UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF A CLASSIFICATION SYSTEM OF POLITICAL MESSAGES IN SOCIAL NETWORKS USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING TECHNIQUES

CARLOS GUILLERMO BERMÚDEZ CRUCES

2018

TRABAJO DE FIN DE GRADO

Título:	Diseño y desarrollo de un sistema clasificador de mensajes políticos en redes sociales basado en técnicas de Natural Language Processing y Machine Learning
Título (inglés):	Design and development of a classification system of political messages in social networks using Natural Language Processing and Machine Learning techniques
Autor:	Carlos Guillermo Bermúdez Cruces
Tutor:	Óscar Araque Iborra
Ponente:	Carlos A. Iglesias Fernández
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE GRADO

DESIGN AND DEVELOPMENT OF A CLASSIFICATION SYSTEM OF POLITICAL MESSAGES IN SOCIAL NETWORKS USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING TECHNIQUES

Carlos Guillermo Bermúdez Cruces

Enero de 2018

Resumen

Con el rápido aumento de la tecnología en la vida cotidiana, las redes sociales se han convertido en una herramienta muy importante para los políticos, ya que les permite captar futuros votantes. Sin embargo, en el lado del ciudadano, la gran cantidad de información recibida puede producir la pérdida de perspectiva.

Este proyecto se centra en el diseño y desarrollo de un sistema clasificador de mensajes políticos según su sesgo, clasificándolos en partidistas o neutrales. Para el desarrollo se ha usado el dataset Political-Social-Media importado de la plataforma Kaggle. De esta manera, el sistema es entrenado con mensajes de Facebook y de Twitter, por lo que una vez desarrollado será capaz de predecir el sesgo de nuevos mensajes de estas dos redes sociales.

Para el diseño del sistema se han usado dos soluciones diferentes: el modelo Bag of Words, y un sistema básico de redes neuronales basado en Word2Vec. Además, se ha explorado una tercera solución formada por la combinación de las dos anteriores.

Finalmente, el sistema es testado con un conjunto de datos diferente al empleado para el entrenamiento. Una vez seleccionados los mejores modelos para Facebook y Twitter, el resultado se implementa en un plugin. Para esta tarea se utiliza Senpy, que permitirá la clasificación de nuevos mensajes en tiempo real.

La última parte del proyecto consistirá en extraer las principales conclusiones del trabajo, así como recopilar las nuevas tecnologías empleadas, con el objetivo de destacar futuras líneas de trabajo.

Palabras clave: Twitter, Facebook, Senpy, Sesgo político, Word2Vec, Python, NLP, Machine Learning.

Abstract

With the progressive increase of technology in the daily life, the use of the social networks has become into a relevant tool for politicians in the aim of catching new possible voters. However in the side of the citizen, the high amount of information can produce a lose of perspective.

This project is focused on the design and development of a system able to classify political messages according to their bias into partisan or neutral. For the development of the system it is used the dataset Political-Social-Media imported from Kaggle. Thus, the system is trained with Facebook and Twitter messages, and will be used to predict new samples of these social networks.

For the design of the system were used two different approaches: The Bag of Words model and a basic Neural Network system. In addition to this, it is also developed a third solution based on the combination of the two previous.

Finally the system is tested with a different set of samples, used only to verify the behaviour of the system. Once are selected the best models for Twitter and Facebook, the final result is implemented in a plugin. For this task, it is employed Senpy that will allows to classify new samples in real time, as well to share the system in an easier way due to the use of linked data.

The last step of our project is to extract the main conclusions of the work and join them with the new technologies learned, in order to stipulate possible future lines of work.

Keywords: Twitter, Facebook, Senpy, Political bias, Word2Vec, Python, NLP, Machine Learning.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This chapter will provide a quick analysis of the current use and effects of the social networks in relation with the political messages, as well the definition of the main tasks and methods employed to develop this project.

1.1 Context

With the progressive increase of technology in the daily life, the use of the social networks has turned into something indispensable for politicians, since it allows to transmit the desired message to a larger number of people, beside trying to catch future voters. In fact, there are reviews like the Annie Hellweg [17] where is studied the impact that Twitter and Facebook have in the intention of vote. In this study, the author discovered that there is a relationship between the content published by politicians and the voters' intention, being clear the importance of this two social networks in the labour of capting voters.

However in the side of the citizens, the high number of speeches, messages, meetings, and others similar actions can cause a feeling of saturation, specially at elections time. This could lead to possible manipulation and deception on the part of the informants. This raises the need to apply certain filters of neutrality to this information, so that the user is

allowed to know if certain message is trying to influence in his vote, and in this way enjoy an experience as objective as possible.

Due to this fact, the use of Natural Language Processing in conjunction with Big Data and Machine Learning, have become very relevant, because it can extract big amount of textual information from social networks for its posterior analysis. In particular, this work will be centered in the analysis part. According to this, the main purpose of this project is to use this NLP techniques with Machine Learning classifiers to design and develop a system that allows to analyze Twitter or Facebook messages written by politicians. The analysis consists mainly in the process of classifying this messages according to their bias. The bias of the message can be split in neutral messages, that are those messages with no hidden intentions, and partisan when refering to those messages which try to produce induced feelings towards some political party or ideology.

Summarizing, this document details the design and the development of the system, which mainly will filter political messages according to their bias, helping citizen to be more aware and less susceptible to manipulation.

1.2 Project Goals

The principal goal of this project is to design and implement a Machine Learning system, which is able to detect and classify different political messages according with the neutrality of the bias. These messages are extracted from social networks like Twitter or Facebook. In this way, focusing on the origins of the messages, smaller objectives can be highlighted. This sub objectives are the natural continuation of the principal.

One of them, is the characterization of the probability distribution of words according to their neutrality and objetivity. For this goal the aim is distinguish those words that are commonly employed in the different contexts, like in the partisan messages in Twitter and Facebook.

As another secondary objetive, in relation with the previous, is to profile the use of social networks in relation to their different use in the political context, analyzing the most relevant characteristics of each one.

This project will mainly focus on the different classification techniques, according to the main aim, trying to classify and analyze correctly the results and optimizing the classifiers with the most relevant features getting in this way the correspond conclusions.

1.3 Project Tasks

The main tasks to be performed in this project are described in the follow list:

- Study the state of the art in relation to the application of Natural Language Processing technologies, applied to the domain of political messages.
- Perform an analysis of system requirements to be developed.
- Taking into account intrinsic aspects of NLP such as linguistic resources, different algorithms and techniques.
- Design the system to be used, and based on the results of the previous task implement the NLP system.
- Validate the developed system, and finally test it with different samples.
- Integrate the system into a plugin so that can be used to classify new political messages in real time.

The language used for the programming will be Python, due to its facility of use and its adequacy to the proposed project. Furthermore, due to its libraries, Python is one of the most used languages currently for the field of the Machine Learning, Big Data and NLP. In this project the following libraries will be used: scikit-learn, pandas, numpy, scipy, as well as others focused on text processing such as NLTK or TextBlob. We also are going to use the library gensim in order to obtain a more complex system.

1.4 Structure of this Project

This section will indicate the basic structure that this document will have, adding a small resume of each chapter.
Chapter 1: It provides the context of the project as well the problem and tasks that will be executed in order to solve it. In particular, it shows an overview of the uses of the social networks in the political field. Furthermore is given a deeper description of the project, including its objectives and methodology.
Chapter 2: It shows the background of the study field as well the current state of the most relevant technologies used in this project.
Chapter 3: This chapter will detail the main architecture of the system, including an overview of the initial system and its architecture once it is implemented into a plugin.

Chapter 4: In this chapter are explained the requirements followed to perform the experiments as well the analysis of the results of each experiment.

Chapter 5: This last chapter provides the main conclusions extracted from the project. Furthermore, it is also given the main problems found during the develop of the project and future lines of work.

CHAPTER 2

Enabling Technologies

This second chapter will provide us an overview of the related technologies used in the develop of this project, as well the most relevant background from which we can extract some conclusions.

2.1 Introduction

This project will focus on the use of two emergent technologies: Machine Learning and Natural Language Processing (NLP). Furthermore, indirectly it also will be treated the term of Big Data that will be explained with the two mentioned then.

NLP [16] is the field that covers any kind of computer manipulation of natural language. This technology will allows to analyze different texts for its study. For this analysis, NLP gives the possibility to extract features from these texts, transforming them into numerical values. The final objective is to integrate the data of NLP in Machine Learning systems in order to evaluate the results obtained and to develop a classifier system in this case. It is relevant to say that there is another concept that couple in a great way with the two previous mentioned. That is the Big Data that will provide a high number of examples or amounts of data. Thanks to this apport, Machine Learning systems are benefited because

for classification, these systems requires high amount of training data. In particular, Big Data is related with our project due to the high number of samples that we will process. In general, having more data means that the classifier will do a better work, because it receives a better train improving the capacity of generalize, and also better test, improving also the evaluation of the results. For all that reasons, in this project these three concepts are correlated and will work together.

For the implementation of these technologies, Python will be used. Python [19] is an object-oriented programming and one of the most famous and important programming languages in the world of Machine Learning [22]. The principal reasons of this relevance are their simple and optimist use, an its facility to add different modules or libraries. In fact, this language is on the top of high-level programming languages. Its main characteristic is the bigger number of libraries that provides, in comparation with other laguages like *R* also used in this field. Particulary, the most useful for this project, as well other relevant tools, are described in the follow sections.

2.2 Python base libraries

Between the high number of libraries that Python has, it is necessary to highlight the labour of these specific libraries due to its importance in this project and in the develop in Python in general.

- ***Scipy:***

  Scipy [11] that is a library of tools and algorithms. The main advantage that provides is the the high-level commands and different classes with the aim of visualizing and treating data. It can be used among others, to join the features of the different feature extraction methods.

- ***Pandas:***

  Pandas [9] is the library which provides high-performance in Python, and makes more simple the use of data analysis tools and structures of data. The basic data structures that Pandas provides are:

  * **Series**: A one-dimensional ndarray composed by an index and the corresponding associated value. This value is stored in this structure and can be an array, a dictionary, a scalar value or even Python objects.
  * **Data-Frames**: It is the primary and two-dimensional Pandas data structure. In general, it is similar to a tabular data, having rows and columns with different

parameters and with heterogeneus content (Python Objects, raw text, etc.) As an example of definition, it can be thought of as a dict-like container for Series objects.

The relation between this defined structures and this project is close because the need to convert the initial dataset that is in CSV format into a DataFrame in order to start to work with data. After the DataFrame is maked, Pandas will also be useful for data managament tasks such as grouping, aligment and split or joining datasets.

- ***Matplotlib:***

  Matplotlib [6] is a Python two and three dimensional plot library, that generates plots, histograms, power spectra, bar charts, errorcharts, scatterplots and other important resources in a simple way to the developer. This resources can be useful in order to visualize the properties of the analyzed data.

- ***Numpy:***

  Numpy [7] is the principal package for scientific computing with Python. It allows to our system to have and manage N-dimensional array object. It is the baseline of rest of libraries.

2.3 Python NLP libraries

In addition to the main libraries of Python, there are others more specific and related with our project purpose. This libraries are oriented to the work in the field of NLP. The most relevant for us are described then:

- ***NLTK:***

  The Natural Language Toolkit [3] is a Python package for Natural Language Processing, which will provide us different useful tools and libraries related to this field. Some examples of the contributed tools are tokenization, stemming , lemmatization and tagging. In particular, some of the multiple resources that offers are:

  * Processing raw-text.
  * N-grams.
  * Lexical analysis.
  * Part-of-speech tagger.

- ***Scikit-learn:***

  Although Scikit-learn [10] is not an NLP exclusive library, it is included in this section due to the high number of tools that provides for the task of NLP. It is built on NumPy, SciPy and Matplotlib, and is the library which will provide the different machine learning algorithms. The main problems related to our project that can solve are classification, dimensionality reduction, model selection and preprocessing.

  * **Classification**: Scikit-learn provides a lot of different classifiers, some of them explained in follow sections. The final aim in this part is to identifying the category that each sample belongs. Continuing with this stage, Scikit Learn will provide also the necessary metrics to evaluate the work that each classifier do.
  * **Dimensionality reduction**: consists mainly in reducing the number of random variable to consider. This is done in order to eliminate the possible noise that some features or variables contribute in the task of classification. This means to choose only those features that really provide information to the classifier, gaining in this way in efficiency.
  * **Model selection**: the main purpose is to perform an approach of the parameters of the classifier with the aim of choose the most adequate to the train stage.
  * **Preprocessing**: allows the feature extraction. It is the stage in which the input data in raw format like text, is transformed in a way so that it can be used by machine learning classifiers.

  In order to make the classification a serie of steps must be followed. The general scheme of this steps is shown in the Figure 2.1. We will explain more in detail these steps with other relevant requirements in the section Experimental setup. Another important technology that Scikit-learn provides is the TF-IDF, that will allow us to transform words into vectors, and in this way allows analyze text due to the classifier receives number. This is the base of the NLP. In the section of feature extraction the TF-IDF is explained in detail.

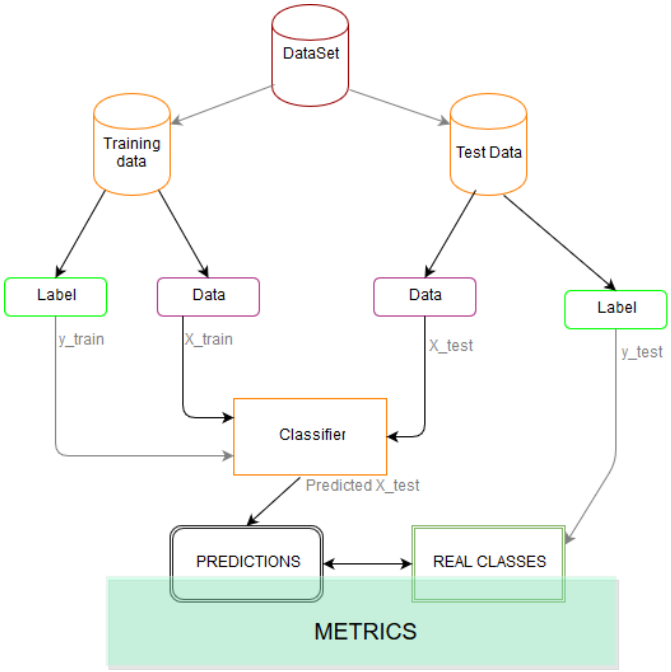


Figure 2.1: Basic classification steps

- ***TextBlob:***

  TextBlob [13] is a Python library focused on the field of NLP. The main tools that it provides are Part-of-Speech tagging (POS), n-grams, language translation and detection or spelling correction among others. In particular for this project we will use POS, with the purpose of feature extraction.

- ***Vader Sentiment Analysis:***

  Vader Sentiment Analysis [14] is a lexicon and rule-based sentiment analysis tool. The main funcionality of this implementation is to extract the polarity of each text. We will employe this analyzer due to its flexibility working in a good way with text of different sources and contexts.

- ***Gensim:***

  Gensim [4] is an open source Python library built on Numpy and SciPy, focused on semantic topic extraction. It allows to process raw texts. In this way, this library revolves around the terms of corpus, vectors and model as principal objectives.

  For this project, Gensim will provide us with the necessary functionalities for using Neural Networks, later explained deeper. In particular, we will use Word2Vec, a model centered in the transformation of words into vectors and detailed later in the section 3.5.2, where is given the scheme of how it is implemented in the system.

2.4 Senpy

Senpy [12] is a framework developed by the GSI at the ETSIT UPM, that implements linked data services of sentiment and emotion analysis. From [12] we can analyze its modular architecture, shown in the Figure 2.2. It is divided in two different modules: the Senpy core, which is the basic component of the system, and the Senpy plugins, that implement the variety of available algorithms. It is based in the ontologies Marl, NIF and Onyx. In this project we will use the first one in order to obtain semantically annotated results. In fact, Senpy can support different formats like XML-RDF, JSON-LD and Turtle. The main functionality of Senpy is that it allows to implement different algorithms offering a common interface, as well to provide common services with purpose of facilitate the deployment and make easier the task of developers. We will use it in order to implement the final system into a plugin, and have easier a way to share it.

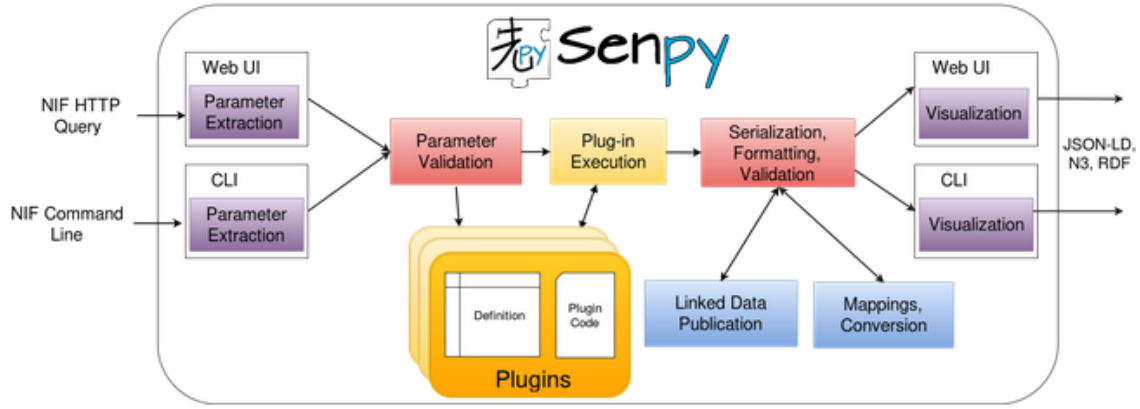


Figure 2.2: Senpy Architecture

2.5 Related background

In this project, the dataset used for training and evaluating the learnign models is Political Social Media Posts [3]. To the extent of our knowledge, there has not been any previous work on detecting partisan and non-partisan messages that use this specific dataset.

This means there are some disadvantages because the results will not be contrasted directly with any others, and there is no conclusions that could be used. In this way, the first results and conclusions about this data will be extracted in this project.

Despite the fact that there are no previous works in this dataset, there are other similar

studies that could contribute with this work. We will divided them according with the kind of way they followed to approach the problem.

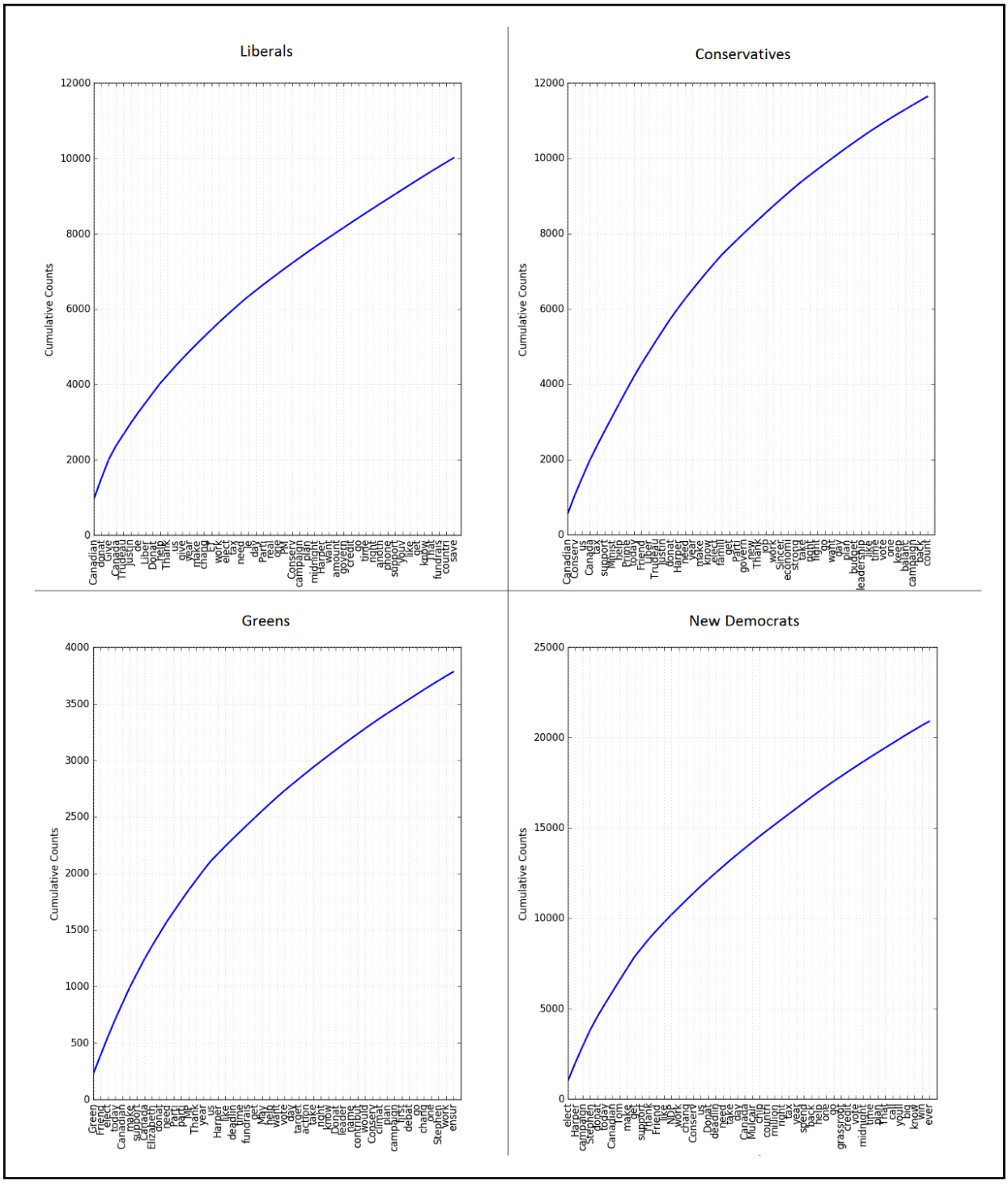
2.5.1 Political bias with Bag of Words models

Taking these works into account an interesting source is the study of the bias of the political messages in Canada that has a quite similar perspective. The canadian study [1] consists in analyze political messages sended by email and classifying them in bias states. In this study, the aspect of the key words have an important value. In fact, they use the most common words to distinguish the class of the sample, it explains between the different parties.

They also explode the classification of those messages that has as principal aim the donation of the reader. In this case, they experiment with the lemma *donat*, that was the most common in the messages with this intention, obtaining very good results in the accuracy. This last experiment can seem very simple, however these two demonstrations can induce us to create methods that can filter by words, in order to cover these words more employed in partisan texts, according in this way with the importance of each one at the time to classify the samples.

The Figure 2.3 shows how the words grouped by lemmas are segmented according to their class. Applying this idea could be a good strategy in order to characterize the different partisan or neutral vocabulary. In fact, this will be the base of our partisan-neutral detector explained in its corresponding section.

Figure 2.3: Most common lemmas by political party



However, it is necessary to say that there are some differences between this project and our project. The main difference is the context, because this dataset contains canadian messages from canadian politicians. It is also important to contrast the source of the messages, email in the developer contribute against our Twitter and Facebook messages. This is relevant due to our message average length will be lower, contributing in this way our samples in a smaller proportion in order to do the classification.

2.5.2 Political bias with Neural Networks

One example of that is the study found in [21]. In this study, they use the IBC [5] as well the OTI [8] dataset. Both datasets consist in political speechs compiled from the two biggest parties in the USA. In particular the Ideological Book Corpus (IBC) is one of the most employed datasets in this task, and consist in near to 4000 sentences annotated from political ideology and divided in liberal, conservatives and neutral sentences. They used these datasets with the aim of develop a system that classifies texts depending of the bias. In order to take a possible comparison of results, they obtain a F1-score of 0.71 using Neural Networks.

However, as the author of the work [21] states, in general "previous work on bias or ideology detection from a textual document is relatively rare in literature". Having this in count, we are going to try to extract the main ideas of these few works in order to have a base in our project.

Other similar work has developed by Mohit Iyyer [18], where the author uses again the mentioned IBC and the Convote dataset. The Convote dataset [2] is composed by the transcripts of US Congress floor at 2005, labeling each speaker with its corresponding political party. In this work they demostrate that applying Neural Networks they obtain a better aproach that with classic models of Bag of Words. They obtain an accuracy of 70.2% on the Convote dataset and 69.3% from the IBC dataset.

From these two works we can extract the idea that Bag of Words models although they have a fairly good performance, they can be improved or replaced by more complex techniques like Neural Networks in this context.

CHAPTER 3

Architecture

This chapter has as objectives to explain the general architecture and the contents of the political messages classifier system, since the data is in its initial stage, to the step in which the classifier takes the processed data in form of vectors and then it is implementated as a plugin.

3.1 Introduction

This sections provide a basic scheme of how the data is porcessed. In general, since the data is in raw format are required a serie of steps to obtain the classification:

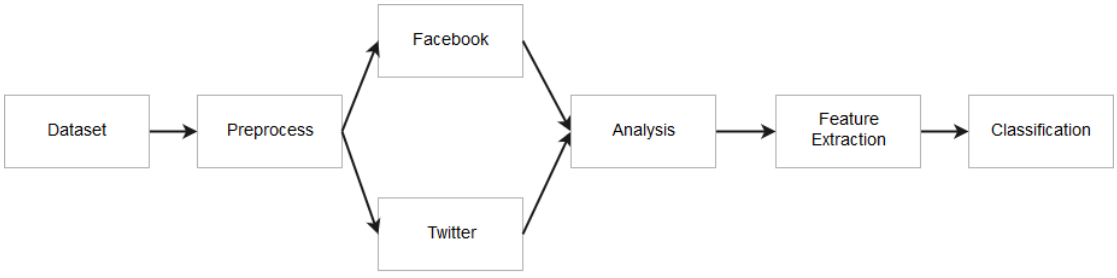


Figure 3.1: Main stages of the data

In the next section we will provide a more detailed description of the general architecture that composes the classifier.

3.2 Overview

In this section we are going to describe the main architecture of the system. As we establish in the figure 2.1 the dataset is divided mainly in training and test sets. Continuing with this idea we are going to analyze more in detail the architecture of the classifer as we can appreciate in the Figure 3.2. The left part of the model represents the extraction of the

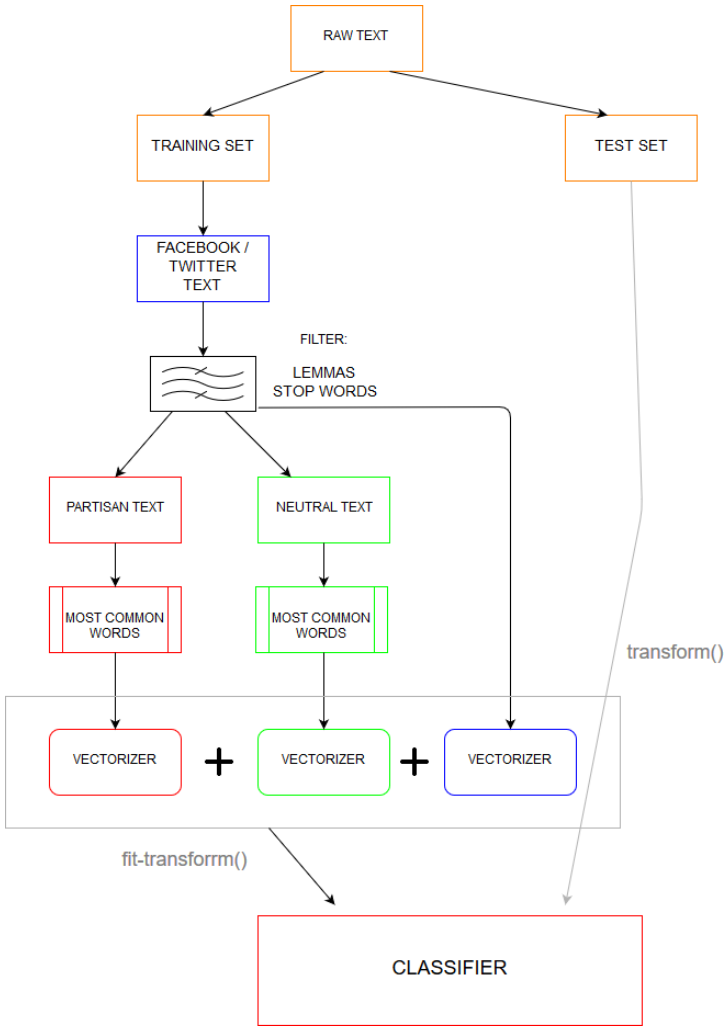


Figure 3.2: General classifier architecture

dictionary, and the blue vector a generic feature. Each extracted feature will be represented in the system by an object in form of vector, so we will need to join this features in an unique object, for the posterior analysis of the classifier. This model will be implementated

as a plugin in Senpy, in the way that the input message is received by Senpy and analyzed following this structure.

3.3 Preprocess

This is the first step by which our data will go through. The current dataset is downloaded from Kaggle [3] in CSV format, but for starting to work with it is needed to convert it into a DataFrame. The library *Pandas* explained in the section 2.2 will do this task, reading the CSV file and converting it into a DataFrame object, allowing in this way to analyze the content of the data.

As we can see in the Table 3.1 there are errors in the decoding process in the raw data. Once the data is readed, this unreadable symbols will have different codifications as **\x86** or **&amp** between a huge number of others. We will try to remove them due to they don't apport information.

Facebook	**Twitter**
By delaying another component of his signature law for a select few, President Obama has yet again confirmed what we&#65533;&#65533;ve long said &#65533;&#65533;&#65533; his health care law is unworkable and harming American families.	What a wonderful night at State Senator Ken Yager&#65533;&#65533;s Chili Supper benefitting the Boys & Girls Clubs of Roane County. http://t.co/DsyoAFZvkR

Table 3.1: Raw text samples

In the next section we will see the most relevant characteristics of the dataset employed, as well as the different techniques and transformations applied on it. However, before perform the analysis, we will remove the stopwords in order to get a clearer results of our data. Stopwords are these words that in general don't apport information due to they have an empty meaning. Examples of this words are prepositions, adverbs, or articles.

3.4 Analysis of the data

The dataset contains several columns with differents features related to political messages generated by Twitter or Facebook. Particularly, the dataset have 5000 instances, with different parameters. In general, we are not going to use most of them due to they can't

apport information to our proposal. Examples of this columns are simple identificators dates, in which the Tweet or Facebook message was collected (not published), and other similar features that are irrelevant. With this in mind we will use the fields explained in the table below, due to they are the most useful for our purpose.

Column	**Meaning**
Source	The origins of the message from which the dataset will be split : Twitter or Facebook .
Text	Represents the Tweet or Facebook message.
Bias	Can be neutral if the author of the message is not trying to influence in a political way on the reader, or partisan in the other case.

Table 3.2: Relevant Fields of the Dataset

We have to take in count that the classifier will need numbers in order to classify samples. However, our target class is not numerical so we will apply the next transformation:

Bias	**Final result**
Neutral	Will be the negative class so it is assigned the value “0”.
Partisan	The partisan class will be the possitive, so it will have the “1” value.

Table 3.3: Transformation to numerical values

Once we have selected the target class and the main fields, we proceed then to split the main dataset into two smaller datasets, with the aim of separate messages according to its source into Facebook and Twitter, and in this way analyze and treat them separately. This means each social network data will be processed by different classifiers.

3.4.1 Facebook

The Facebook dataset is composed by 2500 samples, that correspond to messages collected through this social media network, with the fields explained before. Taking into account that the bias is the target class, we are going to analyze its distribution in order to have a first approach to the problem. We can see the proportion of Neutral and Partisan messages of Facebook in the pie diagram below:

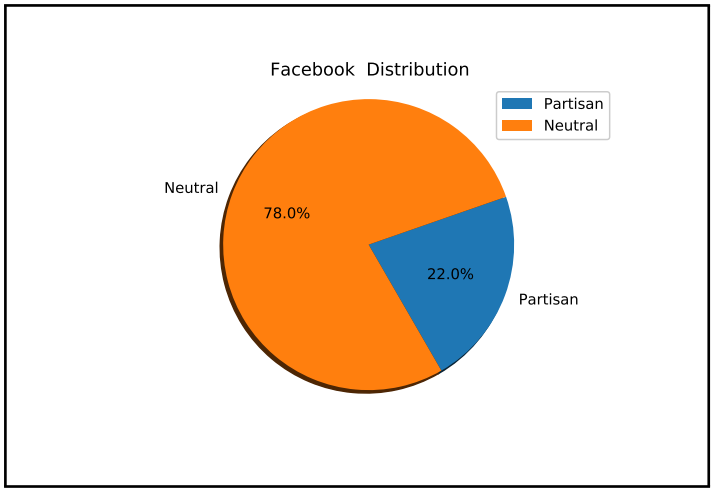


Figure 3.3: Facebook Samples Distribution

We can appreciate that most of the messages belongs to the neutral class. In fact the are near to 80% of neutral messages in the dataset, that means that are 3.5 times more neutral speeches than partisan. As a first solution we can think in performing undersampling with the neutral messages, however, this would mean to lose training data, so we are going to explore other possibilities during the project. We are going to analyze now the most common lemmas depending of the bias class. We can see them in the Table 3.4 .

Bias	Lemmas							
Neutral	today	wa	great	american	state	day	year	nation
Partisan	law	budget	presid	congress	hou	ha	vote	work

Table 3.4: Bias characteristic lemmas in Facebook

As we can see there is a distinction in the top lemmas of each label in Facebook. We will use this characteristic in the feature extraction step. We are comparing lemmas in order to avoid that set of words with the same lexeme, like for example donation and donate, have different counts and the final number of each one becames lower, being this less representative to our analysis.

3.4.2 Twitter

The Twitter dataset contains 2500 samples. This samples correspond with Tweets so they have 140 characters as maximum. As we did in the Facebook part, we are going to analyze

how this samples are distributed according to the bias.

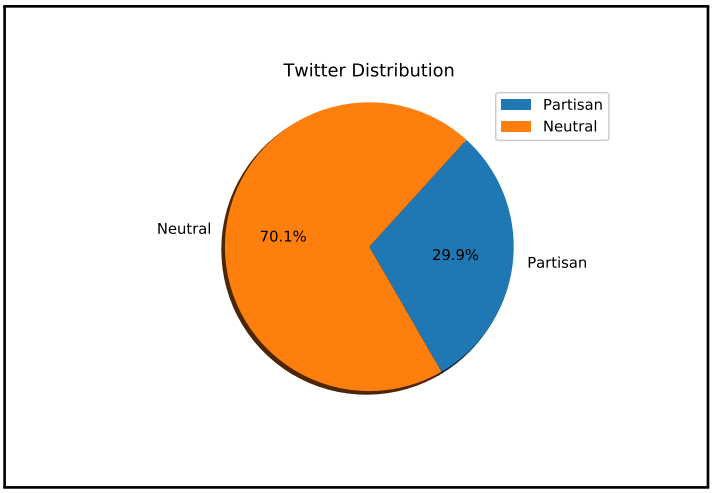


Figure 3.4: Twitter Samples Distribution

We can appreciate that this time the dataset is a bit more balanced that in the case of Facebook. However there is still an important umbalance between the Partisan and Neutral messsages, representing this last more than the 70% of samples.

As we can see in the Table 3.5, the most common lemmas are still different except the lemma today that appears in both bias. We have to say that for this task we also include hastags and mentions, eliminating their corresponding previous symbols. This can explain that the most common lemma in partisan is *obamacar* that comes from obamacare.

Bias	Lemmas							
Neutral	today	thank	great	work	join	new	discuss	support
Partisan	obamacar	hou	today	vote	bill	job	american	ha

Table 3.5: Bias characteristic lemmas in Twitter

3.4.3 Profiling Social Networks

As a quick summary of main characteristic of these two social networks, we analyze the main differences found and extract some conclusions of its general use. Observing the percentage of political messages, we can conclude that Twitter is a social network more used with political aims than Facebook. Furthermore, it seems to be more direct with the readers than Facebook with lemmas like “bill” or “obamacar”, where in Facebook are employed

related terms more elaborated like “budget”.

Finally we also notice that both of them are usually employed with informative aims, due to the high number of neutral messages and the appearance of lemmas like “Today” in both Twitter and Facebook messages.

3.5 Feature Extraction

As it is shown in the general architecture, a process to extract features from the data is required. Considering that, this part will explain the design of the feature extraction module and the main extracted features.

For the text features, the TF-IDF will make the extraction. The Term Frequency-Inverse Document Frecquency represents the importance of each word in the set of documents, providing in this way statistics about the frequency in which the words appears on the document. It is composed by the IDF:

$$IDF(t, D) = log\frac{|D| + 1}{DF(t, D) + 1} \tag{3.1}$$

and by the TF, being the final result:

$$TFIDF(t, d, D) = TF(t, d) * IDF(t, D) \tag{3.2}$$

Where the D represents the number of documents in the corpus, t each term and d each document.

Another important concept are the n-grams. This is referred to the idea of not extract only words as features but extract also couples of words or even higher sets. With this in mind, each n-gram will represent only one feature in the classifier. Each of these features of n-grams will be composed by a certain number of words that are placed in a successive way. Thus, the n-grams can be formed by uni-grams, bi-grams, tri-grams,etc. Knowing this basic concepts, the extracted features are resumed then:

- **Lemmas:** We will use the TF-IDF in combination with a tokenizer, with the aim of extract the lemma of each word. With this we avoid the situation where words with the same lexeme take more than one feature. For example, the words *educational* and *education* will be treat as the lemma *educat*. This feature will be extracted throught n-grams. Finally we have to take into account the number of features. We select it in 300 features, and also an n-gram size limited to 3 in order to not to learn phrases and avoid overffiting.

- **Hastags and mentions:** The number of hastags or mentions to another user that contains each tweet or facebook message. They are represented in each sample with the symbols "#" and "@" respectively.

- **Re Tweets:** The number of re tweets for the Twitter messages. They are represented by the symbol "*RT*".

- **Exclamations and interrogations:** As it is possible that partisan texts show more enphasis or retorical questions, we will extract the number of exclamations or interrogations that each message has.

- **Links:** The number of URLs or links to web pages that each sample contains.

- **Syntactic structure:** The Part Of Speech indicates the category to each word belongs according with its syntactic functions. It can be nouns, adjectives, verbs, etc. An interesting characteristic is that this method can be merge with other tools like n-grams in order to extract the most common syntactic structures. Thanks to the use of TextBlob we can get additionally more specific structures due to it classify verbs in categorical values.

- **Elongated words:** In the English language doesn't exist any word with a character repeated three times consecutively. We will to take advantage of this to detect elongated words as can be the word "Okaaaay".

- **Sentimental categories:** This feature extracts the general polarity of the sample. At the same time it is divided into another three features, which will express the value in form of range between '0' and '1'. This features are:

  * ***Positivity:*** Expresed as a float number that is limited between '0' and '1'. This will represent the level in which the sentence is positive, being the '1' the maximum value.

  * ***Negativity:*** This feature indicates the negative value of the sentence. Same as in the positivity case, this parameter variates between '0' and '1'.

  * ***Neutrality:*** The neutrality express in which proportion the text has neutral inclination. It is representated as the previous are, having in consideration that the sum of this three parameters is equal to one.

3.5.1 Partisan - Neutral Dictionary

It is something usual in NLP, specially talking about Bag of Words models, identify the most relevant features once the data is in vectors form. The objective is to improve them and performs a better classification.

As we could see in the previous explained features, there are a great number of them that can apport an extra differential value to the system. However, due to the fact that the input data to anlayze is composed by text, most of features are formed by lemmas, so they will have a higher impact that the most of the rest of extracted features. Due to this fact, it is logical to take especial attention into this features and try to improve its extraction.

For that reasons, and how was taking the idea introduced in the section 2.5.1 and in particular from [1], one way to improve this features will be to make a domain-adapter dictionary of partisan-neutral terms, in order to adding it to the rest of the features. The main aim behind this idea is to provide to the classifier a train with more partisan words. As we have seen, neutral samples are more common and therefore it is more probable that withouth the dictionary exists vocabulary composed by neutral samples. In this way, we are going to have the same words of neutral and partisan texts.

As two datasets are provided, there will be also two dictionaries, one for Twitter and another for Facebook. Each of these dictionaries will be created from the training data. Particularly, this dictionary will consist in a list of most popular words of the data, in order to help the classifier adding at the same time these relevant features. The first step to create each of these dictionaries is to divide the data into training and test data avoiding by this way make false predictions.

It is relevant to say that the data have to be divided in text format, in order to allow the next step, that is create a vocabulary of most common words. However, like the data is in text format a preprocessor step is required, as with the normal extracted features. In this preprocess we will include the lemmatization of the words, in order to include lemmas as the features of the dictionary.

Once preprocess is done, the new vocabulary is extracted. For that, the dataset will be split again, this time into partisan and neutral texts. Already with this two smaller datasets, both vocabulary are formed by the most common 50 lemmas of each dataset. The next step is to extract these features. To make this, will be used the Count Vectorizer. This time, passing it as parameter the vocabulary created. Now the features are ready to be extracted, so the final stage is to merge these vocabularies in order to join two smaller dictionaries, the neutral most common lemmas and the partisan most common lemmas.

Finally, the result is a dictionary composed by 50 + 50 features, that is added to the rest of the features, those extracted previously from the text.

3.5.2 Word2Vec

All the extracted features we have seen consist basically in the Bag of Words models. However, to obtain a better approach to the problem we are going to follow the studies related with the political bias in which they use Neural Networks to improve their results. In other words we are going to try to analyze also the context. Under this idea we will use Word2Vec.

Word2Vec [15] is a two-layer Neural Network. It works taking as input a corpus of text having as output vectors, so it produces word embeddings. The two mains methods this technology follows are the CBOW and the Skip-gram . Their general architecture [20] is described then:

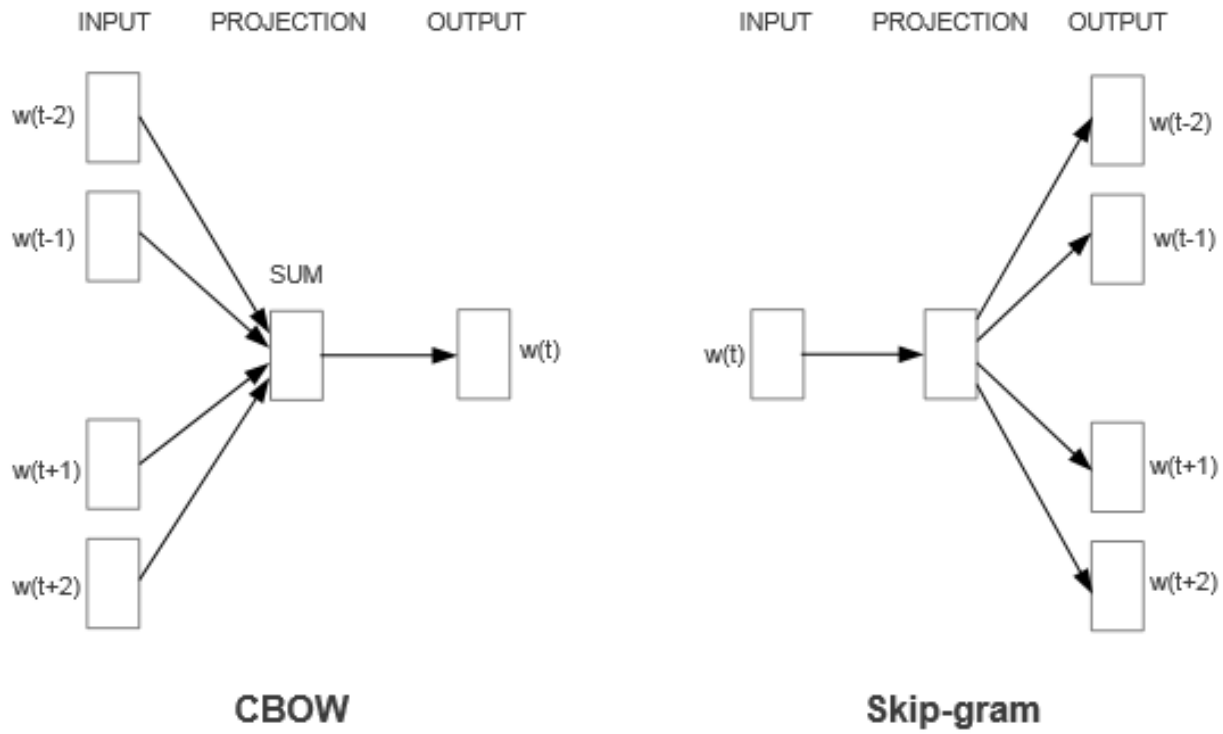


Figure 3.5: Word2Vec methods architecture

In the figure, W(t) represents the current word, and W(t-1) or W(t+1) those that are more near to the current. As we can see, the main difference between the two methods is

that in the CBOW, the principal aim is to predict a word knowing the context, that is the words that are around. On the other hand, in the Skip-gram, the objective is the inverse. It consists in given one word, try to predict those words that are close related.

Having in count the characteristics of this work, we are going to employ the CBOW technique. We are going to analyze more in detail this method. The first task we have to do is to construct the inputs of the future classifier. We will consider the average of all of word vectors in our messages. In this way, we will have the word vectors and every text message will be represented by a 300 dimensional vector. Each of the words represented in the figure are composed by vectors. Going into more detail, we analyze the whole process that happen to obtain the final features. As first approach we have that W_i representing a general word, will be transform into the V_i following the next formula:

$$V_i = W_i^T * E \tag{3.3}$$

In this equation, the variable E represents the **one-hot encoding**. This encoder thus, will transform each word of the sentence into a vector of d dimensions. However this vector V_i is not yet our final vector. To obtain the vector that we are going to use it is necessary to apply the next formula:

$$V_f = \frac{1}{M} \sum_{i=1}^{M} V_i \tag{3.4}$$

This equation 3.4 basically performs the average of all the V_f vectors so we finally obtain V_f. This vector will be the one we are going to use as features, and as the predecesor vectors it has a dimension d, that means it will provide d features.

3.5.2.1 Domain-Adapter Vectors

In order to implement this techniques, we are going to create our own vectors from the vocabulary of the training data. Then we will proceed to train the model with these vectors and once this steps are done, we are going to add this features with the rest, so that the classifier is ready for train. For this vectors we use a dimension d equals to 300.

3.5.2.2 Google Word Vectors

As an alternative way to our own vectors, this vectors are pretrained vectors created by Google[1]. This means that they have yet their own vocabulary. In particular, it contains more than 3 million of words and phrases trained by them in part of the 100 billion of words

[1]https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit

extracted from the Google News dataset. Furthermore it has a predetermined dimension of vectors equals to 300.

3.6 Classification

Once all the previous steps are done, we have a set of features extracted from the cleaned and processed data. Each of this sets is represented by vectors, and are associated with its corresponding label that will determine the class of the sample. This is the stage of classification where we will try to perform an algorithm that can performs the task of divide or group samples into classes in the better possible way. As we said we want to label each sample, in this project it is used the supervised learning. Having this in count in the follow list are described the most important classifiers[10] related to this task, and in particular those related with this work.

- **SVM Classifier:** The Support Vector Machine is a very useful classifier due to its high number of options. In fact, it have different kernels to classify the data. The most relevant to this project are:
  - **Rbf kernel:** The Radial Basis Function performs non-linear classification. It is also called Gaussian because it uses normal curves around the data points.With this curves, it sums this points and basing on that, it takes a decission.
  - **Linear kernel:** Unlike the rbf, this generates linear division of the data. This means that this kernel will consume less resources than the previous one.
- **Decision Tree Classifier:** It is based on the supervised learning with non-parametric method used for classification and regression. It is important to remark that this classifier is propense to be affected by overfitting. Due to that, we have to take in count some aspects like the number of branches. If the maximum number of vertical depth is to high, the classifier will fall in overfitting. However, a very low value of this parameter will cause underfitting.
- **Logistic Regression:** It is a regression model based on the linear division of the data, so it performs a linear model classification. It works determinating the conditional probability and modeling the data with a logistic function. Due to it determinates the conditional probably, this model is considered as a discriminative model, like SVM or Random Forest can be.
- **Stochastic Gradient Descent Classifier:** Commonly called SGD, it is a linear classifier. The main way in wich it works is finding minima or maxima by iteration and

calculating the gradient of the loss of each sample. The differences with the Logistic Regression are based in the use of different loss functions and solvers.

- **Random Forest:** It is a meta estimator that is formed by a combination of predict trees, in which each one is independent and have the same weigth. For the final result, it improves the predictive accuracy and control overfitting.

- **Ada Boost Classifier:** It is a meta-estimator that begins by fitting a classifier on the original dataset and coulding be used in conjunction with others with the aim of improve their performance.

- **Muti Layer Perceptron:** The MLP is a class of Artificial Neural Network. It is composed by an input layer that process the features, an output layer and a variable number of hidden layers. At the same time, each layer is divided in neurons so that each neuron of the hidden layer will receive as input all the neurons of the input layer. The situation is the same along the next hidden layers and finally the output layer have as input the output of the last hidden layer.

- **Extra-Trees Classifier:** It belongs to the family of Decision Tree but with some differences. The main is that Extra Trees makes random splits over the data, so that it creates a decision tree over each part. Then, it uses this trees to control the overfitting, as well to improve the results through the average score of all trees.

3.7 Improving results

In order to improve the use of every algorithm we will explore different approaches to obtain a better solution. For this task we will take into account that each classifier have different hyperparameters which classify the data in different ways. Also we will take into account that for some of these hyperparams, some features can introduce noise instead of information in the classifier. Based on that we will use the methods described in the follow sections, and finally, combine them into a pipeline in order to make bigger the improvement due to prove every combination.

3.7.1 GridSearchCV

This method provided by Scikit-learn will allow us to prove different combinations of hyperparameters in every classifier. With this, it gives the best value of each hyperparameter. The way in which this technology works, as its name describes, is based in performs squares

or grid searchs. This squares are formed by the combination of the different hyperparameters, so that each value of each hyperparameter is scored with all the values of the rest hyperparameters. The final result is a graph formed by squares where the axes are the mentioned hyperparameters. Considering that, we will have a graph of $n + 1$ dimensions being n the number of hyperparameters, and the last, the axis of the score. For take the decision, it uses Cross-validation methods.

3.7.2 SelectKBest

Once the feature extraction is done, this method will provide us the facility to eliminate those features that don't apport information in the classifier. This fact can seems not so relevant, however we can see an example of the performance of a classifier depending the number of features.

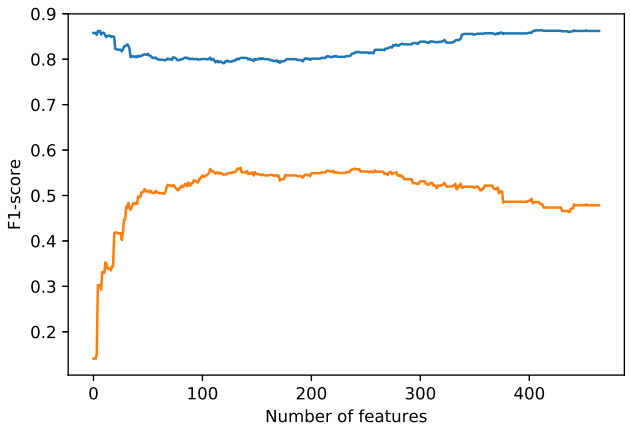


Figure 3.6: F1-score variation with number of features

The top Figure represents the F1-score of the neutral bias with the blue color, and the partisan with the orange. The bot contains a graph of the average of the F1-score. As we can see, exists a significant variation of the F1-score, especially important in the partisan class due to its general low score.

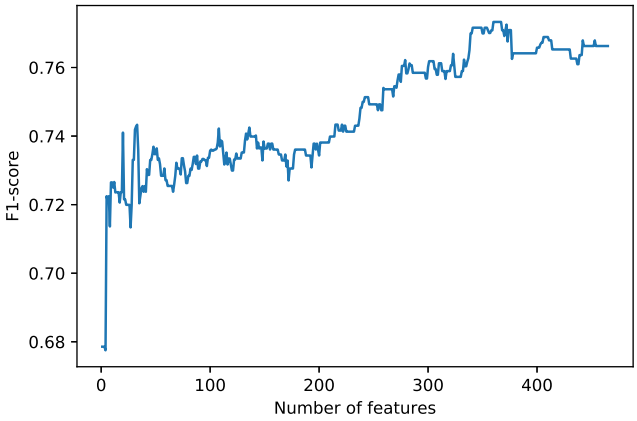


Figure 3.7: Average F1-score variation with number of features

3.8 Implementation in Senpy

Once all the steps are done we proceed to integrate the system into a plugin. For this task, we will use Senpy. In this way the system can be used in two different modes chosen by the user: the Twitter way or the Facebook ones. When selected, the user creates an input with the desired messages to analyze and then, the message pass by the architecture explained in the overview section. The general scheme is summarized in the Figure 3.8. The final result

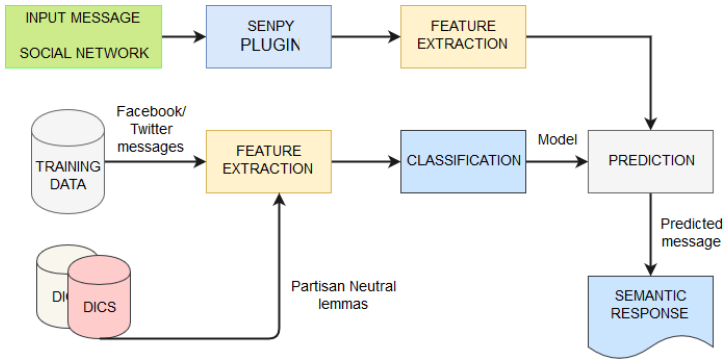


Figure 3.8: System architecture in Senpy

is the class of the message in form of linked data and in a semantic response using marl. We can see below, in the Figure 3.9, an example of this response in JSON-LD format, that the plugin gives with the input of a sample text. The linked data has the advantage that makes the system sharable, extensible and re-usable.

Figure 3.9: Senpy plugin response in JSON-LD format

CHAPTER 4

Evaluation of the results

This chapter has as principal objectives to evaluate the different techniques implemented in the system and select the best result. Furthermore, this chapter provides the main requirements that have to be taken into account in order to perform a correct evaluation in the experiments.

4.1 Introduction

Once the classification step is done, and therefore all the previous stages, it is time to assess the results obtained with each implementation. However, if we want to have a correct evaluation, it is necessary to take into account a serie of requirements in order to evaluate in the most possible real way the results. The next section will explain these requirements needed to have a correct interpretation of results, and the second section will be focus on the interpretation itself.

4.2 Experimental setup

4.2.1 Experiments scheme

In order to obtain correct measures, it is necessary to define the steps to perform the experiments. Firstly, we are going to divide the dataset into training and test set. Due to it is used the TF-IDF (as explain the section 3.5) we can't use cross-validation methods for the validation of the results. Thus, we are going to have a third set that is the validation set. This set is needed in order to tune the different variables of the classifier as well the features. It is important don't use the test set until we obtain that we consider the best approach, otherwise we will doing overfitting, making that the classifier doesn't generalize in a correct way. With this in mind, the final division of the dataset will be:

- **20% Test set**
- **80% Initial training set:**
    - **70% training set**
    - **30% validation set**

Once the dataset is divided correctly, we can proceed to treat the features. For the classifier, each sample of the training data must have the same dimension that each one of the test and validations sets. For that reason, it is necessary to once extracted each feature, make the correspond transformation in the test set and in the validation set.

During the measures we are going to divide the experiments in three different ones. First we are going to measure the improve of the mentioned dictionary. For that, we are going to perform measures with all the features of the system, and then, we repeat the measure removing those features that belong to the dictionary.

The second experiment will consist in measure the response of the system using only Word2Vec features. For this experiment we are not going to use SelectKBest in order to consider all the features.

The third and last experiment will consist in join the two previous experiments, combining the different techniques. The general scheme is represented in the Figure 4.1

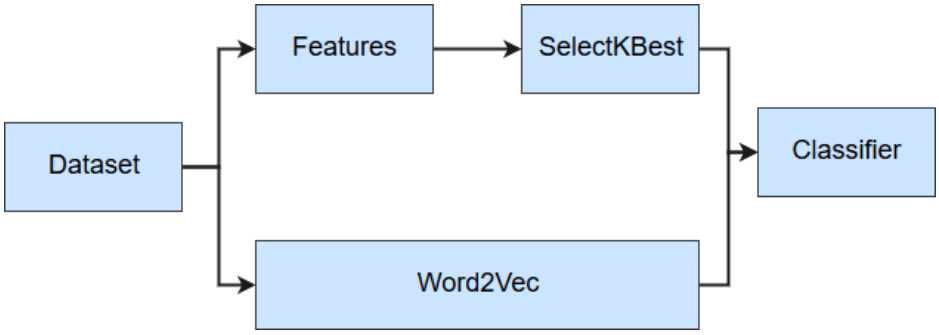


Figure 4.1: Architecture of the third experiment

4.2.2 Metrics

The step to make a correct analyze of the results could becomes into something more relevant even than the choice of any classifier. One of the most famous and popular metrics to this step is the accuracy:

$$Accuracy = \frac{TN + TP}{TP + FP + TN + FN} \tag{4.1}$$

The next table explains in detail what each of these params mean:

TN	True Negative represents the instances of the real class Negative predicted, as Negative by the classifier.
TP	True Positive are the samples classified as Positive having as real class Positive.
FN	Those examples of the real Negative class predicted as Positive class.
FP	Examples that were classified as Positive but were Negative in the real class

Table 4.1: Metrics Params

However this is not enought to know if a classifier is making a good job, in fact, it is a very poor tool to analyze results. One simple example of the low veracity that *accuracy* can offers, is a test set with two labels to predict, but with a proportion of 1:9 numbers of instances between each one. A model like Zero-R (the most simple classifier) will obtain according an accuracy of 90%, because it will classify all examples as the major class.

A similar situation happens with the dataset of this project, where due to the high number of neutral messages, the improve and the results of this parameter could be good.

However the real classification could be classifying most of samples as neutral class. Having this in mind, other measures will be needed in order to obtain a correct approach.

4.2.2.1 F1-score metrics

Also provided by Scikit-learn, this package offers diferent kind of useful tools for the measure. This tools will make possible to check how the errors are distributed around different classes. In particular, with this tool we can analyze the errors in the different ways, described below:

- **Precision:** Shows the proportion between true positives (TP) of one class and the total positives (TP + TN) of this class.

$$Precision = \frac{TP}{TP + TN} \tag{4.2}$$

- **Recall:** The recall gives the proportion between the instances well classified of one class and the instances of the same class bad classified.

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

- **F1-score:** The result F1-score express the measure of the previous metrics. In particular it uses:

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{4.4}$$

4.2.2.2 Confusion Matrix

Like the rest of measures seen, it also can be found in the metrics package of Scikit-learn library. It is a metric that allows to see directly the errors of the classifer , because it

		PREDICTED CLASS	
		POSITIVE	NEGATIVE
REAL CLASS	POSITIVE	TRUE POSITIVE	FALSE NEGATIVE
	NEGATIVE	FALSE POSITIVE	TRUE NEGATIVE

Figure 4.2: Confusion Matrix

provides the distribution of all the samples classified. All the information that apports is the matrix in which the number of True Positives, False Positives, True Negatives and False Negatives are provided together for each class, allowing to identificate the number of each one mentioned.

4.3 Evaluation

Due to there are no other works with this dataset, there is no any baseline to overperform or improve. Therefore, we will center in obtain the better score in specific parameters like the F1-score. Particularly we are going to evaluate the experiments mentioned in the previous section. Furthermore we also will check and contrast the best results obtained in Twitter and Facebook, with the aim to profile each social network.

As we said, due to don't fall in Zero R, it is necessary in this case to choose that results that don't have the neutral class as the principal classification. One example of this is the kernel rbf where for the step of improve of parameters, this result appears.

Class	**Precision**	**Recall**	**F1-score**
neutral	0.76	0.99	0.86
partisan	0.62	0.04	0.08
Avg/total	0.73	0.76	0.67

Table 4.2: Example of tendency to Zero R

We can see that it could seem a good result but, analyzing more in detail the recall of the partisan class, it is near to zero, in other words, this algorithm is classifying almost all the samples as neutral class. For that reason and due to this dataset doesn't belongs to any competition we decided to represent the F1-score in order to have more specific data. In addition to the choice of this metric, we will stipulate a minimum requirement that the final system has to meet, that is to obtain a **minimum F1-score in the partisan class of 0.5**. Furthermore, representing other parameters like accuracy we can wrongly obviate the possible difference in the number of samples as we have seen in the metrics section.

4.3.1 Evaluation of the partisan-neutral dictionary

In this section we analyze the main results of the general system with no adding, and the system after implements this dictionary, checking in this way the possible improve that this feature apports. The results are divided into Facebook (Table 4.3) and Twitter (Table 4.4).

We can see that in general the results are not bad. However in a first analysis it seems that the addition of the dictionary doesn't improve in a great mode the final F1-score. This could be due to the high number of Neutral messages that exists, as we explained in the previous section, that means that a minimal decrease in the F1-score of this class is translated into a relevant lose in the F1-score of the general system.

Analyzing more in detail the results, we can see that both Facebook and Twitter systems reach a good F1-score with the adding of the dictionary, and in both cases the results with the dictionary tends to improve the classifier. For Facebook, the improve means to gain 2% in average, and for Twitter a 2.125%. As we said, there are no big improves, but in general the tendency of this feature is to rise the score.

Classifier	Fb with dictionary	Fb without dictionary
SVM classifier	0.73	0.74
Logistic Regression	0.74	0.68
SGD	0.72	0.68
MultiLayer Perceptron	0.73	0.73
Random Forest	0.73	0.71
Decision Tree	0.71	0.70
Ada Boost Classifier	0.74	0.70
Extra Trees Classifier	0.74	0.74

Table 4.3: F1-score for Facebook with and without dictionary

Classifier	Tw with dictionary	Tw without dictionary
SVM classifier	0.73	0.73
Logistic Regression	0.74	0.73
SGD	0.74	0.68
MultiLayer Perceptron	0.73	0.72
Random Forest	0.70	0.70
Decision Tree	0.71	0.67
Ada Boost Classifier	0.72	0.69
Extra Trees Classifier	0.73	0.71

Table 4.4: F1-score for Twitter with and without dictionary

4.3.2 Evaluation using Word2Vec

In this section we are going to evaluate the system using the features provided by Word2Vec. For that, we divided the results according to the vectors employed as features: the Domain Adapted Vectors and the Google Word Vectors. We can see the results in the tables 4.5 and 4.6 for Facebook and Twitter respectively. As we can appreciate, in general are good results. However, they seem in general to be a bit lower than using our features and our dictionary. We are going to analyze it more in detail. For the first table we can observe that using the domain adapter in general doesn´t improve the system. In fact we obtain 73% of F1-score with Logistic regression, that is a lower value than in the previous experiment. However, in the same table, we reach 0.75% of F1-score with the Google Vectors with three different classifiers, the Multi Layer Perceptron, the Random Forest and with the Ada Boost Classifier. This improves the results obtained before, but only in a 1% of F1-score.

A similar situation happens in the bot Table -the Twitter ones- where the results obtained with the Adapter Domain Vectors are in general lower than the previous experiment and also than the Google Word Vectors. In particular we reach the best score with SVM and using Google vectors, but it doesn't improve the previous work. The low score obtained with the Adapter Domain vectors can be produced by a low number of samples offered by the dataset, so it is probably than using more data this feature improve the system. We appreciate this fact comparing in both tables the scores obtained from the Google Vectors

and the Domain Vectors, where we obtain 2.9% more than F1-score in Facebook and more than 3% in Twitter. As we said, it can be logical if we think that the Google Word Vectors are trained with millions of words, in comparison with the Domain Vectors trained from the current dataset.

We can conclude from these two tables that for Twitter the system doesn't improve with the use of Word2Vec, and for Facebook, the improve that reaches is to small, representing only a 1% of the F1-score. We also appreciates how in this case train the system with a high amount of data allows to generalize better than using the own dataset.

Classifier	**Fb Domain Vectors**	**Fb Google Vectors**
SVM classifier	0.66	0.73
Logistic Regression	0.73	0.74
SGD	0.72	0.71
MultiLayer Perceptron	0.68	0.75
Random Forest	0.71	0.75
Decision Tree	0.67	0.68
Ada Boost Classifier	0.71	0.75
Extra Trees Classifier	0.66	0.72

Table 4.5: Facebook F1-score using Word2Vec

Classifier	Tw Domain Vectors	Tw Google Vectors
SVM classifier	0.69	0.74
Logistic Regression	0.71	0.73
SGD	0.69	0.71
MultiLayer Perceptron	0.68	0.73
Random Forest	0.69	0.72
Decision Tree	0.62	0.66
Ada Boost Classifier	0.68	0.72
Extra Trees Classifier	0.60	0.70

Table 4.6: Twitter F1-score using Word2Vec

4.3.3 Evaluation of Word2Vec combined with features

In the previous experiments we measure the perform and the improve of the system taking the base features and then adding the dictionary to this features. On the other hand we proved the system using Word2Vec features comparing the results of the different vectors. Having this in mind, in this last experiment we are going to measure the combination of these two experiments joining the Word2Vec vectors with the whole features.

In the Table 4.7 we see the results obtained for Facebook and in the 4.8 the obtained for Twitter. The first impressions evidence an improve of the results with these experiments. More in detail, for the Facebook results we can see how adding features suppose a general rise in the F1-score. The domain vectors previously analyzed increase in a way that overcomes the initial experiment, with the SGD classifier and a F1-score equals to 75%. Furthermore it also rises the average F1-score but in a small way with only a 1%.

With respect to the Google Word vectors, the join of the features to the model also supposes a general improve, but again in a small way with an average below to 2%. However, the rise of the F1-score depends in a great way of the classifier, so the average could not be very representative. In fact analyzing each classifier with this combination composed by features plus google vectors, we obtain the highest score of the system with a 79% of F1-score using Logistic Regression model. This means to increase in 5% the best results previously obtained with this classifier and Google Vectors, and in 4% the best result obtained between

all the experiments.

In the Twitter results, we have a similar situation. The domain vectors, also have a relative rise when we add them all the features. This rise consist in the 2% of the F1-score, but unlike in the Facebook results, this improve is not enough to obtain a better score than the model with all the features, so we can conclude in this case it is not resulting a useful implementation. This low results and differences with the same respective vectors in Facebook, could occurs due to the train received. Although both Facebook and Twitter have the same number of samples, the Facebook messages are longer than the 140 characters stipulated by Twitter. This means that each Facebook message could apport more information in each sample than the Twitter ones. On the other hand, analyzing the combination between all the features and Google Word Vectors in Twitter we obtain a significant improve. Again Logistic regression reaches the better result with a 76% of F1-score, improving in 3% the performance of this classifier withouth features, and in 2% the second best result between all the experiments. It is clear then that the mix of high amount of data provided in vectors of Word2Vec plus the customized features is the best option for both classifiers.

Classifier	**All feats**	**Domain**	**D+feats**	**Google**	**G+feats**
SVM classifier	0.73	0.66	0.67	0.73	0.76
Logistic Regression	0.74	0.73	0.74	0.74	**0.79**
SGD	0.72	0.72	0.75	0.71	0.73
MultiLayer Perceptron	0.73	0.68	0.67	0.75	0.75
Random Forest	0.73	0.71	0.67	0.75	0.74
Decision Tree	0.71	0.67	0.70	0.68	0.71
Ada Boost	0.74	0.71	0.70	0.75	0.73
Extra Trees	0.74	0.66	0.71	0.72	0.72

Table 4.7: Facebook comparison between techniques

Classifier	All Feats	Domain	D+feats	Google	G+feats
SVM classifier	0.73	0.69	0.67	0.74	0.75
Logistic Regression	0.74	0.71	0.71	0.73	**0.76**
SGD	0.74	0.69	0.71	0.71	0.72
MultiLayer Perceptron	0.73	0.68	0.71	0.73	0.73
Random Forest	0.70	0.69	0.70	0.72	0.73
Decision Tree	0.71	0.62	0.66	0.66	0.67
Ada Boost Classifier	0.72	0.68	0.68	0.72	0.74
Extra Trees Classifier	0.73	0.60	0.68	0.70	0.71

Table 4.8: Twitter comparison between techniques

Finally we show the F1-score of the best models in order to verify that this value in the partisan class meets the specified requirements. In the follow table we show these results. We can appreciate that the F1-score of the partisan class is higher than 0.5, in fact it reaches

	Facebook			**Twitter**		
Class	**Precision**	**Recall**	**F1-score**	**Precision**	**Recall**	**F1-score**
neutral	0.89	0.82	0.85	0.86	0.78	0.82
partisan	0.56	0.68	0.62	0.56	0.69	0.62
Avg/total	0.80	0.79	0.79	0.77	0.75	0.76

Table 4.9: F1-score by classes in Facebook and Twitter

an acceptable value of 0.62 in both Facebook and Twitter, so we can conclude there are not relevant differences in the classification between these two social networks, and we also can extract the idea that despite the Facebook dataset was more imbalanced than the Twitter ones, it compensates this fact with a higher size of its samples.

CHAPTER 5

Conclusions and future work

This last chapter provides the conclusions obtained from the project. Futhermore, are analyzed the main problems founded during the work, and finally are given the possible lines to develop in the future, in order to improve the system.

5.1 Conclusions

In this project we developed a system able to classify political messages according to their bias in partisan or neutral. The system is based in both machine learning and natural language processing techniques. For the training and test sets were employed Twitter and Facebook messsages, in a way that we characterize these social networks in a separately way and threfore we built a different model for each one.

For the construction of our models we use different techniques and features that were tested in its corresponding experiment. The first experiment measures the score of the system as well the improve of a dictionary composed by the most common lemmas of each class. From this experiment we conclude that the addition of this feature, although not improve the best result in a relevant way (only 1% in Twitter) the general tendency was to rise the score.

The second experiment was focused in analyze the variance of the score of the system when adding basic Neural Networks like Word2Vec, and proving inside this technology different ways to approach the problem. The results obtained reveals that when using pretrained vectors for feed the system, instead adapted domain vectors, the F1-score increases in a relevant way. This fact occurs due to the high amount of information which these external vectors were trained, concluding that the dataset was not enough large to performs a customized learning.

The third and last experiment consists in combine the Word2Vec model with all the features before developed. In this experiment we obtain the best scores for our system with a F1-score of 0.79 in Facebook using Logistic Regression with Google Word Vectors. It also reaches the highest score in the Twitter dataset, also with the same algorithm and the same vectors, obtaining a F1-score equals to 0.76.

Finally, the system was implemented into a plugin in order to allows making real time predictions of political messages. For this task, the tool Senpy was employed.

5.2 Problems Faced

This section describes the most important problems that we had to avoid during the develop of the system.

- **The imbalanced data:** It regards to the high number of samples that belongs to the Neutral class combining with the fact of don't have a very large dataset. It was the main trouble in order to obtain good results in the classification due to techniques like undersample could produce an important waste of resources to train the model. We tried to avoid this problem with different features and in particular, with a dictionary of most common lemmas in each class.

- **The codification of the dataset:** The dataset was created by an external contributor so it had different unuseful symbols which, once the dataset was encoded, creates strange codifications. These codifications only apport noise to our system so we tried to eliminate them.

- **Computer Resources:** The long time that the experiments take was a handicap in order to obtain a better approach refining the model. This consumption of resources was especially notable using the Word2Vec model with the Google trained vectors.

5.3 Future Work

Having in count the work done in this project, there are some possible improvements that can be taken to have a more complet system. This section will explain this main possible lines to develop in the future.

Improve Neural Networks: In this project we used the two layer Neural Network called Word2Vec, but would be interesting, having more resources, to analyze the problem with a higher number of layers.

Collect our own dataset: As we have seen, we solve the main problems with the dataset employed. However, it is limited in the way that we lost information with the mentioned problems. An interest improve could be to collect and analyze later our own dataset, due to it opens new possibilities like to analyze the emoticons used by the writters or also to analyze other context apart from the American politicians. It also would be interesting to train the models periodically with the new vocabulary used by politicians.

Analyze the content of the links: An interesting next step could be, in addition to the count of the number of links that each message contains, to analyze the kind of web that the author wants to share. This could be done creating a database of the most common domains or also trying to analyze the words which contains the link, due to it could apports information of the general aim of the author.

Bibliography

[1] Canadian study. http://blog.canadianstenographer.ca/.

[2] Convote dataset. http://www.cs.cornell.edu/home/llee/data/convote.htmll.

[3] Dataset. https://www.kaggle.com/crowdflower/political-social-media-posts.

[4] Gensim. https://radimrehurek.com/gensim/.

[5] Ibc corpus. https://www.cs.umd.edu/~miyyer/ibc/index.html.

[6] Matplotlib. https://matplotlib.org/.

[7] Numpy. http://www.numpy.org/.

[8] Oti reference. http://www.ontheissues.org/default.html.

[9] Pandas. https://pandas.pydata.org/pandas-docs/stable/generated/pandas.Series.html.

[10] Scikit-learn. http://scikit-learn.org/stable/.

[11] Scipy. https://www.scipy.org/.

[12] Senpy. http://senpy.readthedocs.io/en/latest/senpy.html.

[13] Textblob. http://textblob.readthedocs.io/en/dev/.

[14] Vader sentiment analysis. https://pypi.python.org/pypi/vaderSentiment.

[15] Word2vec. https://radimrehurek.com/gensim/models/word2vec.html.

[16] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[17] Annie Hellweg. Social media sites of politicians influence their perceptions by constituents. *The Elon Journal of Undergraduate Research in Communications*, 2(1):22–36, 2011.

[18] Mohit Iyyer, Peter Enns, Jordan Boyd-Graber, and Philip Resnik. Political ideology detection using recursive neural networks. In *Proceedings of the Association for Computational Linguistics*, pages 1–11, 2014.

[19] Mark Lutz. *Learning Python: Powerful Object-Oriented Programming.* " O'Reilly Media, Inc.", 2013.

[20] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, (12).

[21] Arkajyoti Misra and Sanjib Basak. Political bias analysis.

[22] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.