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 detetction on Twitter

NOMBRE DEL AUTOR

20XX

TRABAJO DE FIN DE GRADO

Título:	Detección de Sarcasmo en Twitter
Título (inglés):	Sarcasm detetction 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
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE GRADO

Sarcasm detetction on Twitter

Enero 2019

Resumen

Palabras clave:

Abstract

Keywords:

Agradecimientos

A Gauss

Contents

List of Figures

CHAPTER 1

Introduction

1.1 Context

Style modified by O. Araque [7], J. Fernando[20], and many others at GSI.

1.2 Project goals

- 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 [16]. 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 [14]

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 [14]:

- *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 [14].

[1] www.thefreedictionary.com

2.2.3.1 Rule-based Approaches [14]

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 [14]:

- *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 [14]:

- *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 [19]:
  - *Description*: This paper presents a Machine Learning (ML) approach to sarcasm detection on Twitter in two languages - English and Czech. The classification was carried out by using Support Vector Machine (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 [1]:
  - *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

Metemos como hemos sacado los tweets?

In this project two main technologies have been used: NLP and ML. The implementation of these tools has been performed in a combined fashion.

NLP has allowed me to transform words into tokens, find syntactical connections between tokens in the same tweet and, finally, construct a numerical vector. ML granted me a way 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 what I am analyzing are Tweets, which contain sentences. Hence, the information that I will use to train the ML algorithm will come in the form of text. According to [8], at one extreme NLP could be as simple as counting

word frequencies to compare different writing styles. At the other extreme, NLP involves "understanding" complete human utterances, at least to the extent of being able to give useful responses to them.

After mentioning its importance, I will clarify what libraries have I used and how.

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]

3.2.1.2 Application

NLTK has provided the following tools to this project:

- *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 [17]. Particularly, I have chosen the SnowballStemmer because it supports stemming in 13 non-English languages, especially Spanish.

- *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 [17]. 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 [17].

- *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 [18]. 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

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

collecting each word feature (i.e. noun, verb, adj, adv) and building up a dictionary with the word features' stats.

These tasks were coordinated by using pipelines.

No me termina de convencer este párrafo. Qué quiero decir?

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. The learning process can be classified in three [12]:

- *Supervised learning*: Widely used for classification applications. A model is trained to make predictions and is corrected whenever the model makes a mistake.
- *Unsupervised learning*: Extract general rules. A model deduces recognizable structures in the input data-set. Result is unknown
- *Semi-supervised learning*: Part of the data-set result is known.

In contrast with ML, NLP is a way to analyze and derive the meaning of human language. To do so, it 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 Natural Language Processing 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.

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.

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

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 [15]. Image 3.1[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 [21]. 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 [10]. 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) [10]. The MultinomialNB has been used as a classifier fed into the pipeline. Its results can be seen.

  añade referencia tabla resultados cuando lo pongas

  .

- *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

[3] https://www.slideshare.net/databricks/dataframes-and-pipelines
[4] https://en.wikipedia.org/wiki/Support_vector_machine

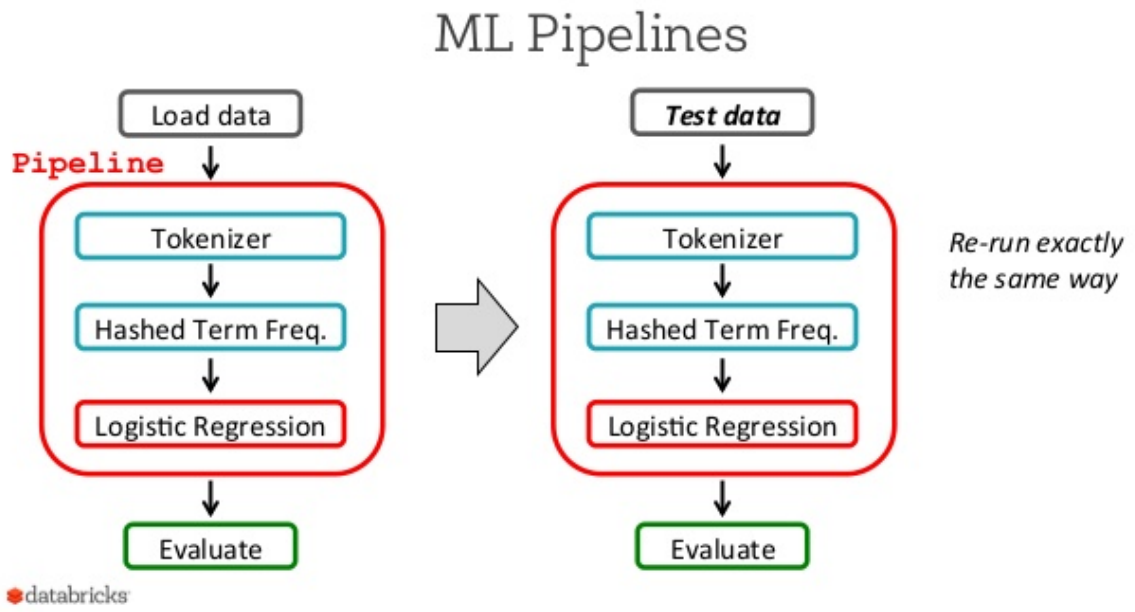

Figure 3.1: A Pipeline. [4]

- *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 [9].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 [11]. Its results can be seen.

  añade referencia tabla resultados cuando lo pongas

  .

3.4 Senpy

Senpy [5] 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).

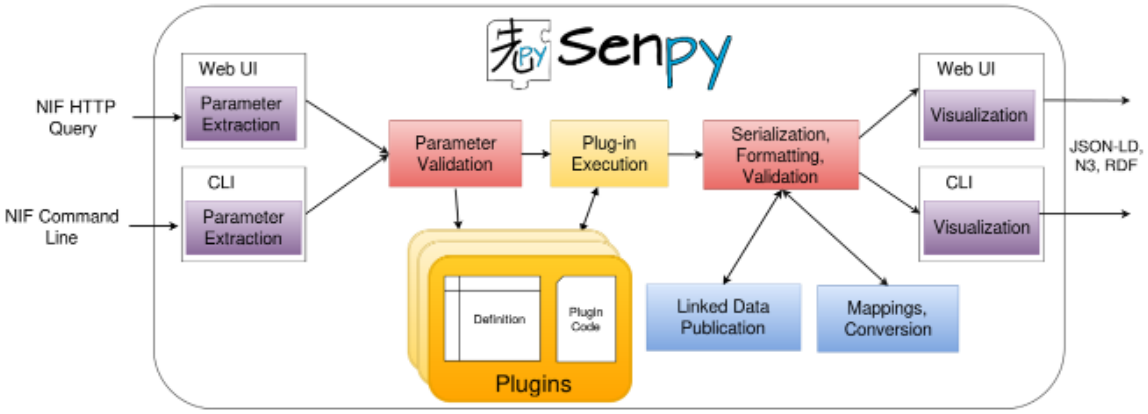

Figure 3.2: Senpy's Architecture [5]

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 3.2 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 [2]. 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:

- *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.

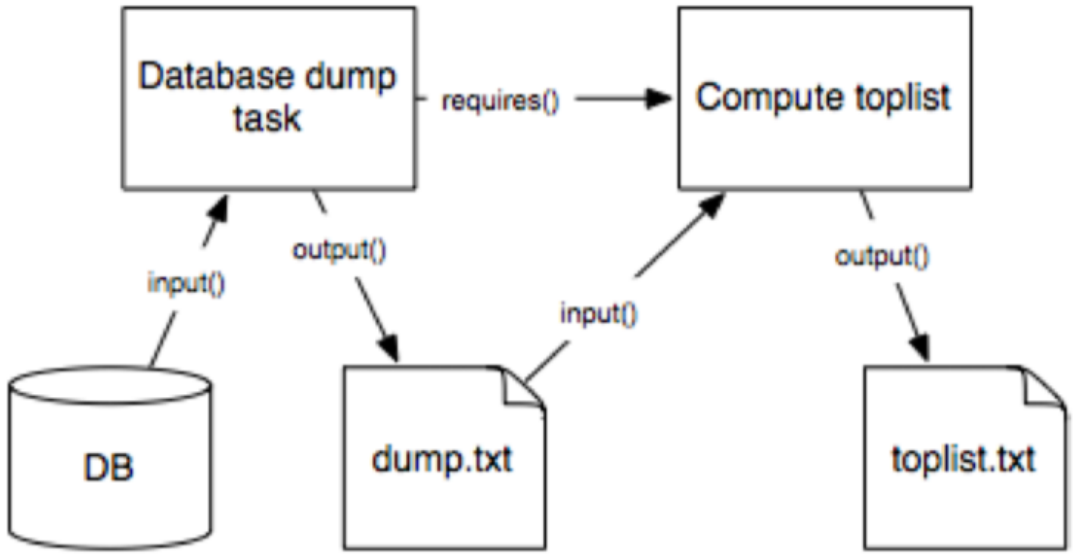

Figure 3.3: Task and Target illustration. [2]

The behaviour of Luigi is briefly depicted in the fig. 3.3.

3.6 Bitter

In order to download the Tweets which integrate my datataset I used Bitter. 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].

3.7 Sefarad

Sefarad [3] is an environment developed to explore, analyse and visualise data.

3.7.1 Architecture

Sefarad is divided in modules:

- 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. 3.4

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

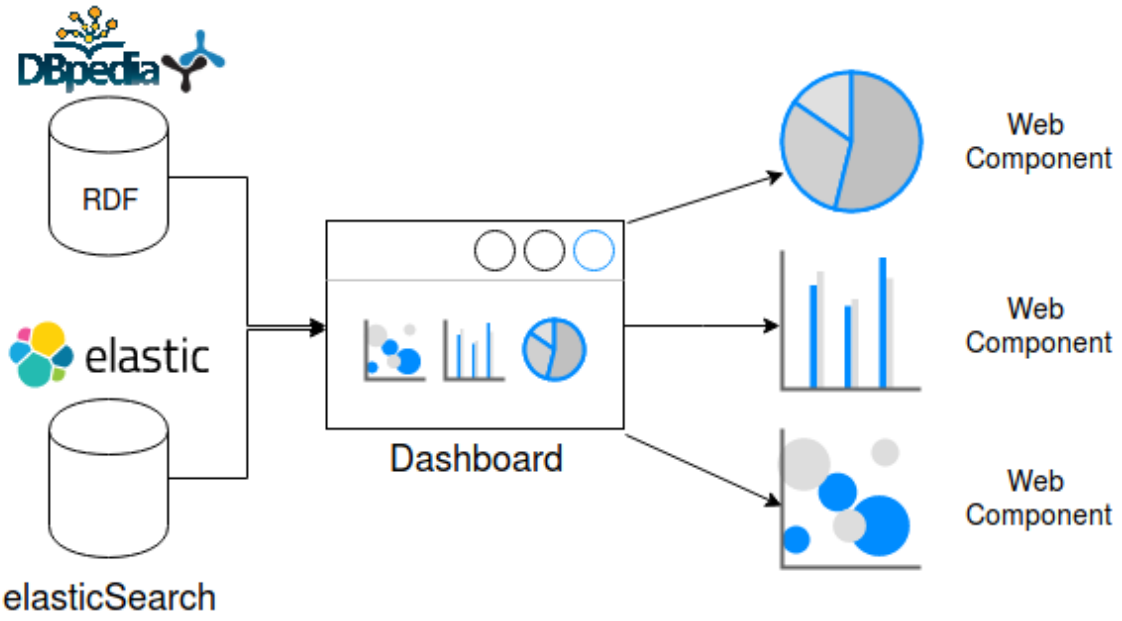

Figure 3.4: Sefarad Architecture [3]

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** [6].

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 [13]. Figure 3.5 shows the idea behind. The most common terms used in Elasticsearch are:

- *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.

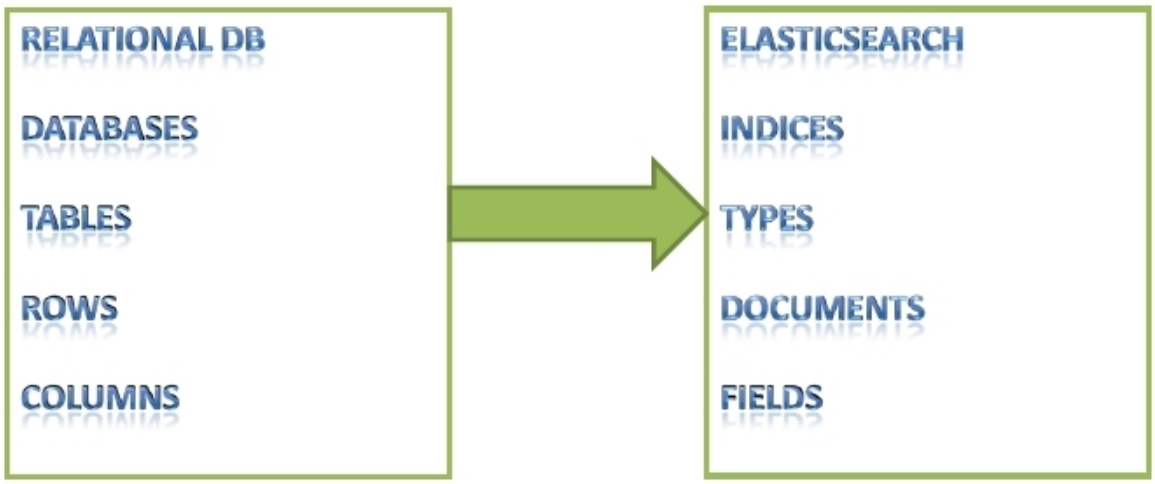

Figure 3.5: Elasticsearch as a Database [13]

- *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

Requirement Analysis

4.1 Introduction

4.2 Use cases

4.2.1 System actors

CHAPTER 5

Architecture

5.1 Introduction

In this chapter, we cover the design phase of this project, as well as implementation details involving its architecture. Firstly, we present an overview of the project, divided into several modules. This is intended to offer the reader a general view of this project architecture. After that, we present each module separately and in much more depth.

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] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[19] 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.

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

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