UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SARCASM DETECTION ON TWITTER

JUAN ÁLVAREZ FERNÁNDEZ DEL VALLADO

2019

TRABAJO DE FIN DE GRADO

Título:	Detección de Sarcasmo en Twitter
Título (inglés):	SARCASM DETECTION ON TWITTER
Autor:	Juan Álvarez Fernández del Vallado
Tutor:	Carlos Angel 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

TRABAJO DE FIN DE GRADO

SARCASM DETECTION ON TWITTER

Enero 2019

Resumen

Palabras clave:

Abstract

Keywords:

Agradecimientos

A Gauss

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

1.2 Project goals

The objective of this project is to define a classifier capable of detecting sarcasm in texts, particularly in tweets. By using a dataset of over 11,000 tweets and Machine Learning (ML) techniques, the objective shall be overcome.

- G1

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is as follows:

Chapter 1 ...

CHAPTER 2

State of the Art

In this chapter a brief introduction on the main learning technologies used for text-based learning will be given. Moreover, an analysis on automatic sarcasm detection will be briefed and finally the main issues found shall be overviewed.

2.1 Introduction

Sarcasm detection is an important component for Natural Language Processing (NLP) very relevant to natural language understanding, dialogue systems and text mining. A challenge is to construct a balanced dataset, where the text is already labeled [22]. In this project, the Tweets were previously classified and a balanced dataset was provided.

2.2 Analysis: Automatic Sarcasm Detection

This section will briefly discuss the analytical tools used to identify sarcasm. It will cover topics such sarcasm definition, data-set issues and analytical approaches available for sarcasm automatic detection.

2.2.1 Sarcasm Definition [20]

The Free Dictionary[1] defines sarcasm as a form of verbal irony that is intended to express contempt or ridicule . The subjective characteristic of sarcasm makes sarcasm very difficult to detect since a word may have a literal meaning and a sarcastic meaning. As a result, sarcasm automatic detection presents a big challenge.
Typically, sarcasm detection has been formulated as a classification problem . In that sense, given a text the goal is to predict whether or not the text is sarcastic. However, the big challenge is to label the text, since the classifier is supposed to distinguish between irony, humour and other sentimental meanings.
Furthermore, sarcasm can also be formulated as detection for dialogue as a sequence labeling task. This technique consists of labelling every word in a sentence and predicting the sarcastic labels (as these words won't be labeled).

2.2.2 Data-set

As a matter of a fact, the studies already done in automatic sarcasm detection depend entirely on the data-set. That is why it is convenient to divide them into three kinds [20]:

- *Short texts*: The social network tweeter is extremely popular due to its popularity among users. It can serve as an example of short text since every tweet is restricted to not exceed an amount of words. One approach to obtain labels for the tweets is to manually mark them as sarcastic or non-sarcastic. Another approach is the hashtag-based supervision. Hashtags can reveal when a tweet is labeled as sarcastic. By introducing the *sarcasm* label, the author makes clear his intention to use sarcasm. This allows the dataset to be bigger. Many works using this variation have been reported since its popularity has constantly grown.
- *Long texts*: The information usually comes from reviews, posts, news, articles.
- *Other data-sets*

It is noteworthy to mention the emphasis made on the Twitter social network since this project will be focusing on a Spanish set of tweets to detect sarcasm.

2.2.3 Approaches used for Sarcasm Detection

In this section, several approaches used for detecting sarcasm will be discussed. In general, approaches for sarcasm detection can be classified into rule-based, statistical and deep learning-based approaches [20].

[1] www.thefreedictionary.com

2.2.3.1 Rule-based Approaches [20]

Rule-based approaches work by identifying sarcasm through a set or rules based upon evidence. The evidence is captured by using rules relying upon sarcasm. For instance, one rule can rely on Google to determine how likely that symbol (word) is while another rule can come from hashtag analysis . By identifying the hashtags present in a tweet, a sarcasm pattern could be found. Furthermore, one could find sarcasm if a negative phrase occurs in a positive sentence. There are more complex rule-based approaches which will not be further mentioned on this document.

2.2.3.2 Statistical Approaches

Statistical approaches are classified in two [20]:

- *Features Used:* Most of these approaches use bag-of-words as features that have been found by statistical sarcasm detection. Apart from finding the features, some other works focus on designing pattern-based features, that are able to indicate sarcastic patterns on the corpus. To allow the classifier to spot sarcastic patterns, these pattern-based features take three real values: exact match, partial match and no match. Furthermore, pragmatic features, like emoticons, can also be considered.

- *Learning Algorithms:* A variety of classifier have already been experimented for sarcasm detection. SVM (Support Vector Machines) is quite often present in sarcasm detection classification techniques.

- *Deep Learning-based Approaches:* Although nowadays deep learning has raised a lot of awareness, there are still few approaches reported for automated sarcasm detection. Similarity between word embeddings as features for sarcasm detection has been used. This augments features based on similarity of word embeddings related to other word pairs and report an improvement on performance. Convolutional neural networks have also been used, reporting an improvement of 2% in performance.

To put it in a nutshell, different techniques for automatic sarcasm detection can be performed. Despite these numerous techniques, sarcasm automatic detection is difficult to detect since a word can have a literal meaning but also a sarcastic meaning that is not so easily labelled.

2.3 Main Issues in Sarcasm Detection

There are three main issues present in sarcasm automatic detection techniques [20]:

- *Data:* Even though hashtag-based labelling can provide large-scale supervision, the quality of the dataset can be doubtful. For example, let us take the hashtag # not. Is this supposed to express sarcasm in the sentence or is it simply used to express a negation? In most of the works, this problem is tackled by removing the # not in the pre-processing step and analyzing the sentence. However, this may as well not be the optimum solution. Another solution is to use as test set some manually labelled tweets and as train set a hashtag labelled set.

- *Features as Sentiment:* The question is how can sentiment be detected in a sentence? In the case of sarcasm, some answers have already been given. If a negative phrase occurs in a positive sentence then that sentence has most likely sarcasm. In a statistical classifier, surface polarity can be used as a feature of the tweet. To capture surface polarity one has to analyze two emotions: activation and pleasantness. This can lead to a 4% improvement in the accuracy.

- *Skewed Data-sets:* Sarcasm is hard to find in an expression. For that reason, skew is reflected in data-sets.

2.4 Figures of Merit

This section is committed to showing other past works in Sarcasm Detection for tweets. The decisive parameters shown in this section will be the F-measure and the classifier itself.

- Sarcasm Detection on Czech and English Twitter [26]:
  - *Description*: This paper presents a ML approach to sarcasm detection on Twitter in two languages - English and Czech. The classification was carried out by using SVM classifiers and Maximum Entropy. Since only SVM has been implemented in this project, the latter will be discarded.
  - *Results:* F-measure (balanced dataset): **0.947**. F-measure (unbalanced dataset): **0.924**. Algorithm: **SVM**

- Irony detection in short texts [2]:
  - *Description*: This docuemnt summarizes the result of analyzing ironic tweets. The language is spanish.
  - *Results:* F-measure (balanced dataset): **0.86**. F-measure (unbalanced dataset): **0.82**. Algorithm: **SVM** F-measure (balanced dataset): **0.82**. F-measure (unbalanced dataset): **0.74**. Algorithm: **Random Forests (RF)**

CHAPTER 3

Enabling Technologies

In this chapter the technologies used to achieve this projects' objective shall be discussed.

3.1 Introduction

In this project two main technologies have been used: NLP and ML. The implementation of these tools has been performed in a combined fashion. Furthermore, other technologies have also been used but they are less relevant than NLP and ML. On the one hand, with NLP it was possible to transform words into tokens, find syntactical connections between tokens in the same tweet and, finally, construct a numerical vector. On the other hand, ML was very useful to construct a classifier, define a model and feed the vectors from the previous step into the model to make predictions.

In short, this is how the project was carried out. Both technologies will be extensively explained in the next sections.

3.2 Natural Language Processing

NLP has a crucial role in this project since tests (tweets) is what this project is trying to deal with. Hence, the information that used to train the ML algorithm will come in the form of text. According to [11], at one extreme NLP could be as simple as counting word frequencies to compare different writing styles. At the other extreme, NLP involves

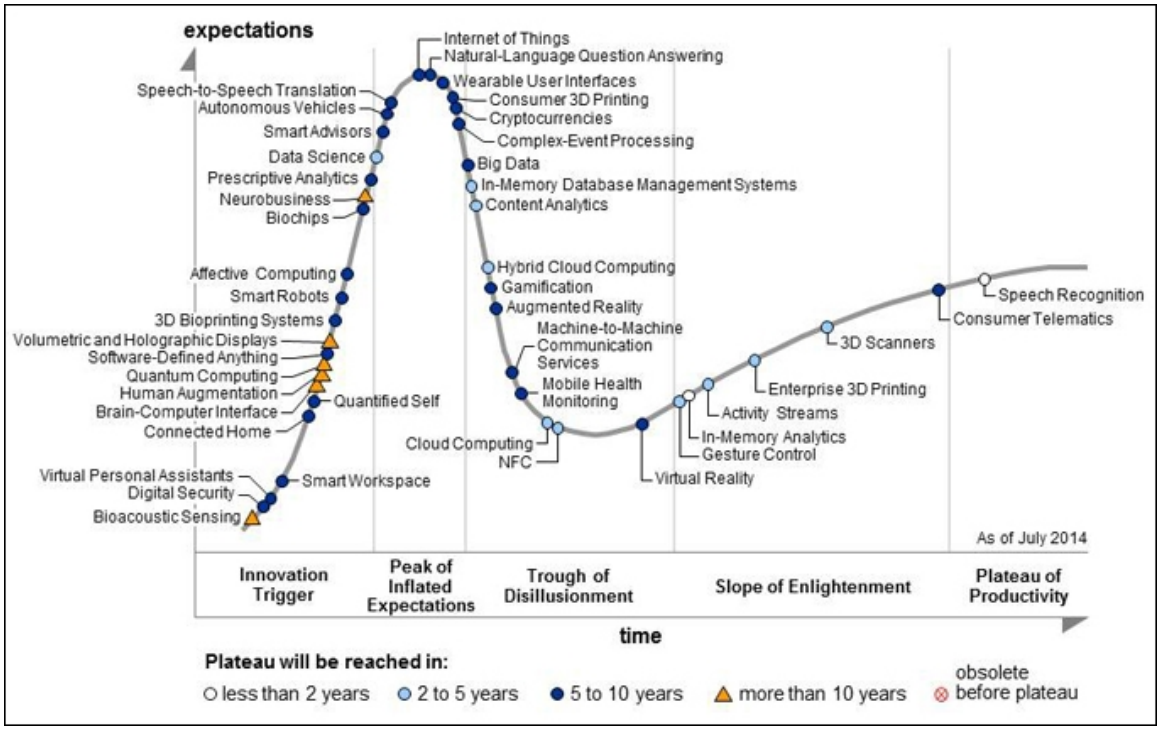

Figure 3.1: Image is taken from `http://www.gartner.com/newsroom/id/2819918`

"understanding" complete human utterances, at least to the extent of being able to give useful responses to them.

Figure 3.1 shows many emerging technologies that are using NLP to provide their services. After making the reader conscious of the potential of NLP, it will be shown what has been achieved in this project with NLP.

Many other examples of NLP applications are machine translation, optical character recognition, language detection, topic modeling and many more [19].

3.2.1 Natural Language Toolkit

3.2.1.1 Definition

Natural Language Toolkit (NLTK) is a platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum. [1].

The following applications explained beneath belong to the NLTK class.

[1] https://www.nltk.org/

3.2.1.2 Application

NLTK has provided the following tools to this project:

- *Tokenizer*: A word (Token) is the minimal unit that a machine can understand and process. So any text string cannot be further processed without going through tokenization. Tokenization is the process of splitting the raw string into meaningful tokens [23]. In this project, tokenization was required to get the words containing a tweet. Furthermore, some of the tokens were removed since they do not add meaningful information (e.g. yo, a). These words belong to a stoplist provided by NLTK for all supported languages [23].

- *Stemmer*: An stemmer is needed to remove afflixes from a word, ending up with a stem. Stemming is frequently used for indexing words. Instead of storing all forms of a word, the algorithm stores only the stems, greatly reducing the size of index while increasing retrieval accuracy [23]. Particularly, I have chosen the SnowballStemmer because it supports stemming in 13 non-English languages, especially Spanish.

- *Part-of-Speech tagging*:

  En tu proyecto no has usado POS porque te fallaba.

  A-part-of-Speech tagger (pos_tag) indicates how a word is functioning within the context of a sentence [25]. This part is crucial since a word can function in multiple ways and we would like to distinguish those cases. POS_tagging has been achieved by collecting each word feature (i.e. noun, verb, adj, adv) and building up a dictionary with the word features' stats.

The applications at the top of this section are more commonly known as the preprocessing part previously needed to any ML application. Since there are plenty of tasks, they could be easily coordinated by using pipelines.

3.3 Machine Learning

When it comes to defining models capable of learning from a text-based source, the two dominant technologies are Natural Language Processing and ML. These two main technologies are concerned in how does the machine process human text.
Given data and the proper statistical techniques, ML automates analytical model building capable of recognizing patterns with the aim of making predictions and learning from the model. To be able to do so, a learning process is required. ML algorithms can be classified in three categories [27]:

- *Regression*: Regression learning is predicting a continous varying variable.

- *Classification*: Classification is predicting a discrete class label.

- *Clustering*: Data clustering is the task of labelling unlabeled data, creating clusters with different properties.

Furthermore, depending on the dataset the learning process can be carried out in two different ways:

- *Supervised learning:* The classifier is trained using a percentage of the total dataset while feeding the rest of the dataset to make predictions and to evaluate how good the classifier is. Particularly, this project will be using supervised learning.

- *Unsupervised learning:* These algorithms base their learning properties on learning patterns present on the dataset. Unlike in supervised learning, there is no training and test dataset partition

Two major problems encountered in a ML appication are underfitting and overfitting. These two problems appear due to the bias generated when training the model.
It turns out that depending on *how much trained* the classifier is, it can make more assumptions. A learned function that can perform well enough on unseen data points as well as on the training data is termed a generalizable function [17]. A generalizable function is achieved through a generalizing fit.

- *Underfitting:* Figure 3.2 shows that for any point in the diagram, the function always assigns the same predicted value.

- *Overfitting:* Figure 3.3 shows that for any point in the diagram, the function performs very good. However, the function has become very unflexible and it will not perform good enough on new points.

In contrast with ML, NLP is a way to analyze and derive the meaning of human language. To do so, NLP provides a means to process lexical properties as well as syntactic recognition of the human language. However, human language can be rarely precise and ambiguous. Even though NLP has a very wide range of applications, such as automatic text summarizing, relationship extracting, stemming, sentiment analysis and more, it is a difficult problem to tackle in computer science.

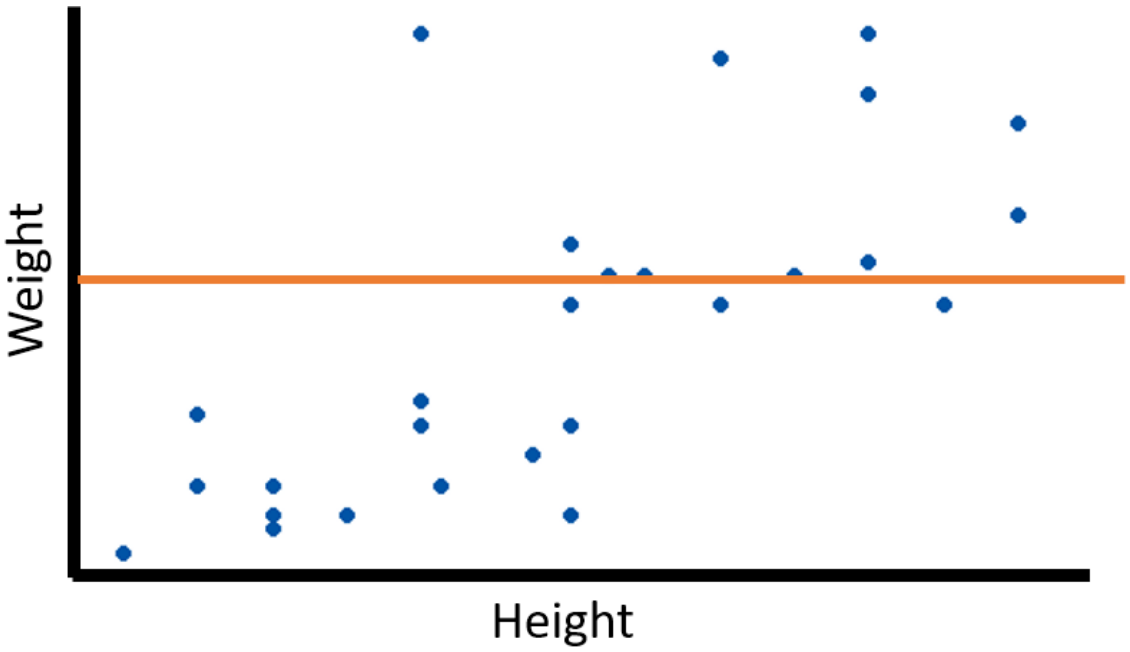

Figure 3.2: Underfitting [17]

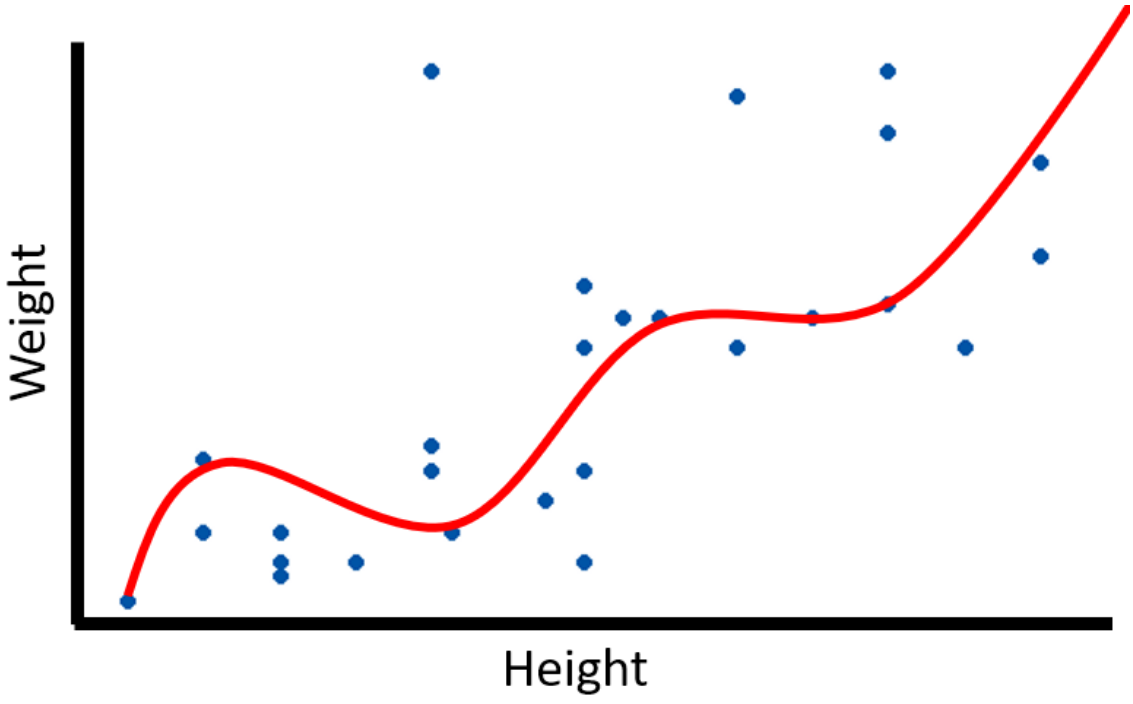

Figure 3.3: Overfitting [17]

3.3.1 Sklearn

3.3.1.1 Definition

Sklearn (a.k.a scikit-learn) is defined [2] as a simple and efficient set of tools for data mining and data analysis. It is built in NumPy and SciPy. Sklearn will provide not only the classifiers needed for the pipeline, but also meningful tools.

3.3.1.2 Application

Sklearn has provided the following tools to this project:

- *Train and Test splitting*: This method is responsible for splitting the main dataset into two parts, one for training the model and the other one for testing the model. It is a very common ML technique. I chose a test set size of 25 % of the main dataset. In addition, testing set will later be used to compute parameters such as accuracy and f1 score.

- *Pipeline*: A pipeline is a set of procedures connected in series, one after the other where the output of one process is the input to the next [21]. Image 3.4[3] represents the idea of a pipeline, though it does not resemble exactly this projects' pipeline.

- *Tf-idf-Vectorizer*: The Tf-idfVectorizer is used for every word. It contains two parts; the *if* part, which represents the word frequency and the *idf* part, which represents inverse document frequency [30]. This term represents a words' weight in a text. The vectorizer has been used in the pipeline to assign each word a number.
  Es esto cierto?

- *Count Vectorizer*: The algorithm consists of representing a token by considering how many times it appears in a text [13]. Moreover, Count Vectorizer has defined the ngram range.

- *Multinomial NaiveBayes*: A multinomial distribution useful to model feature vectors where each value represents a number of occurrences of a term (or its relative frequency) [13]. The MultinomialNB has been used as a classifier fed into the pipeline. Its results can be seen.
  añade referencia tabla resultados cuando lo pongas

  .

[2] http://scikit-learn.org/stable/

[3] https://www.slideshare.net/databricks/dataframes-and-pipelines

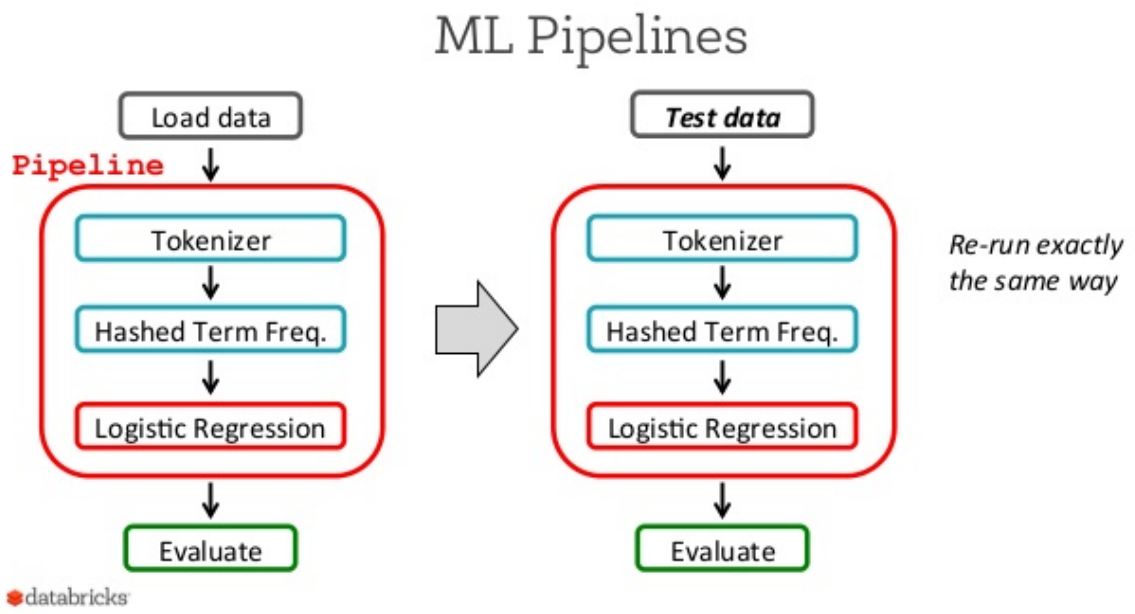

Figure 3.4: A Pipeline. [6]

- *SVM classifier*: Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier[4]. The SVM classifier has been fed into the pipeline as input. Its results can be seen.

  añade referencia tabla resultados cuando lo pongas

- *Kneighbors classifier*: Is a non-generalized ML algorithm that computes distances from one point to all the training dataset points and chooses the k nearest points [12].The KNeighbors classifier has been fed into the pipeline as input. Its results can be seen.

  añade referencia tabla resultados cuando lo pongas

- *Logistic Regression classifier*: Is analogous to multiple linear regression, except the outcome is binary. Various transformations are employed to convert the problem to one in which a linear model can be fit [14]. Its results can be seen.

  añade referencia tabla resultados cuando lo pongas

  .

[4] https://en.wikipedia.org/wiki/Support_vector_machine

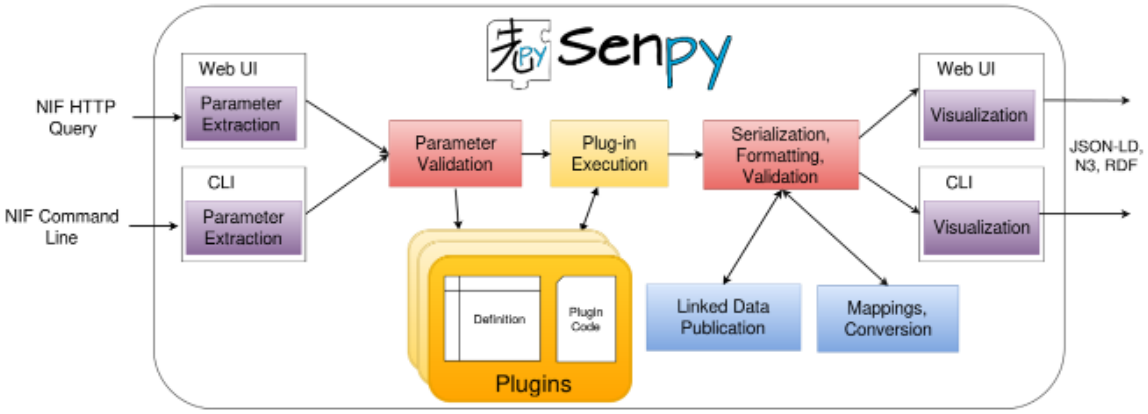

Figure 3.5: Senpy's Architecture [7]

3.4 Senpy

Senpy [7] is a framework for text analysis using linked data. It aims at providing a framework where analysis modules can be easily integrated as plugins and, at the same time, provides the core functionalities (data validation, user interaction, logging, etc).

3.4.1 Architecture

Senpy introduces a modular and dynamic architecture which allows implementing different algorithms in an extensible way, while offering a common interface and offering common services that facilitate development, so developers can focus on implementing new and better algorithms.

To do so, the architecture consists of two main modules:

- Senpy core: Building block of the service.
- Senpy plugins: Analysis algorithms.

Figure 3.5 depicts a simplified version of the processes involved in Senpy analysis framework.

3.5 Luigi

The purpose of Luigi is to run a set of pipelined processes to perform several tasks [3]. In that sense, Luigi will help stitching the tasks together and tends to the workflow management. Luigi is written in Python.

3.5.1 Architecture

There are two fundamental building blocks in Luigi- the *Task* class and *Target* class:

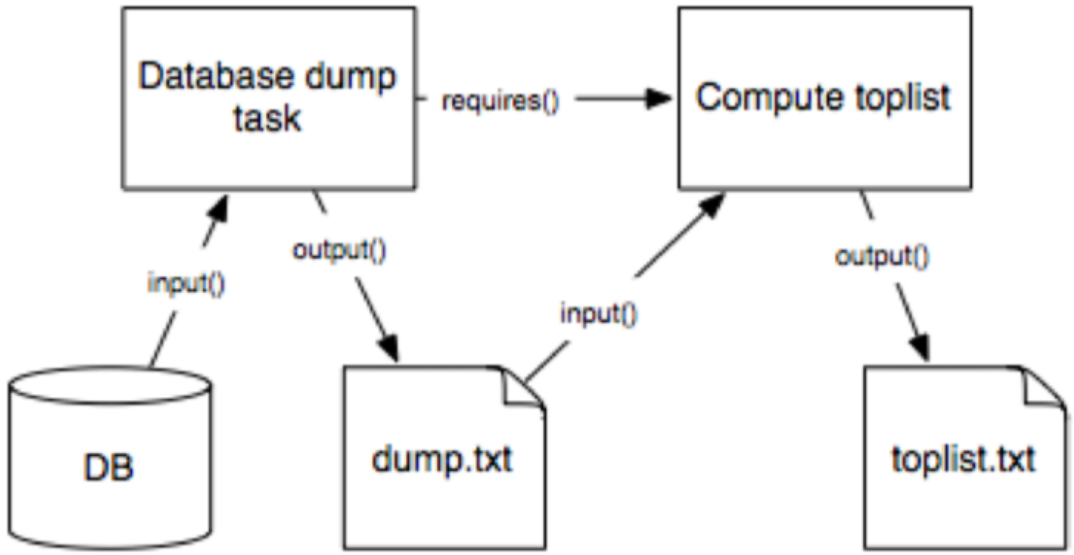

Figure 3.6: Task and Target illustration. [3]

- *Target*: This class corresponds to a file on a disk, or some kind of checkpoint that will be used as the output of a task.
- *Task*: This class is where the computation is done. The *Tasks* consume *Targets* that were created by some other task. Each task define outputs by using the *output()* method.

The behaviour of Luigi is briefly depicted in the fig. 3.6.

3.6 Bitter

In order to download the Tweets which integrate partially the dataset, the tool bitter was used. Bitter is able to automate several actions (e.g. downloading Tweets) by using a Python wrapper over Twitter which adds support for several Twitter API actions [5].
Bitter could be used after installing it and running commands on the UNIX terminal.

3.7 Sefarad

Sefarad [5] is an environment developed to explore, analyse and visualise data.

3.7.1 Architecture

Sefarad is divided in modules:

[5]https://github.com/balkian/bitter

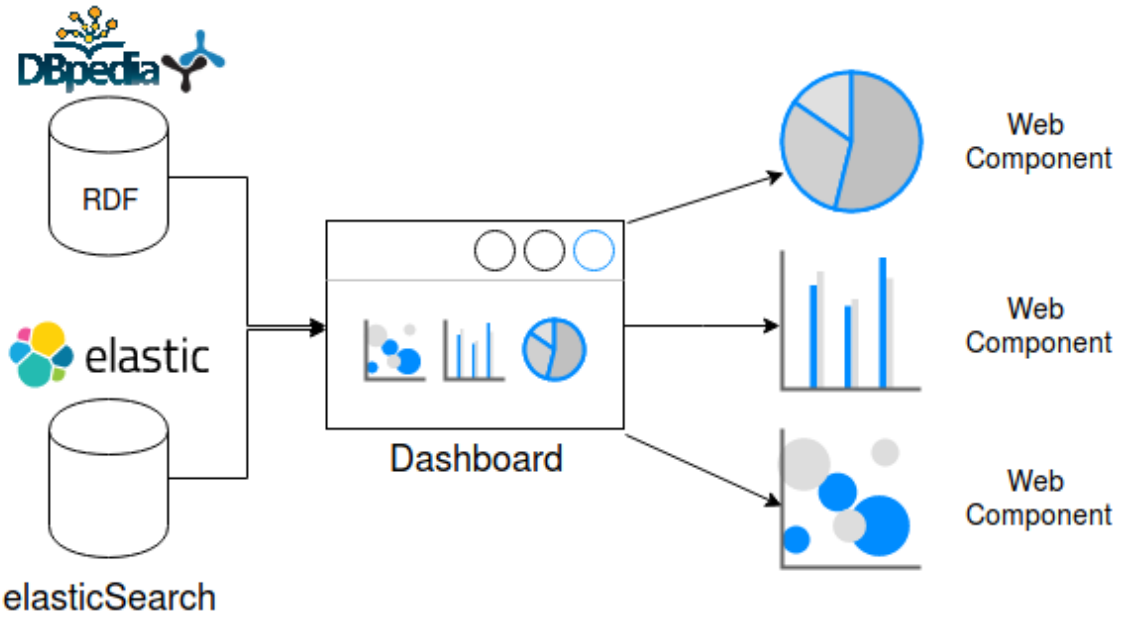

Figure 3.7: Sefarad Architecture [5]

- Visualization: Represent already processed data by using charts. To do so, several dashboards structure the visualization process.
- ElasticSearch: Is the persistence layer which stores the data needed for the visualization process.

The architecture can be easier understood in the fig. 3.7

3.8 Elasticsearch

Elasticsearch is a highly scalable open source search engine with a very powerful analytical engine. The data stored in Elasticsearch is Javascript Object Notation (JSON) formatted. The primary way of interacting with Elasticsearch is via **REST API** [8].

3.8.1 Architecture

This subsection will not explain into detail Elasticsearch. On contrast, a brief description of the architecture will be described and some of the most common terms will also be mentioned.

Even though Elasticsearch is a search engine, it can be understood as a relational database where an index in Elasticsearch is similar to a database that consists of multiple types [16]. Figure 3.8 shows the idea behind. The most common terms used in Elasticsearch are:

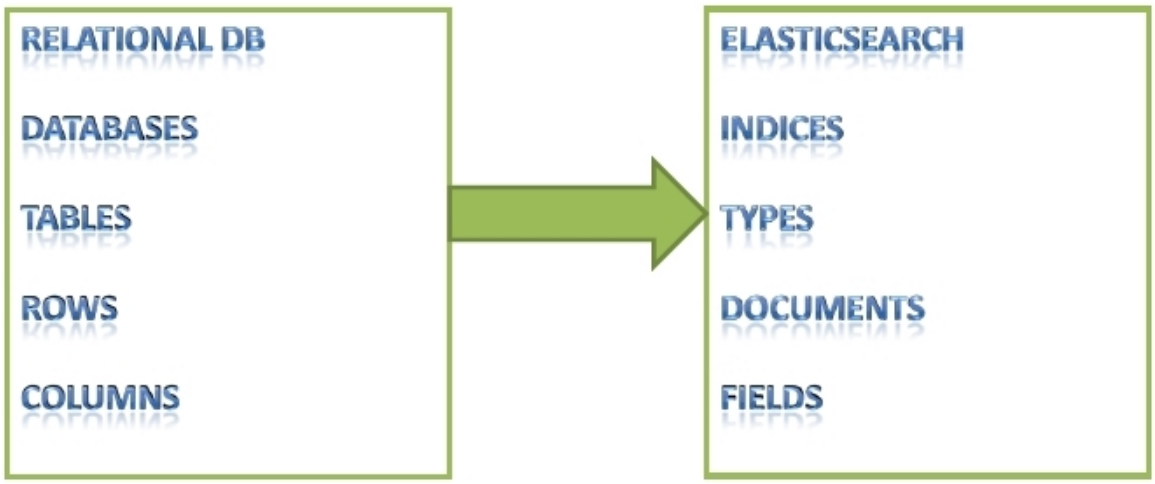

Figure 3.8: Elasticsearch as a Database [16]

- *Node*: An instance of Elasticsearch running on a machine.
- *Cluster*: The name under which one or more nodes are connected.
- *Document*: A JSON object containing the actual data in key-value pairs.
- *Index*: Logical namespace in which Elasticsearch stores data.
- *Doc types*: A class of similar documents. A type consists of a name and a mapping.
- *Shard*: Containers which can be stored on a single node or multiple nodes. A shard can be either primary or secondary. A primary shard is the one where all the operations that change the index are directed. A secondary shard is the one that contains duplicate data of the primary shard and helps in quickly searching the data as well as for high availability; in a case where the machine that holds the primary shard goes down, then the secondary shard becomes the primary automatically.
- *Replica*: Duplicate copy of the data living in a shard for high availability.

CHAPTER 4

Model

4.1 Introduction

The social network Twitter has become extremely popular worldwide. It has developped into a tool that allows anyone to interact directly with people of his interest. Even worldwide leaders, like the president of the United States of America, announce important decisions using Twitter. In other words, Twitters' simplicity and easy-to-use interface has completely transformed the way citizens interact with the society. Altogether with the globalised world that we are living in and the limitless scope that the internet presents makes Twitter a resourceful platform of data.
In this chapter a technical description of the implemented model will be given.
Firstly, an overview of the model shall be presented, followed by the dataset details. Furthermore, the data munging preprocessing will be made clear. Later on, the model building alongside with the classificator options shall be extensively explained, making emphasis on the best classificator results. Finally, the parameter optimization will be briefed altogether with the best founded parameters results.

4.1.1 Overview

In this section a wide view of the system will be given. Figure 4.1 shows the steps followed during the ML stage.

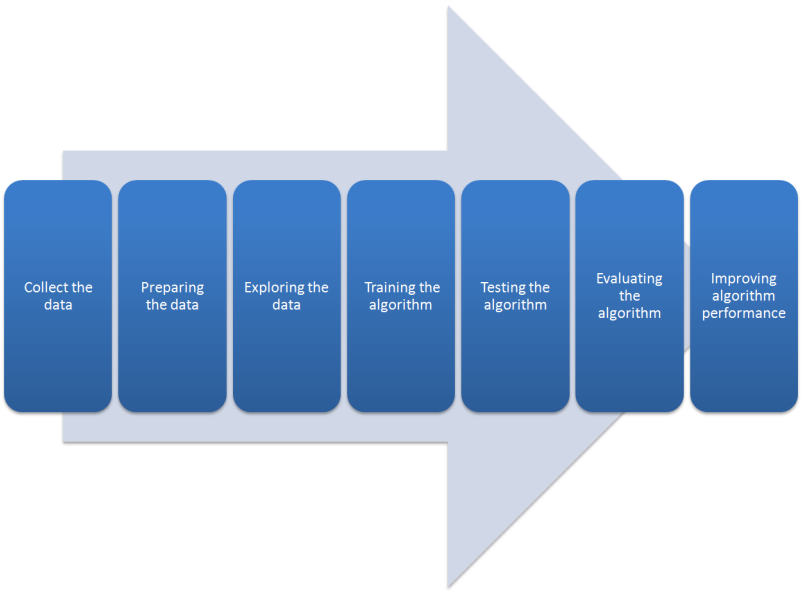

Figure 4.1: The Model [4]

The first two steps are needed to collect the data from the Twitter social network, combine it with more data sources, generate a final *.csv* file and prepare the dataframe (see section 4.2). Once the dataframe is generated, a data exploration and cleaning will be performed (i.e. delete empty tweets, encode categorical variables and so on). It is then when an algorithm is built and train. The algorithm is also known as a classifier and it will be the responsible for predicting sarcasm in new tweets (see section 4.5). Finally, an accuracy report of the classifier will be carried out to evauate the quality of the classifier (seesection 4.6).

4.2 Dataset

This section explains the dataset choice, as well as the main label present in the dataset.

This project is formerly designed to detect sarcasm in spanish written texts, particularly tweets. Therefore, all the datasets used in this project are written in spanish.
The dataset body will contain two rows, the tweet body, i.e. the text, and a binary value expressing the sarcasm nature of that tweet.
The final dataset will be composed of three different datasets.
The initial database [2] consists of 4529 sarcastic tweets and 335 non-sarcastic tweets. At first, the dataset contained only the tweet identifier and the sarcastic label, meaning there was no text present on the dataset. In order to obtain the tweet text, it was necessary to use the bitter tool (see section 3.6). While doing so, many tweet bodies were no longer available to retrieve since the Twitter platform had removed them or the owners had privatized them.

	tweet	ironic
0	Algunas personas sufren en las discos mientras...	1
1	@jacevedoaraya es para sostener el marcador.....	1
2	Alguna de estas imágenes te sacara una sonrisa...	1
3	@_Eurovision2014 en 2013 falta esdm jajajajaja...	1
4	Hooo que buen padre...#Sarcasmo #GH2015	1
5	@JhoynerV ja ja ja ja ja así o más claro, cas...	1
6	@patronbermudez con todo respeto lo principios...	0
7	Gran rapidez todo en la UPO y no iban a ser me...	1
8	¿Que humilde es Simeone no? #ironía #llorón #f...	1
9	¿Alguien se viene a la playa conmigo? #resfria...	1
10	Tantos enamorados que no son novios😵💑💔💗\ny tan...	0

Figure 4.2: Dataset

As a consequence, new sarcastic tweets were provided by a colleague from the Intelligent Systems Group in the UPM.
Even though the numerous amount of sarcastic tweets was a good start, the dataset was completely unbalanced. Consequently, new tweets had to be searched to obtain a properly balanced dataset. Considering the fact that the mayority of the dataset is sarcastic, finding new non-sarcastic tweets was not a difficult duty.
At the end, the result was 5638 sarcastic tweets and 5444 non-sarcastic. That can be rewritten as 49.12% of non-sarcastic and 49.5% of sarcastic tweets.

The fig. 4.2 displays the dataset structure. As explained before, there is one column showing the tweet text and another column showing the sarcastic value of the tweet. The '1' refers to sarcastic, while '0' is non-sarcastic.

4.3 Preprocessing

The preprocessing stage is conceived to dispose of any information that will not be needed during the ML step [29]. The preprocessing was applied individually to each tweet. The changes in the dataset that have been made are:

- *False items*: Since almost half the dataset tweets come from tweeter, it was required to download those tweets from the social network. Some of the captured tweets were false.
- *Categorical values*: Originally the labels of the dataset were either 'True' or 'False'. These two values were encoded into 'True'= 1 and 'False' = 0.
- *Stopwords*: In ML it is a very common practice to delete words which barely provide any information to the text (e.g. 'a', 'I', 'you'). In this project, the list of stopwords was chosen for the spanish language due to the language of the tweets. The list of words can be consulted in [19].
- *Tokenization*:This process consists of splitting a string (tweet) into tokens. Usually a token is the same as a word.
- *Stemming*: This process consists of reducing the tokens into its root. Basically the words are transformed into its root. Stemming was accomplished using a Snowball stemmer.

These steps of preprocessing were all executed defining a custom tokenizer but the categorical value encoding and the false item removal, which were done previously.
It is noteworthy to mention that the custom tokenizer is used later for the feature extraction.
At the beginning of the preprocessing stage, the dataset is split into two parts, a training part and a testing part. The purpose of this split is to train a clasifier using the train sequence and evaluate the accuracy of the classifier using the test sequence. The split is taken randomly. The size of each sequence is 75% for the train sequence and 25% for the test sequence (8311 and 2771). More information on can be found on section 4.6.4.

4.4 Feature Extraction

The feature extraction process consist of reading the text and extracting certain features. Furthermore, those extracted features will be fed to the classifier and finally the model will be completed. As a result, the accuracy of the model will be directly related to the choice

of features. Therefore, it is very important to choose wisely the features to extract.
This section will be committed to extensively explaining the selected features.

The extraction will be performed inside a Pipeline, which will execute each extraction sequentially. In this project, two pipelines are defined, the first aims at extracting the ngrams. With the ngrams extracted from the first pipeline, a second one will extract other features and unify the ngrams with the other extracted features. Even though there are two pipelines defined, the results of the ngrams pipeline are added to the other pipeline. This is done thanks to the Feature Union class provided by Scikit-Learn.

4.4.1 Lexical Features

Usually extracting lexical features involves getting the number of sentences and the number of words in each sentence. However, since this projects' dataset is made out of tweets and the number of characters in a tweet is limited, I have considered that each tweet contains only one sentence. As a consequence, the lexical feature extraction becomes very simple. The extraction consists of tokenizing each sentence and stemming each token (or word) to get the root. At the same time, the words belonging to the spanish stop list are removed (see section 4.3 for more details). In section 4.4.5 is explained what happens with the extracted words in this stage.

4.4.2 Syntactic Features

Esto no aparece en el código

The syntactic feature extraction consists of counting the number of nouns, adjectives, verbs, adverbs, conjunctions, pronouns and numbers present in each tweet. In section 4.4.5 is explained what happens with the extracted words in this stage.

4.4.3 Ngram Features. Pipeline

Ngram extractions consist of grouping the words into bag of words. If we group the words individually then a sentence with five words will be considered as a five words sentence. If we group two words together then a sentence of six words will be considered as a three word sentence. So each pair of words are understood by the estimator as one.
Since it is very difficult to guess what is the optimum bag of words (i.e. number of words) CountVectorizer method provided by the Scikit-Learn class allows us to define a range. Finally, to select the optimum range, a Grid Search (GS) will be performed (see section 4.7.1).

4.4.4 Latent Dirichlet Allocator (LDA) Features

LDA [1] is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar.
In other words, LDA extraction can be used to extract how many topics are being spoken about in the dataset. This can be observed by analyzing each words' presence.
The main parameter of this statistical model is the number of topics. Similarly as with the ngrams features, a GS will be performed to find the optimum number of topics (see section 4.7.1).

4.4.5 Frequency Inverse Document Frequency (TFIDF)

TFIDF [2] is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Particularly, the TFIDF recognizes those words that are rare in the corpus but may be of great importance. The TFIDF uses a word as an argument and outputs the inverse frequency of that word.
The equation used is [18]:

$$tf_{t,d} = \frac{count(t)}{count(alltermsindocument)} \tag{4.1}$$

with

$$idf_t = log(\frac{N}{df_t}) \tag{4.2}$$

and

$$tfidf_{t,d} = tf_{t,d} \times idf_t \tag{4.3}$$

The number of occurences of a word in a complete document is computed with eq. (4.1). Equation (4.2) represents the *idf* of a word. That amount serves as how much information a word provides. As explained before, terms with less frequency are considered to provide more information to the whole document as more common terms. Finally, eq. (4.3) computes the weight of a word.

4.5 Classifiers

This section will explain extensively the classifiers used for the learning process.
Classification [31] is the task of predicting the class to which an object, known as pattern, belongs. The pattern is assumed to belong to one and only one among a number of a priori

[1] https://en.wikipedia.org/wiki/Latent_Dirichlet_allocation
[2] https://en.wikipedia.org/wiki/Tf\begingroup\let\relax\relax\endgroup[Pleaseinsert\PrerenderUnicode{}intopreamble]idf

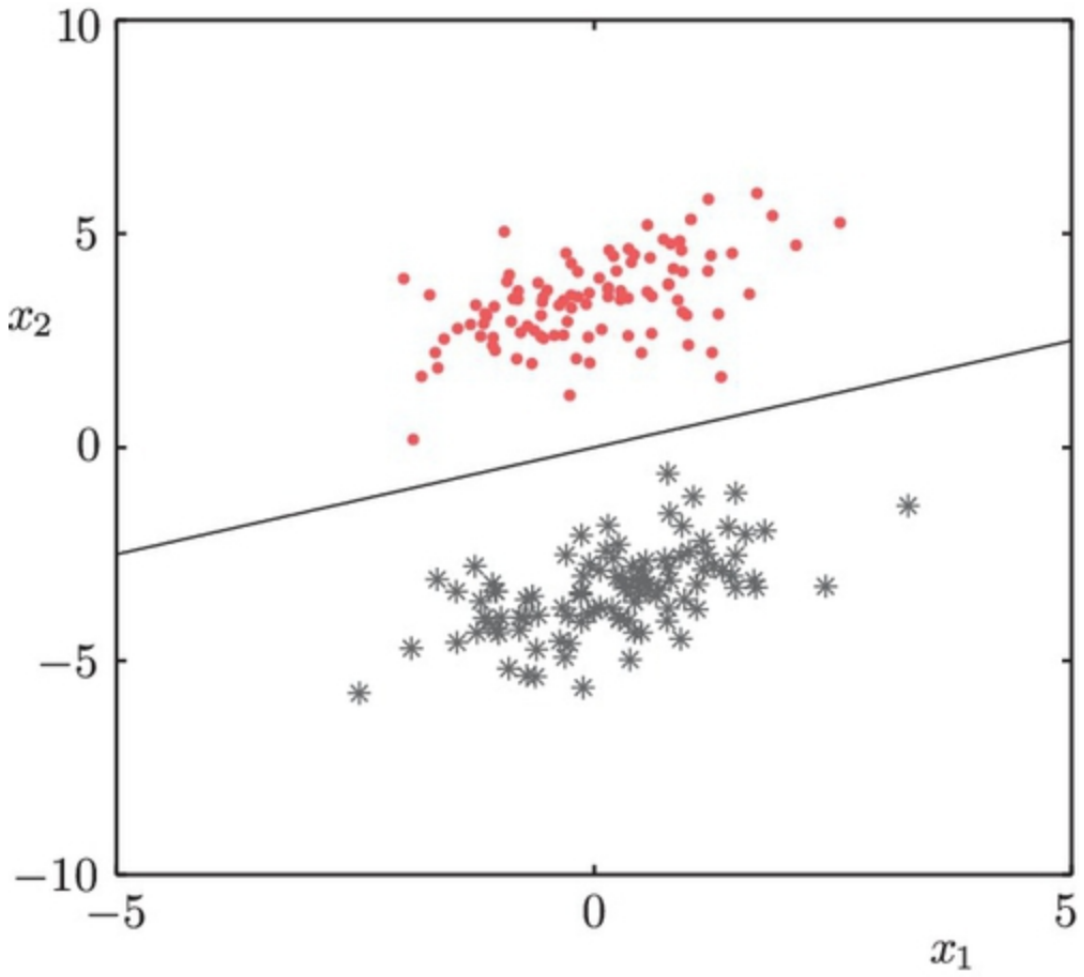

Figure 4.3: Classifier [31]

known classes. Each pattern is uniquely represented by a set of measurements, known as features. One of the early stages in designing a classification system is to select an appropriate set of feature variables. These should "encode" as much class-discriminatory information, so that, by measuring their value for a given pattern, to be able to predict, with high enough probability, the class of the pattern. Figure 4.3 is an example of two linearly separable classes, where a straight line can separate the two classes.
In this project, there will be three classifiers.

4.5.1 Naive Bayes Classifier (NB)

The NB classifier is a typical and popular example of a suboptimal classifier. The basic assumption is that the components (features) in the feature vector are statistically independent; hence, the joint Probability Density Function (PDF) can be written as a product

of l marginals [31]:

$$p(x|\omega_i) = \prod_{k=1}^{l} p(x_k|\omega_i) \tag{4.4}$$

Considering that both ω_i and x are Gaussian variables, it is only necessary to compute the mean and the variance for each pair of variables. That leads to a total of $2 \times l$ unknown variables for a subclass. Computationally speaking, $2 \times l$ complexity is achievable in a reasonable amount of time for a large l. This is the great advantage of the NB.

The NB classifier was defined setting the parameter $alpha = 0.01$. Nevertheless, it is not very important what parameters to set when defining the classifier since a GS will be performed with the purpose of discovering the best classifier parameters. (See section 4.7.1) In this project, the results obtained for the NB with no parameter optimization can be observed in table 4.1. The quality of a classifier will be established using the *F1-Score* field

Variable	Precision	Recall	F1-Score	Support
0	0.81733	0.78781	0.80230	1329
1	0.81074	0.83773	0.82401	1442
avg/total	0.81390	0.81379	0.81360	2771

Table 4.1: Naive Bayes Classifier Classification Report

from the classification report. For more details about this precedent see section 4.6.

4.5.2 Support Vector Machine (SVM)

Support vector machine [32] is a supervised learning algorithm that can be used for classification and regression. It is able to classify data linearly and nonlinearly using kernel methods. Each data point in the training dataset is labeled, as it is supervised learning, and mapped to the input feature space, and the aim is to classify every point of new data to one of the classes. A data point is an N dimension number, as N is the number of features, and the problem is to separate this data using N-1 dimensional hyperplane and this is considered to be a linear classifier.

The SVM was defined setting as parameters $C = 1$ and $kernel =$ linear. In other words, the initial SVM model was the linear SVM model.
With such parameters, the results can be observed in table 4.2. The quality of the classifier was established following the *F1-Score* field from the classification report. For more details see section 4.6.

Variable	Precision	Recall	F1-Score	Support
0	0.92590	0.92387	0.92488	1366
1	0.92614	0.92811	0.92712	1405
avg/total	0.92602	0.92602	0.92602	2771

Table 4.2: Support Vector Machine Classification Report

4.5.3 K-Nearest Neighbors (k-NN)

k-NN is based on the idea that samples that are close with respect to a predefined distance metric are also similar, so they can share their peculiar features. To measure that distance, a distance function is defined. Usually, that distance function is the Minkowski metric [13]. For each data point in the out-sample data, we calculate its distance from all data points in the in-sample data. Each data point has a vector of distances and the K distance which is close enough will be selected and the final decision about the class of the data point is based on a weighted combination of all k neighborhoods [32].

The k-NN was defined setting as parameters $neighbors = 7$.
The results can be observed in table 4.3. The quality of the classifier was established following the *F1-Score* field from the classification report. For more details see section 4.6.

Variable	Precision	Recall	F1-Score	Support
0	0.79561	0.76940	0.78229	1366
1	0.78276	0.80783	0.79510	1405
avg/total	0.78909	0.78888	0.78878	2771

Table 4.3: K-Nearest Neighbors Classification Report

4.5.4 Logistic Regression (LR)

Logistic Regression is a classification method that is based on the probability of a sample belonging to a class. It consists of maximum-entropy classification. Mathematically, the

aim is to compute the minimum of eq. (4.5) [24].

$$min_{w,c}\frac{1}{2}w^T w + C\sum_{i=1}^{n} log(exp(-y_i(X_i^T w + c)) + 1) \tag{4.5}$$

The LR classifier was defined setting the parameter $C = 1$.
For such a parameter, the results obtained can be observed in table 4.4. The quality of the classifier was established following the *F1-Score* field from the classification report. For more details see section 4.6.

Variable	Precision	Recall	F1-Score	Support
0	0.90007	0.90074	0.90040	1360
1	0.90426	0.90361	0.90393	1411
avg/total	0.90220	0.90220	0.90220	2771

Table 4.4: Logistic Regression Classification Report

4.6 Quality of a Classfier

In order to ensure a sufficient classifier accuracy, precision is not always the best indicator. Let us consider a binary class dataset where 99% of the data belongs to one class and only 1% of the data belongs to the other class. Now, if a classifier were to always predict the majority class for every data point, it would have 99% accuracy. But that would not mean that the classifier is performing well. To have a better idea of the classifiers' efficiency we first must define some parameters [10]:

- *True Positive (TP)*: Cases where the actual and the predicted classes are both positive.
- *True Negative (TN)*: Cases where the actual and the predicted classes are both negative.
- *False Positive (FP)*: Cases where the actual class is negative but the predicted class is positive.
- *False Negative (FN)*: Cases where the actual class is positive but the predicted class is negative.

To better understand these definitions let us imagine a classifier that predicts a sarcastic when in fact that tweet is not sarcastic, that would be a false positive. On the other hand, if the classifier predicts that a tweet is not sarcastic when in reality is sarcastic, that would be a false negative.

4.6.1 Precision

Precision is defined as the ratio of number of positive cases that were correct to all the cases that were identified as positive [10]. Mathematically:

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \tag{4.6}$$

4.6.2 Recall

Recall is defined as the ratio of the number of positive cases that were identified to the all positive cases present in the dataset [10]. Mathematically:

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{4.7}$$

4.6.3 F1 Score

is the harmonic average of the precision and recall. It is a metric that conveys the balance between precision and recall [10]. Mathematically:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.8}$$

Since the F1 score takes into account both the precision and the recall (see eq. (4.8)), it is the most suitable for determining a binary classifiers' quality. In this project, a tweet can either be sarcastic or non-sarcastic. Therefore, F1 score report is the appropiate quality measure.

4.6.4 Model Creation

The fig. 4.4 depicts the methodology followed for training and evaluating the classifier. Firstly, the dataset is split into two sets, the training and the test. (In the fig. 4.4, there is another subdivision called *val* but that will not be necessary in this project). Once the split is done, a model is constructed. Finally, the quality of that model is evaluated using the test set and applying the F1 score.

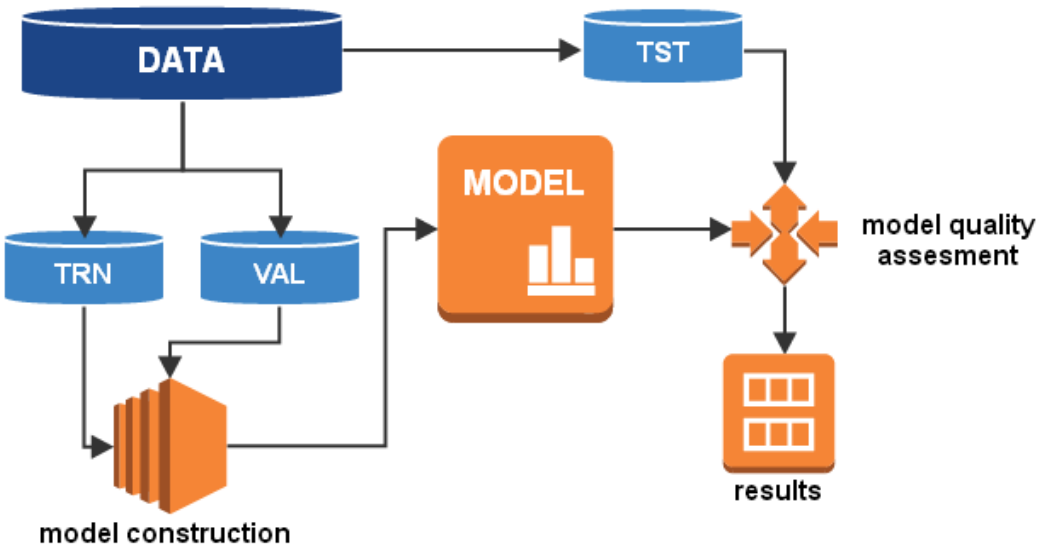

Figure 4.4: Model Construction [1]

4.7 Parameter Optimization

This section will cover how was the parameter optimization performed and what were the results obtained for each classifier.

To understand the parameter optimization, first we must comprehend how is the classifier built in this project.

The simplest definition of text classification is that it is a classification of text based on the content of that text. Now, in general, all the machine learning methods and algorithms are written for numeric features/variables. One of the most important problems with text corpus is how to represent text as numeric features [24]. That stage is accomplished by making use of the pipeline described in section 4.4. Depending on how the feature extraction pipeline was created, the parameters of the final classifier change. Therefore, the classifier is assembled by both the pipeline described in section 4.4 and one of the classifiers chosen in section 4.5.

In order to find the right classifier parameters, it must be carried out a parameter search for the pipeline parameters and the classifier parameters.

From section 4.4, the pipeline extracts three main features; ngrams, words (custom tokenizer) and topics (LDA). From these features, the only parameters that will be searched are the number of topics from LDA and the number of ngrams.

From section 4.5, the parameters searched will depend on the type of classifier. On the following subsections from this section, the topics will be covered. The parameter search

will be done using the GridSearchCV from the Scikit Learn class. The principle of the GridSearchCV is:

1. Define a dictionary of parameters to change.
2. Fit the grid search with the train dataset (this process can take a very long time).
3. Print best score alongside with the best parameters.
4. Print the metrics accuracy score by testing the classifier with the best parameters found. For this, it is required a test dataset.

4.7.1 Grid Search

A grid search is an exhaustive search through a manually-specified subset of the hyperparameter space. The grid search algorithm requires a performance metric, such as cross-validation error or validation-set error to evaluate the best possible parameter [17]. The grid search method comes from the Scikit Learn class.
A grid search was optimizied for each classifier to get the best performance.

4.7.1.1 Naive Bayes Classifier

- **Parameters to optimize**:
  1. *Alpha*. Belongs to classfier
  2. *Number of topics*. Belongs to pipeline
- **Results**

Variable	Precision	Recall	F1-Score	Support
0	0.96481	0.96765	0.96623	1360
1	0.96873	0.96598	0.96735	1411
avg/total	0.96680	0.96680	0.92602	2771

Table 4.5: Naive Bayes Classifier Classification Report Optimized

4.7.1.2 Support Vector Machine

- **Parameters to optimize**:

1. *C*. Belongs to classfier
2. *Number of topics*. Belongs to pipeline
3. *Number of ngrams*. Belongs to pipeline

- **Results**

Variable	Precision	Recall	F1-Score	Support
0	0.91131	0.90662	0.90896	1360
1	0.91044	0.91495	0.91269	1411
avg/total	0.91086	0.91086	0.91086	2771

Table 4.6: Support Vector Machine Classification Report Optimized

4.7.1.3 K-Nearest Neighbors

Since the results of the k-NN were very low (see table 4.3) there was no grid search performed to predict the best classifier parameters. Furthermore, k-NN it is not a recommended classifier for the porpose of the project.

4.7.1.4 Logistic Regression

- **Parameters to optimize**:

1. *Number of topics*. Belongs to pipeline
2. *Number of ngrams*. Belongs to pipeline

- **Results**

Variable	Precision	Recall	F1-Score	Support
0	0.92819	0.90361	0.91573	1359
1	0.90953	0.93272	0.92098	1412
avg/total	0.91868	0.91844	0.91841	2771

Table 4.7: Logistic Regression Classification Report Optimized

CHAPTER 5

Sarcasm Analysis

5.1 Introduction

In this chapter, the architecture of the project will be discussed, as well as the implementation details involving its architecture. The idea of this chapter is to clarify how does the whole system work and later to explain every block separately.
The technologies used in this project can be found with more detail in chapter 3.

5.2 Architecture

No me gusta

This section is committed to introducing the system architecture alongside with the subsystems that make it possible for the project to work.
The subsystems needed for this project are:

1. *Entry system*: This system offers a REST interface

CHAPTER 6

Case study

6.1 Introduction

In this chapter we are going to describe a selected use case. This description will cover the main Wool features, and its main purpose is to completely understand the functionalities of Wool, and how to use it.

6.2 Rule edition

...

CHAPTER 7

Conclusions and future work

In this chapter we will describe the conclusions extracted from this project, and the thoughts about future work.

7.1 Conclusions

7.2 Achieved goals

N1

7.3 Future work

- F1

Bibliography

[1] Classification models. http://algolytics.com/tutorial-how-to-determine-the-quality-and-correctness-of-classification-models-part-1-introduction/.

[2] Hacia la detección de ironía en textos cortos. `http://ru.ffyl.unam.mx/bitstream/handle/10391/4787/VII_CoLiCo_G_Jasso_Mesa_3_2015.pdf?sequence=2&isAllowed=y`.

[3] Luigi documentation. ~ https://media.readthedocs.org/pdf/luigi/latest/luigi.pdf.

[4] *Regression Analysis with R.*

[5] Sefarad architecture. https://sefarad.readthedocs.io/en/latest/sefarad.html.

[6] Spark dataframes and ml pipelines. https://www.slideshare.net/databricks/dataframes-and-pipelines.

[7] What is senpy? https://senpy.readthedocs.io/en/latest/senpy.html.

[8] Abhishek Andhavarapu. *Learning Elasticsearch.* Packt Publishing, 2017.

[9] Oscar Araque. Design and Implementation of an Event Rules Web Editor. Trabajo fin de grado, Universidad Politécnica de Madrid, ETSI Telecomunicación, July 2014.

[10] Rounak Banik. *Hands-On Recommendation Systems with Python.* Packt Publishing, 2018.

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

[12] Giuseppe Bonaccorso. *Machine Learning Algorithms.* "Packt Publishing", 2017.

[13] Giuseppe Bonaccorso. *Machine Learning Algorithms - Second Edition.* "Packt Publishing", 2018.

[14] Andrew Bruce; Peter Bruce. *Practical Statistics for Data Scientists.* "O'Reilly Media, Inc", 2017.

[15] Giuseppe Ciaburro. *MATLAB for Machine Learning.* Packt Publishing, 2017.

[16] Bharvi Dixit. *Elasticsearch Essentials.* Packt Publishing, 2016.

[17] Dipanjan Sarkar; Raghav Bali; Tamoghna Ghosh. *Hands-On Transfer Learning with Python.* Packt Publishing, 2018.

[18] Josh Patterson; Adam Gibso. *Deep Learning.* " O'Reilly Media, Inc.", 2017.

[19] Nitin Hardeniya. *NLTK Essentials.* "Packt Publishing", 2015.

[20] Aditya Joshi, Pushpak Bhattacharyya, and Mark J Carman. Automatic sarcasm detection: A survey. *ACM Computing Surveys (CSUR)*, 50(5):73, 2017.

[21] Russell Jurney. *Agile Data Science 2.0, 1st Edition.* "O'Reilly Media, Inc", 2017.

[22] Mikhail Khodak, Nikunj Saunshi, and Kiran Vodrahalli. A large self-annotated corpus for sarcasm. *arXiv preprint arXiv:1704.05579*, 2017.

[23] Nitin Hardeniya; Jacob Perkins; Deepti Chopra; Nisheeth Joshi; Iti Mathur. *Natural Language Processing: Python and NLTK.* " Packt Publishing", 2016.

[24] Nitin Hardeniya; Jacob Perkins; Deepti Chopra; Nisheeth Joshi; Iti Mathur. *Natural Language Processing: Python and NLTK.* Packt Publishing, 2016.

[25] Benjamin Bengfort; Rebecca Bilbro; Tony Ojedar. *Applied Text Analysis with Python.* " O'Reilly Media, Inc", 2018.

[26] Tomáš Ptáček, Ivan Habernal, and Jun Hong. Sarcasm detection on czech and english twitter. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 213–223, 2014.

[27] Charu C. Aggarwal; Chandan K. Reddy. *Data Clustering.* "Chapman and Hall/CRC", 2013.

[28] J. Fernando Sánchez-Rada. Design and Implementation of an Agent Architecture Based on Web Hooks. Master's thesis, ETSIT-UPM, 2012.

[29] Bhargav Srinivasa-Desikan. *Natural Language Processing and Computational Linguistics.* "Packt Publishing", 2018.

[30] Sinan Ozdemir; Divya Susarla. *eature Engineering Made Easy.* "Packt Publishing", 2018.

[31] Sergios Theodoridis. *Machine Learning.* Academic Press, 2015.

[32] Dr. Param Jeet; Prashant Vats. *Learning Quantitative Finance with R.* Packt Publishing, 2017.