	6	4	5	1	4
6	9	7	6	6	8	4	494	4	2	1	3	0	0	0	1	1	2	2	3	0
7	10	2	5	1	4	24	4	494	2	3	2	1	4	1	1	7	2	2	2	1
8	8	0	3	2	4	6	1	2	534	0	11	0	1	0	1	0	0	0	1	1
9	8	49	3	5	5	6	3	4	0	452	3	1	1	1	3	1	3	1	0	1
10	15	13	11	1	13	13	7	2	8	5	438	2	2	0	1	1	4	2	0	1
11	2	6	2	1	0	12	2	3	7	1	6	490	1	0	1	1	0	2	2	0
12	5	1	3	0	0	28	0	1	1	1	3	1	446	2	3	4	1	0	1	0
13	1	0	1	0	0	5	0	0	0	0	0	0	0	477	1	0	0	0	0	0
14	5	4	7	5	4	9	6	5	1	6	4	2	2	1	365	3	2	4	0	2
15	3	5	3	2	2	12	1	18	0	1	2	1	4	0	3	369	1	1	0	3
16	19	7	10	10	9	10	4	1	7	2	7	2	0	0	3	1	233	4	0	2
17	13	3	7	6	18	3	4	1	2	3	3	0	0	0	2	1	7	200	3	2
18	3	4	3	5	3	4	6	0	3	0	1	1	0	0	1	1	2	1	237	0
19	3	3	3	3	8	8	3	2	0	0	1	4	1	0	2	0	0	1	3	222

Figure 5.2: Confusion matrix for the SVM model using Tf-idf embedding method in Experiment 20#.

Most of the predictions remain in the diagonal. However, there are predictions regarding hashtags with labels 2 and 4 that seems to be problematic. According to Table 4.3 in Section 4.5 this labels correspond to hashtags #the100 and #The100 respectively. In this case the source of the error is that both hashtags are the same and are used to label tweets equally. The classifier is not able to find differences between both categories which reinforces the fact the users handle hashtags in a proper way. The obtained results repeating the experiment again but deleting the tweets from label 2 and the label itself are presented in Table 5.5.

Experiment	Precision	Recall	F1-Score
19# with SVM Tf-idf	**0.844**	**0.835**	**0.839**
20# with SVM Tf-idf	0.812	0.801	0.806

Table 5.5: Comparison between experiment with 19 and 20 hashtags using the SVM model with *Tf-idf* embedding.

5.1.3 Results experiment 30# to 100#

The obtained results in experiments with 30 hashtags to 100 are presented in Appendix C, in Tables C.1, C.2, C.3, C.4, C.5, C.6, C.7 and C.8, respectively. The evolution of the evaluated metrics as the number of hashtags increased taking into account all the embedding methods have been summarized in Figures 5.3, 5.4 and 5.5 for each learning algorithm.

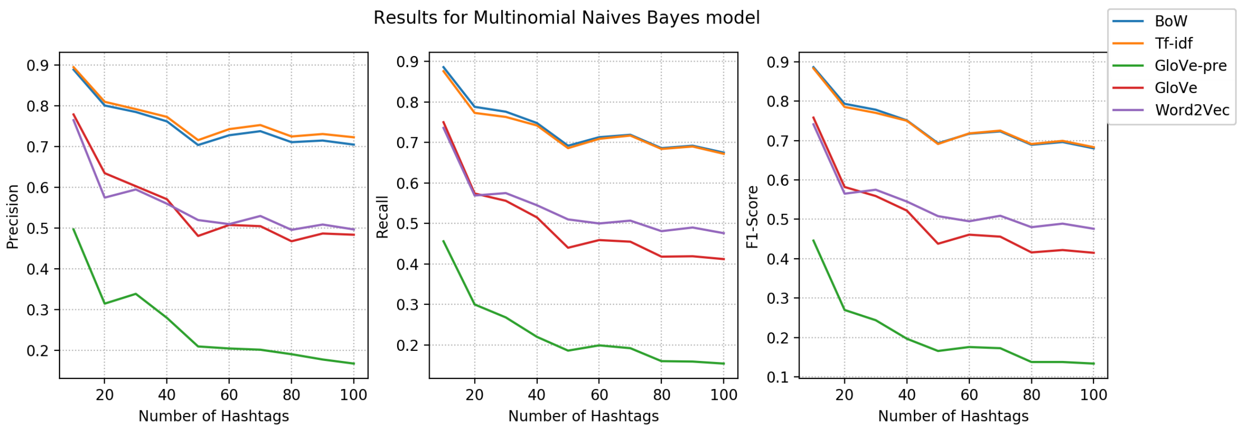


Figure 5.3: Evolution of the Precision, Recall and F1-Score obtained in the Multinomial Naives Bayes model.

The precision, recall and F1-Score tends to decrease as the number of classes increases. Anyway, the decrease is more pronounced when going from 10 to 40 hashtags rather than from 60 to 100. As can be stated, the bests embedding methods are by far the *BoW* and *Tf-idf* while the worst performance is achieved when using the *GloVe-pre*. On the other hand, the *GloVe* and *Word2Vec* achieved a good results when dealing with the simplest case where the classification was only between 10 different classes. However, as the number of classes has been increased, the obtained precision, recall and F1-score have been reduced ending in around 0.5, 0.45 and 0.45 respectively in the experiment with 100 different hashtags.

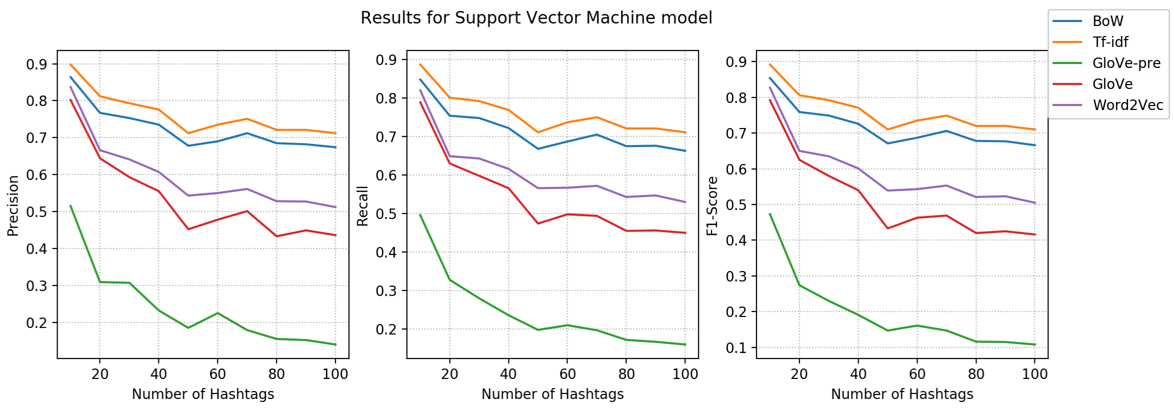


Figure 5.4: Evolution of the Precision, Recall and F1-Score obtained in the Support Vector Machine model.

From Figures 5.3, 5.4 and 5.5, it can be concluded that the best embedding methods are the *BoW* and the *Tf-idf*. However, it can not be stated which of the learning algorithms achieve better results. For this reason, a comparison between them has been evaluated in Section 5.2.1. On the other hand, it should be mentioned the poor performance achieved by *GloVe-pre* algorithm. This is caused because pre-trained vectors from another twitter dataset have been used. The poor performance acheived suggests that the words and expressions used in one dataset differ a lot from the ones used in the other.

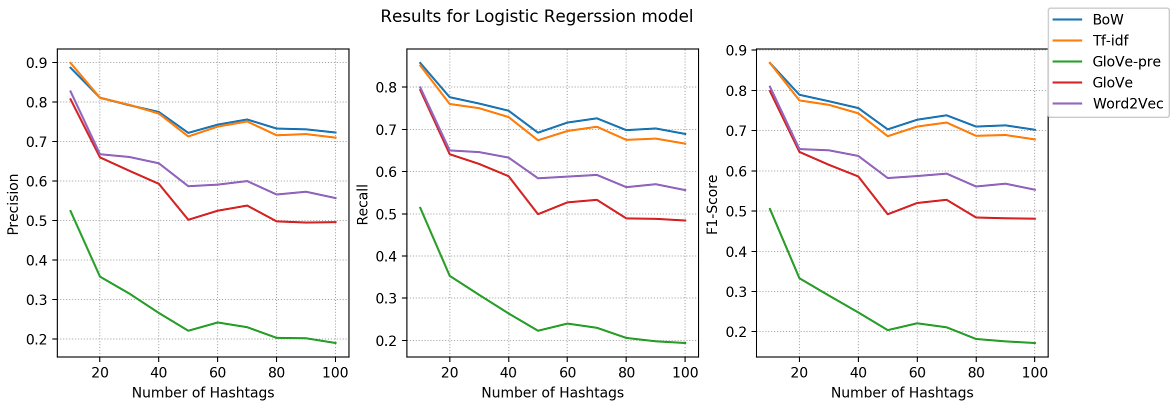


Figure 5.5: Evolution of the Precision, Recall and F1-Score obtained in the Logistic Regression model.

5.2 Other results

In this section the results of other experiments conducted are explained. After analyzing the results of experiments described in Section 4.5, another experiment has been considered in order to detect the model and the embedding method that achieved the best performance. Furthermore, an experiment with 10 different hashtags using a larger dataset and an another one deleting the duplicated or similar hashtags have also been explored and presented.

5.2.1 Comparison between learning algorithms

From Section 5.1 can only be concluded that the *Tf-idf* and *BoW* are the best options to convert the tweets content into a way to be understand by the learning algorithms. In this subsection, a comparison between the learning algorithms when using those two embedding methods have been conducted. The evolution of the precision, recall and F1-Score while increasing the number of classes are presented in Figure 5.6.

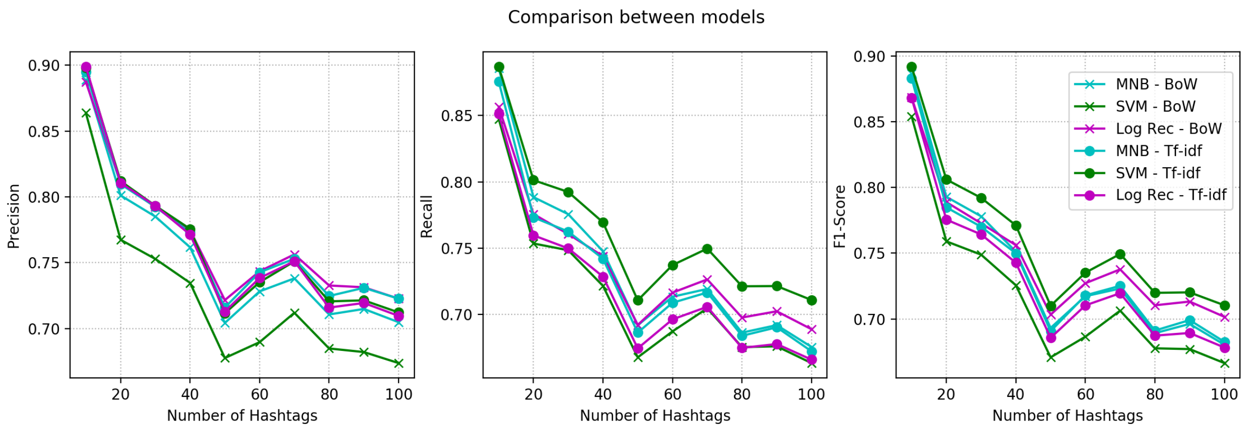


Figure 5.6: Precision, recall and F1-score comparison between the models.

According to Figure 5.6, the best results are achieved when the Support Vector Machine with the *Tf-idf* approach is used. However, when comparing different models and methods, it is also important to evaluate the computational time required for the learning algorithm to be trained. A detailed analysis has been conducted considering the three learning algorithms and the *BoW* and *Tf-idf* embedding methods. The results are presented in Figure 5.7.

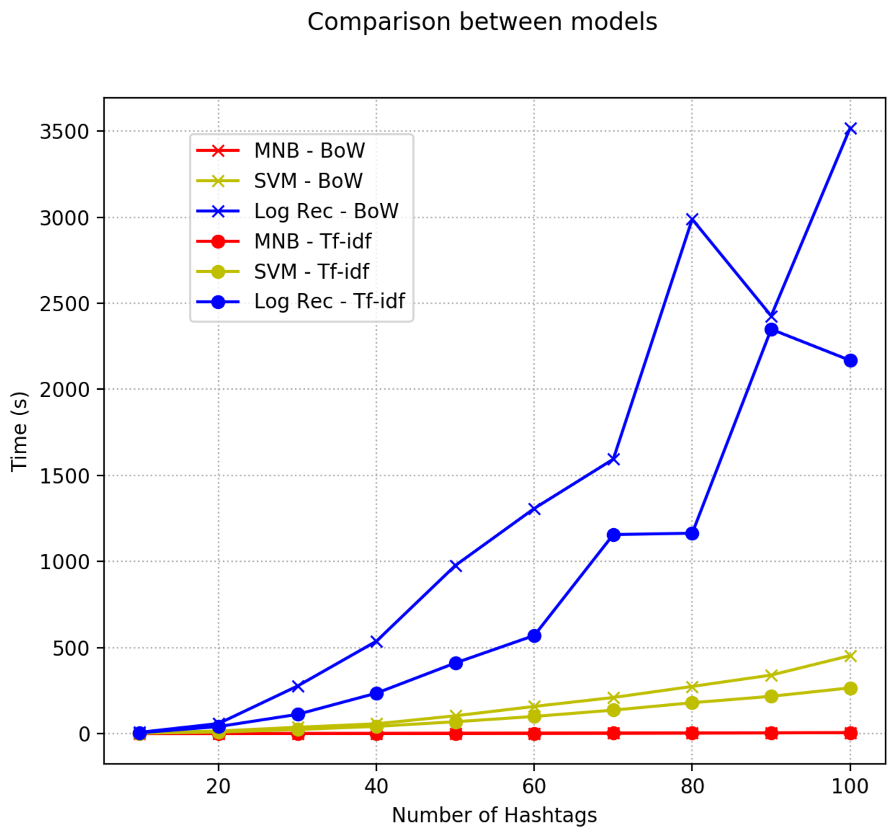


Figure 5.7: Training computational time comparison between learning algorithms.

As can be observed in Figure 5.7, the Logistic Regression algorithm needs more time to be trained than the other methods. As can be expected the required time increase while increasing the number of samples of the dataset used. The required time for training using the *BoW* approach is around 3500 seconds and 2168 for the *Tf-idf*. On the other hand, the fastest algorithm is the Multinomial Naives Bayes approach. It only needs less than 6 seconds to be trained in both methods when dealing with more than 143.000 samples. Finally, the Support Vector Machine algorithm needed 454 seconds using the *BoW* approach and 265 for the *Tf-idf*.

5.2.2 Experiment 10# using a bigger dataset

GloVe and Word2Vec approaches used are usually trained with a larger dataset as stated in [13] and [11] respectively. Since they have achieved state of the art results in other natural language processing problems, one of the reasons of the poor results obtained in the experiments could have been the size of the dataset used. For this reason another experiment has been conducted using a larger dataset. The hashtags and the number of samples used are detailed in Table 5.6.

Hashtag	**Label**	**Training Sample**	**Test Sample**
#BETAwards	0	12489	4343
#TrumpKimSummit	1	12629	4215
#NFLDraft	2	12694	4186
#SaveShadowhunters	3	12713	4140
#TheBachelorette	4	12640	2919
#MetGala	5	11889	3887
#Westworld	6	8852	2919
#PlayStationE3	7	8710	2913
#WorldEnvironmentDay	8	7340	2432
#ShawnMendesTheAlbum	9	7045	2353

Table 5.6: Hashtag and samples used for the experiment.

For this experiment 107001 samples were used for training purposes and 35667 for testing, using the SVM as learning algorithm. The obtained results are presented in Table 5.7.

Experiment	Embedding Method	Precision	Recall	F1-Score
Larger	BoW	0.872	0.866	0.868
Dataset	Tf-idf	0.895	0.891	0.893
	GloVe-pre	0.482	0.46	0.444
	GloVe	0.801	0.788	0.792
	Word2Vec	0.845	0.841	0.843
Original	BoW	0.864	0.848	0.854
Dataset	Tf-idf	0.898	0.887	0.892
	GloVe-pre	0.515	0.496	0.473
	GloVe	0.802	0.789	0.792
	Word2Vec	0.837	0.82	0.827

Table 5.7: Comparison between experiments using a larger dataset and the original dataset.

The obtained results are not better that those obtained in the original experiment in general terms. This suggests that the number of tweets used in the original experiment was correct.

5.2.3 Experiment deleting similar hashtags

The results using 100 different hashtags have shown that with some hashtags the algorithm was not able to distinguish the features between tweets from different hashtags and badly classified an important number of samples. After carefully reviewing the hashtags used, it has been observed that in some cases the hashtags were repeated or have a very similar meaning. Based on the hashtags used in experiment 100#, a new experiment has been performed deleting the corresponding hashtags. The hashtags deleted are presented in Table 5.8.

Hashtags	
#DDay	#the100
#FifaWorldCup_BTS	#TheBachelerotte
#FortniteE3	#ThursdayThoughts
#HappyBirthdaySachin	#TuesdayThoughts
#MondayMotivation	#WednesdayWisdom
#saveshadowhunters	#WorldCup

Table 5.8: Deleted hashtags from the 100# dataset.

For this experiment only the SVM learning algorithm with the *Tf-idf* approach have been considered. The training dataset used is made up of 121666 tweets from 88 different hashtags. The model is tested using 40556 samples. The obtained results are presented in Table 5.9.

Experiment	**Precision**	**Recall**	**F1-Score**
88# with SVM Tf-idf	**0.754**	**0.750**	**0.751**
100# with SVM Tf-idf	0.712	0.711	0.711
80# with SVM Tf-idf	0.721	0.721	0.721

Table 5.9: Results comparison between experiment 88#, 80# and 100#.

The results in this experiment are higher that the ones obtained in the experiment with 100#. It could be possible to think that the reason for this better performance is because the number of classes is lower. However, when comparing the results with the ones obtained in the experiment with 80 classes, the precision, recall and F1-score are also higher.

Chapter 6
Future Work

The study presented in this work has focused in the feature extraction using a word level approach. Further work could extend this study to the character level approach. It would be interesting to see if other state of the art methods in other tasks such as hashtag prediction can outperform the results achieved so far with the same small dataset used in this work. Tweet2Vec, introduced by Dhingra et al. proposed a Bi-directional Gated Recurrent Unit neural network for learning tweet representations [6] scoring a precision of 33.1% on a datasize with 933 hashtags and 2 million tweets for training, outperforming the Word2Vec used as baseline.

Another continuation is to apply this study on tweets and hashtags in other non-english languages and compare the performance using datasets where tweets from several languages are included. Yang et al. introduced a linear translation for multi-language classification creating a translation matrix to bridge the gap between languages for twitter election classification [22].

Finally, another possibility is to conduct a more extensive study of the hashtags usage. Identifying different hashtags types and evaluating the performance of the model using training datasets composed by hashtags from the same type could be an option. Another interesting possibility could be to compare these results to the ones achieved when using hashtags from different classes.

Chapter 7
Conclusions

The main goal of this thesis was to study the proper usage of hashtags by the users when labelling the tweet content. If a learning algorithm was able to handle this classification task based on the tweet content then it could be said that the hashtags were used in a proper way because a relation between tweets from the same hashtag could be found. For this reason, a comparative study of different word level approaches and learning algorithms have been presented.

As stated before, extracting features from Tweets is a challenging task. People write content in an informal way and spelling and grammar errors are commonly found. In addition, the length restriction causes words to be abbreviated or omitted more often than desired. For those reasons, a lot of preprocessing was required and a lot of tweets were discarded when creating the dataset. The idea behind this strong data processing was in order to obtain a dataset with strongly related content to the hashtags, such as keywords or common expressions for that hashtag, instead of having a dataset full of raw data.

From the obtained results it can be concluded that the Support Vector Machine with the Tf-idf approach was the best combination of learning algorithm and embedding method, scoring precision above 70% for classification on 100 classes. The results also showed that if tweets from similar hashtags were deleted, the precision increased to 75% for classification on 88 classes. This states that the algorithm performance depends on the set of hashtags used due to the fact that not all the hashtags are used in the same way. Some of them describe a specific event that is taking place, others help to express random thoughts while others are used as a general topic. However, this problem also takes place when trying to solve this classification problem manually and it can be concluded that hashtags are used in the way they were originally invented, to label related content.

References

[1] M. Abtahi, J. V. Gyllinsky, B. Paesang, S. Barlow, M. Constant, N. Gomes, O. Tully, S. E. D?Andrea, and K. Mankodiya. Magicsox: An e-textile iot system to quantify gait abnormalities. *Smart Health*, 5-6:4 – 14, 2018.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, Mar. 2003.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995.

[5] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science (JASIS)*, 41(6):391–407, 1990.

[6] B. Dhingra, Z. Zhou, D. Fitzpatrick, M. Muehl, and W. W. Cohen. Tweet2vec: Character-based distributed representations for social media. *CoRR*, abs/1605.03481, 2016.

[7] P. W. Foltz. Latent semantic analysis for text-based research. *Behavior Research Methods, Instruments, & Computers*, 28(2):197–202, Jun 1996.

[8] D. W. Hosmer and S. Lemeshow. *Applied logistic regression (Wiley Series in probability and statistics).* Wiley-Interscience Publication, 2 edition, 2000.

[9] W. Ling, I. Trancoso, C. Dyer, and A. W. Black. Character-based neural machine translation. *CoRR*, abs/1511.04586, 2015.

[10] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, UK, 2008.

[11] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[14] I. Pilszy. Text categorization and support vector machines. *6th International Symposium of Hungarian Researches on Computational Intelligence*, 11 2005.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[16] Z. S. Harris. Distributional structure. *Word*, 10:146–162, 08 1954.

[17] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, Inc., New York, NY, USA, 1986.

[18] F. Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, Mar. 2002.

[19] S. Taheri and M. Mammadov. Learning the naive bayes classifier with optimization models. *Int. J. Appl. Math. Comput. Sci.*, 23(4):787–795, Dec. 2013.

[20] S. Tellex, B. Katz, J. Lin, A. Fernandes, and G. Marton. Quantitative evaluation of passage retrieval algorithms for question answering. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 41–47, New York, NY, USA, 2003. ACM.

[21] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer-Verlag, Berlin, Heidelberg, 1995.

[22] X. Yang, R. McCreadie, C. Macdonald, and I. Ounis. Transfer learning for multi-language twitter election classification. *The 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 341–348, July 2017.

Appendix

A Tweets Database

Hashtag	**Samples**	**Hashtag**	**Samples**
#BETAwards	66816	#FathersDay	2270
#TrumpKimSummit	38633	#StormHector	2243
#NFLDraft	31936	#DebateINE	2195
#MondayMotivation	20285	#ALLCAPS	2194
#SaveShadowhunters	19319	#LateLateShawn	2117
#TheBachelorette	18007	#WorldOceansDay	2111
#MetGala	15776	#IAmwayForward	2083
#ARMYHiveStreamingParty	12739	#BBNaija	2057
#mprraccoon	11795	#WorldBookDay	2031
#Westworld	11771	#GreysAnatomy	1965
#PlayStationE3	11623	#BMWMotorrad310	1930
#WorldEnvironmentDay	9872	#RHONY	1929
#ShawnMendesTheAlbum	9398	#heatwave	1929
#SDLive	8799	#HAHN	1906
#The100	8343	#HappyBirthdaySachin	1870
#LHHATL	7261	#Supergirl	1852
#Brooklyn99	5901	#Origin	1820
#NBAAwards	5854	#ComeyTownHall	1714
#the100	5821	#LoveIs	1697
#WorldCup	5708	#SpaceForce	1691
#MTVAwards	5606	#CAGovDebate	1639
#WorldCupRussia2018	5272	#TheBoldType	1601
#BasketballWives	4704	#ITrySoHardBut	1587
#saveshadowhunters	4631	#DaytimeEmmys	1567
#E32018	4546	#Game7	1562
#WednesdayWisdom	4501	#MLBDraft	1557

#INDvAFG	4184	#WWENXT	1497
#HeartMCSeokjin	4175	#TheProblemWithMeIn5Words	1484
#NoTearsLeftToCry	3737	#FortniteProAM	1433
#TuesdayThoughts	3578	#DDay	1384
#MarriedAtFirstSight	3544	#PoseFX	1355
#PickUpShadowhunters	3518	#Voicenotes	1347
#StGeorgesDay	3378	#GreenForGrenfell	1344
#MyHandleExplained	3102	#Riverdale	1335
#TonyAwards	2976	#ENGTUN	1331
#TeenMom2	2892	#TheLightIsComing	1302
#EarthDay	2833	#TheBachelerotte	1291
#civility	2731	#SoldAtDevilsYardSale	1263
#InfinityWar	2681	#WorldBloodDonorDay	1260
#WhateverItTakes	2674	#TheRealityOfDepressionIs	1193
#HappyBirthdayAriana	2664	#AmericanIdol	1190
#ThursdayThoughts	2648	#CaliforniaPrimary2018	1189
#RHOBHReunion	2614	#BTSxCorden	1168
#TeenChoice	2590	#GE14	1154
#RHOA	2506	#Congrats5SOS	1148
#FIFAWorldCup_BTS	2476	#90DayFianceHappilyEverAfter	1123
#MITB	2446	#ChicagoFire	1123
#FridayFeeling	2396	#Redemption18	1120
#LoveIsland	2361	#GlobalRunningDay	1114
#FortniteE3	2293	#BelowDeckMed	1068

Table A.1: Original dataset of retrieved tweets

B Hashtags used in the experiments

Hashtag	Label	Hashtag	Label
#BETAwards	0	#ThursdayThoughts	15
#TheBachelorette	1	#FIFAWorldCup_BTS	16
#Westworld	2	#MITB	17
#PlayStationE3	3	#FridayFeeling	18
#WorldEnvironmentDay	4	#FathersDay	19
#ShawnMendesTheAlbum	5	#StormHector	20
#The100	6	#IAmwayForward	21
#LHHATL	7	#RHONY	22
#WednesdayWisdom	8	#DDay	23
#INDvAFG	9	#GreenForGrenfell	24
#NoTearsLeftToCry	10	#TheLightIsComing	25
#TonyAwards	11	#AmericanIdol	26
#TeenMom2	12	#BTSxCorden	27
#civility	13	#90DayFianceHappilyEverAfter	28
#HappyBirthdayAriana	14	#BelowDeckMed	29

Table B.1: Hashtags used in experiment 30#.

Hashtag	Label	Hashtag	Label
#TrumpKimSummit	0	#LoveIsland	20
#ShawnMendesTheAlbum	1	#StormHector	21
#SDLive	2	#ALLCAPS	22
#LHHATL	3	#IAmwayForward	23
#Brooklyn99	4	#RHONY	24
#NBAAwards	5	#HappyBirthdaySachin	25
#WorldCupRussia2018	6	#Supergirl	26
#BasketballWives	7	#LoveIs	27
#saveshadowhunters	8	#SpaceForce	28
#E32018	9	#TheBoldType	29
#WednesdayWisdom	10	#DaytimeEmmys	30
#INDvAFG	11	#Game7	31
#NoTearsLeftToCry	12	#FortniteProAM	32
#PickUpShadowhunters	13	#DDay	33

#TeenMom2	14	#PoseFX	34
#HappyBirthdayAriana	15	#GreenForGrenfell	35
#ThursdayThoughts	16	#ENGTUN	36
#RHOBHReunion	17	#TheLightIsComing	37
#FIFAWorldCup_BTS	18	#TheBachelerotte	38
#MITB	19	#Congrats5SOS	39

Table B.2: Hashtags used in experiment 40#.

Hashtag	**Label**	**Hashtag**	**Label**
#BETAwards	0	#InfinityWar	25
#TrumpKimSummit	1	#RHOBHReunion	26
#NFLDraft	2	#FIFAWorldCup_BTS	27
#MondayMotivation	3	#MITB	28
#SaveShadowhunters	4	#LoveIsland	29
#TheBachelorette	5	#StormHector	30
#MetGala	6	#WorldOceansDay	31
#ARMYHiveStreamingParty	7	#IAmwayForward	32
#Westworld	8	#WorldBookDay	33
#WorldEnvironmentDay	9	#heatwave	34
#ShawnMendesTheAlbum	10	#RHONY	35
#SDLive	11	#Supergirl	36
#The100	12	#ComeyTownHall	37
#Brooklyn99	13	#SpaceForce	38
#the100	14	#ITrySoHardBut	39
#WorldCupRussia2018	15	#DaytimeEmmys	40
#BasketballWives	16	#MLBDraft	41
#saveshadowhunters	17	#PoseFX	42
#WednesdayWisdom	18	#GreenForGrenfell	43
#INDvAFG	19	#TheLightIsComing	44
#NoTearsLeftToCry	20	#TheBachelerotte	45
#TuesdayThoughts	21	#SoldAtDevilsYardSale	46
#PickUpShadowhunters	22	#GE14	47
#StGeorgesDay	23	#ChicagoFire	48
#TeenMom2	24	#GlobalRunningDay	49

Table B.3: Hashtags used in experiment 50#.

Hashtag	Label	Hashtag	Label
#BETAwards	0	#FridayFeeling	30
#TrumpKimSummit	1	#FortniteE3	31
#NFLDraft	2	#FathersDay	32
#MondayMotivation	3	#StormHector	33
#MetGala	4	#DebateINE	34
#ARMYHiveStreamingParty	5	#LateLateShawn	35
#PlayStationE3	6	#WorldOceansDay	36
#WorldEnvironmentDay	7	#BBNaija	37
#ShawnMendesTheAlbum	8	#GreysAnatomy	38
#The100	9	#BMWMotorrad310	39
#LHHATL	10	#RHONY	40
#Brooklyn99	11	#HAHN	41
#NBAAwards	12	#Origin	42
#the100	13	#ComeyTownHall	43
#WorldCup	14	#LoveIs	44
#BasketballWives	15	#SpaceForce	45
#saveshadowhunters	16	#TheBoldType	46
#WednesdayWisdom	17	#DaytimeEmmys	47
#HeartMCSeokjin	18	#MLBDraft	48
#TuesdayThoughts	19	#WWENXT	49
#PickUpShadowhunters	20	#FortniteProAM	50
#StGeorgesDay	21	#DDay	51
#MyHandleExplained	22	#Voicenotes	52
#TonyAwards	23	#TheRealityOfDepressionIs	53
#TeenMom2	24	#AmericanIdol	54
#WhateverItTakes	25	#BTSxCorden	55
#HappyBirthdayAriana	26	#GE14	56
#ThursdayThoughts	27	#ChicagoFire	57
#RHOBHReunion	28	#90DayFianceHappilyEverAfter	58
#FIFAWorldCup_BTS	29	#BelowDeckMed	59

Table B.4: Hashtags used in experiment 60#.

Hashtag	Label	Hashtag	Label
#BETAwards	0	#FortniteE3	35
#NFLDraft	1	#FathersDay	36
#MetGala	2	#StormHector	37
#ARMYHiveStreamingParty	3	#DebateINE	38
#mprraccoon	4	#ALLCAPS	39
#Westworld	5	#LateLateShawn	40
#PlayStationE3	6	#WorldOceansDay	41
#WorldEnvironmentDay	7	#IAmwayForward	42
#ShawnMendesTheAlbum	8	#GreysAnatomy	43
#SDLive	9	#RHONY	44
#LHHATL	10	#HAHN	45
#NBAAwards	11	#Supergirl	46
#the100	12	#Origin	47
#WorldCup	13	#ComeyTownHall	48
#MTVAwards	14	#LoveIs	49
#WorldCupRussia2018	15	#SpaceForce	50
#BasketballWives	16	#CAGovDebate	51
#saveshadowhunters	17	#TheBoldType	52
#WednesdayWisdom	18	#ITrySoHardBut	53
#INDvAFG	19	#DaytimeEmmys	54
#HeartMCSeokjin	20	#Game7	55
#NoTearsLeftToCry	21	#MLBDraft	56
#TuesdayThoughts	22	#FortniteProAM	57
#MyHandleExplained	23	#PoseFX	58
#TonyAwards	24	#GreenForGrenfell	59
#TeenMom2	25	#Riverdale	60
#EarthDay	26	#ENGTUN	61
#HappyBirthdayAriana	27	#TheLightIsComing	62
#ThursdayThoughts	28	#SoldAtDevilsYardSale	63
#RHOBHReunion	29	#WorldBloodDonorDay	64
#TeenChoice	30	#CaliforniaPrimary2018	65
#RHOA	31	#GE14	66
#FIFAWorldCup_BTS	32	#Congrats5SOS	67
#FridayFeeling	33	#90DayFianceHappilyEverAfter	68
#LoveIsland	34	#BelowDeckMed	69

Table B.5: Hashtags used in experiment 70#.

Hashtag	Label	Hashtag	Label
#BETAwards	0	#FathersDay	40
#TrumpKimSummit	1	#StormHector	41
#MondayMotivation	2	#DebateINE	42
#TheBachelorette	3	#ALLCAPS	43
#MetGala	4	#WorldOceansDay	44
#mprraccoon	5	#IAmwayForward	45
#Westworld	6	#BBNaija	46
#PlayStationE3	7	#GreysAnatomy	47
#WorldEnvironmentDay	8	#BMWMotorrad310	48
#ShawnMendesTheAlbum	9	#RHONY	49
#SDLive	10	#heatwave	50
#The100	11	#HappyBirthdaySachin	51
#LHHATL	12	#Supergirl	52
#Brooklyn99	13	#ComeyTownHall	53
#NBAAwards	14	#LoveIs	54
#WorldCup	15	#SpaceForce	55
#MTVAwards	16	#CAGovDebate	56
#WorldCupRussia2018	17	#TheBoldType	57
#BasketballWives	18	#ITrySoHardBut	58
#saveshadowhunters	19	#DaytimeEmmys	59
#E32018	20	#Game7	60
#WednesdayWisdom	21	#MLBDraft	61
#INDvAFG	22	#WWENXT	62
#HeartMCSeokjin	23	#TheProblemWithMeIn5Words	63
#TuesdayThoughts	24	#FortniteProAM	64
#MarriedAtFirstSight	25	#DDay	65
#PickUpShadowhunters	26	#PoseFX	66
#StGeorgesDay	27	#Voicenotes	67
#MyHandleExplained	28	#ENGTUN	68
#TonyAwards	29	#TheLightIsComing	69
#TeenMom2	30	#TheBachelerotte	70
#EarthDay	31	#SoldAtDevilsYardSale	71
#civility	32	#WorldBloodDonorDay	72
#InfinityWar	33	#AmericanIdol	73

#HappyBirthdayAriana	34	#CaliforniaPrimary2018	74
#ThursdayThoughts	35	#BTSxCorden	75
#RHOBHReunion	36	#GE14	76
#MITB	37	#Congrats5SOS	77
#FridayFeeling	38	#ChicagoFire	78
#LoveIsland	39	#BelowDeckMed	79

Table B.6: Hashtags used in experiment 80#.

Hashtag	**Label**	**Hashtag**	**Label**
#BETAwards	0	#FathersDay	45
#TrumpKimSummit	1	#StormHector	46
#NFLDraft	2	#DebateINE	47
#SaveShadowhunters	3	#ALLCAPS	48
#TheBachelorette	4	#LateLateShawn	49
#MetGala	5	#WorldOceansDay	50
#ARMYHiveStreamingParty	6	#IAmwayForward	51
#mprraccoon	7	#BBNaija	52
#Westworld	8	#WorldBookDay	53
#PlayStationE3	9	#GreysAnatomy	54
#WorldEnvironmentDay	10	#BMWMotorrad310	55
#ShawnMendesTheAlbum	11	#heatwave	56
#The100	12	#RHONY	57
#LHHATL	13	#HAHN	58
#Brooklyn99	14	#HappyBirthdaySachin	59
#NBAAwards	15	#Supergirl	60
#the100	16	#Origin	61
#WorldCup	17	#ComeyTownHall	62
#MTVAwards	18	#LoveIs	63
#WorldCupRussia2018	19	#SpaceForce	64
#BasketballWives	20	#TheBoldType	65
#saveshadowhunters	21	#ITrySoHardBut	66
#E32018	22	#DaytimeEmmys	67
#WednesdayWisdom	23	#Game7	68
#INDvAFG	24	#MLBDraft	69

#HeartMCSeokjin	25	#WWENXT	70
#NoTearsLeftToCry	26	#FortniteProAM	71
#TuesdayThoughts	27	#DDay	72
#MarriedAtFirstSight	28	#PoseFX	73
#PickUpShadowhunters	29	#Voicenotes	74
#MyHandleExplained	30	#GreenForGrenfell	75
#TonyAwards	31	#Riverdale	76
#TeenMom2	32	#ENGTUN	77
#EarthDay	33	#TheLightIsComing	78
#civility	34	#TheBachelerotte	79
#InfinityWar	35	#SoldAtDevilsYardSale	80
#WhateverItTakes	36	#WorldBloodDonorDay	81
#HappyBirthdayAriana	37	#TheRealityOfDepressionIs	82
#ThursdayThoughts	38	#AmericanIdol	83
#RHOBHReunion	39	#BTSxCorden	84
#TeenChoice	40	#GE14	85
#RHOA	41	#Congrats5SOS	86
#FIFAWorldCup_BTS	42	#90DayFianceHappilyEverAfter	87
#MITB	43	#Redemption18	88
#LoveIsland	44	#GlobalRunningDay	89

Table B.7: Hashtags used in experiment 90#.

Hashtag	**Label**	**Hashtag**	**Label**
#BETAwards	0	#FathersDay	50
#TrumpKimSummit	1	#StormHector	51
#NFLDraft	2	#DebateINE	52
#MondayMotivation	3	#ALLCAPS	53
#SaveShadowhunters	4	#LateLateShawn	54
#TheBachelorette	5	#WorldOceansDay	55
#MetGala	6	#IAmwayForward	56
#ARMYHiveStreamingParty	7	#BBNaija	57
#mprraccoon	8	#WorldBookDay	58
#Westworld	9	#GreysAnatomy	59
#PlayStationE3	10	#BMWMotorrad310	60

#WorldEnvironmentDay	11	#RHONY	61
#ShawnMendesTheAlbum	12	#heatwave	62
#SDLive	13	#HAHN	63
#The100	14	#HappyBirthdaySachin	64
#LHHATL	15	#Supergirl	65
#Brooklyn99	16	#Origin	66
#NBAAwards	17	#ComeyTownHall	67
#the100	18	#LoveIs	68
#WorldCup	19	#SpaceForce	69
#MTVAwards	20	#CAGovDebate	70
#WorldCupRussia2018	21	#TheBoldType	71
#BasketballWives	22	#ITrySoHardBut	72
#saveshadowhunters	23	#DaytimeEmmys	73
#E32018	24	#Game7	74
#WednesdayWisdom	25	#MLBDraft	75
#INDvAFG	26	#WWENXT	76
#HeartMCSeokjin	27	#TheProblemWithMeIn5Words	77
#NoTearsLeftToCry	28	#FortniteProAM	78
#TuesdayThoughts	29	#DDay	79
#MarriedAtFirstSight	30	#PoseFX	80
#PickUpShadowhunters	31	#Voicenotes	81
#StGeorgesDay	32	#GreenForGrenfell	82
#MyHandleExplained	33	#Riverdale	83
#TonyAwards	34	#ENGTUN	84
#TeenMom2	35	#TheLightIsComing	85
#EarthDay	36	#TheBachelerotte	86
#civility	37	#SoldAtDevilsYardSale	87
#InfinityWar	38	#WorldBloodDonorDay	88
#WhateverItTakes	39	#TheRealityOfDepressionIs	89
#HappyBirthdayAriana	40	#AmericanIdol	90
#ThursdayThoughts	41	#CaliforniaPrimary2018	91
#RHOBHReunion	42	#BTSxCorden	92
#TeenChoice	43	#GE14	93
#RHOA	44	#Congrats5SOS	94
#FIFAWorldCup_BTS	45	#90DayFianceHappilyEverAfter	95
#MITB	46	#ChicagoFire	96

#FridayFeeling	47	#Redemption18	97
#LoveIsland	48	#GlobalRunningDay	98
#FortniteE3	49	#BelowDeckMed	99

Table B.8: Hashtags used in experiment 100#.

C Experiment results for 30# to 100#

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.785	<u>0.776</u>	<u>0.778</u>
	Tf-idf	<u>0.792</u>	0.763	0.771
	GloVe-pre	0.339	0.268	0.244
	GloVe	0.603	0.556	0.559
	Word2Vec	0.595	0.575	0.575
SVM	BoW	0.753	0.748	0.749
	Tf-idf	**<u>0.793</u>**	**<u>0.792</u>**	**<u>0.792</u>**
	GloVe-pre	0.307	0.280	0.231
	GloVe	0.593	0.598	0.581
	Word2Vec	0.641	0.643	0.635
Log Reg	BoW	0.792	<u>0.761</u>	<u>0.773</u>
	Tf-idf	<u>0.793</u>	0.750	0.764
	GloVe-pre	0.315	0.308	0.291
	GloVe	0.626	0.618	0.615
	Word2Vec	0.661	0.646	0.651

Table C.1: Results obtained in Experiment 30#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.762	<u>0.748</u>	<u>0.751</u>
	Tf-idf	<u>0.773</u>	0.742	0.750
	GloVe-pre	0.282	0.221	0.197
	GloVe	0.571	0.515	0.522
	Word2Vec	0.56	0.545	0.545
SVM	BoW	0.735	0.722	0.726
	Tf-idf	**<u>0.776</u>**	**<u>0.769</u>**	**<u>0.771</u>**
	GloVe-pre	0.232	0.236	0.191
	GloVe	0.555	0.566	0.541
	Word2Vec	0.607	0.616	0.601
Log Reg	BoW	<u>0.775</u>	<u>0.744</u>	<u>0.756</u>
	Tf-idf	0.771	0.729	0.743
	GloVe-pre	0.266	0.264	0.248
	GloVe	0.593	0.589	0.586
	Word2Vec	0.645	0.633	0.637

Table C.2: Results obtained in Experiment 40#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.704	<u>0.692</u>	<u>0.693</u>
	Tf-idf	**<u>0.716</u>**	0.686	0.691
	GloVe-pre	0.211	0.186	0.166
	GloVe	0.481	0.44	0.438
	Word2Vec	0.521	0.512	0.508
SVM	BoW	0.678	0.668	0.671
	Tf-idf	<u>0.712</u>	**<u>0.711</u>**	**<u>0.710</u>**
	GloVe-pre	0.185	0.198	0.147
	GloVe	0.452	0.474	0.433
	Word2Vec	0.543	0.566	0.539
Log Reg	BoW	<u>0.722</u>	<u>0.692</u>	<u>0.703</u>
	Tf-idf	0.713	0.674	0.686
	GloVe-pre	0.221	0.223	0.204
	GloVe	0.502	0.499	0.492
	Word2Vec	0.587	0.584	0.582

Table C.3: Results obtained in Experiment 50#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	**Embedding Method**	**Precision**	**Recall**	**F1-Score**
MNB	BoW	0.728	<u>0.713</u>	0.717
	Tf-idf	<u>**0.743**</u>	0.709	<u>0.718</u>
	GloVe-pre	0.205	0.199	0.176
	GloVe	0.508	0.459	0.461
	Word2Vec	0.510	0.501	0.495
SVM	BoW	0.69	0.687	0.687
	Tf-idf	<u>0.735</u>	<u>**0.737**</u>	<u>**0.735**</u>
	GloVe-pre	0.225	0.211	0.161
	GloVe	0.478	0.498	0.463
	Word2Vec	0.55	0.567	0.543
Log Reg	BoW	<u>**0.743**</u>	<u>0.716</u>	<u>0.727</u>
	Tf-idf	0.738	0.696	0.71
	GloVe-pre	0.242	0.24	0.221
	GloVe	0.525	0.527	0.52
	Word2Vec	0.591	0.588	0.587

Table C.4: Results obtained in Experiment 60#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.738	<u>0.719</u>	0.723
	Tf-idf	<u>0.753</u>	0.717	<u>0.725</u>
	GloVe-pre	0.202	0.192	0.173
	GloVe	0.505	0.455	0.456
	Word2Vec	0.53	0.507	0.509
SVM	BoW	0.712	0.705	0.706
	Tf-idf	<u>0.751</u>	**<u>0.750</u>**	**<u>0.749</u>**
	GloVe-pre	0.179	0.197	0.147
	GloVe	0.501	0.494	0.469
	Word2Vec	0.561	0.572	0.553
Log Reg	BoW	**<u>0.756</u>**	<u>0.726</u>	<u>0.738</u>
	Tf-idf	0.751	0.706	0.721
	GloVe-pre	0.23	0.23	0.211
	GloVe	0.538	0.533	0.528
	Word2Vec	0.6	0.592	0.593

Table C.5: Results obtained in Experiment 70#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	**Embedding Method**	**Precision**	**Recall**	**F1-Score**
MNB	BoW	0.711	<u>0.686</u>	0.689
	Tf-idf	<u>0.725</u>	0.684	<u>0.691</u>
	GloVe-pre	0.191	0.16	0.138
	GloVe	0.468	0.418	0.416
	Word2Vec	0.496	0.481	0.480
SVM	BoW	0.685	0.675	0.678
	Tf-idf	<u>0.721</u>	<u>**0.721**</u>	<u>**0.721**</u>
	GloVe-pre	0.155	0.172	0.116
	GloVe	0.433	0.455	0.421
	Word2Vec	0.528	0.543	0.521
Log Reg	BoW	<u>**0.733**</u>	<u>0.698</u>	<u>0.710</u>
	Tf-idf	0.716	0.675	0.687
	GloVe-pre	0.203	0.206	0.182
	GloVe	0.498	0.489	0.484
	Word2Vec	0.566	0.563	0.561

Table C.6: Results obtained in Experiment 80#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	**Embedding Method**	**Precision**	**Recall**	**F1-Score**
MNB	BoW	0.715	<u>0.692</u>	0.696
	Tf-idf	<u>**0.731**</u>	0.690	<u>0.699</u>
	GloVe-pre	0.178	0.159	0.138
	GloVe	0.487	0.419	0.422
	Word2Vec	0.509	0.49	0.489
SVM	BoW	0.682	0.676	0.677
	Tf-idf	<u>0.721</u>	<u>**0.721**</u>	<u>**0.721**</u>
	GloVe-pre	0.152	0.167	0.115
	GloVe	0.449	0.456	0.425
	Word2Vec	0.527	0.547	0.523
Log Reg	BoW	<u>**0.731**</u>	<u>0.702</u>	<u>0.713</u>
	Tf-idf	0.719	0.678	0.689
	GloVe-pre	0.202	0.198	0.176
	GloVe	0.495	0.488	0.482
	Word2Vec	0.573	0.57	0.568

Table C.7: Results obtained in Experiment 90#. The <u>underlined</u> numbers are the highest values obtained in each model while the **bold** ones are the best in all models.

Model	Embedding Method	Precision	Recall	F1-Score
MNB	BoW	0.705	0.675	0.68
	Tf-idf	**0.723**	0.672	0.683
	GloVe-pre	0.168	0.154	0.134
	GloVe	0.484	0.412	0.415
	Word2Vec	0.497	0.476	0.476
SVM	BoW	0.674	0.663	0.666
	Tf-idf	0.712	**0.711**	**0.711**
	GloVe-pre	0.14	0.16	0.108
	GloVe	0.436	0.45	0.416
	Word2Vec	0.512	0.53	0.505
Log Reg	BoW	**0.723**	0.689	0.702
	Tf-idf	0.71	0.666	0.678
	GloVe-pre	0.19	0.194	0.172
	GloVe	0.496	0.484	0.481
	Word2Vec	0.557	0.556	0.553

Table C.8: Results obtained in Experiment 100#. The underlined numbers are the highest values obtained in each model while the **bold** ones are the best in all models.