Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Integrated system based on the automation of the extraction and characterization, using natural processing language, of user ratings in sales platforms

Estudante: Pablo Durán de las Heras
Dirección: José Carlos Dafonte Vázquez
Ángel Gómez García

A Coruña, xuño de 2022.

Dedicatoria

Acknowledgements

Firstly, I would like to show gratitude to my tutors José Carlos Dafonte Vázquez and Ángel Gómez García for their time in mentoring and guiding this project. More personally, I would like to show my gratitude to my close friends for their help when developing the project. Finally, I could not finish these acknowledgements without expressing my gratitude to my parents and my sisters for the great support through all these years.

Abstract

At present, more and more importance is given to what people think, what they think and what are your preferences. With the rise of social networks and online stores, these data are accessible more than ever in a simple and global way, so it is increasingly important to know manage and analyze all this information. Thus, it is not surprising that over time more companies are focusing their interest on sentiment mining, which allows them to identify possible business opportunities, maintain a good reputation of the company in the networks or improve the marketing your products. For all these reasons, we are interested in building a tool capable of extracting from all the reviews that exists throughout the Internet, what is the opinion of a product/service (its most praised characteristics, its greatest shortcomings...), so that, through a good analysis, we can extract important information in an automated way.

This paper will discuss how to build a tool capable of analyzing and retrieving reviews from the World Wide Web. Building two main systems, one for extracting reviews with the use of scraping tools and another for analyzing sentiment in texts with the use of machine learning and natural language processing.

Resumo

Na actualidade cada vez dáselle máis importancia a qué opinan as persoas, qué é o que pensan e cales son as suas preferencias. Coa subida da popularidade das redes sociais e as tendas online, estes datos estan máis accesibles que nunca, de manera sinxela e global, co que cada vez é máis importante saber xestionar e analizar toda esa información. Así, non é de extrañar que co tempo máis compañías estén centrando o seu interés na minaría de sentimentos, que lles permite posibles oportunidades de negocio, mantener unha boa reputación de empresa nas redes e mellorar o marketing.

Por todo isto, interésanos construir unha ferramenta capaz de extraer de todas as reseñas que existen ao longo de internet, cal é a opinión que se ten dun produto/servizo, de forma que, a través dunha boa análise podamos obter información de importancia de maneira automatizada.

Neste traballo falarase de como construir unha ferramenta capaz de analizar e obter as reseñas da World Wide Web. Construindo un sistema de extracción de reviews co uso de ferramentas de scraping e outro de análise de sentimentos en textos co uso de aprendizaxe automático e procesamento da linguaxe natural.

Keywords:

- Sentiment analysis
- Feature extraction
- Scraping
- Natural language processing
- Artificial intelligence

Palabras chave:

- Análise de sentimentos
- Extración de características
- Scraping
- Procesamento da linguaxe natural
- Intelixencia artificial

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sentiment analysis or opinion mining is the study of opinions, feelings and emotions expressed in a text.

Much of the existing research on textual information processing has been focused on the mining and retrieval of factual information [1], e.g., Information retrieval, Web search, Text classification, Text clustering, and many other text mining and natural language processing tasks. Little work had been done on the processing of opinions until only recently. Yet, opinions are so important that whenever we need to make a decision we want to hear others' opinions. This is not only true for individuals but also true for organizations...

One of the main reasons for the lack of study on opinions is the fact that there was little opinionated text available before the World Wide Web. Before the Web, when an individual needed to make a decision, he or she typically asked for opinions from friends and families. When an organization wanted to find the opinions or sentiments of the general public about its products and services, it conducted opinion polls, surveys, and focus groups. However, with the Web, especially with the explosive growth of the user-generated content in the past few years, the world has been transformed.

The Web has dramatically changed the way that people express their views and opinions. They can now post reviews of products at merchant sites and express their views on almost anything in Internet forums, discussion groups, and blogs, which are collectively called the user-generated content [2].

However, finding opinion sources and monitoring them on the Web can still be a formidable task because there are a large number of diverse sources, and each source may also have a huge volume of opinionated text (text with opinions or sentiments). In many cases, opinions are hidden in long forum posts and blogs.

Building a sentiment analysis system is a complex process in which 5 parts are essential:

- Data collection: It is defined by obtaining a large number of opinions on a certain topic in particular.

- Pre-processing of the text: Part especially important since during this phase all the information that is of no interest to the system is eliminated, the parts with negative contexts are differentiated from the positive ones and lemming or stemming is used, with this the text is simplified and the fragment the text to aid the sentiment classifier in analyzing it [3].
- Characteristics extraction: This part consists of the extraction of those characteristics that are most important within the topic to be analyzed. A statistical analysis makes it possible to obtain these characteristics in a quick and generic way and then to be able to remove those misidentified characteristics [4].
- Classification of sentiment: The main step in which, using some classification system, the sentiment of the text is obtained, positive or negative [5].
- Visualization of the result: The objective of any system is to be able to present information to the user in an accessible way, in this part the data obtained is shown in the form of graphs in order to understand the information obtained [6].

Thus, the great challenge that arises in this field is to be able to automate both the extraction of these opinions and their subsequent analysis in order to obtain their sentiment and the most relevant keywords for a certain topic, and thus finally visualize all this information in a understandable and useful way.

1.1 Objectives

The main objective of this work is the study of the sentiment analysis problem and the implementation of a web application capable of extracting and visualizing the required sentiment data.

The specific objectives are the following:

- **Design and implementation of a sentiment classification system.**

  Using machine learning and natural language processing this system should be able to classify any text in one of the following classes:

  - Positive: Text that expresses a positive sentiment about the subject.
  - Negative: Text that expresses a negative sentiment about the subject.

- **Design and implementation of a REST service for the sentiment classification system.**

  For an easy use and interconnection with other services an API REST should be available exposing the main functionalities of the sentiment classification system.

- **Design and implementation of a data extraction system.**

  To be able to facilitate the analysis of any subject in the WWW a system capable of scraping any website to retrieve those text that express a sentiment about a certain topic has to be built. This service is important to populate a database with reviews that later can be analyzed by the sentiment classifier.

- **Design and implementation of a REST service data extraction system.**

  In the same way that an API REST was needed to expose the functionalities of the classifier an API REST to expose the scraping functionalities has to be built to interconnect both systems.

- **Design and implement a user web interface.**

  The final step is to build a user interface capable of visualize the gathered data and make easy to understand the results. Thanks to the REST API services created the data and functionalities should be already available to consume.

1.2 Structure of the degree thesis

According to the objectives, this thesis structures the content in a specific order, namely:

- Chapter 1. **Introduction**: This chapter consists in a short introduction of what is going to be explained in this work and also the main objectives.
- Chapter 2. **State of the art**: This chapter covers the analysis of the sentiment analysis problem while drawing an image of the projects and companies around this matter.
- Chapter 3. **Planning and methodology**: This chapter explains the engineering methodology and states the project planning.
- Chapter 4. **Technologies and tools**: This chapter showcases the different technologies and tools used in the project.
- Chapter 5. **Analysis**: This chapter analyzes the main purpose of the project and its architecture.
- Chapter 6. **Development**: This chapter explains the development organization at the same time that the procedures to achieve the different goals to finish the project.
- Chapter 7. **Testing**: This chapter is focused in the different tests made for both back-end and front-end. From unit to integration testing and also end to end.

- Chapter 8. **Results** : This chapter summarizes the main results according to the objectives.
- Chapter 9. **Conclusions and future research**: This chapter includes the conclusion of the project and enumerates some of the future research lines.

Chapter 2

State of the art

In this chapter we will study the different ways in which sentiment mining has been covered, as well as examples of applications based on sentiment analysis.

2.1 Supervised Learning

Sentiment classification can easily be treated as a supervised learning classification problem where there are two classes, positive and negative sentiment, that can be assigned. Since most websites where a user can reflect their opinion about a product, service... contain some type of rating system (1-5 stars), it is easy to classify those reviews in any of the 2 previous classes with the aim of populate the learning system.

SVM [7] and naive Bayesian [8] are two very common methods of transforming sentiment analysis into a supervised learning problem. The work of Pand et al [9] when classifying film reviews as positive or negative, experimenting with the different supervised learning systems, demonstrated the good functioning of the SVM and Bayesian naive when using uni-grams (individual words) as features when training the system. Bo Pang and Lillian Lee [10] also base their work on the use of an SVM classifier but they focus their attention on optimizing the text that is provided to the system, thus, they try to eliminate all objective information from the texts to leave only the one that represents the sentiment that the user is reflecting, thereby improving the precision of the system proposed by Pand, converting the reviews into much smaller information extracts while maintaining the information with the polarity of opinion.

2.2 Unsupervised Learning

It is not hard to imagine that opinion words and phrases are the dominating indicators for sentiment classification. Thus, using unsupervised learning based on such words and phrases would be quite natural. The method used by Turney [11] is such a technique. It performs classification based on some fixed syntactic phrases that are likely to be used to express opinions. The are three main ideas in this method:

- Score the polarity of a review based on the total polarity of the phrases in it.
- Use patterns of part of speech tags to pick out phrases that are likely to be meaningful and unambiguous with respect to semantic orientation.
- Finally, these potentially-meaningful phrases are then scored using Pointwise mutual information (PMI) to seed words on known polarity.

There are similar techniques for unsupervised learning like the explained by
Dave et al [12].

2.3 Opinion Lexicon Generation

Some of the methods mentioned above require the use of opinion words, words that express a certain sentiment. There are three main approaches that have been investigated to obtain the collection of opinion words with their polarity: the manual approach, the dictionary-based approach, and the corpus-based approach.

The manual [13][14][15] approach is the most basic one and so the most time consuming. Despite this, combined with automated approaches as the final check for the mistakes of the systems, this method can be really useful. The automatic Dictionary-based approach [16][17] is based on bootstrapping using a small set of seed opinion words and an online dictionary, like WordNet [18]. The strategy is to first collect a small set of opinion words manually with known orientations, and then to grow this set by searching in the WordNet for their synonyms and antonyms. The newly found words are added to the seed list. The iterative process stops when no more new words are found. After the process completes, manual inspection can be carried out to remove and/or correct errors.

Researchers [19][20][21] have also used additional information in WordNet and additional techniques (e.g., machine learning) to generate better lists.

2.4 Existing tools

Current available sentiment analysis tools like Lexalytics [22], MonkeyLearn [23] or MeaningCloud [24] are mainly focused in serving an API for text analysis, none of them have as main purpose making a tool around the evolution and sentiment analysis of a product with a easy front-end and a system to retrieve reviews of them. The main differentiator of the tool developed in this work is that it contain 3 interconnected systems that work all as one, a scraping system, a sentiment analysis and feature extractor system and a front-end.

Chapter 3

Planning and methodology

Before getting into details about the tool development, this section will explain which engineering methodology was used in this project and why.

3.1 Engineering methodology

The methodology used in this project was Scrum [25][26], a lightweight, iterative and incremental framework for managing complex jobs. It is designed for teams of ten members or less, who divide their work into objectives that can be completed in time-framed iterations, called sprints, that do not exceed one month and usually two weeks.

These sprints are made up of a series of tasks with a particular work weight [27], that represents a estimation of how much time will cost to do the task. Using that weight, the tasks that make up the sprint can be defined.

To obtain the estimation the developers jointly analyse each task and approximate how much time could cost to do it, this are taken from the backlog, a set of tasks that have been defined but neither estimated or started.

Scrum was chosen over other methodologies since it is purposefully designed to help to solve problems where the solution is not clear and experimentation plays a big role [28]. In this work there was great uncertainty about how the system could be built, the elements needed and even if it was possible to achieve the the fact that the work team was made up of only one person (the author of this work), which led to adapting the methodology itself.

3.2 Tool development

For the development of each sprint the agile software development life cycle (figure 3.1) was used. In each iteration, a set of essential requirements were defined, which later were used to design and develop the iteration. Finally the developed solution was tested and reviewed to see if it matched the needed requirements.

With each sprint new functionalities were added at the same time that corrections from other sprints where done.

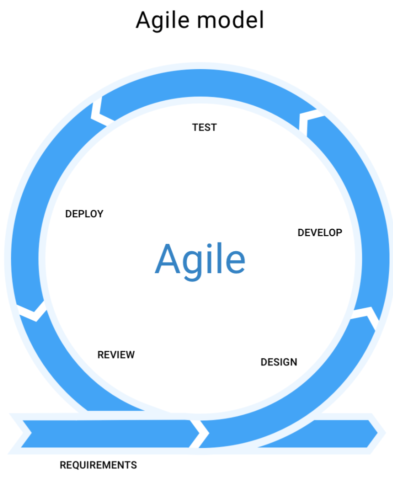
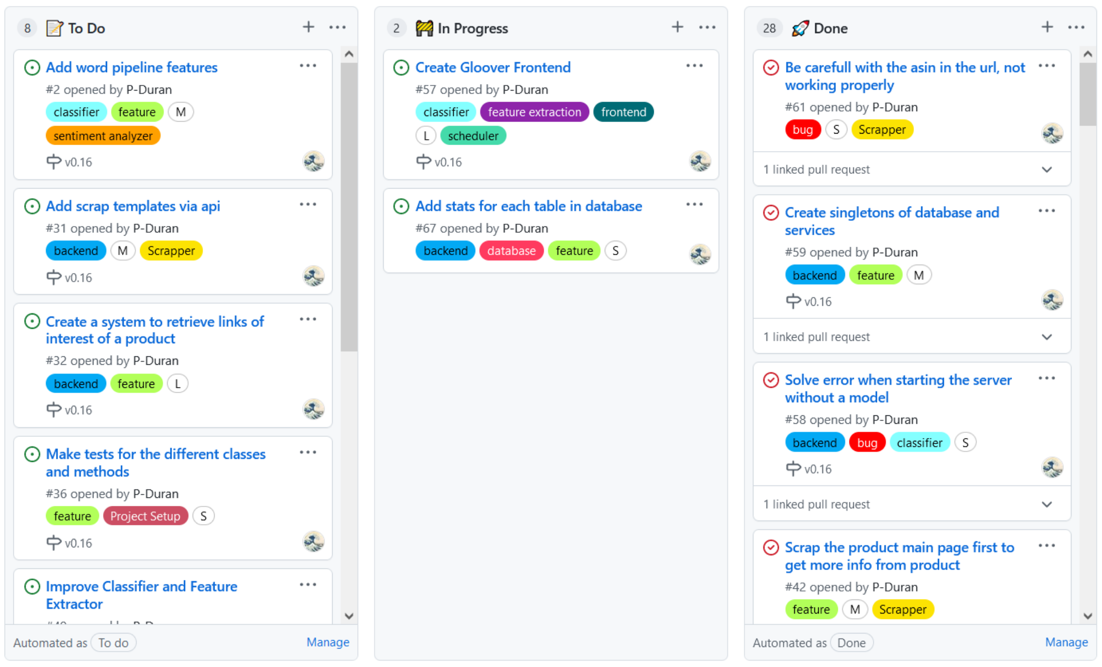


Figure 3.1: Agile software development life cycle

To keep track of the progress of each sprint a Kanban [29][30] board was used. This board is composed of several columns indicating a possible state for a task or tasks to be. For this project three columns where used (see figure 3.2):

- **To do**: Contains the tasks that have not been started
- **In progress**: Contains the tasks that are being done in the current sprint
- **Done**: Contains all the tasks that have been finished



8 To Do

Add word pipeline features
#2 opened by P-Duran
classifier feature M sentiment analyzer
v0.16

Add scrap templates via api
#31 opened by P-Duran
backend M Scrapper
v0.16

Create a system to retrieve links of interest of a product
#32 opened by P-Duran
backend feature L
v0.16

Make tests for the different classes and methods
#36 opened by P-Duran
feature Project Setup S
v0.16

Improve Classifier and Feature Extractor

Automated as To do Manage

2 In Progress

Create Gloover Frontend
#57 opened by P-Duran
classifier feature extraction frontend L scheduler
v0.16

Add stats for each table in database
#67 opened by P-Duran
backend database feature S

28 Done

Be carefull with the asin in the url, not working properly
#61 opened by P-Duran
bug S Scrapper
1 linked pull request

Create singletons of database and services
#59 opened by P-Duran
backend feature M
v0.16
1 linked pull request

Solve error when starting the server without a model
#58 opened by P-Duran
backend bug classifier S
v0.16
1 linked pull request

Scrap the product main page first to get more info from product
#42 opened by P-Duran
feature M Scrapper

Automated as Done Manage

Figure 3.2: Kanban board

At the end of each iteration, a git [31] tag was assigned representing the resulting version of the sprint. This version tag followed the Semantic Versioning [32] pattern of `MAJOR.MINOR.PATCH` where:

- **MAJOR**: version when you make incompatible API changes
- **MINOR**: version when you add functionality in a backwards compatible manner
- **PATCH**: version when you make backwards compatible bug fixes

3.2.1 Project planning and monitoring

In scrum, at the end of each sprint, there is a sprint review to evaluate the outcome of the work and determine the future goals to set for the next sprint [33]. This process involves all the team, where developers show the state of the tasks and the product manager starts planning the following goals with that information. For this project both developers and manager functions were done by the same author.

In terms of time cost and project planning, the following project phases were initially defined:

- **Researching about Sentiment Analysis**
  - Study of sentiment classification
  - Study of feature extraction
  - Study of artificial intelligence technologies
- **System implementation**
  - Design and architecture
  - Implementation of the sentiment analysis system
  - Design and architecture
  - Implementation of the document retrieval system
  - Implementation of a web user interface
  - Improvements

In terms of project monitoring, the figure 3.3 shows a Gantt diagram with the final time cost of each project phase at the end of the project. Some delivery dates could not be reached since, as expected, some tasks required more time than initially thought. For example, the implementation of the REST API for the scraper system was delayed due to a incompatibility of the scraper library with the web framework used in the project.

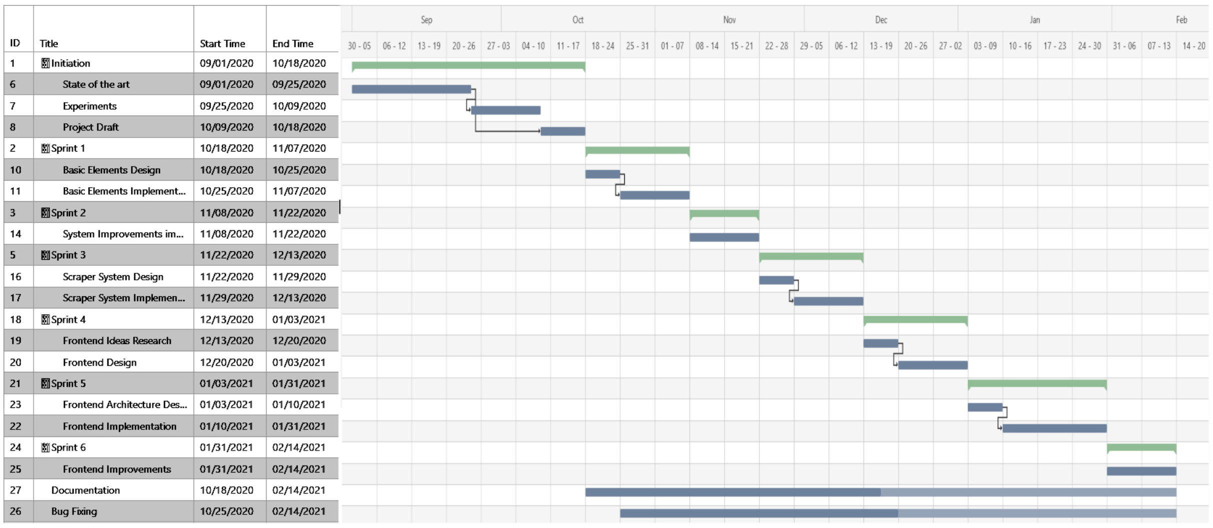

ID	Title	Start Time	End Time
1	Initiation	09/01/2020	10/18/2020
6	State of the art	09/01/2020	09/25/2020
7	Experiments	09/25/2020	10/09/2020
8	Project Draft	10/09/2020	10/18/2020
2	Sprint 1	10/18/2020	11/07/2020
10	Basic Elements Design	10/18/2020	10/25/2020
11	Basic Elements Implement...	10/25/2020	11/07/2020
3	Sprint 2	11/08/2020	11/22/2020
14	System Improvements im...	11/08/2020	11/22/2020
5	Sprint 3	11/22/2020	12/13/2020
16	Scraper System Design	11/22/2020	11/29/2020
17	Scraper System Implemen...	11/29/2020	12/13/2020
18	Sprint 4	12/13/2020	01/03/2021
19	Frontend Ideas Research	12/13/2020	12/20/2020
20	Frontend Design	12/20/2020	01/03/2021
21	Sprint 5	01/03/2021	01/31/2021
23	Frontend Architecture Des...	01/03/2021	01/10/2021
22	Frontend Implementation	01/10/2021	01/31/2021
24	Sprint 6	01/31/2021	02/14/2021
25	Frontend Improvements	01/31/2021	02/14/2021
27	Documentation	10/18/2020	02/14/2021
26	Bug Fixing	10/25/2020	02/14/2021

Figure 3.3: Gantt diagram

3.2.2 Materials and cost estimate

This section includes the minimum resources needed for building the tool and its estimated cost.

Human Resources

To develop the project in the best way possible a team of developers and managers have to be built. This team consists in two back-end developers, one front-end developer and one project manager with programming knowledge. With this organization the different developers can focus in a certain area of the project at the same time that the project manager organize the tasks, helps the developers and monitors the evolution of the project.

The estimated cost of the team per month would be 8.333,33€, where each developer has a monthly salary of 1.666,67€ while the project manager a salary of 3.333,33€

Materials and infrastructure

The following materials and infrastructure elements were the minimal components proposed to build system:

- A unique server for both back-end and front-end deployment
- A pull of proxies for web scraping
- 4 personal computers to develop the tool

Where the estimate cost of a server is of 100€/month, a subscription to a proxy handler that covers the needs of the project is around 30€/month and a good personal computer is 1.000€.

3.2.3 Estimated and monitored cost

The estimated cost is calculated based on the cost of each resource per day. For the developers and project manager the weekends also counted for daily cost, as hiring a person requires to pay vacations and weekends. Taking into account the previous sections, the cost estimate is summarized in table 3.1.

Estimated and final cost						
Resource	Amount	Cost per day	Estimated time	Final time	Estimated cost	Final cost
Project Manager	1	111,11€	158	184	17.555,38€	20.444,24€
Developer	3	166,67€	158	184	26.333,86€	30.661,76€
Server	1	3,33€	120	146	399,6€	486,18€
Proxy	1	1,0€	85	101	85,0€	101,0€
Personal Computer	4	-	-	-	5.016,0€	5.016,0€
Total					**49.476,42€**	**56.709,18€**

Table 3.1: Estimated and final cost table

Chapter 4

Technologies and tools

This section showcase the different technologies and tools required to build the project. Since this work has both front-end and back-end, the different elements will be structured in those categories to better explain and understand the use of each one and the relationship between them.

4.1 Back-end

This section enumerates the different technologies used to build the different parts of the back-end, the sentiment classification system, feature extractor and the scraping system.

4.1.1 Python

Python [34] is a high-level, interpreted, general-purpose programming language. It's simplicity, flexibility, popularity and the incredible amount of packages and libraries around it, made easy the choice to build the project using this technologies, since it was needed a fast development language popular in machine learning (required for most of the sentiment analysis tasks).

4.1.2 Flask

Flask [35] is a micro web framework written in Python. It is classified as a micro-framework because it does not require particular tools or libraries to run it.

For this project it was used to build the multiple REST API services required to expose all the back-end functionalities.

Alternatively Django [36] was also was taken into account, since it is a more complete framework, but flask is easier, much simpler and the project was not large enough to require all the features and complexity offered in Django.

4.1.3 Scikit-learn

Scikit-learn [37] is a free software machine learning library for Python. It stands out for featuring various classification, regression and clustering algorithms including support-vector machines, random forests, gradient boosting, k-means and DBSCAN,

The library provides simple APIs to use any type of classifier, good performance, great amount of documentation and a big user base. All of this made easy the choice to use this library instead of more complex ones like PyTorch or TensorFlow (that also focused in deep-learning).

4.1.4 NLTK

NLTK [38] is a platform for building Python programs to work with human language data. It provides a really simple API to use the different features and contains all the main operations required to work with human language data.

It's simple use, functionality and user base made this library the choice instead of Spacy [39], more powerful, complex. but less used and documented.

4.1.5 Scrapy

Scrapy [40] is a free and open-source web-crawling framework written in Python. It was used in the project to obtain the required reviews of a web site, it has not direct competitors so it was a direct choice.

4.2 Front-end

This section enumerates the different technologies used to build the different parts of the front-end, the web application and designs.

4.2.1 Typescript

Typescript [41] is a programming language, syntactical super-set of JavaScript and adds optional static typing to the language.

JavaScript (JS) [42] is the main programming language used to develop web applications but the lack of typing makes developing in it really difficult and buggy, so Typescript was chosen because it has all the good features of JS but with the advantage of adding typing, also the majority of the content made for JS (libraries) are also available for Typescript, so there is almost any limitation.

4.2.2 React

React [43] is a free and open-source front-end JavaScript library for building user interfaces based on UI components.

It's simplicity, knowledge about it and popularity made the choice easy, since it has an enormous potential and a really low learning curve.

4.2.3 Figma

Figma [44] is a vector graphics editor and prototyping tool which is primarily web-based.

One of the best tools for prototyping, free and easy to use.

Chapter 5

Analysis

This chapter analyzes the main purpose of the project and its architecture, to understand the requirements, problems and solutions proposed in next chapters.

5.1 Mission statement

The main goal of the project was to build a system capable of allowing a user to analyze the general sentiment of any product/topic and its features based on a collection of documents reflecting a particular opinion about it. This allow any person/organization to make decisions both for buying a product or understanding the shortcomings and strengths of it.

The system is based in four main elements:

- System to classify the polarity of a text
- System to extract product features
- System for web scraping
- System for visualizing data

Text classifier

A system that extracts the sentiment expressed in a text. It is the core system as it allows to understand the sentiment of text and with it, understand the people opinion about a certain product/topic.

Feature extractor

This system is in charge of extracting the product features, it works examining a set of documents and finding the most popular words that contains an opinion.

With this system it is possible to understand what are the people opinion targets when reviewing a product.

Web scraper

The web scraper is an element of the project that allows a user to extract opinionated text from any internet web page. It plays a key role as it serves with documents the previous systems and makes easy for an user to obtain and analyze any source of reviews from the internet.

Data visualization

Finally the data visualization, a really important component as it allows to visualize the extracted data in a meaningful way, making it easy to understand and analyze.

5.2 Actors and use cases

The tool is intended to allow the users to extract and analyze the polarity of a product/topic at the same time that visualizing the obtained data in a simple way, so the following main use cases are defined:

- User requests the reviews of a certain product from a web page
- User uses both classifier and feature extractor to analyze a set of documents
- User performs a request to visualize the extracted data

This project was conceived as a web application with a server-client architecture in which the server hosts, delivers and manages most of the resources and services to be consumed by the client. In this case, the web application was the client while the server consisted in the classifier, feature extractor and web scraper (see figure 5.1).

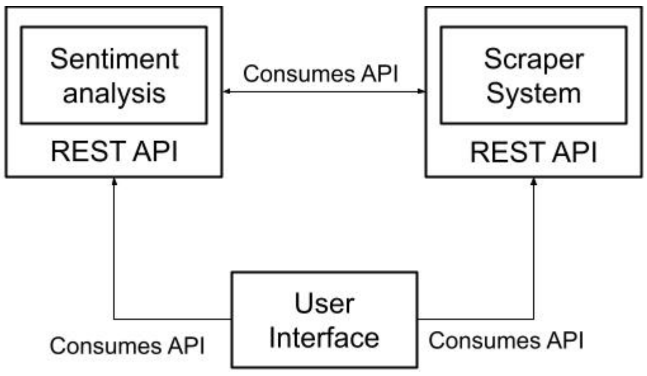


Figure 5.1: Basic architecture

Therefore, two actors within the protocol are defined:

- A Relying Party, that supplies exposes the methods that allow to perform action in the system through REST endpoints as well as serving the user interface with client-side.
- The user, that uses the system communicating with it, both using the REST API or user interface.

Chapter 6
Development

This project was made under the Scrum methodology, an agile methodology based in the use of sprints of a certain time length where tasks are defined to be done. This chapter deals with, in one hand, the different sprints that the development of the project required, and, on the other, the requirement analysis and technical details of the tool.

6.1 Sprints organization

The development of this project required eight sprints, based on the development of new features and the refinement of the previous ones, that differed from the initially defined structure. Those sprint were the following:

- **Sprint 1**: Building the key elements
- **Sprint 2**: System Improvements
- **Sprint 3**: Scraper System
- **Sprint 4**: Gloover REST API and Database Integration
- **Sprint 5**: Front-end Design and Prototype
- **Sprint 6**: Front-end Implementation
- **Sprint 7**: Gloover Refinement
- **Sprint 8**: Front-end Refinement

6.2 Sprint 1: Building the key elements

At the beginning of the development, a first sprint was proposed based on the construction of two key elements for the correct operation of the tool. On the one hand a classification system and on the other a feature extraction system. The first subsections argue the project setup, architecture and technologies while the following subsections will define and expand the idea and implementation of the classification and the feature extraction systems.

6.2.1 General Aspects

For the development of the project Python [34] was selected as the main programming language. Python is one of the most popular and complete languages in the artificial intelligence field, it has a large base of users and libraries focused exclusively on this topic [45]. Also, this language allows the project to evolve and grow without many restrictions because of it flexible nature and the great amount of libraries that cover almost all the needs that could occur in the development process.

With this in mind, there are two main libraries that were used in this first iteration:

- Scikit-learn [37]: Contains a wide variety of classification systems and the tools to tune them, all of those necessary to build the classificator.
- NLTK [38]: A set of libraries for natural language processing that are needed to build both classification and feature extractor systems.

Both libraries have a great community and good documentation that made them a good choice for this project.

As a note, it has to be said that Spacy [39] as natural language processing library is more powerful and efficient that NLTK, but the lack of documentation and users made this one the non-preferred option, although in some specific situations it's use was preferred over NLTK for performance reasons.

With the technology already explained It's time to define the main architecture that the elements defined in this sprint. The main idea behind the classifier is to be able to create a system that given a certain text it will extract the sentiment that the person who wrote it was trying to express, while the main purpose of the feature extractor is to obtain the key ideas that are manifested in a set of reviews.

6.2.2 Text Classifier

Text classification is a machine learning technique that assigns a set of predefined categories to open-ended text. In this work those categories are two, whether a text is positive or negative. The text classificator system is the one in charge of this task, because of this, it is a essential part of this work, one of the main pillars.

To be able to build such a crucial system, at first a Support vector machine (SVM) classifier was proposed to be used due to its proven effectiveness in the text analysis field [9]. In addition a pipeline with the data extraction, transformation, normalization... needed for a text to be able to be fed as training data in the SVM was designed.

Pipelines

The pipeline module of *scikit-learn* allows you to chain transformations and estimators together in such a way that you can use them as a single unit. This chain of transformations are composed by transformers and *scikit-learn* provides an easy way of composing custom ones following the next code (listing 6.1, figure 6.1).

```python
class Transformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        """The initialization needed"""
        pass

    def transform(self, dataframe, y=None):
        """The workhorse of this transformer"""
        return dataframe.apply(some_function)

    def fit(self, df, y=None):
        """Returns `self` unless something different happens in
    train and test"""
        return self
```

Listing 6.1: Transformer code

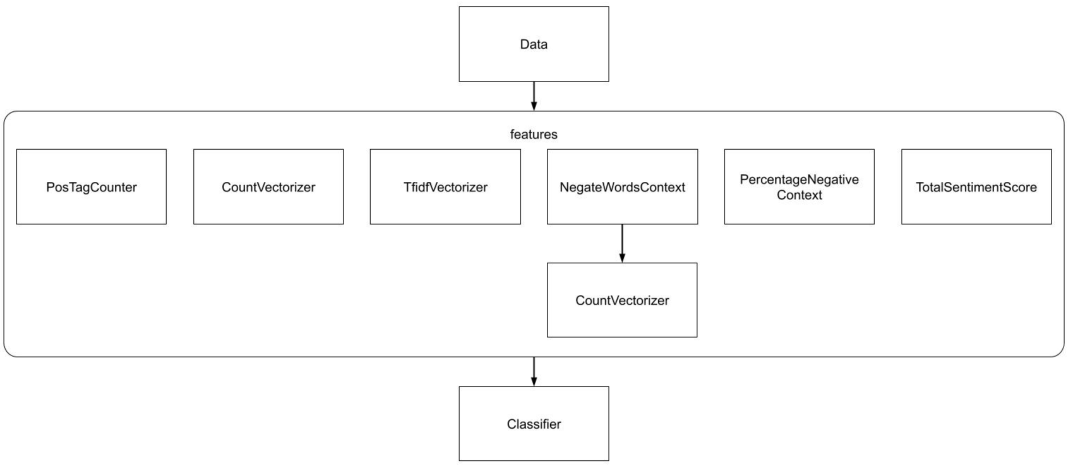


Figure 6.1: Transformers pipeline architecture

The pipeline designed for this project, represented in the figure 6.1, was composed by the following transformers or features:

- **CountVectorizer**: Extracting the n-gram counts of a text is one of the main transformations that are needed to convert a text into data that a classifier can read. Widely used in text analysis researching, it is proved to be one of most helpful and powerful features in this field [46][47].

  The n-grams are a contiguous sequence of n items from a given sample of text or speech. In text classification are really useful because they work as discriminative features, this cab be shown with the next example: Imagine it is needed to classify whether a road name is from China, India or Britain and there is a table (table 6.1) with the road names that contain the bi-gram ”ck”.

Road names that contain the bi-gram ”ck”		
British	Chinese	Indian
Alnwick	Boon Teck	
Berwick	Hock Chye	
Brickson	Kheam Hock	

  Table 6.1: Example table of road names.

  Given that example, if a classifier receives a read containing ”ck” it can say with confidence that that road is not Indian.

- **PostTagCounter**: This transformer is in charge of extracting the number of verbs, adjectives and nouns from a text, which provides a simple way to represent its main structure.

  For the construction of this element first of all the text needs to be tokenized and the part of speech tag of each token needs to be extracted. NLTK provides a good toolkit for tokenizing and Part-of-speech (POS) tagging a text, but since NLTK makes a fine-grained tag detection that is not needed for this transformer a simple conversion (table 6.2) can be done to simplify the outcome of the POS tagging.

Example of tag simplification		
NLTK Tag	Description	Simplified Tag
JJ	adjective ‘big’	Adj
JJR	adjective, comparative ‘bigger’	Adj
JJS	adjective, superlative ‘biggest’	Adj

  Table 6.2: Tag simplification table.

- **TfidfVectorizer**: The Term frequency–inverse document frequency (TF-IDF) is a numerical statistic that is intended to reflect how important a word is to a document in a collection.

In our case the TfidfVectorizer transformer obtains the TF-IDF of each word in a text and gives it a score of how useful a word is to the entire document.

Like the CountVectorizer this feature a big asset in text classification[48].

- **PercentageNegatedContext**: This feature provides the percentage of text that is in a negated context. With this we mean that the system extract the quantity of words that are modified and affected by a negation.

  For a better explanation let's see this example: Given the next text *"This was a good day but I didn't like that at the end it rained."* we can clearly see that there are two main contexts where the information is presented, on hand we have *"This was a good day"*, and in the other *"I didn't like that at the end it rained."* both express a sentiment and both could express a positive one if only there wasn't the *"didn't"* modifier in the second one that modifies completely the entire meaning of the phrase.

  Recognizing those negated contexts are really important because the polarity of the words affected by this negated context are inverted in the majority of the cases, in the example we can clearly see that *"like"*, which usually express a positive sentiment, in the example was a strong negative one.

  As shown detecting and working with this casuistry can help to improve the success of our classifier.

  To extract those contexts first it is needed to search for a negation word and once localized the closest punctuation mark is retrieved. All the words between the negation word and the punctuation mark are defined as negated context.

  A negation word is any word matching the following regular expression [49] (listing 6.2).

```
    (?:
    ^(?:never|no|nothing|nowhere|noone|none|not|
        havent|hasnt|hadnt|cant|couldnt|shouldnt|
        wont|wouldnt|dont|doesnt|didnt|isnt|arent|aint
    )$
    )
    |
    n't

```

Listing 6.2: Regular expression of negation words

- **TotalSentimentScore**: This transformator estimates the sentiment score of a text assigning to each word a sentiment score obtained from a sentiment lexicon.

A sentiment lexicon is a lexicon in which each word has a polarity score assigned that represents how positive/negative the word is. Usually this lexicons are hand-crafted and don't contain the score for a word in a negated context.

In this project the lexicon used is the SentiWords lexicon [50] a lexicon of roughly 155.000 English words. Since the lexicon selected does not have any score for negated context, any word contained in those contexts reversed its polarity.

- **NegateWordsContext**: Adds the tag _NEG to each word in a negated context.

  It iterates through each word of a text and if it is inside a negated context appends the _NEG tag to the word.

Transformers Summary List	
Transformer Name	Description
CountVectorizer	Presence or absence of contiguous sequences of 1, 2, 3, and 4 tokens
PostTagCounter	Extracts the number of verbs, adjectives and nouns in a text
TfidfVectorizer	Convert a collection of raw documents to a matrix of TF-IDF features
PercentageNegativeContext	Extracts the percentage of negated contexts of a text
TotalSentimentScore	Calculates the sentimental score of a text given a lexicon
NegateWordsContext	Ads the tag _NEG to every word in the negated context of a text

Table 6.3: Tag simplification table

6.2.3 Feature Extractor

The classification of a text is a really useful as a global indicator of the sentiment of the document, but it does not provide the detail of which aspects are the ones that hold the negative sentiment and which ones don't, a negative review about a product does not mean that the user who wrote it hated all the product features.

To obtain that level of detail a feature extractor is needed, a system capable of identifying the object features that have been commented, and later determine if what the text says about those features is positive or negative.

Although there are multiple methods for the extraction of features, both based on machine learning and statistical methods, for this work we will focus on statistical methods since there are reduced amount of text features data sets and the statistical approach reduces the complexity of the project at the same time that provides decent results.

The system proposed performs the following tasks to achieve the extraction of the features with the architecture reflected in the diagram (figure 6.2):

- Elimination of all words that are not important from a feature perspective (Noise removal).

- Selection of the possible features.
- Removal of the wrong detected features.

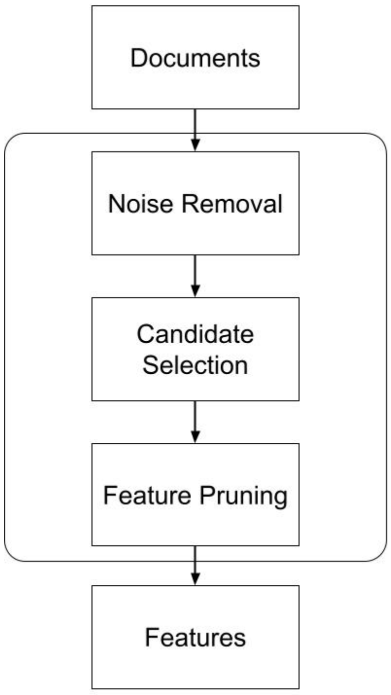


Figure 6.2: Feature extraction architecture

Noise Removal

Removing the text parts that do not contain any valuable information for the feature extractor is key to localize the important features in an efficient and correct way, the less words to process the more accurate the extraction will be.

This process is composed from various parts that allow the text to minimize its information at the same time that retains the possible features. The first of them is word tokenizing the text, this consists in extracting and splitting the text into words (tokens). Once the text is converted into a list of tokens the next step is POS tagging those tokens to be able to filter everything that is not a word capable of becoming a feature.

The next step consist in Lemmatizing the obtained tokens. Lemmatizing is the of grouping together the inflected forms of a word so they can be analysed as a single item. For example, 'walked' could be lemmatized as 'walk'. With this process we can be able to reduce the number of possible combinations of a word to help the extractor.

An object feature is any word that describes a product characteristic and most of the times this words are nouns, so at the end of the process every non noun token is removed.

Sometimes and adjective could appear alone referencing a noun eg: This item is large, but to simplify we consider only nouns as possible features.

Feature Selection

This is the core element of the system. In this part we will study how to extract all the possible features from a collection of documents using an statistical approach.

To be able to find the features, we consider that in a collection of documents with an opinion about a certain object, the features will appear directly or indirectly throughout the collection with a high frequency, since they are the subject of the opinion.

Starting from this assumption localizing the feature can be done using association rule mining [51], a process that finds all frequent item-sets. In this case, the item-sets will be a collection of words.

To perform the association rule mining the apriori algorithm was used [52][53], feeding it with the collection of preprocessed documents and assigning a maximum length of the item-sets of 3, since it is consider that there is no feature that contains more than 3 words.

At the end of this task a item-set of candidate features is obtained with both individual word features and multi-word features.

Feature Pruning

This part explains the selection algorithm created to split the final features from the candidates.

As said before, an object feature is any word that describes a product characteristic, and it is, most of the time, followed by some kind of opinion about it. It is not unreasonable to think then, that a feature will have a modifier that express a certain sentiment or attribute.

What can differentiate the real features from the wrong detected is if an adjective is modifying them. To be able to do that every document is tokenized and POS tagged, then the feature is searched across the collection of documents and for each occurrence it is noted if an adjective modifies it. If the proportion of occurrences with modifier reaches a certain threshold the feature is selected, otherwise it is removed.

Once this process is finalized the object features from the document collection are extracted successfully.

6.3 Sprint 2: System improvements

With the main elements of the system build, this sprint was focused in the improvements needed to finalize the implementation of both text classifier and feature extractor. On the one hand the pipeline of transformators was found to be extremely slow when processing a big amount of documents and applying expensive operations to them (figure 6.3). On the other, as noted before, the lexicon used hadn't sentiment scores for negated contexts, To try to improve the classificator the construction of an ad hoc lexicon was proposed with both sentiment score and negated context sentiment score. Finally, the extraction of the sentences containing each feature to be able to obtain its generalized polarity needed to be built.

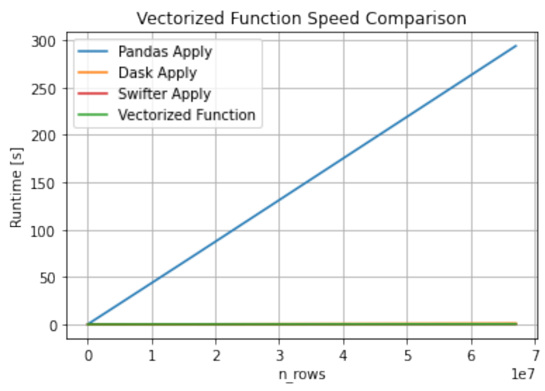


Figure 6.3: Time comparison between default pandas apply and multiple core swifter apply

6.3.1 Lexicon Generation

The majority of the existent sentiment lexicons are hand-made, do not take into count the current context of the word or the relationship with the document topic. Thus, for example, the word unpredictable could have a negative meaning if we are talking about a software product but a positive one if it used when describing the plot of a movie.

Those problems could be solved with the generation of an automatic lexicon based on the topic of the collection to analyze.

There are multiple ways to extract the sentiment score of a term but Pointwise mutual information (PMI) is a simple robust and well proved method for NLP tasks [54][55], a measure of association used in information theory and statistics. In this work PMI is calculated as follows:

$$PMI(w,p) = log^2 \frac{freq(w,p) * N}{freq(w) * freq(p)}$$

Given a collection of documents, $freq(w,p)$ is the number of times that a word w occurs in the document with polarity p, $freq(w)$ is the frequency of the word w in the corpus,

$freq(p)$ is the number of tokens in the documents with polarity p and N the total number of tokens in the corpus. With this in mind, the sentiment score of a word (/w/) in a collection of documents can be calculated using this formula:

$$SentimentScore(w) = PMI(w, positive) - PMI(w, negative) \tag{6.1}$$

The collection of documents used was a data-set of Amazon reviews about electronic devices [56]. Therefore, as Amazon reviews rank products on a scale of 1 to 5, a $positive$ document is any document with a score $>= 3$ and a $negative$ one is any document with a score < 3.

To be able perform the task of creating a sentiment lexicon, the work was divided in two:

- Creation of an index from the collection of documents.
- Generation of the custom lexicon.

Document Indexation

The formulas for calculating the sentiment score of a word require the frequency of the word in the collection, the number of tokens in it... Because of this, the creation of an index was proposed. An index allows to perform the operations required to extract the score efficiently, and at the same time that gives the possibility of storing different indexes referring different document collections.

To build the lexicon each review from the data-set was transformed in a document with the following fields (table 6.4).

Document to index	
Field	Description
Polarity	The polarity of the review (positive/negative)
Plain Text	The entire text of the review
Not Negated Context	The review text without the negated contexts
Negated Context	The review text only with the negated contexts

Table 6.4: Index Document

The separation between plain text, not negated context and negated context was needed because the $freq(w, p)$ has to be able to extract the frequency of w only in the reviews with the polarity p.

Lexicon Generation

For the lexicon, three independent scores were calculated, the general score of the world, the score for no negated contexts and the score for negated contexts.

To calculate those, the formula 6.1 was applied to each text field of the index documents (table 6.4). The words selected to build the lexicon were the used in SentiWords [50] to be able to compare the results from the custom sentiment lexicon against a well proven one.

At the end, the custom lexicon was able to generate results that were really promising. For almost every word analyzed, the scores followed the next pattern, for positive words like "good", the next behaviour was observed:

$$NegatedContextScore < GeneralScore < NoNegatedContextScore$$

While for negative words like "bad"

$$NoNegatedContextScore < GeneralScore < NegatedContextScore$$

This results are similar to what a person could think when using a word in different contexts, a positive word has its most negative polarity when used in a negated context and its most positive in non negated context, while a negative word has the most positive polarity in the negated contexts and the negative one in the non negated contexts. But one problem was found, in some cases the scores were not reasonable, since PMI does not work well with low frequency terms.

When comparing it with SentiWords[50] it was seen that for the most common opinion words the algorithm worked fine, but the rest of the lexicon did not have a good consistency needed to be able to improve the classificator.

6.3.2 Pipeline Optimization

The pipeline needed some optimization to be able to process the text in a more efficient way, To achieve this performance improvement the python library Swifter [57] was used. Swifter tries to vectorize the function applied to a data-frame and use parallel processing to lower the times of applying a function over a data-frame. Using the library in the transformers increased the performance notably, also some minor code improvements where needed to achieve the desired processing speed.

6.3.3 Feature Sentence Extraction

To analyze what product features are the ones that hold the positive opinion and which ones holds the negative, the extraction and classification of the sentences where those features appear is needed.

With a system that returned the product features from a document collection, the implementation of this functionality was easy. First, the location of each occurrence of the different features across the collection is needed, once the sentences are detected the sentiment classifier classifies them into positive or negative.

At the end of the process a collection of JSON elements with the following structure was generated (listing 6.3).

```
{
  "review_id": "361f201e-183b-461f-affb-49f7815f82d0",
  "product_id": "B06XP1JZ53",
  "start": 10,
  "end": 14,
  "word": "feel",
  "sentence": "they both feel accurate and enjoyable.",
  "polarity": 1,
}
```

Listing 6.3: Example of analysed sentence output

The next diagram shows the full workflow of the sentence extraction (figure 6.4):

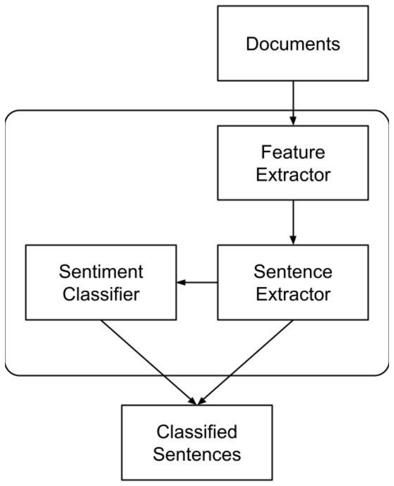


Figure 6.4: Sentence extraction architecture

6.4 Sprint 3: Scraper System

From now on, two independent systems will be used, so to avoid confusions, the system containing the feature extractor and text classifier will be called the Gloover system while the system referring to the scraper, the Scraper system.

The Gloover system created did not had a way to retrieve automatically the documents from the WWW, the only source of information available to the system was a predefined file with a limited number of reviews [56].

The focus of this sprint is the construction of a web crawling system able to retrieve any piece of text information capable of been processed and analyzed by the Gloover system. To simplify the sprint, the main focus of the web crawler was the Amazon marketplace and the reviews that each product contains.

To develop the scraper system three main tasks were done:

- Choosing the web scraping technology
- Building the scraper system
- Linking the scraper system with Gloover

6.4.1 Scraping Technology

Automatically extracting information from the WWW is a complex task if done well, multi-threading, avoiding ip blocking and navigating a web page to extract the right data is a long and difficult task. Fortunately there are plenty of packages and libraries that do a great job solving all this issues. For this project, Scrapy [40] was the library selected to develop the system, because it is a high-level web crawling and web scraping framework, which means that you can abstract from the details and use the easy and flexible Application programming interface (API) that the tool provides you.

Although Scrapy does a real good job managing the elements needed to access and extract information from a web page, it not gives you a generic way to transform the html of the web page into structured data. To make up for this lack, a second library was selected, Selectorlib [58], a python library that reads a YAML file defining the structure of the data to obtain, and extracts it from a html provided.

6.4.2 Building the tool

Scrapy gives you an easy way to build a scraping system, the spiders, a place where you define the custom behaviour for crawling and parsing pages for a particular site.

The spider built for the project was developed with the maximum flexibility in mind, to achieve this, the use of selectorlib was crucial, all the web page particularities relapsed in the YAML file that defines the elements of the scraped site instead of in the spider implementation. The designed spider only contained the basic implementation needed to navigate thought the pagination if required and the normalization of the data [59].

Scrapy provides two main ways to run it, the scrapy command-line tool and the scrapy API that allows you to start scrapy from a script.

At first, the script API was planned to be used, Gloover would call Scrapy via script to retrieve reviews and store them, but the incompatibilities that this method have, used in conjunction with Flask [35], a library to build a REST API, required to think in an alternative way to start and use the system. A lot of research was needed to finally design a system capable of been used by Gloover.

Inspired by the GitHub project [60] based on the same idea, a REST API was designed (table 6.5), that allowed not only to use Scrapy in the most flexible way, but to use it by any project if required.

REST API Methods		
Method	Resource	Description
POST	/scrape	Starts a scraping job with the given URL and a certain spider
GET	/jobs	Return the state of the scraping jobs
DELETE	/jobs/<scraper_id>	Stop the scraping job with id <scraper_id>
GET	/spiders	Returns the available spiders
GET	/containers	Returns the folders and files containing the scraped data
GET	/containers/<container_id>	Returns the content of the data container with id <container_id>

Table 6.5: REST API Methods

With the type of system decided, the next step was implementing the REST API and all the elements needed. Since the process of scraping can be a really long task, a REST API request that waits for the system to retrieve all the information could return a timeout or block the system. To solve this problem, an asynchronous method to scrape is needed, and with it, the following components have to be built:

- An asynchronous request system to scrape and retrieve the data.
- A job system that assigns each scraping request to an individual job .
- A storage system to save the scraped data linked with the job that produced them.

Asynchronous requests

Sometimes a REST API operation might take a considerable amount of time to complete. Instead of letting the client wait until the operation completes an immediate response can be returned and process the request asynchronously.

When making a scrape request (POST */scrape*), a shell command that launches scrapy is assigned to a job, that will execute it as soon as possible. This task is usually really expensive and can take minutes if not more. Once the request has been made, the id of the job created is immediately returned to the client. With that id the sender can check the status of the job (GET */jobs*) and if finished the scraped data can be retrieved using the job id (GET */containers/<id>*). The full workflow is shown in the diagram (figure 6.5).

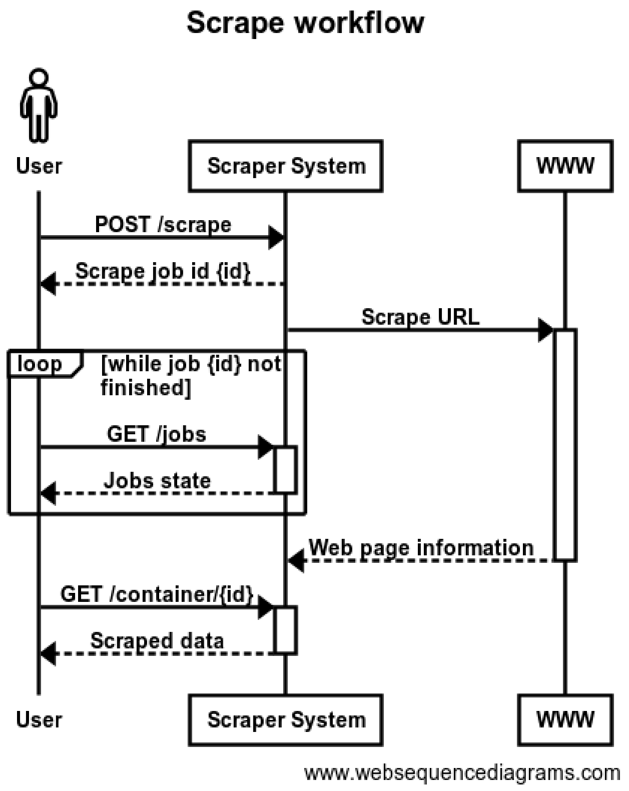


Figure 6.5: Scrape and retrieve data workflow

Job scheduler

In the last section we were talking about jobs (units of work), as an element needed for the asynchronous calls. Those jobs have to be built and managed in some way. This section will explain the methods used to make those tasks.

A job scheduler is a computer application for controlling unattended background program execution of jobs. There are plenty of packages in python that can be used to build one, but for this project Apscheduler [61] was chosen.

Apscheduler allows you to build a system capable of creating jobs, scheduling and launching them in background, managing the state of the jobs and storage the state if needed. With all of this tools building the scraper system is relatively easy, just a mater of making the Apscheduler API reachable via the REST API created.

Storage Structure

Scrapy provides an easy way to store the scraped data in many different ways, from files to databases. For this project the storing method used was the file system, since is an easiest and simpler way, but the right approach would have been using a database.

Since the scraped data is stored in a file system, some kind of structure to order all the files that could be created have to chosen.

Every scrape has a job id assigned to it, so the easiest way to identify each output file of each scraping process is using the job id as file name. With this approach retrieving the output data is as easy as looking for a file with a certain name, but despite the strong and easy that this method is, lacks some kind of hierarchy between files. Each file is scraped using a certain spider focused in obtaining the information of a particular web page, so structuring the files was based on which spider was used to generate them. With this is mind, is easy to see that the output of the scraping process can be structured also in folders, one per existent spider. With this approach a simple transformation can be done, turning the file structure into a formatted JSON.

The following example shows the file structure generated by the scraper and the output received when calling (GET /containers) (listing 6.4) (In this example no reddit files where obtained).

- reddit
- amazon
  - 2c5524f3-4c8a-4429-ae03-8298285770ca.json
  - 06a45942-5cbd-426b-9864-3cf5005c3f5a.json

```
{
    "containers": {
        "reddit": {},
        "amazon": {
            "2c5524f3-4c8a-4429-ae03-8298285770ca": {
                "id": "2c5524f3-4c8a-4429-ae03-8298285770ca",
                "last_modified": "2021-05-18 19:40:06.299821",
                "path":
"generated/amazon/2c5524f3-4c8a-4429-ae03-8298285770ca.json",
                "spider": "amazon"
            },
            "06a45942-5cbd-426b-9864-3cf5005c3f5a": {
                "id": "06a45942-5cbd-426b-9864-3cf5005c3f5a",
                "last_modified": "2021-04-02 19:05:49.108819",
                "path":
"generated/amazon/06a45942-5cbd-426b-9864-3cf5005c3f5a.json",
                "spider": "amazon"
            }
        }
    }
}
```

Listing 6.4: GET /containers output

6.4.3 Linking the Scraper system with Gloover

To connect both systems, the first task was implementing a facade REST API method for scraping from Gloover.

The facade method sends the scrape request and stores the id returned, to check the state of the job. When finished, Gloover makes a request to retrieve the scraped information to be used in both classificator and feature extractor.

Finally, for manual notification that a job finished a second method was also made, that allowed the system to retrieve any scraped data.

The workflow between the two system is the same as the shown in the figure 6.5.

6.5 Sprint 4: Gloover REST API and Database Integration

In the previous sprints a system fully capable of extracting and analyzing documents from any resource of the WWW was built. In this sprint the main goal was to create a REST API interface to use the system in a easy way, thinking in the future implementation of the front-end, and to persist all the extracted and generated data into a database.

In the following subsections the implementation and design of the REST API and database will be the main focus.

6.5.1 Database

There are plenty of database, from relational databases to non-relational, each one with their pros and cons [62][63][64]. For this project MongoDB [65], a non-relational database, was used. Non-relational databases do not store the data in relational form and mostly do not use SQL as query language, this approach makes them flexible, fast and scalable since relational databases are not designed to store large amounts of data and their architecture tend to be complex and much less flexible.

Given that this system will scrape and produce a great amount of data that later needs to be store, and it does not have complex relationships between elements, MongoDB is a great choice, with a lot of documentation, users and integrations with already used python packages [66].

For the design tables and relationships needed to create the database, what the Gloover system produced and consumed was analyzed, to understand the needs of each element and the way to persist them. See table 6.6.

Produced and consumed data	
Element	Description
Products	Obtained from the Scraper System, represent a product
Reviews	Obtained from the Scraper System, represent a user review about a product
Features	Extracted by the feature extractor, represent a product feature
Feature sentences	Extracted by the feature extractor, represent a review sentence where a product feature appears

Table 6.6: Produced and consumed data by Gloover

All of those elements needed to be persisted to allow consuming them in the future at the same time that let the chain of process to obtain them asynchronous, you don't need to get the reviews, obtain the products, then the features and finally the sentences in the same workflow, since you don't lose the data once is persisted in a database. The elements aforementioned were persisted in the database following the figure 6.6 were the relationships and properties of each of them are represented.[1]

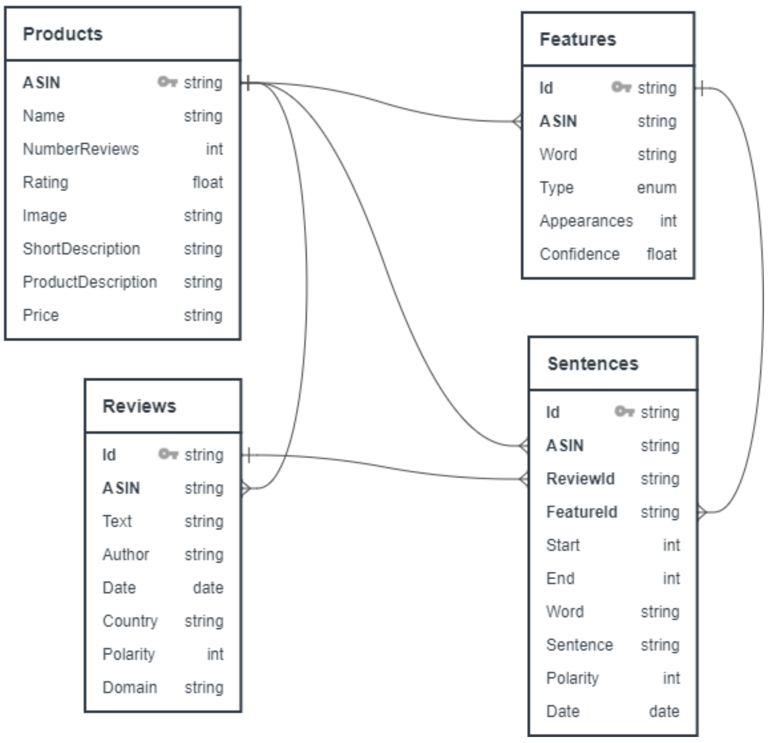


Figure 6.6: Database diagram

6.5.2 REST API Creation and Design

Until now, the way to execute the Gloover system was via command line, this approach restricted and complicated its use since other systems could not interact with it. To solve this issue a REST API was built.

Each functionality of the system needed to be exposed in the API, from the classification to the feature extraction, in addition to the retrieval of the Database (DB) elements stored.

The table 6.7 shows the summary of the endpoints created.

[1] Since this work is focused on Amazon reviews, each product is identified by the Amazon Standard Identification Number (ASIN)

REST API Methods		
Method	Resource	Description
POST	/scrape	Sends a scraping request to the Scraper system
POST	/scrape/notify/<scraper_id>	Notifies the system that the scrape with id <scraper_id> has finished
POST	/classifier/classify	Returns the polarity of a text
GET	/database/reviews	Returns the reviews stored in the database
GET	/database/reviews/<id>	Returns the review with id <id>
GET	/database/products	Returns the products stored in the database
GET	/database/features	Returns the product features stored in the database
GET	/database/sentences	Returns the feature sentences stored in the database
PUT	/database/features/update	Updates/Creates the features of a certain product and stores them in the database
PUT	/database/sentences/update	Updates/Creates the feature sentences of a certain product and stores them in the database

Table 6.7: Gloover REST API Methods

6.5.3 Final Structure

At the end five main services were build, with some degree of interconnection.

- **Sentiment Classifier**: Service in charge of classifying a text in one of the two classes positive/negative. It requires to load all the persisted reviews in the database to build a classification model, it also saves the class of a certain classified text in the database.
- **Feature Sentence Extractor**: Service that extracts the sentences were a feature appears. It requires the features of a product either from the database or the feature extractor, and also sends the obtained sentences to the classificator to know the polarity of them.
- **Feature Extractor**: Service focused in the extraction of features of a product, it requires to load all the database reviews of a certain product to work and when obtained it also saves them into the database.
- **Database Manager**: Service in charge of saving data into MongoDB.
- **REST API**: Exposes all the functionality mentioned above, both for executing tasks and retrieving the result of that execution.

The figure 6.7 shows the final structure of the project, once this sprint was finished.

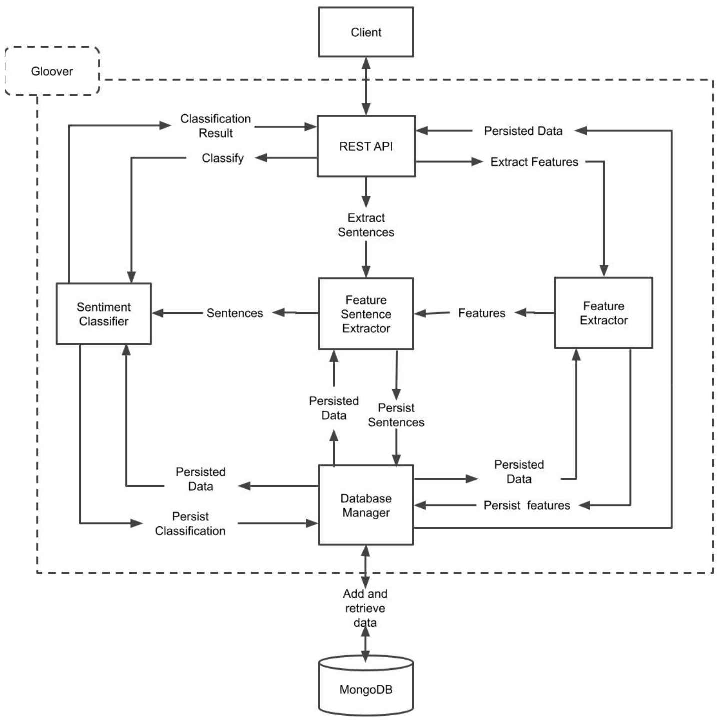


Figure 6.7: Gloover diagram

6.6 Sprint 5: Front-end Design and Prototype

With all the pieces built and connected together, the system was fully operational, so it was essential to have a way to show all the data that had been obtained, from the reviews that were extracted and later saved in the database, through the characteristics extracted from those reviews and the polarity of each phrase where each feature was contained. With this idea in mind, this sprint was dedicated to the visual design of the front with its different views and components, that is, to make the prototype of what the front would be.

6.6.1 General Aspects

Figma [44] is a vector graphics editor and prototyping tool which is primarily web-based, allows you to model and design the user interface. It was decided to use this tool to design the front-end since it is free, has an overwhelming flexibility and there was already experience using the tool.

Prototyping consists in the creation of a test model that simulates the behaviour of the real system. It is key process because allows both users and developers to refine the project, from a user perspective, improving the usability, and from a developer perspective helping to understand the requirements and improve the estimations of the project.

For the development of the prototype four tasks were defined:

- Design guidelines and architecture.
- Design and prototype of the home screen.
- Design and prototype of the product screen.

6.6.2 Design guidelines and architecture

As a base on which to mount the prototype, certain design guides and generic components that would be part of each view were defined, using material design[67] as the basis. These design guides allow to maintain the visual coherence of the entire application while the design of the generic components and views allows to obtain an idea of the final result of the product at the same time that makes the task of implementing the front easier since you only have to focus on the programmatic part.

Design Guidelines

The main design guidelines proposed were the following:

- Color palette: A series of colors were proposed for each part of the application, the main color, success, error and warning action colors as well as background colors. See figure 6.8.
- Paddings: A series of rules was established on the paddings to be followed in the components 20px horizontal 30px vertical.
- Borders: The border of the components should have a 25px radius.
- Font: All the text should have as font Roboto.
- Icons: The icons to use should be the collection of Feather Icons [68].

This seeks to maintain the aforementioned coherence so that any new contribution made in the design of the application maintains visual coherence with what already exists.

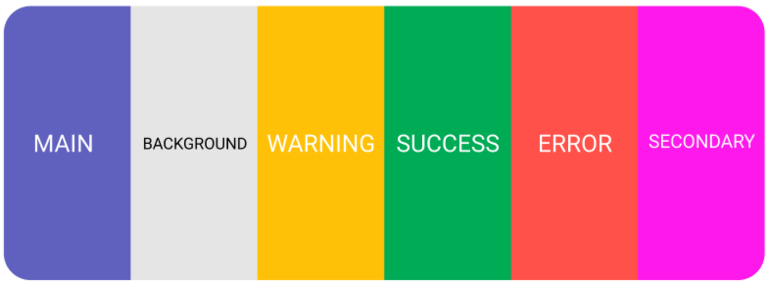


Figure 6.8: Color palette

Front-end Architecture

For the construction of the different user interface views, a series of guidelines were defined. These guidelines seek the simplest and most flexible way to build the individual views.

This designed system is based on the division of the visible space into three main parts. On the one hand the navigation bar, which allows you to navigate through the different views of the interface, on the other the app bar, that contains the main actions common to all the different views, and finally the content. While the first two remain unchanged between view and view, the last one will show all the identifying information of each one.

To make the content part as flexible as possible, the space was divided horizontally into 12 parts. Each information component had to occupy a percentage of those 12 parts, see figure 6.9. Thus, for example, if it is necessary to show a table of elements with many columns, an element that occupies 12/12 portions of the space will be defined, that is, it will fill the entire horizontal space, while if we have two elements that will not occupy the entire space, it will be assigned to each one a proportion of the space of 6/12, thus each occupying half of the viewing space. With this system it is possible to define, in a simple and clear way, the structure of the user interface homogeneously and forgetting to structure each view in a unique and individual way. It also allows you to reorganize the elements easily. What's more, it speed up the process of reorganizing the existent elements in different ways depending on the user device used, assigning different portions of the space based on the screen size.

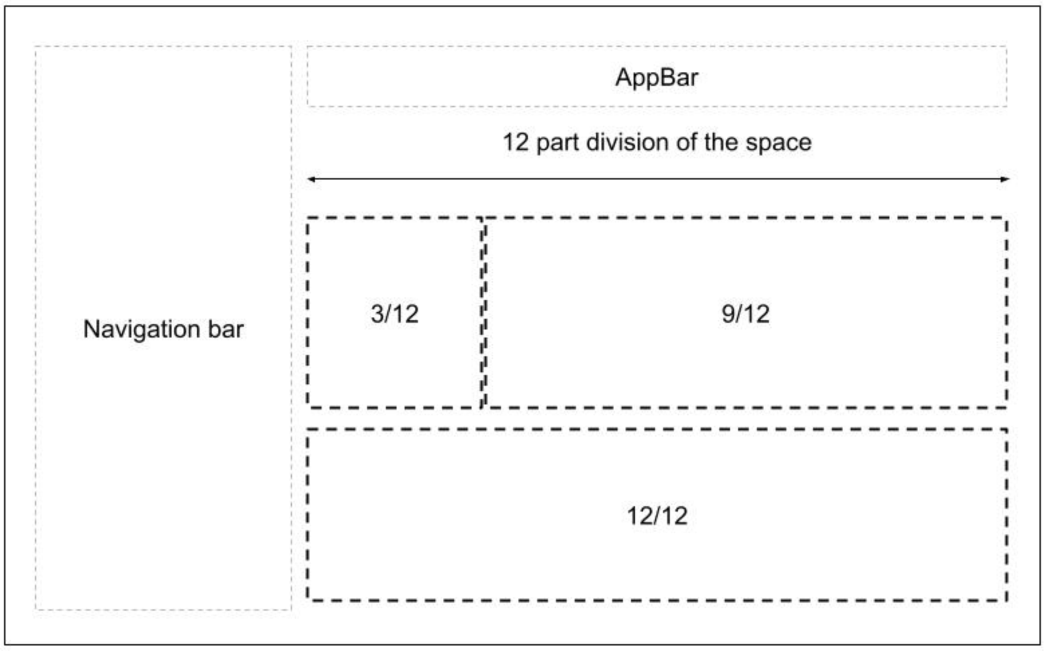


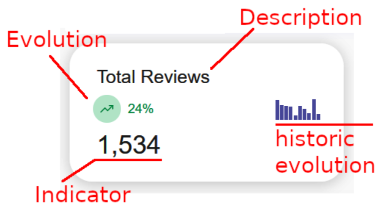
Figure 6.9: Front-end design architecture

6.6.3 Design and Prototype Home Screen

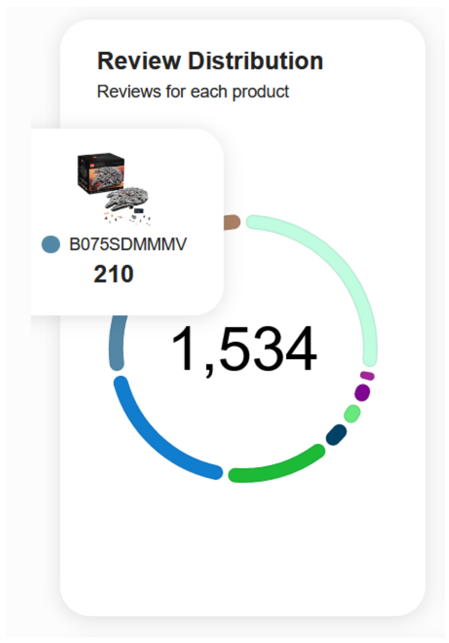
This view is made up of all the information regarding the set of reviews and products collected. It is essential for the user to have an overview of the status of the system, a quick way to check which products contain reviews and how many they have, as well as having the possibility to obtain the entire list of existing products from which to go to the product detail view.

With this in mind four components where designed:

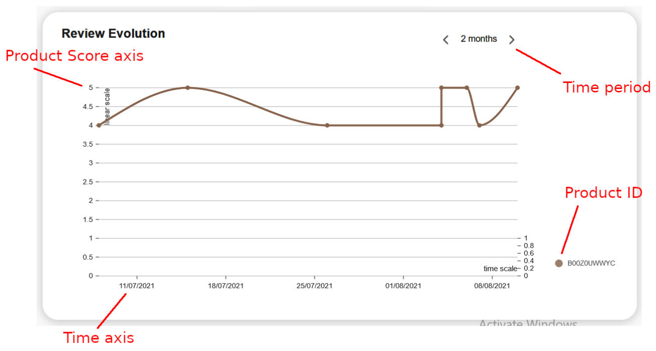
- Statistic Component: This component occupies 3/12 of the space, is designed to show a numeric number and allow the visualization of its evolution in time. See figure 6.10a.
- Pie Chart Component: This component occupies 3/12 of the space, is designed to show the distribution of the product reviews in addition to the total number of reviews. See figure 6.10b.
- Line Chart: This component occupies 9/12 of the space, is designed to show the evolution in time of the review score of a product/products. See figure 6.10c.
- Product Table: This component occupies 12/12 of the space, is designed to show the list of products with its information. It is the way to access the product screen of each of them. See figure 6.10d.



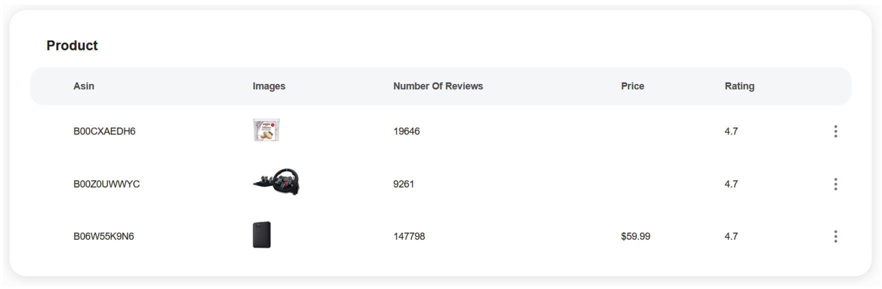
(a) Statistics Small Component



(b) Review distribution chart



(c) Reviews historic line chart



Product

Asin	Images	Number Of Reviews	Price	Rating
B00CXAEDH6		19646		4.7
B00Z0UWWYC		9261		4.7
B06W55K9N6		147798	$59.99	4.7

(d) Products Table

Figure 6.10: Front-end components design

6.6.4 Design and Prototype Product Screen

This view allows you to show the detailed information of a product: its score, name, description, its main characteristics, the reviews of the product itself, the polarity of each one of them, as well as how well or badly it is spoken about a certain product feature.

It is the most complex view and the one that has the greatest amount of information, as well as being the one that gives meaning to the front-end by allowing us to check what is the user's opinion about a certain product and its features.

For displaying the information, the following components were defined:

- Product Information: Provide the name, description and rating of the product, occupies 9/12.
- Product Positivity: Shows the percentage of positive reviews of the product, occupies 3/12.
- Product Feature Ranking: This component occupies 3/12 of the space, its main purpose is to rank the different product features based on a specific characteristic.
- Product Feature Viewer: Component with the graphical information of the feature. It shows, of each feature, the evolution in time of its positivity, in order to study the evolution of the opinion of consumers, at the same time that displays the score assigned to it. Occupies 12/12.
- Product Feature Table: Consists in a table with of all the features with their score, number of occurrences, positivity and also the list of reviews where each feature appears with the polarity of the sentence in which that feature appeared. Occupies 12/12.

6.7 Sprint 6: Front-end Implementation

This sprint was defined by the choice of technology to use and the implementation of the front-end. Once the design and prototype had been completed, it was necessary to move on to the implementation, transferring all the flows defined in the prototype and turning it into reality. For this, it was decided to use React [43], a JavaScript (JS) library designed for the creation of user interfaces focused on the single page application model. This decision was made since the desired application model fitted very well with the concept of one-page application, in addition to being a framework with a large documentation, user base, easy to use and existed previous development experience. Furthermore, it contains a large number of packages that implement the functionalities and basic components to speed up the construction of user interfaces.

6.7.1 Typescript

JavaScript is an extremely versatile language, every data structure can be considered an object and the dynamic typing of variables makes it extremely flexible. But this versatility and flexibility come at a price, the maintainability and readability of the code is very poor, the bigger the project, the easier it is to lose sight of what each element is, in addition to allow programming mistakes more frequently. For this reason, if the project was going to be more than an experiment, it was necessary to consider the use of a tool that covered these JavaScript gaps, while maintaining its main benefits. The tool selected was Typescript [41][69], a programming language built on JavaScript that adds a series of features that substantially improve it, at the same time that keeps almost full compatibility with JS. The table 6.8 shows the main differences between Javascript and Typescript.

JavaScript vs TypeScript		
Characteristic	JavaScript	Typescript
Static typing	No	Yes
Generic types	No	Yes
Structural typing	No	Yes
Enumerates	Can be simulated	Yes
Tuples	No	Yes
Interfaces	No	Yes
Object Oriented Programming	Yes	Yes but with a simpler syntax similar to other languages

Table 6.8: JavaScript vs TypeScript

6.7.2 React UI Framework

From the start of the development, the idea of using a React Framework containing the main React components (buttons, tables, inputs...) was proposed. It had no sense to build from scratch those components, loosing time building and testing them, instead of using a framework flexible enough to meet the needs required, and to adapt to the chosen design.

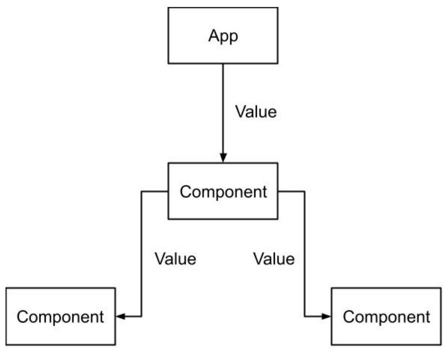
Two main options were proposed:

- Material-UI [70]: React framework with a huge number of components based on the material design system but with the flexibility of using your own.
- Ant-Design [71]: React framework with containing a set of components following the ant design specification.

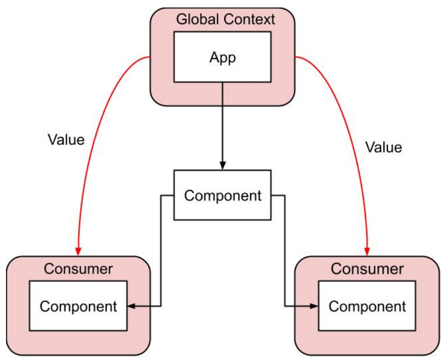
Of the two options, Material-UI was chosen since the design proposed contained many similarities with the material design system and the flexibility that the framework provided, compared to Ant-Design, when customizing the components, was important.

6.7.3 Global State

One of the key elements when building the front-end, was the use of a global state containing the basic information needed for all the views to work. A global state provides a way to share values between components without having to explicitly pass that value through every level of the component tree. The information that needed to be contained in that global state was the list of existent products and the basic review stats. The figure 6.11 show the different data flow that a global state application have in comparison with a non global context application.



(a) Normal data propagation

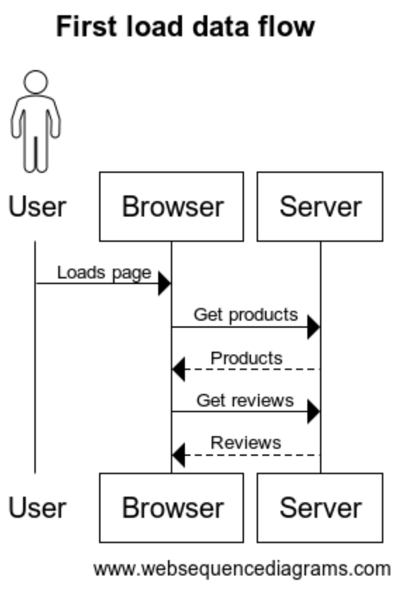


(b) Global context data propagation

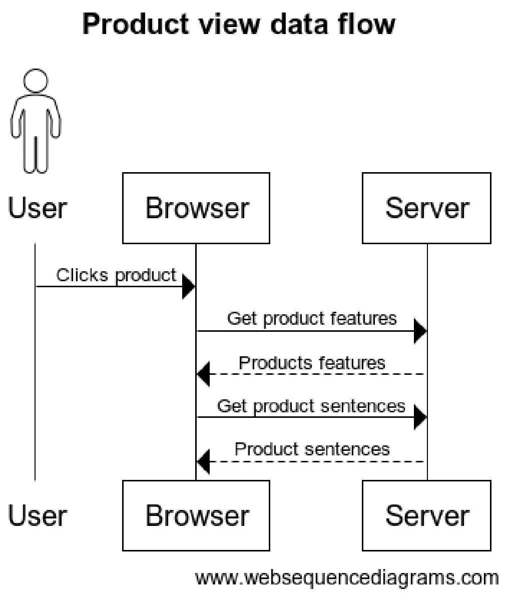
Figure 6.11: Different component tree data flows

6.7.4 Data flow between front-end and back-end

The application was thought to be a tool to visualize data, so the main flows between front and back were the data retrieval. On one hand, the first data load, retrieving the general data needed for the global state, and in the other, the data retrieval once you enter on the product view. Both flows are shown in the figure 6.12.



(a) First load

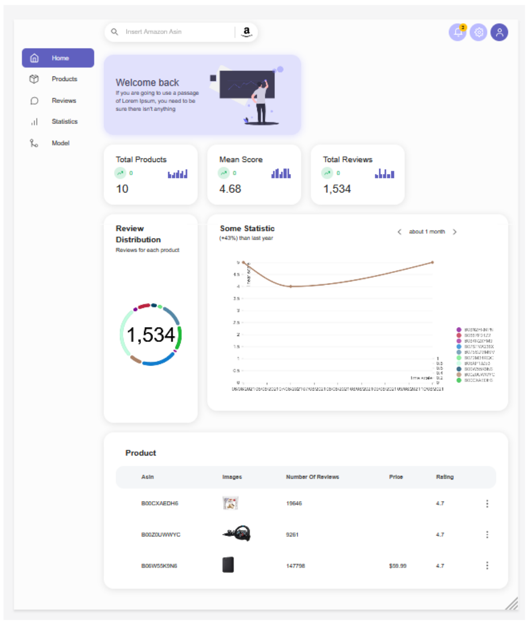


(b) Product view load

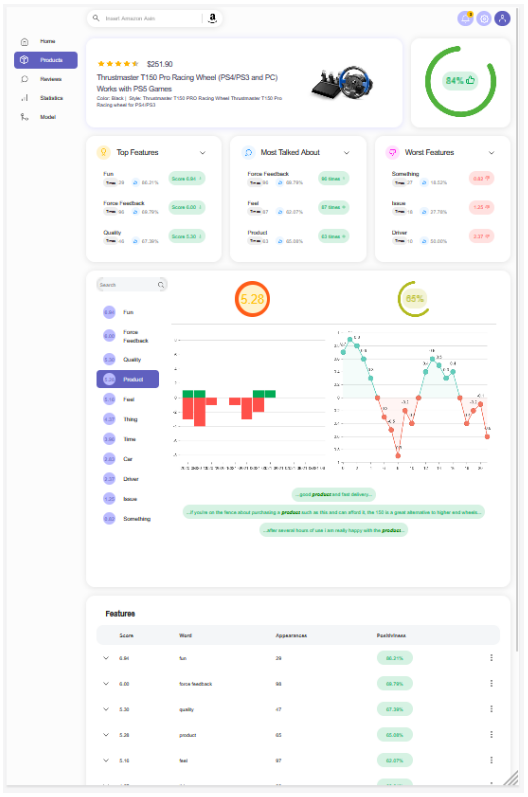
Figure 6.12: Data flows for data retrieval

6.7.5 Final Result

The final result was a simple, minimalist and extensible design following the principles created, that made the foundations to build the front-end with the knowledge of which elements to build, how to organize them and the different flows between views. The figures 6.13 show the final result of the sprint.



(a) Home view


Thrustmaster T150 Pro Racing Wheel (PS4/PS3 and PC) Works with PS5 Games
84%
Top Features
Most Talked About
Worst Features
5.28
Features

(b) Product view

Figure 6.13: User interface design

6.8 Sprint 7: Gloover refinement

With everything connected, from the user interface to the Scraper system and Gloover, some details had to be refined. With the use and manual testing of the tool, it was shown that some elements didn't work as well as expected. For example, the initialization of the tool costed too much time as the classification model had to be trained each time (no model persistence) and the ranking system made for the features was too simple (percentage of positiveness), so did not provide good results.

6.8.1 Classification model persistence and management

Until now, when starting Gloover, it created a classification model from a series of reviews defined in a file [56]. Starting this process every time it was launched, implied that the system could take minutes to start, since it had to process all the reviews and train the classifier with them to generate the model. Logically, this could be solved by having a classification model management system, loading a previously trained model when starting Gloover.

For this, a series of actions were proposed to be able to manage the models, which are reflected in the following endpoints table 6.9.

REST API Methods		
Method	Resource	Description
GET	/classifier/models	Get the stored models
GET	/classifier/models/current	Get the current loaded model
POST	/classifier/models	Create a new model based on a file or database documents
PUT	/classifier/models/<id>	Loads the model with id <id>
DELETE	/classifier/models/<id>	Deletes the model with id <id>
GET	/classifier/models/<id>/test	Get the accuracy of a the model with id <id> against the database

Table 6.9: Gloover model management REST API methods

Creating a model

The first step was the model creation method. A function that made possible creating a certain model passing the route of a file or indicating that the current database would be the training data.

Without modifying too much the structure of the current system, as it had already a way to load the reviews from a file, the REST API method was built following those requirements.

Saving a model

The key step and main reason to build the managing system. Saving a created model was extremely important, as the process of creating one each time, was slowing the initialization of the entire system.

To be able to save a model it was necessary to use joblib [72], a python package that allows, among other things, to load and save sklearn classification models. With this package saving a model was extremely easy, as it provides a function to dump a model into a file with anything but the file path and the model as parameters. The process of saving a model was decided to start automatically each time a model was created.

Loading a model

Once a model was created it needed to be loaded in some way. Joblib, in addition to allow saving a model, it offers a way to load it, and as simple as the saving method, it only requires the model path as parameter.

Since the main problem to solve was the creation of a model each time the system started, the automatic load of the model was proposed as a way to solve the issue. That process consist in finding the most recent created model and loading it when launching Gloover.

Testing a model

When creating or using a classification model is important to know how accurate its classification is. As a way to check the accuracy of a model. a method for testing it was created, allowing a user to select a certain model and test it against a collection of documents.

Deleting a model

Creating and loading a model is really important but deleting a model that is not useful is also an important matter. When testing different configurations for extracting the most accuracy from a classification model, a lot of different ones can be created and not all of them are going to be useful, so the creation o a method to delete them was designed. As joblib save the models in a file, deleting them was as easy as deleting a text file using the built in functions of python.

6.8.2 Feature ranking

Prior to this sprint, the features were ranked using the % of times that a feature appeared in positive phrases, the higher the percentage, the higher its score. Although it may seem like a simple and effective way to organize the information, it present the next problem: A feature with 1000 appearances with 70% positives would appear in the ranking below a feature with 10 appearances with 80% positivity.

Is easy to see that this method favors low frequency elements and is not a great solution, since a feature that has been talked about a great amount of times has to be given more importance than one that is marginally mentioned as its reviews could not be representative.

Lower bound of Wilson

The solution to the prior problem was the use of the lower bound of Wilson [73] [74] to obtain the score of a feature. The lower bound of Wilson gives a way to balance the proportion of positive ratings with the uncertainty of a small number of observations following the formula shown in figure 6.14.

p = (n° of positive ratings)/(Total ratings)

n = Total ratings

$Z\alpha/2$ = Quantile of the standard normal distribution

$$\left(\hat{p} + \frac{z_{\alpha/2}^2}{2n} \pm z_{\alpha/2}\sqrt{[\hat{p}(1-\hat{p}) + z_{\alpha/2}^2/4n]/n}\right)/(1 + z_{\alpha/2}^2/n).$$

Figure 6.14: Lower bound of Wilson

In figure 6.5 we can see how, despite the fact that feature No.2 has a percentage of positivity of 81.25%, it is ranked lower than feature No.1 with 75.6% of positivity, since it gives more weight to the fact that No.1 have a total of 336 appearances.

```
[
    {
      "feature_id": "8364d003-21da-49d7-99cb-824141740e80",
      "id": "91cf2f31-3dd2-424f-916d-b83df787e05a",
      "negative": 82,
      "positive": 254,
      "score": 0.7073014668439815
    },
    {
      "feature_id": "aec6d7be-18c4-4689-b70d-d747915c23d3",
      "id": "0f10f103-5871-4373-84dd-2386f8b4ed25",
      "negative": 6,
      "positive": 26,
      "score": 0.6469084479862404
    }
  ]
```

Listing 6.5: Real server response when asking for feature statistics

6.9 Sprint 8: Front-end refinement

The last sprint was based on the development of two new functionalities on the front, being able to scrape a product and be able to check the status of the scraping from it.

The ability to scrape a product from the front-end was the only remaining key flow, one of the essential parts for using the tool. But given the ease of use of the REST API for this, its incorporation in the user interface had been delayed. For developing the features, two main tasks where done:

- Adding a button to be able to scrape from the front-end.
- Including the visualization of the state of the scraped data

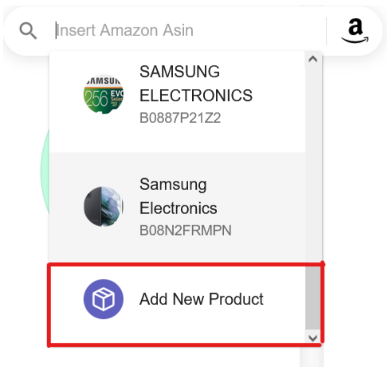
6.9.1 Add scrape button

One of the user interface flows that where not implemented was a method to start a scraping from the front-end.

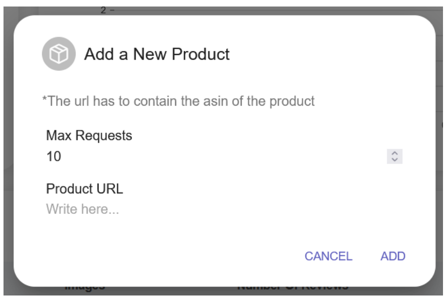
This flow was defined as follow:

- The user uses the search bar to search a certain product, if the product does not exist, an option allowing him to add a new one is enabled.
- The user clicks on the button and a form to fill the scrape fields appears.
- The user fills the scrape fields and it can be submitted.
- Once submitted, the response is stored in the global state.

The result of the process can be checked in the figures 6.15.



(a) Add product button



*The url has to contain the asin of the product

Max Requests
10

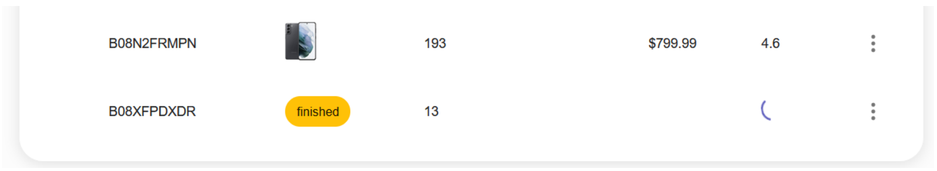
Product URL
Write here...

(b) Add product form

Figure 6.15: UI elements for product scraping

6.9.2 Visualization of scrapping status

The other important flow was the visualization of the scraping status (see figure 6.17), that is, once the scraping of the product had started following the previous flow, it was necessary to check the process progress. For this, it was decided that the products that were being scraped would be displayed in the list of products but showing a label that indicated the status of the scraping instead of the normal information, see figure 6.16. Once finished the product would appear in the list as the rest of them.



B08N2FRMPN		193	$799.99	4.6	⋮
B08XFPDXDR	finished	13			⋮

Figure 6.16: UI scraping state

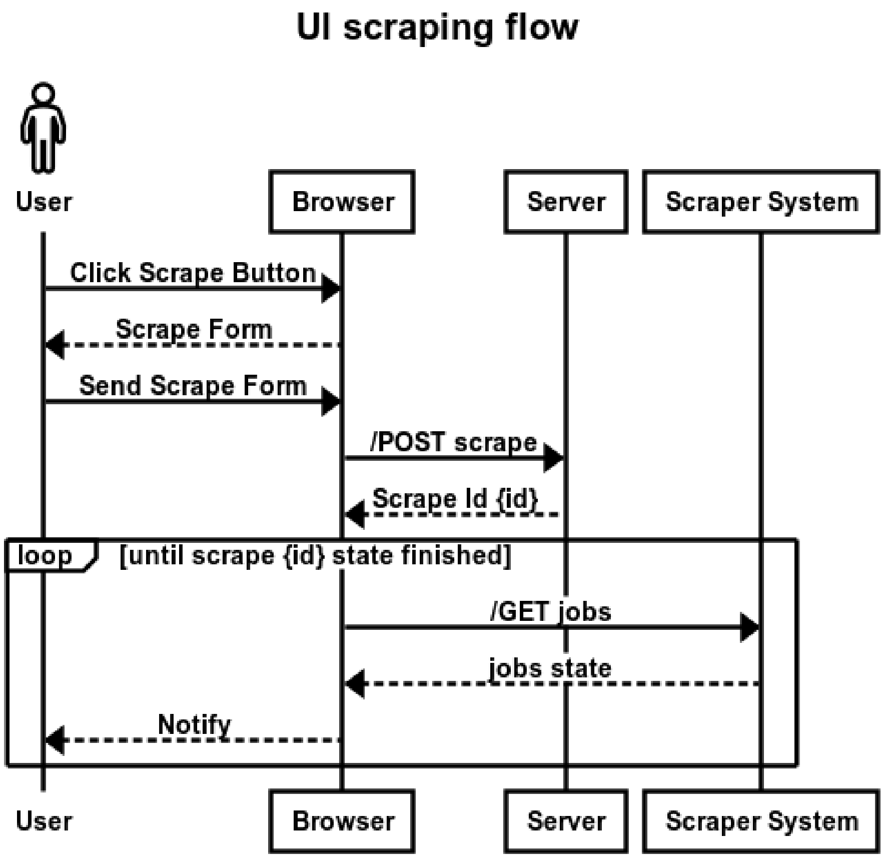


Figure 6.17: UI scraping flow

Chapter 7
Testing

This section showcases the testing made to validate the correctness of the different features of the work.

7.1 Back-end

Most of the complex element of the project were located in the back-end, since it was the computational part, which meant that it required the majority of the tests.

This section is divided in two parts, corresponding with the two micro-services of this work, were testing will be discussed.

7.1.1 Gloover

For this part of the project multiple tests were made, one for each functionality. The approximation used for all the test was unit-testing. Unit tests are automated tests written to ensure that a service of an application behaves as intended.

In this case three unit test were developed using the python core assertions to check that the expected result of each service returned the correct information. This test are those referring to:

- **Feature Selector**: Unit test checking that the functionality is able to select the different product features from an array of reviews. See listing 7.1.

```
if __name__ == "__main__":
    reviews = load_reviews("testing/reviews.json")
    product_asin = "TEST-ASIN"
    simple, complex = feature_selector(product_asin, reviews)
    assert simple.__contains__({"word": "price"}), True
    assert simple.__contains__({"word": "screen"}), True
    assert simple.__contains__({"word": "color"}), True
    assert len(complex), 0
```

Listing 7.1: Example of unit test for the feature selector

- **Sentences Extractor**: Unit test checking that the feature extractor is capable of obtaining the sentences were each feature appears in a set of reviews. See listing 7.2.

```
  if __name__ == "__main__":
    reviews = load_reviews("testing/reviews.json")
    product_asin = "TEST-ASIN"
    simple, complex = load_features("testing/features.json")
    feature_sentences = sentence_extractor(reviews, product_asin,
    simple, complex)
    assert feature_sentences.__contains__({"sentence": "the color
    is beautiful", "feature": "color"}), True
    assert feature_sentences.__contains__({"sentence": "the screen
    has resolution of only 720p", "feature": "screen"}), True
    assert feature_sentences.__contains__({"sentence": "price is
    good", "feature": "price"})
```

Listing 7.2: Example of unit test for the sentences extractor

- **Classifier**: Test to check that the saved test model is able to classify correctly two simple sentences. See listing 7.3.

```
if __name__ == "__main__":
    classifier = Classifier(test_size=0.2, train_size=0.2,
    update_existent=True)
    assert (classifier.classify(['This is the best']), True)
    assert (classifier.classify(['This is the worst']), False)
```

Listing 7.3: Example of unit test for the classifier

7.1.2 Scraper

For the scraper system, since mocking the calls to the different web pages with its content was really complex, manual testing was done.

Multiple tests were done to check that everything worked fine, from scraping test pages like ”`https://scrapeme.live/shop/`”, to scraping real products in amazon making sure that all the data was obtained in the correct way.

The manual testing was enough for this part, because the amount of features was and also the mutability of the code was low. Only one real feature was develop and it was not going to change much.

7.2 Front-end

For this section End-to-end (E2E) testing was made, a technique that tests the entire software to ensure the application flow behaves as expected. It ensures all integrated pieces work together as expected.

During the development of the front-end each feature was accompanied by its corresponding E2E test. This tests were done launching the back-end and front-end simultaneously and using the different UI features to check the result of the each action.

7.3 End of sprint functional testing

Finally for each end of sprint a functional testing was made in conjunction with the thesis directors to check that all the functional requirements of each sprint were met. This testing consisted in running the application and checking that the objectives were correctly achieved.

Chapter 8
Results

At the end of all the sprints, the result was the creation of an autonomous and automatic sentiment analysis system for Amazon products, but with the possibility of an easy extension to other data sources.

Thanks to the study and development of this system, it was possible to deeply explore both sentiment analysis and natural language processing matters, since in order to obtain information on what is the opinion contained in a text, it is necessary pre-process the text. With this exploration, new knowledge about statistics and parallel processing was acquired, since most of the NLP algorithms are based in the statistical approach, regarding the parallel processing, given that the system required a big amount of data to work, optimizing and improving the performance of it was necessary.

In addition, since a scraping system and a front-end were built, it required to reaffirm the use of spiders/web crawlers and even proxies, improving the experience and knowledge about them, also skills and fluency in the use of react and web design were improved since the use of typescript with react was decided to use to build the front-end.

All this knowledge acquired and achievements during the development of this work are reflected in the following section.

8.1 Gloover development

Regarding the implementation of the sentiment analysis system, all the objectives were achieved.

8.1.1 Text classifier

Gloover's main tool is the text classifier, which was the first objective to be reached. In the final system, this classifier was built using a SVM with a chain of text transformers and the use of a dataset of reviews as training. The result was a system capable of differentiating between a negative review and a positive one with 80% accuracy.

8.1.2 Feature Extractor

The other essential piece of the system is the feature extractor, an element capable of obtaining those differentiating features of a product through the analysis of reviews on said product. With this, a tool capable of detecting with certain precision those elements that make up a product was sought. At the end a precision close to 70% was reached in the detection of characteristics. This precision was measured manually, checking the result of the system and discarding those undesirable characteristics.

Some examples of precision when detecting features can be shown in table 8.1.

Product features precision table			
Product	Correctly detected	Wrongly detected	Precision
Nintendo Switch	20	9	68.96%
Acer Swift	14	7	66.67%
Horizon Zero Dawn	24	12	70.58%
Lego Star Wars 75313	18	7	72.00%

Table 8.1: Product features precision table

8.1.3 REST API

Another of the main objectives was to expose the functionalities of the system through a REST API in order to have a front-end that would facilitate the use and understanding of the tool (see table 8.2). This part was relatively easy since the python flask package makes it very easy to build this type of tools.

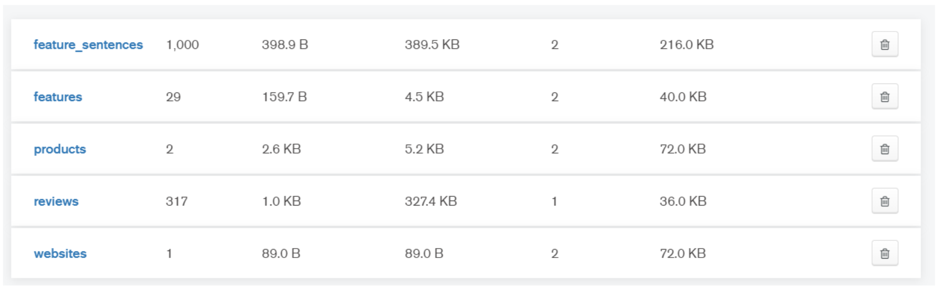
REST API Methods		
Method	Resource	Description
POST	/scrape	Sends a scraping request to the Scraper system
POST	/scrape/notify/<scraper_id>	Notifies the system that the scrape with id <scraper_id> has finished
POST	/classifier/classify	Returns the polarity of a text
GET	/database/reviews	Returns the reviews stored in the database
GET	/database/reviews/<id>	Returns the review with id <id>
GET	/database/products	Returns the products stored in the database
GET	/database/features	Returns the product features stored in the database
GET	/database/sentences	Returns the feature sentences stored in the database
PUT	/database/features/update	Updates/Creates the features of a certain product and stores them in the database
PUT	/database/sentences/update	Updates/Creates the feature sentences of a certain product and stores them in the database
GET	/classifier/models	Get the stored models
GET	/classifier/models/current	Get the current loaded model
POST	/classifier/models	Create a new model based on a file or database documents
PUT	/classifier/models/<id>	Loads the model with id <id>
DELETE	/classifier/models/<id>	Deletes the model with id <id>
GET	/classifier/models/<id>/test	Get the accuracy of a the model with id <id> against the database

Table 8.2: Gloover REST API methods

8.1.4 Data persistence

Finally, another of the most important objectives was to allow the data to persist in a database in order to have a good management of the processed data, as well as to be able to serve the users with historical information about the data.

An example of the resulting mongo database structure can be checked in figure 8.1



feature_sentences	1,000	398.9 B	389.5 KB	2	216.0 KB
features	29	159.7 B	4.5 KB	2	40.0 KB
products	2	2.6 KB	5.2 KB	2	72.0 KB
reviews	317	1.0 KB	327.4 KB	1	36.0 KB
websites	1	89.0 B	89.0 B	2	72.0 KB

Figure 8.1: MongoDB example collections

8.2 Scraping system development

For this part of the project two main objectives were to be achieved.

8.2.1 Scraper

The first one was the construction of the scraper, a tool to obtain data from web pages in a structured and automatic way. Thanks to scrapy, reaching this goal was relatively easy as the mentioned package exposes a rich amount of functionalities that facilitated the construction process.

8.2.2 REST API

The second one was exposing the functionality of scrapy via REST API, to be able to use the tool by the front-end and Gloover. Again the use of flask made the goal really reachable.

8.3 Front-end development

Regarding the front, two main objectives were achieved.

8.3.1 Data visualization

The main point of the development of the front-end was to be able to visualize all the data obtained by the back-end. This visualization needed to be simple and understandable, so multiple components were needed to achieve this, each one with a specific piece of information.

8.3.2 Use of Back-end functionalities

Another main goal was to be able to facilitate the use of certain functionalities of the back-end in an easier way, since calling the API directly can be tedious, from constructing the request to reading the response.

Because of all of this matters, some of this API calls were simplified by the front-end using simple buttons and forms.

8.4 Bonus objectives

In addition to the points mentioned above, an extra objective was achieved. The dockerization [75] of the application to facilitate the installation and deployment [76] of the tool.

Since three independent projects were created, and installing, building and connecting all of them is a difficult task, dockerizing all the project, including databases, using docker compose was a great goal. Docker compose allows to achieve those objectives in two simple steps, the creation of the configuration file and the use of the command `docker-compose up` . The following listing shows the configuration file `docker-compose.yml`(listing 8.1) that was needed to build all docker containers.

```yaml
version: '3.7'
networks:
  gloover_network:
    driver: bridge
services:
  gloover_ws:
    container_name: gloover_ws
    build: .
    ports:
      - "5000:5000"
    environment:
      MONGODB_DATABASE: gloover_db
      MONGODB_USERNAME: gloover_user
      MONGODB_PASSWORD: password1
      MONGODB_HOSTNAME: mongo
    volumes:
      - .:/var/www
    links:
      - mongo
    networks:
      - gloover_network
  gloover_scraper:
    container_name: gloover_scraper
    build: ./gloover_scraper
    ports:
      - 9080:9080
    volumes:
      - ./gloover_scraper:/var/www/app
    networks:
      - gloover_network
  mongo:
    image: mongo
    container_name: mongo
    ports:
      - 27017:27017
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: Secret
    volumes:
      - mongo-data:/data/db
      - mongo-configdb:/data/configdb
    networks:
      - gloover_network
volumes:
  mongo-data:
  mongo-configdb:
```

Listing 8.1: Docker compose file for the project

Chapter 9

Conclusion and future research

IMPLEMENTING a data mining system is a complex task, even more so if you add the fact that a scraping system and a user interface were also developed alongside it. Despite this, at the end of the development of this work, a complete system was obtained that allows extracting product reviews from any information source using scraping, analyzing said reviews to extract the most relevant characteristics of a product and making an analysis of feelings of the same intensely in the obtained reviews and characteristics. All this accompanied by a user interface that allows you to use all these systems in a simple way and view the extracted information in an orderly and legible manner.

9.1 Conclusion

The little information related to the mining of feelings, compared to other areas, made the acquisition of knowledge on the subject slower than expected, in addition to this was added the fact that it was an area little worked on before, with which also existed a constant learning on the subject.

Thus, during the development of the work it was learned about the use of different classifiers to categorize reviews and related to it, the different necessary preprocessing in the texts to facilitate that categorization. In turn, the use of various mathematical formulas applied to the field of sentiment mining were also a great tool when it came to improving the project and learning about mathematical theorems.

In addition, given the existence of a scraping system and a front end, it was also necessary to document data mining and the use of proxies (to avoid IP blocking), and web development and react, although on this last part I already There was a good base of knowledge given by previous personal projects.

In summary, it was a work full of new frontiers to be explored, on topics that were lacking in context at first, which made research on it a continuous learning process. In addition to the breadth of the project, touching on numerous topics, from sentiment mining, to web crawling and web development, you have to think that it was necessary to devise the entire project thinking of it not as an isolated system but as a complete and interconnected product. Although it is true that the most fundamental bases already existed thanks to the degree, and important topics for the project such as text processing had been touched in the computing mention studied, it was quite an experience that managed to expand the existing frontiers in many of the computer science environment and that will serve later to improve this same project and develop new ones.

9.2 Future research

Despite the fact that the objectives have been met, the project could benefit from improvements in some areas. This section includes some of this potential improvements and research lines:

1. **Job based feature review analysis and extraction:** In the current state of Gloover, the analysis and feature extraction of the review products is not job based which limits the amount of requests that the system is capable of accepting, and also since those executions are not asynchronous the REST API can throw a timeout exception, so making something similar to what is happening in the scraping should do the work.

2. **Add user authentication:** One of the main shortcomings is the lack of user authentication in the project, the security improvement of using authentication is significant. Also, with the inclusion of this feature some new functionalities could be added:

   - Seen only the products that you selected/scraped on your dashboard.
   - Limit the amount of scraping request per user.
   - Block some functionalities that only the admin should use.

3. **Feature merging:** The feature extractor could be improved if, when extracting features for a product, two extracted similar features like ”battery” and ”battery life” for example, were merged into one so the post analysis would be more accurate since those two features are basically the same.

4. **New sections in front-end:** The front-end needs to be improved adding more sections, right now for time reasons, the only two sections that exist are the dashboard where there is a summary of the available data and the product section where you can see the details of a product, but more sections could be added.

    - Classification Model Section: A section with all the available options to manage classification models, from training a new one to using an already trained.
    - Reviews section: A section to check for all the reviews available.
    - Comparison section: A section where you could compare two products to check the differences between them.

Appendices

Appendix A

User Manual

THIS appendix describes the process to install and run the project in your local computer in addition to a little tutorial of how to propperly use the tool.

A.1 Installation

For the installation and deployment of the project, docker [77] was used for the back-end and npm [78] for the front-end.

A.1.1 Back-end installation

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

A.1.2 Prerequisites

To install and run the project you will need the following:

- Docker & Docker compose [79]
- SentiWords lexicon [50] located in `resources/lexicons/SentiWords_1.1.txt`
- Amazon product reviews dataset [56] located in `resources/datasets/reviews_Cell_Phones_and_Accessories_5.json`

Installation

First clone this repo to your local machine: `https://github.com/P-Duran/gloover.git`

Setting up environment

- Make sure docker is running
- Open a terminal in main project's folder

Once all the prerequisites are met, the final step to run the project is executing the following command `docker-compose up --build`, to tell docker to build all the containers with their scripts.

Configuring MongoDB

Once docker has successfully built the project, a MongoDB container will appear without any configuration. To leave the database ready to use, the following steps are required:

- Access MongoDB container `docker exec -it mongo bash`
- Once inside the container, log in to the MongoDB root administrative account
  `root@893759837:/# mongo -u <root-user> -p <root-password>`
  (Mongo admin ceredentials can be found here)
- Create gloover_db database `mongo> use gloover_db`
- Create the application user that will be allowed to access the database
  `mongo> db.createUser({user: <user>, pwd: <password>, roles: [{role: 'readWrite', db: <database>])`
  (Gloover app credentials can be found here)

Check that it's up and running

Now everything should be correctly configured and installed. To verify a simple API call could be done to `GET http://localhost:5000/info` and the following response should be returned `status=True, message='Server is running!'`

A.1.3 Front-end installation

Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes.

A.1.4 Prerequisites

- Install NVM [80] and install last version of npm `nvm install "latest"`
- Install your preferred web browser

Installation

First clone this repo to your local machine: https://github.com/P-Duran/gloover_frontend.git

To install the front-end go to the main project folder and run `npm install & npm start`

Check that it's up and running

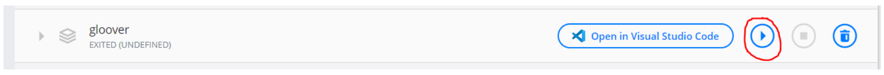
If everything went fine a server listening in `http://localhost:3000` should be accessible via web browser.

A.2 Using the tool

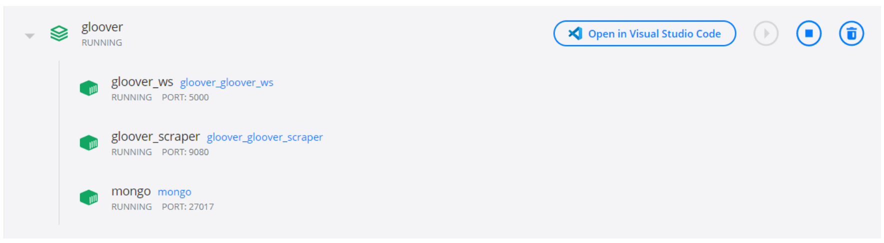
This section explains how to use the tool, from launching the back-end and front-end to understanding the UI.

Starting the back-end

If all the installation process worked correctly and docker is running in your computer, to start the back-end it is only required to run the container *gloover* as shown in figure A.1



(a) Gloover container not started

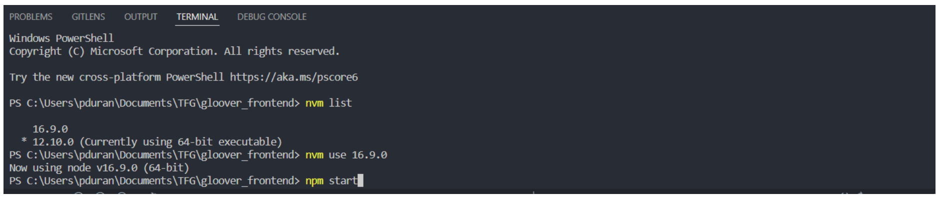

gloover
RUNNING
Open in Visual Studio Code
gloover_ws gloover_gloover_ws
RUNNING PORT: 5000
gloover_scraper gloover_gloover_scraper
RUNNING PORT: 9080
mongo mongo
RUNNING PORT: 27017

(b) Gloover container up and running

Figure A.1: View of gloover container management in docker desktop

Starting the front-end

In this step, same as before, if the previous intallation process worked, it should be easy to run it, only go to the react main project folder and run `npm start` in a terminal (also choose the correct npm version if nvm in use) as shown in figure A.2.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\pduran\Documents\TFG\gloover_frontend> nvm list

    16.9.0
  * 12.10.0 (Currently using 64-bit executable)
PS C:\Users\pduran\Documents\TFG\gloover_frontend> nvm use 16.9.0
Now using node v16.9.0 (64-bit)
PS C:\Users\pduran\Documents\TFG\gloover_frontend> npm start
```

Figure A.2: Running the front-end

Once the startup of the react project has finished, it should be available in `http://localhost:3000` from any browser, as show in figure A.3

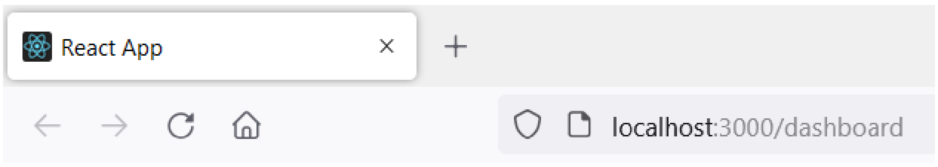
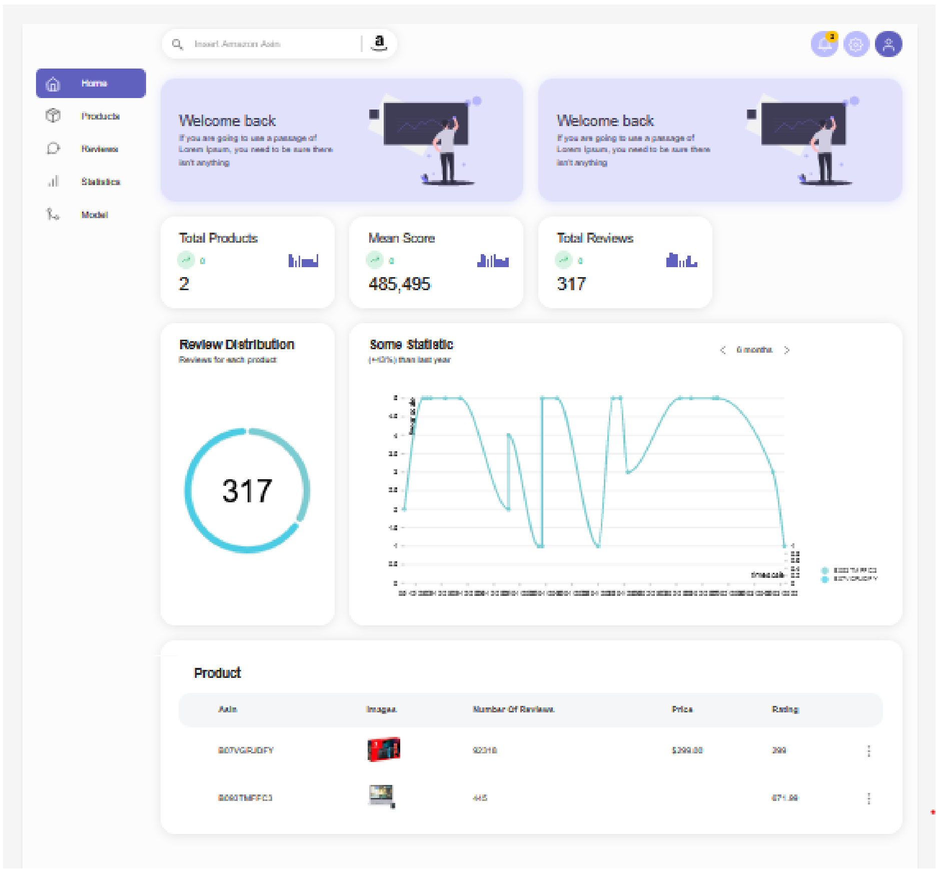


Figure A.3: Accessing the front-end via browser

A.2.1 Main Page

Once the front-end is correctly launched and can be accessed via browser, the main page will appear, figure A.4.

Here the user can visualize multiple product data in various formats, to understand the product reviews.


Insert Amazon Asin

Home
Products
Reviews
Statistics
Model

Welcome back
If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything

Welcome back
If you are going to use a passage of Lorem Ipsum, you need to be sure there isn't anything

Total Products
0
2

Mean Score
0
485,495

Total Reviews
0
317

Review Distribution
Reviews for each product
317

Some Statistic
(+43%) than last year
6 months

Product

Asin	Images	Number Of Reviews	Price	Rating
B07VGRJDFY		92316	$299.00	299
B080TMFFC3		445		671.99

Figure A.4: Main Gloover front-end page

Analyzing the historic review data of a product

The front-end exposes a lot of information about products, one of these features has been built for analyzing the historic data of a product as shown in figure A.5.

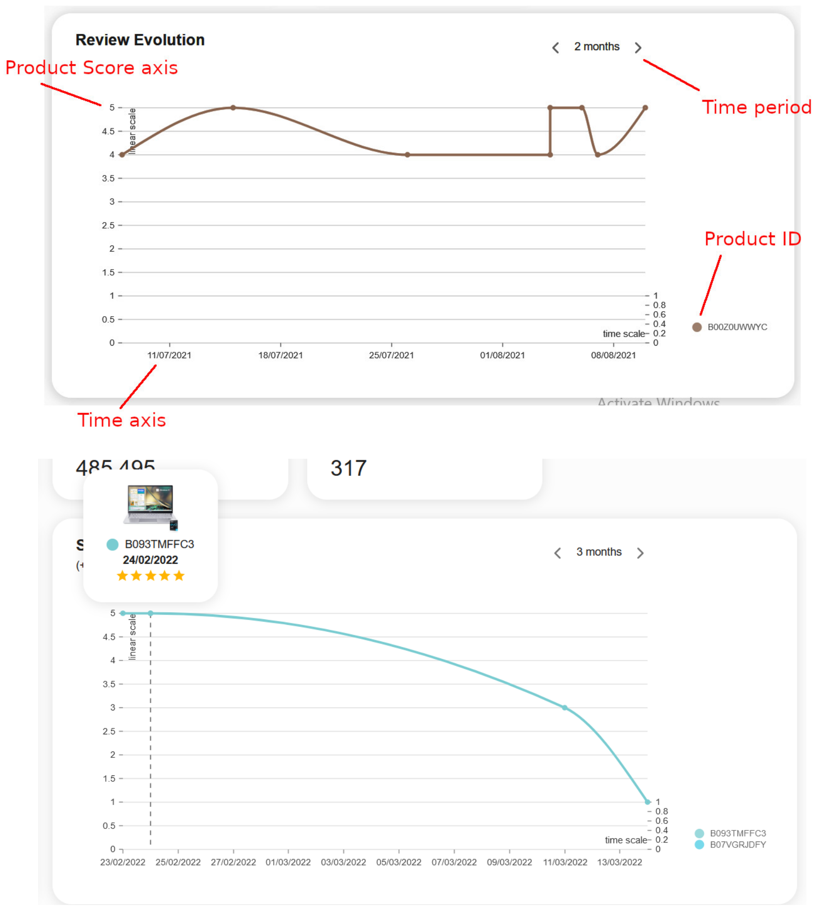


Figure A.5: Historic view of the reviews of a product

This component can be used to check the review evolution of a specific product in a time period.

Checking the review distribution of analyzed products

To know how many reviews in total have been analyzed and how many correspond to each product a component with a pie graphic can be used, see figure A.6. This graphic exposes the information of how many reviews have been analyzed for each product and also shows the total amount of reviews analyzed.

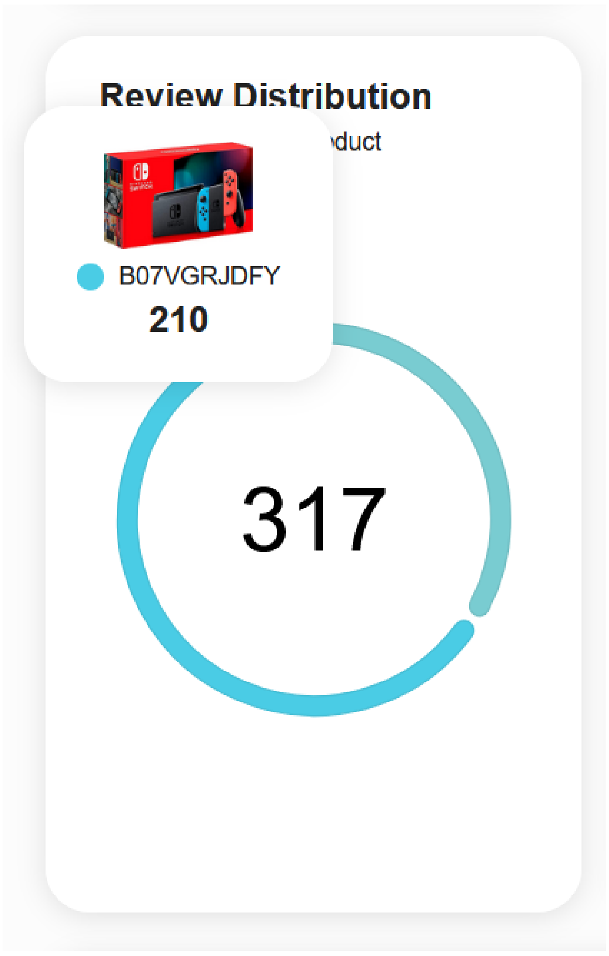
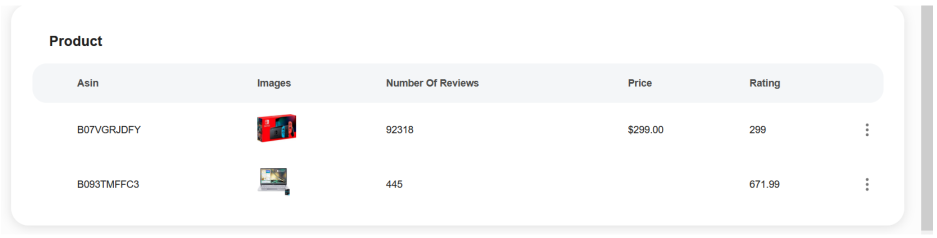


Figure A.6: Review distribution graphic

Product list

Finally, in the main page there is an analyzed product list that shows all the products with reviews analyzed by the system (see figure A.7). This list shows some information like total number of reviews, price and score, but the important functionality is that it allows to navigate to each product page (when clicking on each list element), where a lot of information about the specific element can be shown.



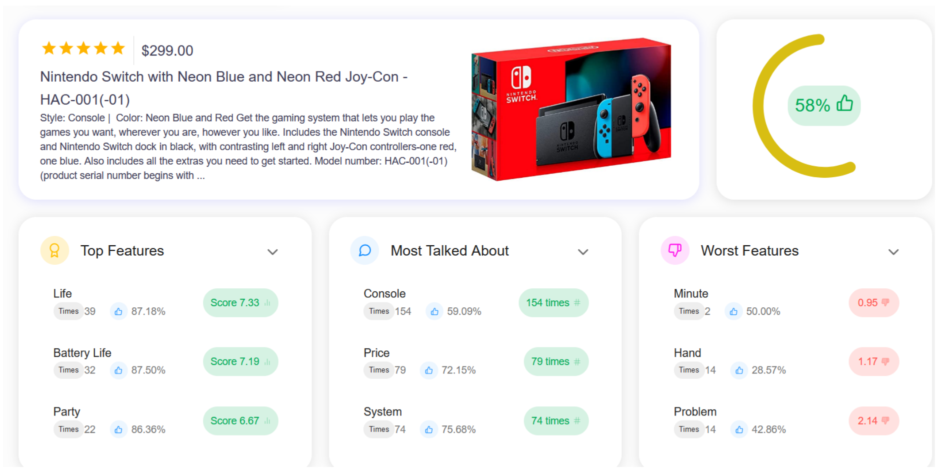
Product

Asin	Images	Number Of Reviews	Price	Rating
B07VGRJDFY		92318	$299.00	299
B093TMFFC3		445		671.99

Figure A.7: List of products

Product detail view

Once inside a product detail view all the information related to that particular product is displayed, from score, description, price and percentage of positive reviews, to the features of that same product and how positive each one of them are. See figure A.8.



★★★★★ | $299.00

Nintendo Switch with Neon Blue and Neon Red Joy-Con - HAC-001(-01)

Style: Console | Color: Neon Blue and Red Get the gaming system that lets you play the games you want, wherever you are, however you like. Includes the Nintendo Switch console and Nintendo Switch dock in black, with contrasting left and right Joy-Con controllers-one red, one blue. Also includes all the extras you need to get started. Model number: HAC-001(-01) (product serial number begins with ...

58%

Top Features

Life
Times 39 87.18% Score 7.33

Battery Life
Times 32 87.50% Score 7.19

Party
Times 22 86.36% Score 6.67

Most Talked About

Console
Times 154 59.09% 154 times

Price
Times 79 72.15% 79 times

System
Times 74 75.68% 74 times

Worst Features

Minute
Times 2 50.00% 0.95

Hand
Times 14 28.57% 1.17

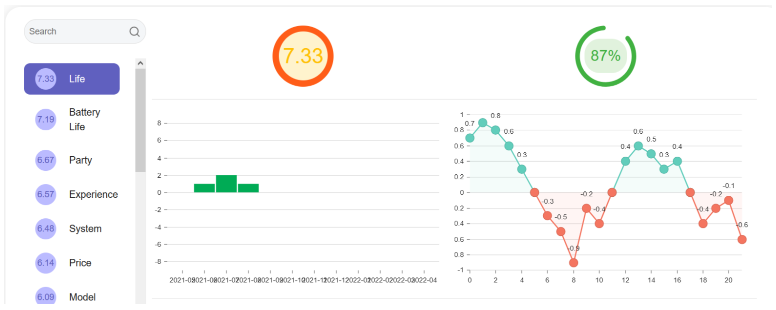
Problem
Times 14 42.86% 2.14

Figure A.8: Product detail view

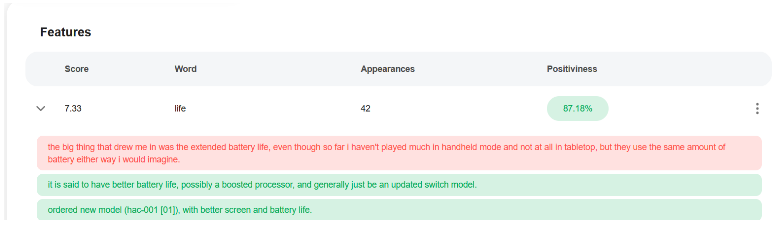
Product feature analysis

Every product has some features attached to it, those features have an score telling how good/bad are and each of them have sentences referring to in the reviews. To analyze those features two components where made:

- **Chart view** (figure A.9a): Contains charts of each feature were how many positive/negative references have been made in the reviews of that exact feature.
- **List view** (figure A.9b): Contains the sentences and reviews where each feature appears along with some basic information of the feature like appearances, score and positiveness.



(a) Chart view of features



Features

	Score	Word	Appearances	Positiviness
	7.33	life	42	87.18%

the big thing that drew me in was the extended battery life, even though so far i haven't played much in handheld mode and not at all in tabletop, but they use the same amount of battery either way i would imagine.

it is said to have better battery life, possibly a boosted processor, and generally just be an updated switch model.

ordered new model (hac-001 [01]), with better screen and battery life.

(b) List view of features

Figure A.9: Components for displaying analyzed data about product features

List of Acronyms

API	Application programming interface. 35
ASIN	Amazon Standard Identification Number. 41
DB	Database. 41
E2E	End-to-end. 61
JS	JavaScript. 16, 50
PMI	Pointwise mutual information. 6, 31, 33
POS	Part-of-speech. 26, 29, 30
SVM	Support vector machine. 24, 25
TF-IDF	Term frequency–inverse document frequency. 26, 27
WWW	World Wide Web. 3, 34, 35, 40

Glossary

Gloover The system created in this work that contains the text classifier and feature extractor along with an API Rest and the database access. vii, 34–36, 39–41, 70

Information retrieval the process of obtaining information system resources that are relevant to an information need from a collection of those resources.. 1

Lemmatizing To group together the inflected forms of (a word) for analysis as a single item. 29

part of speech Category of words (or, more generally, of lexical items) that have similar grammatical properties. Commonly listed English parts of speech are noun, verb, adjective, adverb, pronoun, preposition, conjunction, interjection, numeral, article, or determiner. 26

Text classification The task is to assign a document to one or more classes or categories. This may be done ”manually” (or ”intellectually”) or algorithmically.. 1

Text clustering The application of cluster analysis to textual documents. 1

transformer Black box that accept a matrix as input and return a matrix of the same shape as output with a specific transformation applied. 25, 33

web crawling A Web crawler, sometimes called a spider or spiderbot and often shortened to crawler, is an Internet bot that systematically browses the World Wide Web, typically used by search engines for the purpose of Web indexing. 35

YAML YAML is a human-readable data-serialization language. 35

Bibliography

[1] B. J. Jansen and S. Y. Rieh, “The seventeen theoretical constructs of information searching and information retrieval,” *Journal of the American Society for Information Science and Technology*, vol. 61, no. 8, pp. 1517–1534, 2010. [In line]. Avaliable in: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.21358

[2] L. Lessig, “The future of ideas: The fate of the commons in a connected world,” 01 2002.

[3] E. Haddi, X. Liu, and Y. Shi, “The role of text pre-processing in sentiment analysis,” *Procedia Computer Science*, vol. 17, p. 26–32, 12 2013.

[4] D. M. Asghar, A. Khan, S. Ahmad, and F. Kundi, “A review of feature extraction in sentiment analysis,” *Journal of Basic and Applied Research International*, vol. 4, pp. 181–186, 01 2014.

[5] X. Fang and J. Zhan, “Sentiment analysis using product review data,” *J Big Data*, vol. 2, 12 2015.

[6] M. Sadiku, A. Shadare, S. Musa, C. Akujuobi, and R. Perry, “Data visualization,” *International Journal of Engineering Research and Advanced Technology (IJERAT)*, vol. 12, pp. 2454–6135, 12 2016.

[7] T. Evgeniou and M. Pontil, “Support vector machines: Theory and applications,” vol. 2049, 01 2001, pp. 249–257.

[8] I. Rish *et al.*, “An empirical study of the naive bayes classifier,” in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[9] B. Pang, L. Lee, and S. Vaithyanathan, “Thumbs up? sentiment classification using machine learning techniques,” in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. Association for Computational Linguistics, Jul. 2002, pp. 79–86. [In line]. Avaliable in: https://aclanthology.org/W02-1011

[10] B. Pang and L. Lee, “A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts,” in *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*, Barcelona, Spain, Jul. 2004, pp. 271–278. [In line]. Avaliable in: https://aclanthology.org/P04-1035

[11] P. Turney, “Thumbs up or thumbs down? Semantic orientation applied to unsupervised classification of reviews,” *Computing Research Repository - CORR*, pp. 417–424, 12 2002.

[12] K. Dave, S. Lawrence, and D. Pennock, “Mining the peanut gallery: Opinion extraction and semantic classification of product reviews,” *Mining the Peanut Gallery: Opinion Extraction and Semantic Classification of Product Reviews*, vol. 775152, 10 2003.

[13] S. R. Das and M. Y. Chen, “Yahoo! for amazon: Sentiment extraction from small talk on the web,” *Management Science*, vol. 53, no. 9, pp. 1375–1388, 2007. [In line]. Avaliable in: https://pubsonline.informs.org/doi/abs/10.1287/mnsc.1070.0704

[14] S. Morinaga, K. Yamanishi, K. Tateishi, and T. Fukushima, “Mining product reputations on the web,” 07 2002.

[15] R. Tong, “An operational system for detecting and tracking opinions in on-line discussions,” in *Working Notes of the SIGIR Workshop on Operational Text Classification*, New Orleans, Louisianna, 2001, pp. 1–6.

[16] M. Hu and B. Liu, “Mining and summarizing customer reviews,” 08 2004, pp. 168–177.

[17] S.-M. Kim and E. Hovy, “Determining the sentiment of opinions,” 01 2004.

[18] C. Fellbaum, *WordNet*. John Wiley Sons, Ltd, 2012. [In line]. Avaliable in: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781405198431.wbeal1285

[19] A. Andreevskaia and S. Bergler, “Mining WordNet for a fuzzy sentiment: Sentiment tag extraction from WordNet glosses,” in *11th Conference of the European Chapter of the Association for Computational Linguistics*. Trento, Italy: Association for Computational Linguistics, Apr. 2006, pp. 209–216. [In line]. Avaliable in: https://aclanthology.org/E06-1027

[20] A. Esuli and F. Sebastiani, “Determining term subjectivity and term orientation for opinion mining,” 04 2006.

[21] J. Kamps, M. Marx, R. Mokken, and M. Rijke, “Using wordnet to measure semantic orientations of adjectives,” 04 2004.

[22] "Lexalytics," https://www.lexalytics.com/semantria/, accessed: 2020-08-31.

[23] "Monkeylearn," https://monkeylearn.com/, accessed: 2020-08-31.

[24] "Meaningcloud," https://www.meaningcloud.com/products/sentiment-analysis, accessed: 2020-08-31.

[25] J. Sutherland and J. Sutherland, *Scrum: The Art of Doing Twice the Work in Half the Time.* Random House Business, 1986.

[26] L. Rising and N. Janoff, "The scrum software development process for small teams," *Software, IEEE*, vol. 17, pp. 26 – 32, 08 2000.

[27] "2020 scrum guide," https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100, accessed: 2020-08-31.

[28] L. Cocco, K. Mannaro, G. Concas, and M. Marchesi, "Simulating kanban and scrum vs. waterfall with system dynamics," vol. 77, 05 2011, pp. 117–131.

[29] D. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business.* Blue Hole Press, 2010. [In line]. Avaliable in: https://books.google.es/books?id=RJ0VUkfUWZkC

[30] E. Brechner, *Agile Project Management with Kanban*, ser. Best practices. Microsoft Press, 2015. [In line]. Avaliable in: https://books.google.es/books?id=21DXoQEACAAJ

[31] "Git," https://git-scm.com/, accessed: 2020-08-31.

[32] "Semantic versioning," https://semver.org/, accessed: 2020-08-31.

[33] D. Marshburn, "Scrum retrospectives: Measuring and improving effectiveness," 2018.

[34] "Python," https://www.python.org/, accessed: 2020-08-31.

[35] "Flask," https://flask.palletsprojects.com/en/2.0.x/, accessed: 2020-08-31.

[36] Django, "Django project," https://www.djangoproject.com/, accessed: 2020-08-31.

[37] "Scikit-learn," https://scikit-learn.org/stable/, accessed: 2020-08-31.

[38] "Nltk," https://www.nltk.org/, accessed: 2020-08-31.

[39] "Spacy," https://spacy.io/, accessed: 2020-08-31.

[40] Zyte, "Scrapy," https://github.com/scrapy/scrapy, 2020.

[41] Typescript, "Typescript," https://www.typescriptlang.org/, accessed: 2020-08-31.

[42] Javascript, "Javascript," https://www.javascript.com/, accessed: 2020-08-31.

[43] "React," https://es.reactjs.org/, accessed: 2020-08-31.

[44] "Figma," https://www.figma.com/, accessed: 2020-08-31.

[45] S. Raschka, J. Patterson, and C. Nolet, "Machine learning in python: Main developments and technology trends in data science, machine learning, and artificial intelligence," *Information*, vol. 11, no. 4, 2020. [In line]. Avaliable in: https://www.mdpi.com/2078-2489/11/4/193

[46] W. Cavnar and J. Trenkle, "N-gram-based text categorization," *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, 05 2001.

[47] H. J. Escalante, T. Solorio, and M. Montes, "Local histograms of character n-grams for authorship attribution." 01 2011, pp. 288–298.

[48] Z. Yun-tao, G. Ling, and W. Yong-cheng, "An improved tf-idf approach for text classification," *Journal of Zhejiang University-SCIENCE A*, vol. 6, pp. 49–55, 2005.

[49] C. Potts. Sentiment symposium tutorial: Linguistic structure. [In line]. Avaliable in: http://sentiment.christopherpotts.net/lingstruc.html#negation

[50] L. Gatti, M. Guerini, and M. Turchi, "Sentiwords: Deriving a high precision and high coverage lexicon for sentiment analysis," *IEEE Transactions on Affective Computing*, vol. 7, no. 4, p. 409–421, Oct 2016. [In line]. Avaliable in: http://dx.doi.org/10.1109/TAFFC.2015.2476456

[51] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215. Citeseer, 1994, pp. 487–499.

[52] R. J. Bayardo, "Efficiently mining long patterns from databases," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '98. New York, NY, USA: Association for Computing Machinery, 1998, p. 85–93. [In line]. Avaliable in: https://doi.org/10.1145/276304.276313

[53] C. Pujari, Aiswarya, and N. P. Shetty, "Comparison of classification techniques for feature oriented sentiment analysis of product review data," in *Data Engineering and Intelligent Computing*, S. C. Satapathy, V. Bhateja, K. S. Raju, and B. Janakiramaiah, Eds. Singapore: Springer Singapore, 2018, pp. 149–158.

[54] P. Turney, “Mining the web for synonyms: Pmi-ir versus lsa on toefl,” 09 2001, pp. 491–502.

[55] Peter and Littman, “Measuring praise and criticism: Inference of semantic orientation from association,” *ACM Transactions on Information Systems*, vol. 21, pp. 315–, 10 2003.

[56] S. University. Amazon product reviews datasets. [In line]. Avaliable in: http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/

[57] jmcarpenter2, “Swifter,” https://github.com/jmcarpenter2/swifter, 2020.

[58] Scrapehero, “Selectorlib,” https://github.com/scrapehero/selectorlib, 2020.

[59] SelectorLib, “Using selectorlib with scrapy,” https://selectorlib.com/scrapy.html, accessed: 2020-08-31.

[60] xbound, “scrapy-api,” https://github.com/xbound/scrapy-api, 2020.

[61] “Apscheduler,” https://github.com/agronholm/apscheduler/tree/3.x, accessed: 2020-08-31.

[62] K. Fraczek and M. Plechawska-Wojcik, “Comparative analysis of relational and non-relational databases in the context of performance in web applications,” 04 2017, pp. 153–164.

[63] H. Phiri and D. Kunda, “A comparative study of nosql and relational database,” *Zambia ICT Journal*, vol. 1, pp. 1–4, 12 2017.

[64] “Acid vs. base,” https://neo4j.com/blog/acid-vs-base-consistency-models-explained/, accessed: 2020-08-31.

[65] “Mongodb,” https://www.mongodb.com, accessed: 2020-08-31.

[66] PyMongo, “Pymongo documentation,” https://pymongo.readthedocs.io/en/stable/, accessed: 2020-08-31.

[67] Google, “Material design,” https://material.io/design, accessed: 2020-08-31.

[68] “Feathericons,” https://feathericons.com/, accessed: 2020-08-31.

[69] G. M. Bierman, M. Abadi, and M. Torgersen, “Understanding typescript,” in *ECOOP*, 2014.

[70] “Material-ui,” https://mui.com/, accessed: 2020-08-31.

[71] “ant-design,” https://ant.design/, accessed: 2020-08-31.

[72] "Joblib," https://joblib.readthedocs.io/en/latest/, accessed: 2020-08-31.

[73] E. Miller, "How not to sort by average rating," https://www.evanmiller.org/how-not-to-sort-by-average-rating.html, accessed: 2020-08-31.

[74] A. Agresti and B. A. Coull, "Approximate is better than "exact" for interval estimation of binomial proportions," *The American Statistician*, vol. 52, no. 2, pp. 119–126, 1998. [In line]. Avaliable in: http://www.jstor.org/stable/2685469

[75] "Dockerization examples," https://docs.docker.com/samples/, accessed: 2020-08-31.

[76] "Docker deployment," https://docs.docker.com/compose/production/, accessed: 2020-08-31.

[77] Docker, "Docker compose," https://docs.docker.com/compose/, accessed: 2020-08-31.

[78] npm, "Npm," https://www.npmjs.com/, accessed: 2020-08-31.

[79] Docker, "Docker desktop," https://www.docker.com/products/docker-desktop, accessed: 2020-08-31.

[80] "Nvm," https://github.com/coreybutler/nvm-windows, accessed: 2020-08-31.