UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

DESIGN AND DEVELOPMENT OF A MACHINE LEARNING SYSTEM FOR OPINION AND NATURAL LANGUAGE ANALYSIS IN SOCIAL MEDIA. APPLICATION TO THE RIDE-HAILING AND RADICALIZATION DOMAINS.

ÁLVARO DE PABLO MARSAL
2021

TRABAJO DE FIN DE MÁSTER

Título:	Diseño y desarrollo de un sistema de aprendizaje automático para el análisis de opinión y lenguaje natural en redes sociales. Aplicación a los dominios de ride-hailing y radicalización.
Título (inglés):	Design and Development of a Machine Learning System for Opinion and Natural Language Analysis in Social Media. Application to the Ride-Hailing and Radicalization Domains.
Autor:	Álvaro de Pablo Marsal
Tutor:	Óscar Araque Iborra
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

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

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MÁSTER

Design and Development of a Machine Learning System for Opinion and Natural Language Analysis in Social Media. Application to the Ride-Hailing and Radicalization Domains.

2021

Resumen

El análisis del contenido de los posts escritos en redes sociales ha establecido una importante línea de investigación en los últimos años. El estudio de dichos textos, así como su relación entre ellos y la dependencia de los mismos con respecto a la plataforma en la que se escriben, permiten analizar el comportamiento de los usuarios y la opinión de los mismos con respecto a diferentes dominios.

La aplicación de técnicas y algoritmos de Inteligencia Artificial, concretamente de la rama del Procesamiento del Lenguaje Natural (PLN), ha logrado avanzar en este sentido, de tal manera que es posible desarrollar modelos que predicen la temática de los posts o la manera en que se habla de un determinado tema. Esto es fundamental para entender la opinión de los usuarios sobre un tema concreto, para conocer el grado de satisfacción de los clientes de un servicio o incluso para identificar posibles mensajes de odio o con un claro contenido extremista.

En este proyecto se ha desarrollado un sistema que analiza de manera automática y en tiempo real el contenido de los posts escritos en distintas redes sociales con el objetivo de analizar el lenguaje utilizado y la opinión de estos usuarios sobre diferentes temas, concretamente, relacionados con plataformas de movilidad (Ride-Hailing) o temáticas de carácter extremista y radical.

El sistema desarrollado está basado en diversas técnicas de PLN y de Aprendizaje Automático, como son el Modelado Temático, el Análisis de Sentimientos o la creación de modelos de clasificación, entre otros. Los resultados de este análisis se pueden observar gracias a la implementación de un módulo de visualización, donde se muestran los resultados de manera agregada y pueden filtrarse para realizar análisis personalizados.

Finalmente, este proyecto trata de estudiar la viabilidad de utilizar los modelos de clasificación desarrollados con otros tipos de datos con los que los modelos no han sido entrenados. Esto permitiría el desarrollo de aplicaciones cuyo uso pudiera extenderse a otros medios, facilitando la reutilización de modelos que han sido generados en base a conjuntos de dtaos limitados.

Palabras clave: Redes Sociales, Inteligencia Artifical, PLN, Aprendizaje Automático, Modelado Temático, Análisis de Sentimientos, Ride-Hailing, Radicalización, Aprendizaje por Transferencia

Abstract

The analysis of the content of posts written on social media has established an important line of research in recent years. The study of these texts, as well as their relationship with each other and their dependence on the platform on which they are written, allows to analyze the behavior of users and their opinions with respect to different domains.

The application of Artificial Intelligence techniques and algorithms, specifically from the branch of Natural Language Processing (NLP), has made progress in this regard, in such a way that it is possible to develop models that predict the subject matter of posts or the way in which a certain topic is discussed. This is essential to understand the opinion of users on a particular topic, to know the degree of satisfaction of the customers of a service or even to identify hate speeches or messages with a clear extremist content.

In this project, it has been developed a system that analyzes automatically and in real time the content of posts written in different social media to analyze the language used and the opinion of these users on different topics, specifically related to mobility platforms (Ride-Hailing) or topics of extremist and radical character.

The developed system is based on several NLP and Machine Learning techniques, such as Topic Modeling, Sentiment Analysis, or the creation of classification models, among others. The results of this analysis can be observed thanks to the implementation of a visualization module, where the results are shown in an aggregated way and can be filtered to perform a customized analysis.

Finally, this project tries to study the feasibility of using the developed classification models with other types of data with which the models have not been trained. This would allow the development of applications whose use could be extended to other media, facilitating the reuse of models that have been generated based on limited data sets.

Keywords: Social Media, Artificial Intelligence, NLP, Machine Learning, Topic Modeling, Sentiment Analysis, Ride-Hailing, Radicalization, Transfer Learning

Agradecimientos

Quiero agradecer a todas las personas que me han apoyado en el desarrollo del proyecto durante este año, especialmente a Óscar Araque, ya que sin su ayuda este proyecto no habría sido posible. Gracias a todos los miembros del GSI, tanto compañeros como profesores, con los que a lo largo de estos años he aprendido tanto.

Gracias a todos.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This project has been done as part of the Cátedra Cabify - UPM (Cabify - UPM Chair)[1] in the Intelligent Systems Group (GSI).

In this section, the objectives of the project and an introduction to the developed system will be related.

1.1 Context

This project arose from the need to develop a multidomain and multiplatform language and opinion analysis tool focused on social media. Social media platforms have been growing both in the number of platforms available and in the number of users who use them daily. These platforms have become a tremendously useful source of information, especially some of them in which users write about current issues, often giving their opinion or even forming debates.

For this reason, the develop of a system which allows to monitor these social media, allowing to analyze the posts of different users writing about different domains will be a useful tool. This tool could make it possible to analyze opinions, complaints, current issues on different topics, or different biases that may occur in certain population segments, such as extremist or violent attitudes.

[1] https://catedra-cabify.gsi.upm.es/

With the objective of creating a tool of these characteristics, it has been developed a system based on Artificial Intelligence techniques, such as Machine Learning and Natural Language Processing procedures. The developed system has the function of analyzing the language used in different social media to know the opinion and the way of speaking of the different users that use these media, so important nowadays. It has been developed to be able to analyze posts from different domains, for studying the behaviour of social media users when talking of a specific topic.

The system is able to obtain social media posts from different sources and to process them in different ways depending on the source. This processing has been designed to be as efficient as possible for the final performance, and to be as accurate as possible.

Then, the developed system applies different techniques to the posts to obtain information such as to know what users talk about or how they talk in the different domains of study. Finally, new unseen data will need to be processed and analyzed in the same way to maintain system functionality over time and to understand new needs or new user reactions.

The system also needs to provide a persistence layer and a visualization module to facilitate the analysis and observation of social media. Moreover, it should be useful in different use cases, such as the competitive analysis of a company, the security of a certain entity or state, and more.

Therefore, this project will focus on the development of a system that processes text from different sources, from different domains, and that is available in real time. In addition, the system will be developed based on virtualization technologies, so this tool will be portable to different machines.

1.2 Project goals

This project aims to cover different needs in a single system. The system must be able to analyze different sources, process them, and display results that can be easily analyzed by the observer.

Also, the system needs to be developed in a generic way to analyze data belonging to different domains unified in the same system. The mainly goal of the system will be the integration of every needed component in an easy way, allowing, among others, the processing of texts, the validation of the texts, the classification of texts, the extraction of information based on NLP techniques such as sentiment analysis, NER or POS, the topic extraction, data persistence and visualization of the analysis. Moreover, the system will need to be developed to work in a virtualized way.

In addition , this project will cover the different studies that have been done to show the feasibility of the project, especially regarding the extrapolation of the study to other sources, where it is necessary to include machine learning mechanisms, introducing and applying both traditional and more modern techniques, such as Deep Learning or automatic translation of texts with transformers.

In conclusion, the main goals of this project can be resumed in three parts:

- The development of the real-time system.
- The application of the system to different domains.
- The study of the feasibility of extending the trained models to other social media sources and types of data.

1.3 Structure of this document

The remaining of this document is structured as follows:

Chapter 2: On Background. This chapter will explain the context on which this project is based, with an introduction to the state of the art of the technologies and processes carried out in the project, as well as an explanation of the most important technologies and tools used in the development of the system.

Chapter 3: System Architecture. The general architecture of the developed system will be explained here. This section includes both the different processes of which the developed system is composed, as well as the procedure that has been carried out to develop the different components and functionalities of which the complete system is composed.

Chapter 4: Use Case - The Ride-Hailing Domain. This chapter will discuss the application of the developed system in the domain of the study of posts published in different social networks about Ride-Hailing companies, such as Uber, Lyft, or Cabify. The procedure will be explained, as well as the results that have been reached.

Chapter 5: Use Case - The Radicalization Domain. In this chapter, radical tweets, in particular those who are related to the extremism ideologies, pro-Islam and anti-Islam ideologies, will be analyzed with the developed system. The obtained results for this domain will be discussed here.

Chapter 6: Cross-Source Validation. Transfer Learning. This chapter will explain the application of the developed system to the study of other social media sources. This chapter builds on Chapter 4 and will attempt to analyze the feasibility of extrapolating the results obtained by analyzing one social network to other social networks.

Chapter 7: Conclusions. The results and conclusions of the analysis and development of the project will be discussed here.

Appendixes. Some figures and more detailed explanations will be explained here.

CHAPTER 2

Background

2.1 State of the Art

Artificial Intelligence and, more specifically, Natural Language Processing (NLP) techniques, have been applied to social media data on numerous occasions. The objectives behind the application of these types of techniques cover many studies, as varied as the study of the opinion of the population on public health issues based on the content of posts on Twitter [15], the analysis of different social media sources to develop suicide identification and prevention techniques [12], or the detection of different ways of speaking and expressing opinions, such as the detection of hate speech in these media [43], among many others applications.

The techniques used in NLP range from simple text transformations to more complex representations that have been developed in recent years with incredibly good results. Some of these techniques can be the Tf-idf representation, which is useful to determine word relevance in documents [37] or other representations based on Neural Networks which can infer interesting features from texts such as the similarity between words within a semantic context [24].

Moreover, beyond text representation, which is necessary for algorithms to understand and work with texts, other methods for extracting other types of features from texts have come to the fore in recent years. One of them can be Sentiment Analysis, which analyzes texts extracting from them the intent of the person who writes them, that is, if he/she wants to express an idea with negative or positive connotations. This kind of analysis has been used in many approaches, such as the identification of subjectivity [26], or to know the opinion of the population in certain situations, performing opinion mining techniques based on sentiment analysis to know opinions in social media [25], as we intend to do in this project.

On the other hand, for opinion mining, in addition to knowing the intent of the messages, it is important to know their subject matter. Therefore, several studies have applied Topic Modeling algorithms to social network posts for this purpose [19] [36]. As stated, topic modeling results can be joined with other opinion mining techniques, such as sentiment analysis, to understand and to analyze the way in which users write on social media on a specific topic, such as the prediction of the stock market [34] or to know how people talk about some important events, such as the global climate change [13], among others.

Other applications of these technologies applied to social media have to do with the approach of modern cities. These are increasingly looking for the transformation of cities into sustainable cities, with green areas and sustainable transport, among others, and these techniques can be used both to know the opinion of the population on these changes or to know the needs of the residents of these cities [44]. On the other hand, many of these changes have been introduced by certain companies, such as those involved in sustainable passenger transport or those that are suppliers of public furniture. Therefore, an important application of this type of technology is the analysis of the competitiveness between different companies to analyze the opinion of customers or users, to increase the quality of the service offered or to anticipate problems that may arise [33] [54].

Furthermore, NLP techniques, as well as different Machine Learning approaches, have been used to detect radical content in texts, specially in social media or newspapers content [14]. Sentiment analysis has also been used in the detection and classification of radical texts on social media [1]. These techniques can be used also to search for certain linguistic markers with which to identify this type of content.

It is important to note that every of these techniques are modeled with data collected from some sources and usually the generated models are applied to classify new unseen data from the same sources. However, some NLP applications could need to analyze new unseen data from other sources, in this case from other social media sources. In this sense, some studies have demonstrated that the application of these techniques can be used to predict data in other sources, both as applied to issues of radicalism or hate speech identification [39], as well as in other domains [58] [21].

For all these reasons, the different branches of Natural Language Processing have proven to be tremendously useful for the classification of different domains discussed in social media.

Thus, numerous studies have used these techniques and presented more than convincing results on the performance of the use of these technologies.

2.2 Enabling Technologies

This section introduces those technologies used and implemented in the project development process. These technologies range from standard Python libraries to other libraries for Machine Learning, Natural Language Processing, or Deep Learning, as well as other technologies such as virtualization tools and more.

2.2.1 APIs and Web Scraping Tools

This project has been focused on social media sources such as Reddit, Twitter, or the Google Play Store. With the aim of obtaining data from these sources, several APIs and libraries have been used.

2.2.1.1 Pushshift API

Pushshift[1] is a Big Data project which provides different statistics and analytics about Reddit data. In addition, it provides an API that serves Reddit objects in a real-time way. There exist other Reddit scraping tools, such as PRAW[2] (Python Reddit API Wrapper), but the Pushshift API is recommended when attempting to collect large amounts of data.

Each API request returns a set of Reddit posts (maximum 100 posts per request). The API request supports different parameters, such as the name of the subreddit, the date range in which we want to obtain the posts, the type of posts (they can be submissions or comments), and more. Therefore, it is possible to specify other parameters, such as the name of the author, and thus obtain aggregated data based on the specified parameters.

Moreover, each post data is composed by several interesting parameters, such as the text of the post, the title, and other metadata fields. An explanation of the API response content can be seen on the Pushshift API documentation[3].

2.2.1.2 Twint

Twint[4] (Twitter Intelligent Tool) is a Python library that allows to obtain data from Twitter. It is a great alternative to the official Twitter API, as it only allows to collect around 3200 tweets per day.

[1]https://search.pushshift.io/

[2]https://praw.readthedocs.io/

[3]https://reddit-api.readthedocs.io/

[4]https://github.com/twintproject/twint

Twint works by searching for specific words in the tweets. Therefore, tweets can be collected based on specific hashtags, account names, or other keywords.

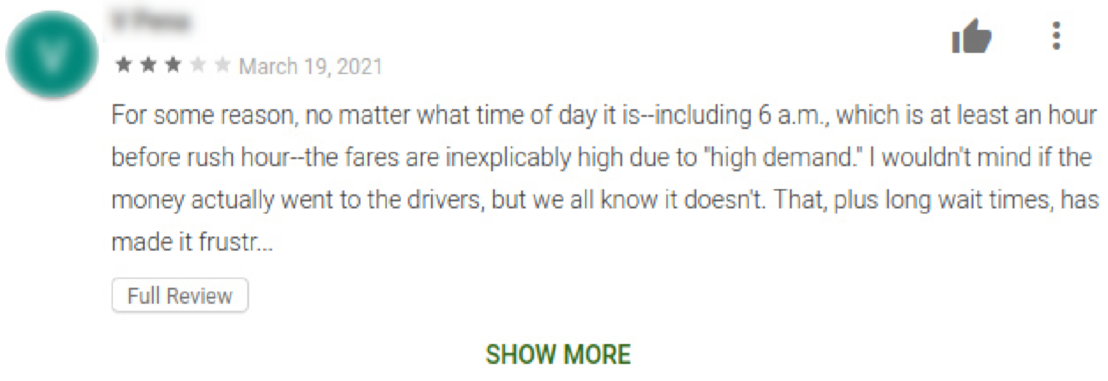
Tweets are retrieved in a JSON format and the information that can be obtained from each tweet is very extensive, including the user name, the tweet identifier, the number of retweets, the content of the post and more.

2.2.1.3 Google Play Scraper

To obtain data from Google Play Store reviews, different methodologies have been tested.

Firstly, a scraper tool was developed trying to obtain the data directly from the HTML code of the web page of each app. In this approach, different libraries were tested, such as BeautifulSoup[5], which allows to dissecting an HTML or XML content and extract in an easy way information of the page, Scrapy[6], which allows to obtain the information directly from the website based on spiders (classes that make the requests and obtain the information), and Selenium[7], a tool that allows to automate the interaction with a website and to collect the DOM tree data resulting from that interaction.

To obtain the required information about the reviews on the Google Play website, it is mandatory to run JavaScript code to perform actions such as scrolling the page or clicking a few buttons. This website displays a button every time it is scrolled four times. Additionally, when texts are long, it is necessary to click on the *Full Review* button in the English version or *Opinión completa* in the Spanish version to visualize the whole text. Figure 2.1 and Figure 2.2 show the visual interface of Google Play, where these buttons can be seen.



★★★★★ March 19, 2021

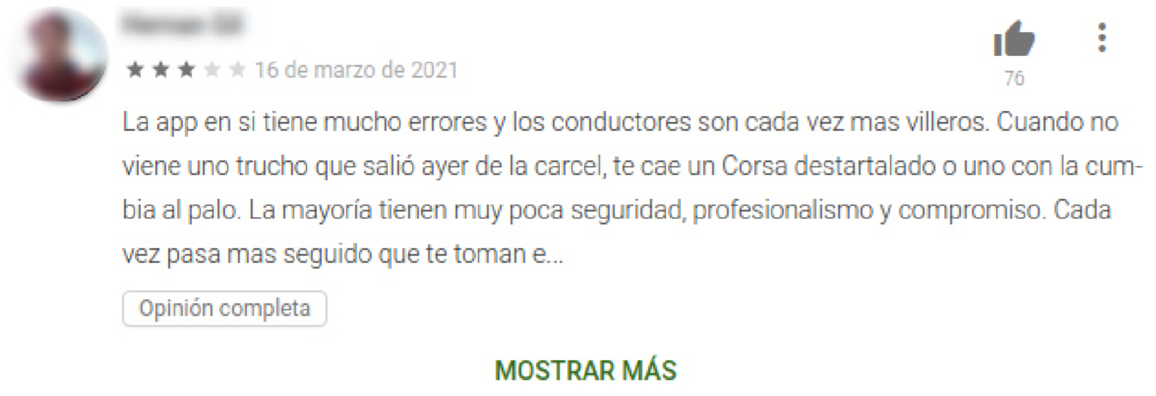
For some reason, no matter what time of day it is--including 6 a.m., which is at least an hour before rush hour--the fares are inexplicably high due to "high demand." I wouldn't mind if the money actually went to the drivers, but we all know it doesn't. That, plus long wait times, has made it frustr...

Figure 2.1: English Google Play Store Buttons

[5]https://www.crummy.com/software/BeautifulSoup/

[6]https://scrapy.org/

[7]https://www.selenium.dev/


★★★★★ 16 de marzo de 2021

76

La app en si tiene mucho errores y los conductores son cada vez mas villeros. Cuando no viene uno trucho que salió ayer de la carcel, te cae un Corsa destartalado o uno con la cumbia al palo. La mayoría tienen muy poca seguridad, profesionalismo y compromiso. Cada vez pasa mas seguido que te toman e...

Figure 2.2: Spanish Google Play Store Buttons

Because of this, a scraper tool based on Selenium was developed, discarding other options since it was necessary to run JavaScript code and BeautifulSoup and Scrapy do not natively support it.

By having to scroll several thousand times to retrieve all possible reviews of each of the analyzed applications, the RAM memory fills up. Due to the memory limitation, although the tool worked correctly, it was decided to look for another approach to collect reviews from the Google Play Store.

For this reason, it was decided to use the google-play-scraper library[8]. This library collects almost every Google Play Store review or some reviews if two dates are set. It allows to collect reviews sorted by relevance or by date (although the review sorting functions do not work well), to set the language of the app version, to filter reviews by their rating, and much more.

Reviews are saved in a JSON format with the information of each review: user name, date, the content of the review, possible responses to the content of the review, score of the review, relevance of the review, and more.

2.2.2 Natural Language Processing Tools and Processes

The analysis of the content of posts, tweets, or reviews are the main information source to understand how people communicate in social media, as well as what opinions they have on certain topics, what they talk about, etc.

Because of this, several processes, such as sentiment analysis, stemming, lemmatizing, and others have been applied to the texts to obtain information. All of these processes are included in Natural Language Processing technologies.

[8]https://github.com/JoMingyu/google-play-scraper

2.2.2.1 Stanford CoreNLP

The Stanford CoreNLP[9] is a tool written in Java that provides several annotators based on Natural Language Processing techniques. It is available for texts in Arabic, Chinese, English, French, German and Spanish. The annotators are just processes that are applied to the texts and return an output. Processes such as Sentiment Analysis, Named Entity Recognition (NER), Part of Speech (POS) Tagging, tokenizing by words or sentences, dependency and constituency parses and more can be applied to the texts. Also, one of the most powerful and useful capabilities offered by Stanford CoreNLP is that it provides a Python module which makes it possible to call the tool by using code written in Python.

In this project, only Sentiment Analysis, NER and POS have been used. However, other annotators are essential to get the output we want. The annotators used when processing texts are Tokenizer, Sentence Split (ssplit), POS, Parser, Lemmatizer, NER and Sentiment Analysis. Stanford CoreNLP works with a pipeline of annotators, in which the input is the raw text and the output is the text with its annotations. A representation of this can be seen in Figure 2.3.

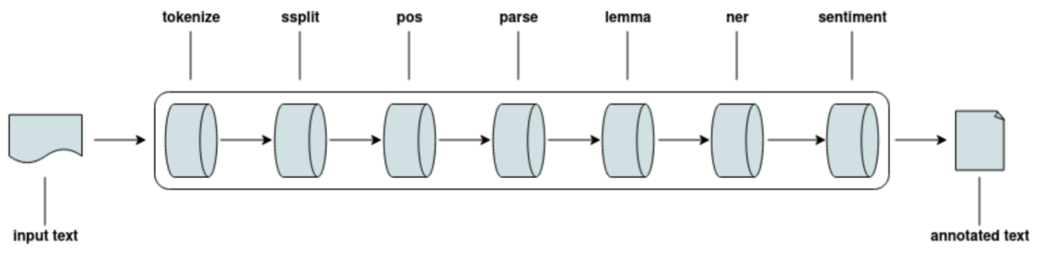
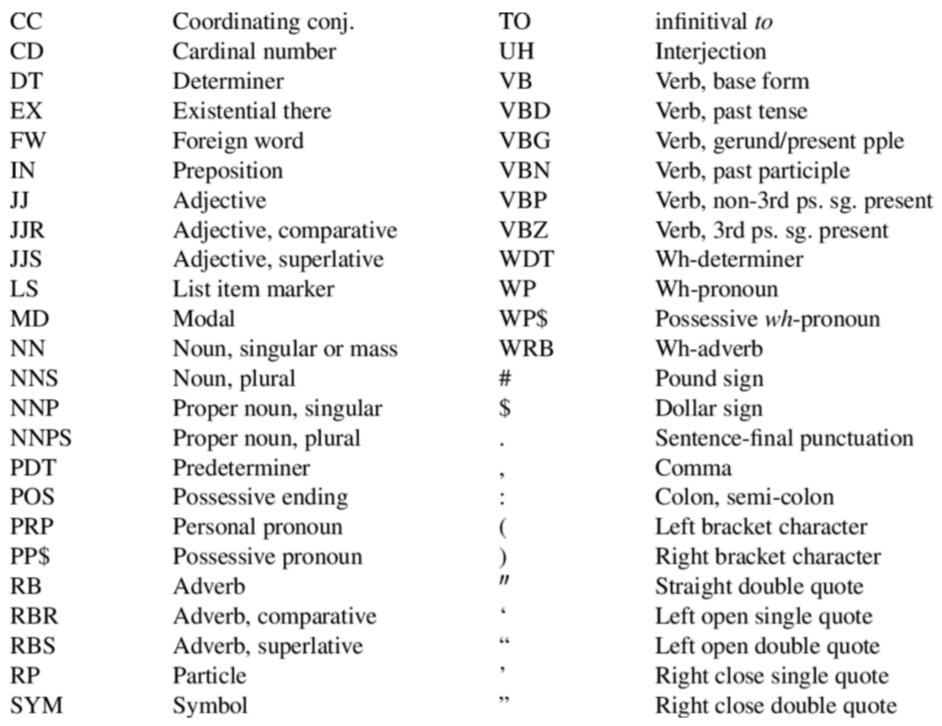


Figure 2.3: Stanford CoreNLP pipeline used in this project

- **Tokenizer Annotator:** It splits texts into words. This splitting will be different depending on the language of texts. Some words are split into different parts ("isn't" to ["is", "n't"] or "you've" to ["you", "'ve"]) to make it easier to process the different tokens with other annotators.
- **Sentence Split (ssplit) Annotator:** It splits texts into sentences.
- **Part of Speech (POS) Annotator:** Tokens are labeled with its corresponding POS tag. The Stanford CoreNLP uses a Java implementation of the log-linear part-of-speech tagger described in [50]. Also, as the project is based on social media analysis, a caseless model[10] (trained on Twitter data) has been used. In English texts, POS tags are belonging to the Penn Treebank tagset [28]. Figure 2.4 shows these tags.

[9]https://stanfordnlp.github.io/CoreNLP/
[10]https://stanfordnlp.github.io/CoreNLP/caseless.html



CC	Coordinating conj.	TO	infinitival *to*
CD	Cardinal number	UH	Interjection
DT	Determiner	VB	Verb, base form
EX	Existential there	VBD	Verb, past tense
FW	Foreign word	VBG	Verb, gerund/present pple
IN	Preposition	VBN	Verb, past participle
JJ	Adjective	VBP	Verb, non-3rd ps. sg. present
JJR	Adjective, comparative	VBZ	Verb, 3rd ps. sg. present
JJS	Adjective, superlative	WDT	Wh-determiner
LS	List item marker	WP	Wh-pronoun
MD	Modal	WP$	Possessive *wh*-pronoun
NN	Noun, singular or mass	WRB	Wh-adverb
NNS	Noun, plural	#	Pound sign
NNP	Proper noun, singular	$	Dollar sign
NNPS	Proper noun, plural	.	Sentence-final punctuation
PDT	Predeterminer	,	Comma
POS	Possessive ending	:	Colon, semi-colon
PRP	Personal pronoun	(	Left bracket character
PP$	Possessive pronoun	)	Right bracket character
RB	Adverb	"	Straight double quote
RBR	Adverb, comparative	‘	Left open single quote
RBS	Adverb, superlative	“	Left open double quote
RP	Particle	’	Right close single quote
SYM	Symbol	”	Right close double quote

Figure 2.4: The Penn Treebank POS tagset [48]

- **Parser Annotator:** Performs a syntactic analysis of sentences from input texts. It is an essential function to analyze the dependencies of the different parts that make up the texts to be able to correctly analyze them with other annotators.

  Stanford CoreNLP provides various parsers that, depending on the situation, will work better or worse. As in the POS annotator case, a caseless model has been used as the parser annotator.

- **Lemmatizer Annotator:** Performs the process of lemmatization. This process transforms words into their lemmas, as it can be seen in Table 2.1.

Original token	Lemmatized token
eats	eat
gaming	game
terrified	terrify
went	go

Table 2.1: Lemmatization Process Examples

- **Named Entity Recognition (NER) Annotator:** This is the process in which those words that represent an entity are labeled with an entity-tag. These entities could be useful to know if a text is talking about an important person, a city, or about time-related terms. As in both POS and Parse Annotators, a caseless model has been used for the NER Annotator.

The Stanford NER Annotator provides models with several tags, which are: *Organization*, *Date*, *Money*, *Criminal charge*, *Title*, *Duration*, *Number*, *Person*, *Miscelaneous*, *Religion*, *Ordinal numbers*, *Location*, *Set*, *City*, *Time*, *Percent*, *Cause of death*, *State or Province*, *Nationality*, *Country*, *Ideology*, *E-mail*, *URL* and *Handle*.

- **Sentiment Analysis Annotator:** This annotator provides a sentiment analysis tool for labeling the sentences of each of the texts with sentiment. The model used in the Stanford CoreNLP for sentiment analysis is based on [45].

  The relationship between the sentiment value that the model provides with each sentiment name is shown in Table 2.2.

Sentiment Value	**Sentiment Label**
0	Very Negative
1	Negative
2	Neutral
3	Positive
4	Very Positive

Table 2.2: Sentiment Analysis Values

2.2.2.2 NLTK

The NLTK (Natural Language Toolkit) Python library [27] is one of the most popular libraries in the Natural Language Processing domain. It provides a lot of useful functions and packages for almost every NLP processes.

The most important functions and facilities that this library provides and that have been used during the development of the project are:

- **Tokenizers:** NLTK provides several tokenizers, either for generic use or for their use in specific situations. They can be mainly used to split text into words or sentences. In this project they have been used:

  – **nltk.tokenize.word_tokenize:** Used in generic texts and in almost every situation (mainly used in Reddit and Google Play texts).

  – **nltk.tokenize.TweetTokenizer:** Used with posts from Twitter.

  - **nltk.tokenize.RegexpTokenizer:** Based on regular expressions, it has been used in some specific situations.

- **NTLK Corpora:** NLTK provides a lot of pretrained models and corpora[11]. They can be downloaded by typing *nltk.download("name_of_the_corpus").* The corpus used in this project are:
  - **StopWords Corpus:** It is available for most languages, including English and Spanish. It is a set of tokens consisting of words such as "is", "you", "me", or "of", commonly called "*stopwords*". It makes easier their elimination from the texts in case it is necessary in any of the processes.
  - **WordNet Corpus:** A lexical database to identify synonyms and similar words to other words. It has other applications, such as help in the lemmatization process.
  - **Punk Tokenizer Models:** Pre-trained models created to identify and tokenize texts into sentences.

- **nltk.stem package:** It provides models for stemming and lemmatizing words.
  - **Lemmatizer:** As explained before, lemmatization is the process of transforming words into their lemmas. The NLTK package used in this project for this purpose was the WordNetLemmatizer, which is based on a WordNet model.
  - **Stemmer:** Stemming is the process of removing morphological affixes from words, leaving only the word stem. In Table 2.3 is shown an example of this process.

Original token	**Stemmed token**
eats	eat
gaming	game
terrified	terrifi
went	went

Table 2.3: Stemming Process Examples

  The NLTK package used in this project for stemming is the SnowballStemmer.

- **FreqDist:** Implements a function to count the occurrences of tokens in a sample of texts.

[11]http://www.nltk.org/nltk_data/

- **nltk.textcat:** It is a module for language identification using the TextCat [10] algorithm, which is based on the Zipf's Law [52] and the n-gram occurrences in texts.

2.2.2.3 WordFreq

WordFreq is a Python library "for looking up the frequencies of words in many languages, based on many sources of data" [46]. It has different models in almost every implemented language, usually being a small one to save memory and a larger one for more precise processes.

It implements a version called zipf_frequency, which is based on the Zipf's Law [52] and gives the probability of occurrence of a word in a specific language in a logarithmic scale

2.2.2.4 CLD3

CLD3[12] (Compact Language Detector v3) is a model generated by a neural network for language identification created by Google. The network is composed by three layers and the language prediction is based on the n-grams of words.

2.2.2.5 Langdetect

Langdetect[13] Python library is another language identifier library. It supports 55 different languages and it can directly detect the language of a text or give the probability of belonging to several different languages.

2.2.3 Machine Learning and Deep Learning

The developed system has some processes in which machine learning techniques are essential. Python has some of the most used Artificial Intelligence libraries and in this project some of them have been used.

2.2.3.1 Gensim

The Gensim library [38] is a Python library for Natural Language Processing that implements unsupervised Machine Learning algorithms such as Topic Modeling or Word Embeddings models. It can handle very large collections of text data and each algorithm can be parallelized.

Among the most important algorithms implemented in this library are LDA, Word2Vec, FastText, LSI, and other NLP techniques such as n-gram generation, lemmatization, and other basic NLP processes.

[12]https://github.com/google/cld3
[13]https://github.com/Mimino666/langdetect

2.2.3.2 Scikit-Learn

The Scikit-Learn library (also known as *sklearn*) [35] is one of the most important Machine Learning libraries in Python. It implements virtually all existing Machine Learning algorithms, using a very easy-to-use API.

It includes both regression algorithms, classification, multiclass classification, unsupervised learning, neural networks, clustering algorithms, and more. In addition, it implements hyperparameter optimization methods as well as data modeling techniques and other functions of interest. It is one of the most complete Machine Learning available libraries.

2.2.3.3 XGBoost

XGBoost[14] is a Machine Learning library which implements the gradient boosting algorithm. It implements a common Machine Learning framework in several languages, such as Python, C++, Java or R, among others. The Python version uses the same API as Scikit-Learn, so is an easy-to-use library in this language.

2.2.3.4 Hugging Face

In this project, translation is an important task which needs to be done with powerful tools that allow for the most efficient and correct translation possible. There exist several tools which allow to translate texts, such as Google Translator API[15] or Deepl API[16]. However, due to the use restrictions of these APIs and the needs of the project, a transformer provided by the Hugging Face library was used.

The Hugging Face library [57] is a Natural Language Processing library which includes, among others, the Transformers library. This library implements some transformers such as BERT or translator transformers.

A transformer [53] is a neural network architecture which works as an encoder-decoder system, using a mechanism known as *attention*, which sets different weights to different parts of the input data. They are mainly used in the NLP field and they are becoming a new paradigm in these Machine Learning tasks.

In this project, the Marian Machine Translator [22] tool was implemented. This translator is included in the Hugging Face library as a transformer, following the described below architecture.

[14]https://xgboost.readthedocs.io/
[15]https://cloud.google.com/translate/
[16]https://www.deepl.com/es/docs-api/

2.2.4 Virtualization and Orchestration Tools

With the aim of developing the complete system with all its components, some of them have been virtualized and connected between them. Therefore, powerful tools such as Docker and Luigi have been used, the latter being necessary to make sense of the system in the form of a consistent and persistent process pipeline.

2.2.4.1 Docker and Docker-Compose

Docker [29] is a virtualization tool based on containers. Each container is an isolated virtual system that runs its own code over its own Operative System or platform. It is one of the most widely used virtualization tools, since each container is capable of abstracting itself from the machine that hosts it, unlike other virtualization tools based on virtual machines, which are highly dependent on the host.

Therefore, several containers have been instantiated, each containing a system process. Furthermore, docker-compose has been used to interconnect each of the containers that contain the virtualized systems.

2.2.4.2 Luigi

Luigi[17] is a Python package created by Spotify engineers that works as an orchestrator for building complex pipelines or batch jobs. It can easily create pipelines of processes in which the output of a process becomes the input of the next process. It is based on tasks, being each task (each process) a unit of work.

It also allows having a global vision for error control of each of the parts that make up the execution of the pipeline.

2.2.5 Data Storage Tools

One of the main targets of the project is the data persistence, needed to be able to analyze the data in an aggregated way and to have as much data as possible, as well as to be able to continue collecting data since the system is launched.

In addition, data must be stored in a format that is understandable, accessible, and available when needed.

For all this, storage technologies such as ElasticSearch have been used, as well as Linked Data techniques for data modeling.

[17] https://github.com/spotify/luigi

2.2.5.1 ElasticSearch

ElasticSearch[18] is a free and open data analytics engine based on the Java Apache Lucene library. It supports several data types, such as textual data, data based on documents (JSON), and both structured and unstructured data. It stands out for its ease of use through simple REST APIs with which complex queries can be performed, with data aggregation and many other facilities. It is developed in Java and it has numerous clients developed in different languages, including Python, Ruby, and Java.

One of the main characteristic of ElasticSearch is that it only supports the JSON format as a response, so formats like CSV or XML are not supported. Even so, the JSON format is widely supported by numerous programming languages, so it makes ElasticSearch the perfect tool to use, for example, in Big Data analysis.

ElasticSearch JSON data is sorted by indexes, where each of these indexes will be where the result of a kind of analysis or another is stored. Each analysis will be stored in documents inside its corresponding index.

2.2.5.2 Semantic Technologies and Ontologies

Semantic technologies can be seen as technical approaches that facilitate or make use of the interpretation of meaning by machines [18]. That is, semantic technologies make it possible to express data sets that make sense among themselves and, above all, among other data sets. This is useful for creating consensuses between data from different sources but representing the same thing. Data modelled with semantic technologies are commonly known as Linked Data.

To talk about Linked Data, it is necessary to define in some way the relationship between data, such as the classes they belong to, possible properties they have, subclasses, etc. In the Artificial Intelligence and Semantic Web domains, the abstraction that models and defines these relationships is called Ontology. These ontologies are modelled by languages such as OWL (Web Ontology Language) [20].

There are different specifications and technologies that model different areas of knowledge. Some of the most common, which have been used in this project, are defined below.

- **RDF:** The Resource Description Framework (RDF) [23] is a set of specifications for the World Wide Web Consortium (W3C). It is based on the concept of *triples*, which creates a relation *subject* - *predicate* - *object* between web resources, allowing searches for resources based on their relationships to other web resources.

[18]https://www.elastic.co/

- **DC:** The Dublin Core (DC) [55] is a set of 15 properties for describing web resources, based on RDF, which includes the *title* of the resource, the *date*, the *creator* or the *language*, among others.
- **SIOC:** The Semantically-Interlinked Online Communities (SIOC) [9] is a semantic-web technology which defines methods for the interconnection between online blogs or social media web pages. These functionalities aim to ensure that posts from different social media, for example, are linked to other posts from other social media, thus providing a better understanding of how people are talking in different networks and what they are talking about in these networks.
- **MARL:** An Ontology for Opinion Mining (MARL) [56] is an ontology for opinion mining, in particular sentiment analysis. It has been developed by the Intelligent Systems Group. It includes classes and properties to measure the polarity of a text, its source, or the aggregated opinion value, among others.

2.2.6 Visualization

With the aim of showing data in a comfortable and simple way, a visualization system based on web technologies has been developed.

2.2.6.1 HTML and CSS

As the visualization system is based on web technologies, HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are the main languages used in the visualization system. One of the main HTML frameworks is bootstrap, which allows to structure the web page in columns and rows, making easy development and allowing a comfortable and simple visualization.

2.2.6.2 JavaScript Frameworks: Polymer and D3

JavaScript is one of the most used programming languages. It is mainly used in the development of dynamic web pages, executing the JavaScript code on the client side.

As the Dashboard is based on the Sefarad project (which will be discussed below), the graphics displayed on the web are made using both the Polymer and D3 libraries, both written in JavaScript.

- **Polymer:** Polymer[19] is an open-source JavaScript library created by Google researchers and developers. It is used to create dynamic web pages based on web components, which are cells that display graphs and panels in an interactive way.

[19]https://github.com/Polymer

Polymer is used in several interactive web pages, such as Google Earth, Google Music, Youtube or Youtube Gaming and by companies like McDonald's or Coca-Cola[20].

- **D3 (Data-Driven Documents):** D3[21] is a JavaScript library that tries to make easier the creation of graphs and tables from data. It uses other web technologies such as HTML5, CSS, or SVG.

  It allows to set the properties of graphs in a comfortable way, allowing to set colors, sizes, interactions, and more.

2.2.6.3 Sefarad

Sefarad[22] is an environment developed by the Intelligent Systems Group (GSI in Spanish) to analyze and visualize data. It is based on web components, each of them shows a different diagram in which to view and analyze data. These web components are developed with the Polymer library, and some specific functionalities are based on D3.

Sefarad provides an ElasticSearch-based persistence layer and this functionality facilitates data aggregation. This is useful for displaying data in a way that makes it easier to see what they mean. In addition, it is possible to filter the data by showing only the data you want to see at that moment, facilitating analysis and accessibility.

Moreover, Sefarad provides a SPARQL query engine to perform Linked Data based searches to sites such as DBPedia or to the data stored in ElasticSearch itself. The architecture of Sefarad can be seen in Figure 2.5

2.2.7 Standard Python Libraries

To develop the whole project, some other libraries have been used.

2.2.7.1 Pandas

Pandas[24] is one of the most used Python libraries in the Data Science field. It has many functions for data manipulation, based on the DataFrame concept, which is nothing more than an organized table of data (rows) and features (columns). It allows to find mistakes in data in a comfortable and easy way, as well as find duplicate data and more. It is a fundamental tool in the area of Python data science.

[20]https://github.com/Polymer/polymer/wiki/Who's-using-Polymer%3F
[21]https://d3js.org/
[22]https://github.com/gsi-upm/sefarad-3.0
[23]http://gsi.upm.es:9080/software/projects/sefarad/
[24]https://pandas.pydata.org/

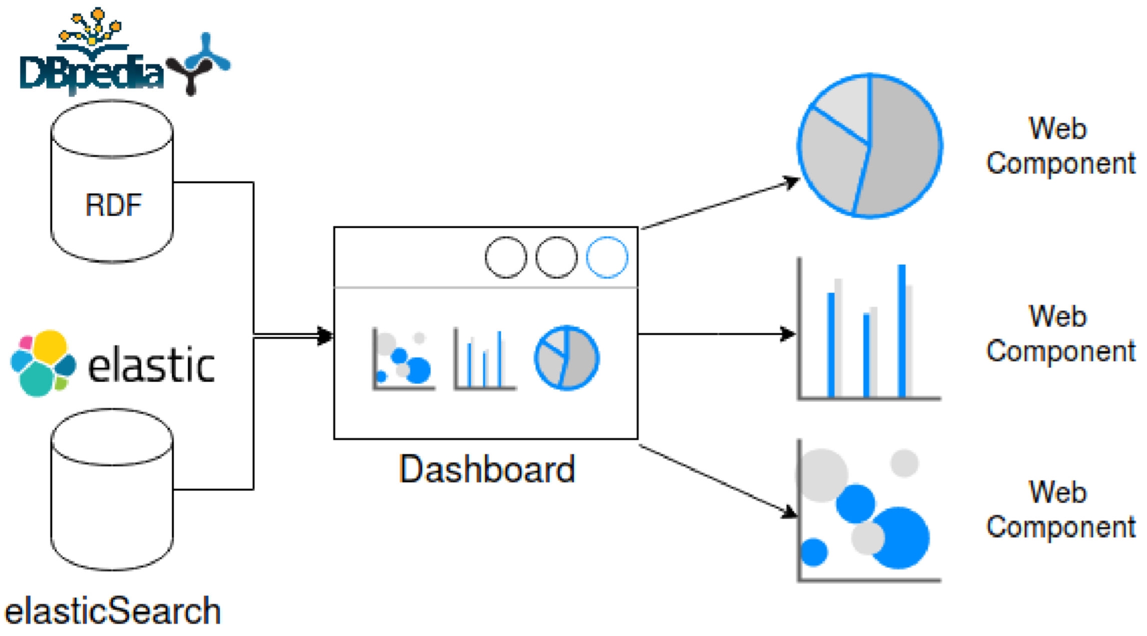


Figure 2.5: Sefarad Architecture[23]

2.2.7.2 Numpy

Numpy[25] is also a fundamental Python tool. It is an open source library for working with complex data such as n-dimensional arrays and other numerical computing tools, such as complex operations, Fourier Series, linear algebra operations, random sequence generation, and more.

2.2.7.3 SciPy

SciPy[26] is an open source Python library for mathematics, science, and engineering. It works with packages such as Numpy, Pandas or SymPy, all of them in the same package, making it an essential tool.

2.2.7.4 Re

The Re[27] Python library provides regular expression matching operations. It is useful to find patterns in texts, as well as to change substrings or to modify text extracts if they match what is expressed in the regular expression.

[25]https://numpy.org/
[26]https://www.scipy.org/
[27]https://docs.python.org/3/library/re.html

2.2.7.5 Matplotlib and Seaborn

Matplotlib[28] and Seaborn[29] libraries allow to graph and plot figures. They are useful to see the data and to analyze it.

2.2.7.6 Pickle and Joblib

Pickle[30] and Joblib[31] libraries are useful to serialize objects and, therefore, be able to store them. This is very useful to save objects such as Machine Learning models or different objects which can be used later or integrated into a defined architecture.

2.2.8 Other Python Libraries

2.2.8.1 GSITK

GSITK[32] is a library on top of scikit-learn that eases the development process of NLP machine learning. It has been developed by the Intelligent Systems Group. Among others, it provides some useful functionalities, such as the managing of datasets or features to facilitate the creation of machine learning pipelines. In addition, it provides some packages such as SIMON [5], which is a feature extractor based on Word Embeddings and a specific lexicon and which has been used in the development of the project.

2.2.8.2 Geopy

GeoPy[33] is another Python library used for geolocation and geocoder tasks. In this project, it has been used to get the coordinates of some geolocations such as cities, countries, or places around the world.

2.3 Machine Learning Algorithms

2.3.1 Topic Modeling

Topic models are unsupervised machine learning algorithms that try to make clusters (topics) with data, based on the words appearances within documents and the similarity between these documents [2]. For this reason, they are useful to extract information in large corpora and to classify texts.

[28]https://matplotlib.org/
[29]https://seaborn.pydata.org/
[30]https://docs.python.org/3/library/pickle.html
[31]https://joblib.readthedocs.io/
[32]https://github.com/gsi-upm/gsitk
[33]https://github.com/geopy/geopy

There are various topic modeling algorithms such as Latent Semantic Analysis (LSA), Correlated Topic Model (CTM), or Latent Dirichlet Allocation (LDA). In this project, the LDA algorithm has been used and the following is an explanation of how this algorithm works.

2.3.1.1 The LDA Algorithm

The Latent Dirichlet Allocation (LDA) algorithm [7] is a generative probabilistic model mainly used on text data. This algorithm is based on the distribution of words within documents, viewing these documents as random mixtures of words over latent topics, being each of these topics a set of weighted tokens. From here on, the used terminology corresponds to that used in [7].

LDA algorithm defines the next terms:

- A *word*, w_n, which is the basic unit of data, represented as a vector with all positions equal to zero except the position where the word is stored in a dictionary composed of all the distinct words in the corpus, which is represented by 1 in this vector.
- A *document*, which is a whole text, composed by a sequence of N words denoted by $\boldsymbol{w} = (w_1, w_2, ..., w_N)$.
- A corpus, which is a collection of M documents denoted by $D = \{\boldsymbol{w_1}, \boldsymbol{w_2}, ..., \boldsymbol{w_M}\}$

Given α and β, which are corpus-level parameters, a set of N topics $\boldsymbol{z}$ and a set of N words $\boldsymbol{w}$, the joint distribution of a topic mixture θ, which is a k-dimensional Dirichlet random variable is given by

$$p(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = p(\theta|\alpha) \prod_{n=1}^{N} p(z_n|\theta) p(w_n|z_n, \beta), \tag{2.1}$$

where $p(z_n|\theta)$ is θ_i for the unique i such that $z_n^i = 1$, $p(w_n|z_n, \beta)$ is a multinomial probability conditioned on the topic z_n for a word w_n, and $p(\theta|\alpha)$ is the probability density on the $(k-1)$-simplex of θ, which lies in the $(k-1)$-simplex if $\theta_i \geq 0, \sum_{i=1}^{k} \theta_i = 1$. This probability density of θ, $p(\theta|\alpha)$, is

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \theta_1^{\alpha_1 - 1} ... \theta_k^{\alpha_k - 1}, \tag{2.2}$$

where α is a k-vector with $\alpha_i > 0$, and $\Gamma(x)$ is the Gamma function. Finally, integrating over θ, the marginal distribution of a document can be calculated as follows:

$$p(\mathbf{w}|\alpha, \beta) = \int p(\theta|\alpha) \left(\prod_{n=1}^{N} \sum_{z_n} p(z_n|\theta) p(w_n|z_n, \beta) \right) d\theta. \tag{2.3}$$

A graphical representation of a LDA model can be seen in the next figure.

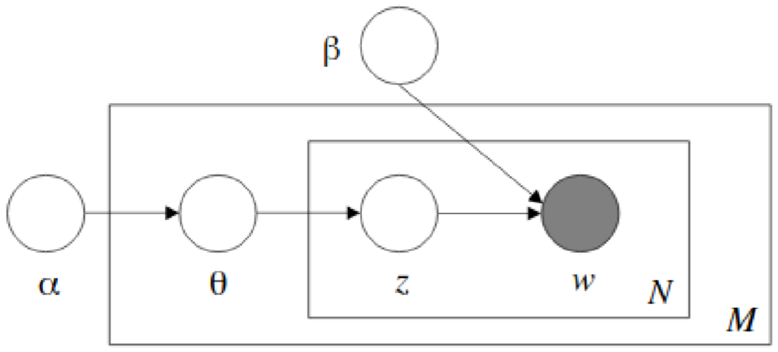


Figure 2.6: LDA Graphical Model [7]

As it can be appreciated, LDA is a hierarchical three-layer model. In the above figure, M represents documents and N is the number of words within a document. Also, α (per-document parameter) and β (per-topic parameter) are corpus-level parameters, θ_d is the topic distribution for document d and z_{dn} and w_{dn} are word-level variables, which z_{dn} means the topic for the n-th word in document d and w_{dn} means that n-th word of the document d.

After the generation of a LDA model, it can be used to predict topics for any text. It will generate a k-dimensional vector with its elements $\in [0, 1]$, which indicate the weight of each topic in the text and, therefore, it is possible to get an idea of what the text is about.

2.3.1.2 Measures of Performance

As LDA is highly dependent on the choice of hyperparameters, especially the parameter k (the number of topics), it is necessary to measure the performance of the generated model. The main available metrics are the Coherence Score and the Perplexity. Both are implemented in the Gensim library.

- **Coherence Score:** The Coherence Score [40] is the main metric used to measure the performance of the model. This metric defines a way in which the coherence between the topics is measured. This is an important task in topic models because, at least in the LDA algorithm, the number of the required topics must be set before training. A lower value of k results in broad topics, and an extremely higher value results in indescribable topics [47].

  There are several versions of this metric, but in this project the C_V version has been used, which is based on four parts [47]:

- **The segmentation of data into words** and the pairing of each of the top-N words belonging to a topic with every top-N word of another topic. The authors refer to W as the set of the top-N words belonging to a topic, to S_i as the segmented pair of each word $W' \in W$ paired with all other words $W^* \in W$ and to S as the set of all pairs.
- **The computation of the probability** of single words $p(w_i)$ and the joint probability of two words $p(w_i, w_j)$, calculated as the number of documents within the corpus in which w_i or (w_i, w_j) occurs divided by the number of documents. To take into account the frequencies and distances between different words, the C_V metric incorporates a sliding window and reduces the number of documents and the appearance of the words to the number of documents and word occurrences within that window.
- **The computation of a confirmation measure** ϕ for every $S_i = (W', W^*)$ which measures how strongly is the relationship of W' with W^*. This relationship is calculated representing W' and W^* as vectors $\vec{v}(W')$ and $\vec{v}(W^*)$, created by pairing them to every word in W, as shown in equation 2.4. The association between w_i and w_j is measured with the NPMI (Normalized Pointwise Mutual Information) function, which is calculated with the equation 2.5. In 2.4 and 2.5, ϵ is used to account for the logarithm of zero and γ is used to apply a bigger weight to higher NPMI measured values

  Finally, each confirmation measure ϕ for each S_i is calculated by the cosine similarity between $\vec{u}$ and $\vec{w}$, being $\vec{v}(W') \in \vec{u}$ and $\vec{v}(W^*) \in \vec{w}$, as shown in 2.6.

$$\vec{v}(W') = \left\{ \sum_{w_i \in W'} \text{NPMI}(w_i, w_j)^{\gamma} \right\}_{j=1,\ldots,|W|} \tag{2.4}$$

$$\text{NPMI}(w_i, w_j)^{\gamma} = \left(\frac{log \frac{P(w_i,w_j)+\epsilon}{P(w_i)P(w_j)}}{-log(P(w_i, w_j) + \epsilon)} \right)^{\gamma} \tag{2.5}$$

$$\phi_{S_i}(\vec{u}, \vec{w}) = \frac{\sum_{i=1}^{|W|} u_i \cdot w_i}{||\vec{u}||_2 \cdot ||\vec{w}||_2} \tag{2.6}$$

- **The aggregation of the measured ϕ values**, calculated as the arithmetic mean of all the confirmation measures, as shown in 2.7 .

$$C_V = \frac{\sum_{i=1}^{|W|} \phi_i}{|W|} \tag{2.7}$$

The main reason of using C_V is because this metric is, with respect to the other existing ones, the one that has the highest correlation with respect to the perception of topics by humans [40].

The higher the value of the Coherence Score, the greater the independence between sets (topics) and, therefore, the more different they will be from each other. Therefore, the objective in terms of optimizing the model will be to obtain the highest number of topics with the highest coherence value, in a process that also involves human perception and other metrics and processes.

- **Perplexity**

  The perplexity is a measure of how much a model is surprised if it sees new unseen words. There are some ways to measure it, but in this project it has been used the Gensim *log_perplexity* function, which implements the below log-likelihood approach [41], where w_d is a set of unseen documents, and k and α are two of the hyperparameters of the model (number of topics and per-document topic weight hyperparameters, respectively). Higher values of perplexity imply a higher performance of the model.

$$perplexity = log(p(w|k,\alpha)) = \sum_d log(p(w_d|k,\alpha)) \tag{2.8}$$

The generated LDA models in this project have been selected based on these two metrics, but mainly the Coherence Score since it returns results that are easier to analyze. Perplexity results are not always correlated with the results derived from human observations, so a priori it will be more effective to analyze the Coherence Score, although in this project both metrics have been analyzed.

2.3.2 Word Embeddings

Word Embedding is a NLP technique which is used to transform words into numerical vectors. These vectors tend to be near, within the vector space, to other vectors which are semantically related. For this reason, this technology makes it possible to relate words based on their transformation into vectors of real numbers, making it possible to extract meaning relationships that other methods, such as the Bag of Words or Tf-idf formats, do not allow.

In this project, two approaches of Word Embedding techniques, Word2Vec and FastText, both implemented in the Gensim library, have been implemented and tested.

2.3.2.1 Word2Vec

The Word2Vec approach [30] is a Word Embedding algorithm developed by Google. It arises as an alternative to the representation of texts in bag-of-words format, where the semantic characteristics of the words are not taken into account, but only their distribution throughout the corpus.

The Word2Vec algorithm is presented with two variations: the Continuous Bag-of-Words Model and the Continuous Skip-gram Model. Both are based on neural networks trained in two steps: the learning of the word vectors using simple models and the training of the n-gram NNLM (Neural Net Language Model) [6] on top of these vectors. Figure 2.7 shows the architecture of both models.

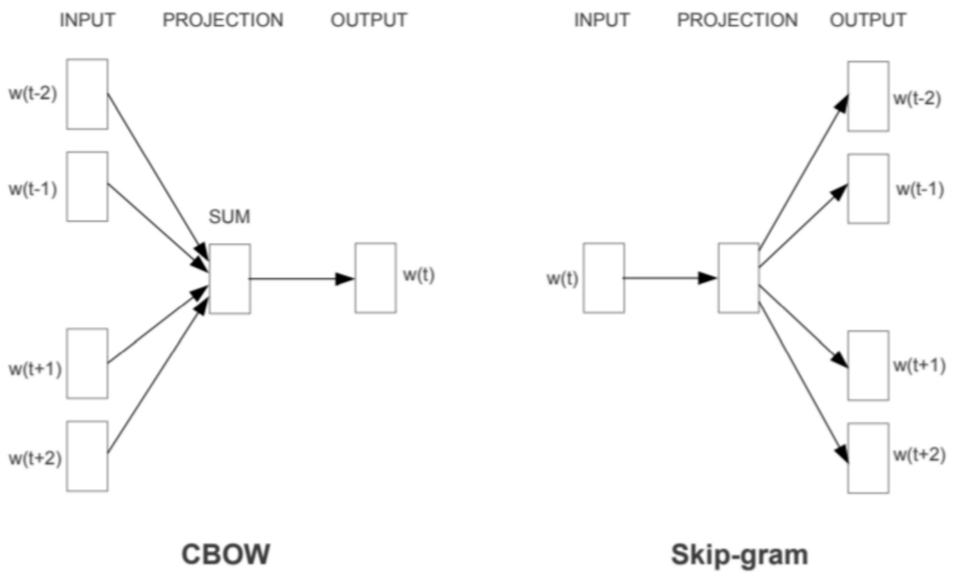


Figure 2.7: CBOW and Skip-gram Word2Vec models [30]

The main differences between them are:

- **CBOW model:** Words are predicted when given their context. The architecture is similar to a feedforward NNLM and the non-linear hidden layer is removed and the projection layer is shared for all words. Every word is projected into the same position (average vector) and the order of words does not affect the projection. For this reason, it is called Continuous Bag-of-Words model. Future words are also used and improve the performance of the model, but it is also more computationally expensive the more future words are used.

- **Continuous Skip-gram model:** Surrounding words are predicted when any word is given. Each word is the input of a log-linear classifier with a continuous projection layer, and it predicts words within a certain range before and after the input word. Furthermore, increasing the range makes the model better but more expensive. Words that are more distant are given a lower weight than those that are closer, since each word depends to a greater extent on the words that are closer to it, although words that are more distant can also give context information.

In this project, the Word2Vec generated models have been used the CBOW approach, since the Skip-gram approach has been applied with the FastText variation which will be explained below.

2.3.2.2 FastText

The FastText approach [8] is another Word Embedding algorithm developed by Facebook. The main characteristic of this approach is that, when creating vectors, the model takes into account the morphological differences between the different words. To do this, FastText implementations consider subword units instead of the whole word, and represent words by a sum of its character n-grams.

This model is derived from Continuous Skip-gram models [30], which, as explained before, try to obtain the surrounding words (context) when a word is given.

Nevertheless, in the FastText approach, instead of analyzing the context of each word, the context of the n-grams of each word formed by its characters is analyzed. To achieve this, the special symbols $<$ and $>$ are added at the beginning and at the end of each n-gram. For example, the word *where*, with $n = 3$ will be represented as $<$*wh, whe, her, ere, re*$>$ and also it will include the whole word $<$*where*$>$ [8]. It is important to note that, in this case, the word *her* will be represented as $<$*her*$>$ and it will be different to the tri-gram *her* derived of the division of the word *where*.

This way of representation makes it possible to take into account, for example, the suffixes and prefixes of the words in the corpus. Finally, each word is represented as a tuple containing the index of the word in the corpus dictionary and the set of n-grams, which are previously hashed.

The performance of this model, in contrast to the skip-gram model, will take into account the n-grams context instead of the whole word context, which will also take into account because the whole word is included in the n-grams set.

2.3.3 SIMON

SIMON [5] is a tool developed by the Intelligent Systems Group (GSI), which is a feature extractor originally designed for sentiment analysis. SIMON needs two things for working:

- A **lexicon**, which is a set of words related to the domain. E.g., in the sentiment analysis domain, the lexicon will contain words which express negative, positive, or neutral sentiments.
- A **Word Embeddings** model, which computes words as vectors.

The idea behind SIMON is that it generates a training matrix of dimensions (d_N, l_N), where d_N is the number of documents and l_N is the length of the lexicon. Features are generated by a similarity function which measures the similarity of the embeddings vector of each document with every word of the lexicon, as it can be seen in the next figure.

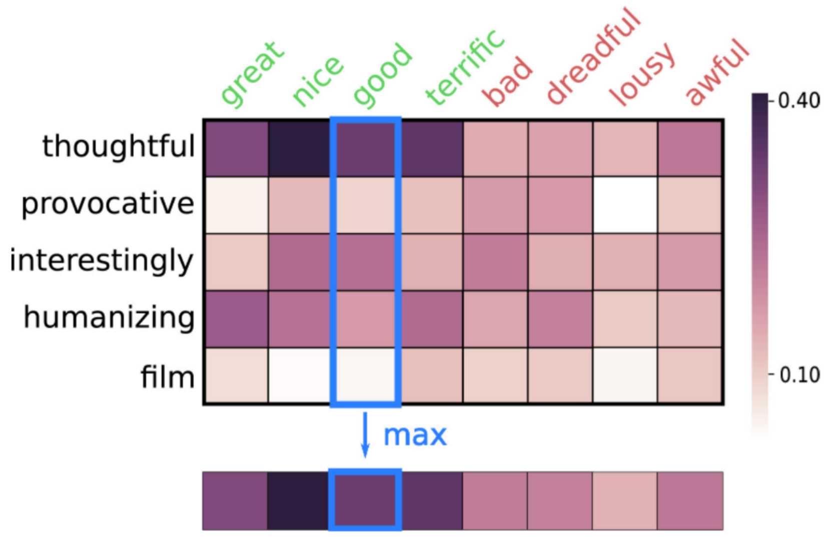


Figure 2.8: SIMON features generation [5]

SIMON can be easily implemented in a scikit-learn pipeline, and it is possible to apply functions that act as feature selectors. Specifically, SIMON provides a parameter which allows to set a percentile function and select only some of the features, with the aim of selecting the best features and avoiding those which introduce noise into the system. When sentiment analysis is performed, SIMON generates features depending on the relationship of the words of a document with the lexicon words, which can be labelled as positive or negative words. The use of SIMON for this type of analysis has been extensively demonstrated [4].

2.3.4 Tf-idf representation

As well as Word Embeddings, Tf-idf is another methodology used for text representation. This format transforms a Bag of Word matrix or a token count matrix into another matrix in which the frequency of the words in every document is taken into account. This transformation is made based on the next expression

$$\text{tf-idf}(t,d) = tf(t,d) * idf(t) = tf(t,d) * \left(\log\left(\frac{1+n}{1+df(t)}\right) + 1 \right), \tag{2.9}$$

where t is a token, d is a document, $tf(t,d)$ is the term-frequency of a token in a document (how many times t appears in d), n is the number of documents and $df(t)$ is the document frequency of t (the number of documents in which appears t divided by the total number of documents in the corpus).

2.3.5 Logistic Regression

The Logistic Regression algorithm is a predictive supervised algorithm used in classification problems. In particular, in this project the Multiclass Logistic Regression algorithm based on the One vs. All technique have been used in order to be able to classify data into multiple classes. This algorithm is a generalization of the binary Logistic Regression, which only can predict two classes based on the Sigmoid function and a threshold (usually located at 0.5).

$$f(x) = \frac{1}{1+e^{-}x} \quad \text{(Sigmoid function)} \tag{2.10}$$

If the probability of an input data is $\geq$ threshold, it will be classified as belonging to one of the categories. If the probability is $\leq$ threshold, it will be classified as belonging to the other category.

For multiclass classification, the One vs. All implementation generates as many different classifiers as the number of classes. Each classifier tries to classify one category versus the rest of categories. The combination of each classifier generates the full Logistc Regression model.

2.3.6 Gradient Boosting

The Gradient Boosting algorithm is a Machine Learning supervised algorithm for regression and classification tasks. It produces a classification model based on an ensemble of decision trees.

A decision tree is one of the most used Machine Learning algorithms. Consists of a decision selector based on certain conditions. An example of a decision tree can be seen in Figure 2.9.

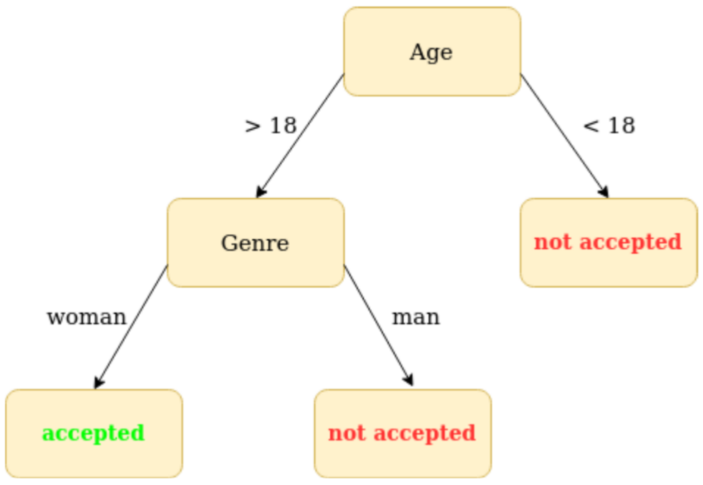


Figure 2.9: Decision Tree Example

The Gradient Boosting starts creating weak classifiers (decision trees) and then generalizes them creating an ensemble model.

The used library for the implementation of this algorithm is XGBoost. This library allows to implement the algorithm using the scikit-learn API, so it is easy to assign parameters to the algorithm. The used implementation is the XGBClassifier, which allows multiclass classification and has the following parameters:

- **objective:** To specify the learning task for which the model is trained. in this project, the objective used has been *multi:softprob*, which allows to train the model for multiclass prediction.
- **random_state:** It establishes a random seed for reproducibility.
- **eval_metric:** The metric used to evaluate the model. In this project, the *mlogloss* metric has been used.
- **num_class:** Number of classes that the model must identify.
- **learning_rate:** The learning rate of the algorithm $\in [0, 1]$.
- **max_depth:** Defines the maximum depth of the base learners.
- **min_child_weight:** Defines the minimum weight for a child to be considered part of the tree.
- **gamma:** Minimum loss reduction required to make a further partition on a leaf node of the tree.
- **colsample_bytree:** Subsample ratio of columns when constructing each tree.

CHAPTER 3

System Architecture

3.1 Introduction

The developed system is composed of several components that need to work together in a coordinated manner to provide the total required service. The different processes that constitute the system have been virtualized and are launched in Docker containers. The overall system is assembled as a pipeline, so that each input of a process is the output of the previous process. The global system is launched with docker-compose and data is retrieved, analyzed, and stored once every 24 hours, so that data can be displayed in real time.

In addition, many of the processes are accessible through a web interface that is enabled on different ports. The next figure shows the schema of the system architecture, which will be explained in this section, as well as each process that makes up the entire system.

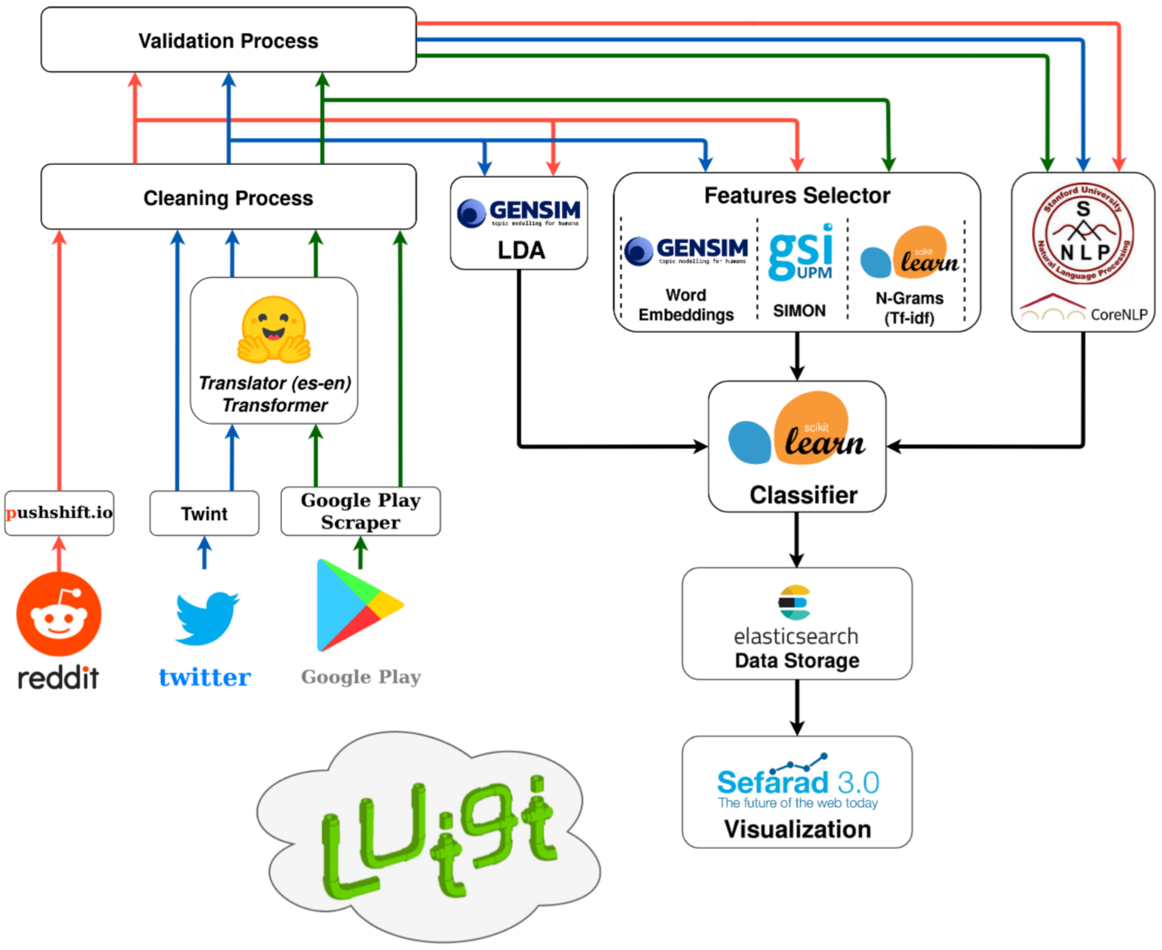


Figure 3.1: Full System Architecture

3.2 Collecting data

The data on which we have worked during the development of the project and for which the system has been developed are mainly texts. Specifically, as previously mentioned, these texts have been collected from social media sources such as Reddit, Twitter and the Google Play Store.

As explained in Section 2.2.1, several tools and libraries have been used to extract texts from these social media sources. The main data have been collected from Reddit (Ride-Hailing domain) and Twitter (Radicalization domain) and almost all the processes carried out have been based on these posts. In addition, Twitter data and Google Play data belonging to the Ride-Hailing domain have been collected to study the impact of extrapolating this analysis to other sources and types of data.

The Collecting Process also includes a Cleaning Process and a Validation Process, which are in charge of processing data for its correct later analysis, and a Translation Subprocess, which main function is to translate Spanish texts into English texts.

3.2.1 Scraping Process

The Scraping Process retrieves the required data. As said before, the Pushshift API, the Twint library, and the google-play-scraper library have been finally used for collecting data from Reddit, Twitter, and Google Play, respectively.

Firstly, the Pushshift API, which has been used for Reddit posts, has many parameters to make searches as specific as possible. The requests to the API used in this project uses several parameters to collect data, such as *before* and *after* parameters, to choose the dates between the collected posts were posted, *size*, to set the size of the request, *sort*, to sort in a descendent or ascendant way the retrieved posts, or the *subreddit* parameter, to choose the subreddit. Also, the endpoint of the API needs a parameter to set which kind of post will be retrieved. This field can be either *comment* or *submission*. An example of this request can be seen below:

```
https://api.pushshift.io/reddit/submission/search?subreddit=
uber&before=1618790400&after=1618012800&size=100&sort=desc
```

The above request will get 100 submission posts (it only returns 43, since only 43 submissions were published in the *r/uber* subreddit between those dates) posted in the *r/uber* subreddit, published between April 10 and 19, 2021.

As explained before, Reddit is composed by two types of post: **submissions** and **comments**. A submission is a main post, composed by the title, the text (called “selftext” in the API response) and another metadata information, such as the score (punctuation of the post), images or videos, hyperlinks, Reddit awards, etc. On the other hand, a comment is a post which hangs from another post or comment. They are composed by the text (called “body” in the API response), the score, multimedia data, and more.

Furthermore, Twitter data has been collected using the Twint library, as said before. The search for the respective tweets has been made based on certain hashtags and usernames related to the domain.

An example of Python code that collects tweets featuring a particular hashtag posted between two specific dates is shown below.

```
import twint

c = twint.Config()
c.Search = '#hashtag' # The term that must be in the retrieved tweets
c.Hide_output = True
c.Store_object = True
c.Since = '%YYYY-%mm-%dd' # The date since tweets will be retrieved
c.Until = '%YYYY-%mm-%dd' # The date until tweets will be retrieved

twint.run.Search(c)
```

Finally, Google Play data have been collected by the google-play-scraper Python library. Requests have been made searching for the names of specific applications.

A piece of sample code in which all reviews of a particular app are collected is shown below:

```
from google_play_scraper import reviews_all

result = reviews_all(
    'app_name', # Name of the app
    lang='es', # Language of the app
    country='es', # Country of the app
    sleep_milliseconds=0, # Sleep time between requests
    filter_score_with=None # Filter reviews by score value
)
```

3.2.1.1 Translation Subprocess

Some collected data sources are written in Spanish. The performed analysis and architecture has been developed only to analyze texts written in English. Processes such as data cleaning or almost every NLP process applied to the texts are specifically designed for English texts. For this reason, as explained in 2.2.3.4, the Marian Machine Translator transformer has been implemented in the system architecture to translate all texts that are written in Spanish. Texts written in another language are not considered, since they are rejected in the Cleaning Process that will be described below.

The Marian transformer needs a special tokenizer and a pretrained model to work. This model has been downloaded from the Language Technology Research Group of the University of Helsinki web page, accessible from the Hugging Face website[1]. This research group provides a lot of packages called *opus*, which are specifically designed for translation

[1]https://www.huggingface.co/Helsinki-NLP

purposes. In particular, the *opus-mt-es-en* package, which has been trained from Spanish to English translation, has been used.

For translation, a label is added to the text indicating the language into which it is to be translated (in this case, “>>en<<”). After this, the text is loaded and tokenized. The tokenizer is configured so that, in the encoding process, the different tokens are not encoded as simple numbers, but return tensors. In this case, it has been configured so that these tensors are PyTorch[2] objects. After that, these tensors enter the decoding process, from which the translated text is extracted.

An example of the code is shown below.

```
# Loading tokenizer
tokenizer = MarianTokenizer.from_pretrained('Helsinki-NLP/opus-mt-es-en')

# Loading pretrained model
model = MarianMTModel.from_pretrained('Helsinki-NLP/opus-mt-es-en')

src_text = ['>>en<<' + text] # Adding the language label

# Encoding the text as tensors
tensors = self.model.generate(**tokenizer(src_text, return_tensors=``pt'',
    padding=True))

# Translating text
translated_text = [tokenizer.decode(t, skip_special_tokens=True) for t in
    tensors]
```

3.2.2 Cleaning Process

The Cleaning Process is done once the data is obtained. Its function is to clean the data, meaning to clear the texts, leaving them machine-readable, without strange characters that may affect the operation of other algorithms and processes.

The main component of the process, which is common to every data source, is based on regular expressions that try to clean the text as much as possible. These expressions reject some characters that do not provide information to the text and introduce noise. Moreover, these expressions try to fix some misspelled words that are often misspelled when writing on social media interfaces from a cell phone, a computer, or another similar device. An example of this cleaning process could be:

- **Original sentence:** *“Uber is operating/working in Barcelona and they dont pay taxes. What can be done? https://www.oneweb.com”*

[2]https://pytorch.org/

- **Cleaned sentence:** *"Uber is operating / working in Barcelona and they don't pay taxes. What can be done?"*

Before texts are readable and well-written, it is important to know what kind of words contain. In Twitter and Google Play data, the language of texts is known because the way in which the texts are collected specifies the language of the page and the texts. However, in Reddit data, although the subreddits are supposed to be in English, some texts are written in other languages and usually contain expressions that are not suitable for translation to perform the intended analysis. For this reason, Reddit texts go through another process of language identification after the cleaning phase.

Different approaches have been tried for language identification, specially with short texts, where the most common techniques are a dictionary-based identification or the use of n-grams [51]. As Reddit is composed by short and large texts, in the Cleaning Process the language identification process has been divided in two phases.

The first phase consists of a search for the probability that the text belongs to some language. It is made with the CLD3 library. If its English probability is the higher, the text is considered to be in English and the text is added to the data set.

The second phase is a dictionary-based approach and it is for texts that have been rejected in the first one. In this phase, several dictionaries of different languages are loaded and compared with the words of the text. The language of the text will be the language of the dictionary that has the closest match to the words in the text. If this language is not the English language, the text is definitely rejected.

The approach made in the second phase is necessary to avoid the noise produced by social media related words that can be identified as other language words. After this Cleaning Process, every cleaned English-written text is included in the whole data set and it passes through the next process.

3.2.3 Validation Process

After cleaning, posts need to be validated to be finally integrated into the data set to be analyzed and visualized. This validation is based on a rules-based approach. These rules are different depending on the source of the posts, although there are some shared rules. Figure 3.2 shows how this process works.

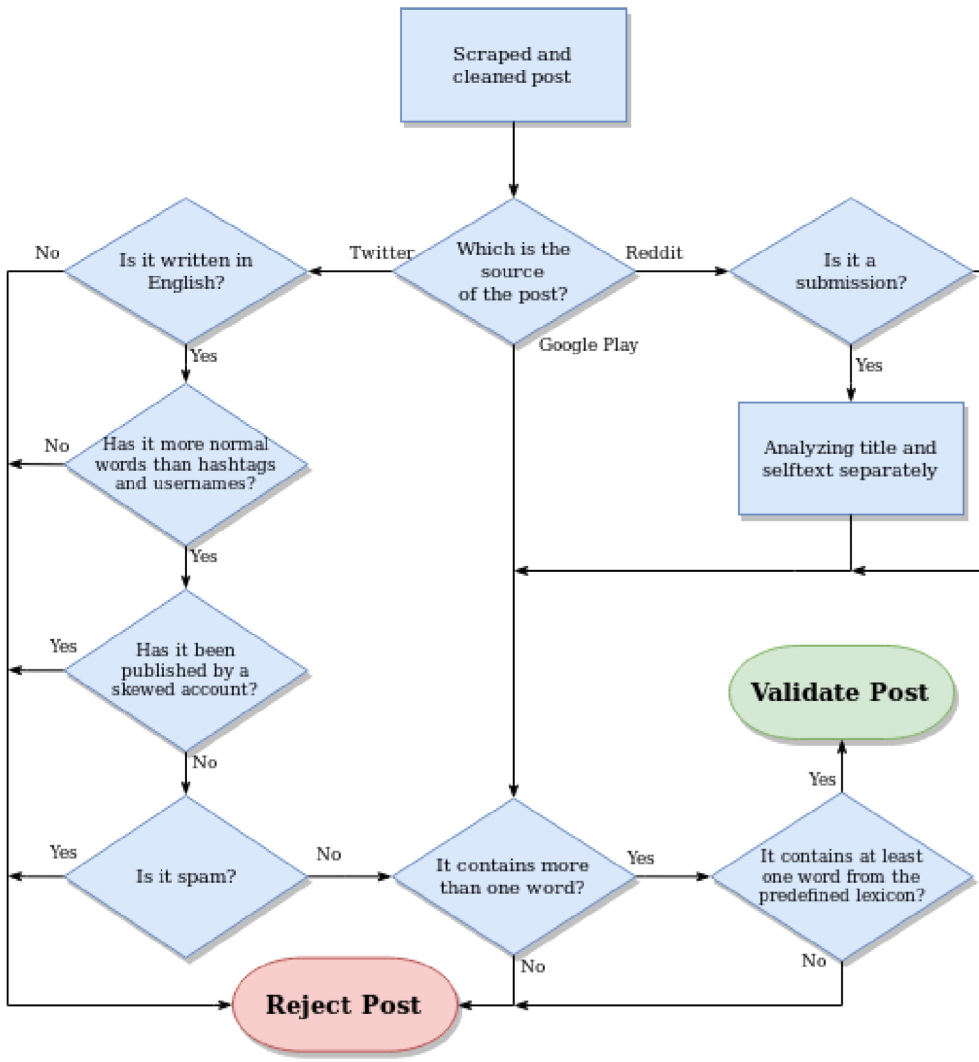

Scraped and cleaned post
Which is the source of the post?
Twitter
Reddit
Google Play
Is it written in English?
No
Yes
Has it more normal words than hashtags and usernames?
No
Yes
Has it been published by a skewed account?
Yes
No
Is it spam?
Yes
No
Is it a submission?
No
Yes
Analyzing title and selftext separately
It contains more than one word?
Yes
No
It contains at least one word from the predefined lexicon?
Yes
No
Validate Post
Reject Post

Figure 3.2: Validation Flow Chart

As it can be seen, the validation process is slightly different depending on the source of the data. Google Play and Reddit posts are usually more formal and well-written than Twitter posts. Additionally, the way in which Twitter posts are collected is less specific than the others. For this reason, Twitter posts have a more exhaustive validation process.

- **Twitter Validation Process:** As said before, Twitter data is collected by searching for specific words, in particular hashtags and user names. For this reason, its validation process is more strong.

  The first step is to analyze the language of the post. This is easier than in the Reddit case because the Twint library retrieves the language of the collected posts. If the text is not written in English, it will be rejected.

  The second step is to analyze the text of the post for hashtags (words starting by the

Twitter special character #) and usernames (words starting by the Twitter special character @). If the text contains a lot of these kinds of words, the analysis of the text will not provide much information. For this reason, if the text had more hashtags and usernames than normal words, it would be rejected.

Next, the user that had post the tweet is analyzed. If the post was posted by an official account with a strong relationship with the analyzed domain, the post will be rejected. Also, if the username information of the post contains words related to the domain, this post will be rejected too. With this rule, it is intended to analyze data provided by users that express their opinion in a disinterested way. For instance, if *@cabify_espana* or *@uber_is_cool* would publish a post and the system collects this post, it will be rejected.

The next filter consists on checking if a tweet is spam or not. The words of the tweet are compared with a vocabulary that contains expressions related to spam. If there is a coincidence, the tweet is rejected.

Another mandatory rule is that texts need to have more than one word. Posts with less than two words are rejected.

Finally, if the intersection between an extracted vocabulary (extracted from the topic modeling procedure, Section 3.4.1) and the words of the tweet is not null, the post will be accepted.

- **Reddit Validation Process:** Reddit posts can be either submissions or comments. If the analyzed post is a submission, the title and the text of the submission are analyzed in a separately way. The process can reject the title and accept the post text, and vice versa.

  The text will pass through a two-step process. First, the text must have more than one word. Finally, the text must have at least a word that matches any of the words that make up the vocabulary mentioned above. If the text passes these steps, the text will be accepted.

- **Google Play Validation Process:** Google Play reviews follow the same steps than Reddit texts.

3.3 Enrichment Process

The Enrichment Process includes all the processes that add interesting data to the retrieved texts. This data addition is based on the processing of texts to extract data with which to analyze the texts from the intended perspective in this project.

The Enrichment Process involves Sentiment Analysis and NLP Data Extraction Process.

3.3.1 Sentiment Analysis and NLP Data Extraction Process

The Sentiment Analysis and NLP Data Extraction Process is the process in which a large amount of information is extracted from texts. The result of this process is an annotated text including some relevant information. As it was explained in 2.2.2.1, the Stanford CoreNLP was used in this process.

The Stanford CoreNLP can be executed in different ways. Mainly, both the server and client version can be used, depending on how it is to be used.

The client version is useful when data is stored in files. These files can be divided in different files, containing each one a text. This process can increase the speed of the execution, allowing multi-threading execution too. It can be launched as a docker container, building a predefined image which contains an environment with the Java compiler installed, such as Alpine [3] and downloading inside the container the Stanford framework, or directly as a shell command, having previously downloaded the Stanford package. Once texts file has been split and stored in different files, the client version can be executed as follows:

```
#Create a file that contains every .txt path.
ls tmp/text*.txt > all-files.txt

#Run Stanford CoreNLP client version
java -mx12g -cp ``*'' edu.stanford.nlp.pipeline.StanfordCoreNLP
-parse.model edu/stanford/nlp/models/lexparser/englishPCFG.caseless.ser.gz
-pos.model edu/stanford/nlp/models/pos-tagger/english-caseless-left3words-
    distsim.tagger
-ner.model edu/stanford/nlp/models/ner/english.all.3class.caseless.distsim.
    crf.ser.gz,edu/stanford/nlp/models/ner/english.muc.7class.caseless.
    distsim.crf.ser.gz,edu/stanford/nlp/models/ner/english.conll.4class.
    caseless.distsim.crf.ser.gz
-annotators tokenize,ssplit,pos,parse,lemma,ner,sentiment  -filelist all-
    files.txt  -outputFormat json -timeout 100000 -threads 4
```

As it can be seen, the Stanford CoreNLP framework specifies some parameters that must be defined. There are several parameters that can be set to perform the analysis. The above parameters are:

- **-mxXXg:** It specifies the amount of RAM that the core will be allowed to use. For large amounts of text, it is recommended to use at least 8 GB of RAM, especially if multi-threading is to be used. In this case, a maximum of 12 GB of RAM has been allocated.

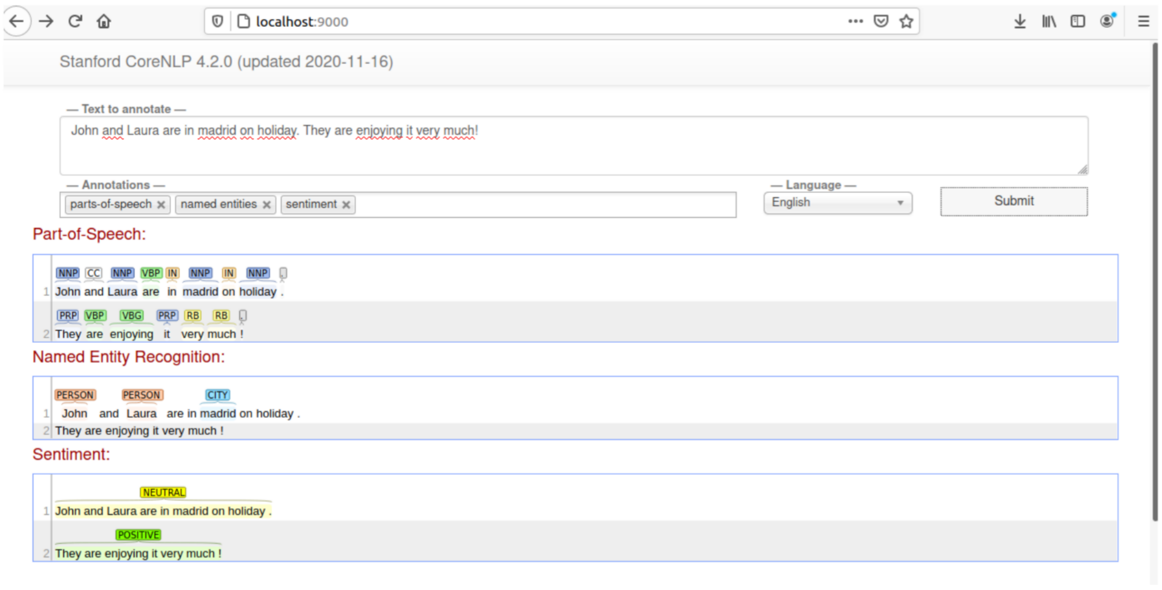
[3] https://alpinelinux.org/

- **-cp:** Used to select the jar files that will be loaded in the current directory. It is set as "*", which means that all available jar files will be loaded.
- **edu.stanford.nlp.pipeline.StanfordCoreNLP:** The version of the Stanford Core NLP. This version is the client version, which allows to send lists of text files to annotate them.
- **-parse.model, -pos.model, -ner.model:** They are the caseless models which have been used in the project.
- **-annotators:** Specifies the annotators to be used to annotate the texts. As explained before, to perform sentiment analysis, POS-tagging, and entity extraction, it is necessary to load other annotators that process the texts previously to perform such analysis and annotations.

  The annotators used in this project are *tokenize* (to tokenize texts in words), *ssplit* (sentence split), *pos* (POS-tagging), *parse* (to apply a parser and annotate texts according to this parser), *lemma* (to perform the lemmatization process), *ner* (NER extraction) and *sentiment* (sentiment analysis).
- **-filelist:** A list containing the path of every text
- **-outputFormat:** The output format of the annotated texts. It can be set as JSON or XML, among others.
- **-timeout:** The maximum time that any thread can be processing a text. If it takes longer than the time specified here (measured in milliseconds), the text is rejected.
- **-threads:** The number of concurrent threads used in the analysis. It is recommended to use as many threads as the number of cores available in the used machine.

This way of using the Stanford CoreNLP is useful in some situations, as explained above. However, with the aim of integrating this framework in the system, it is necessary to use the server version, which will be listening for new input data.

The framework has been integrated in the system as a Docker container, built from the same Alpine image. The executed program in this version is *edu.stanford.nlp.pipeline.StanfordCoreNLPServer*, which starts a server listening on port 9000. For testing, the server can be accessed from a web interface where a text can be entered and annotators can be selected in an interactive way, viewing the result of the process from the same web interface. This web interface can be seen in Figure 3.3. It is important to note that the example shows how caseless models work.



localhost:9000

Stanford CoreNLP 4.2.0 (updated 2020-11-16)

— Text to annotate —

John and Laura are in madrid on holiday. They are enjoying it very much!

— Annotations —

parts-of-speech × named entities × sentiment ×

— Language —

English

Submit

Part-of-Speech:

```
  NNP  CC  NNP   VBP  IN  NNP    IN  NNP
1 John and Laura are  in  madrid on  holiday .
  PRP  VBP VBG      PRP RB   RB
2 They are enjoying it  very much !
```

Named Entity Recognition:

```
  PERSON     PERSON        CITY
1 John   and Laura  are in madrid on holiday .
2 They are enjoying it very much !
```

Sentiment:

```
  NEUTRAL
1 John and Laura are in madrid on holiday .
  POSITIVE
2 They are enjoying it very much !
```

Figure 3.3: Stanford CoreNLP Web Interface Example

For the automation of this process, the rest of the parameters are not specified from the command line, but, using the Python module *pycorenlp.corenlp.StanfordCoreNLP*, they can be specified from the same code in which this process is launched.

3.4 Classification Process

The Classification Process is the process in which texts are annotated with labels that identify the main topic of the text. This process includes several processes which are based on machine learning for the identification of such topics. Firstly, the LDA algorithm is used to analyze the topics that are latent in the corpus. Once the topics have been extracted, it is necessary to develop a technique to find these topics in new texts that have not yet been seen. This technique will be based on Word Embeddings models. Finally, a Machine Learning classifier has been developed to predict new labels.

The procedures carried out for this task, as well as the implementation of various algorithms and data modeling will be explained in this section.

3.4.1 Topic Modeling

Topic extraction is useful to know in which terms social media users are talking and it is one of the fundamental processes in the development of the system. In this project, two models, one per domain, were generated with the aim of classifying by topic each text of the corpora.

The performed procedure is similar in both domains and the scheme can be seen in Figure 3.4.

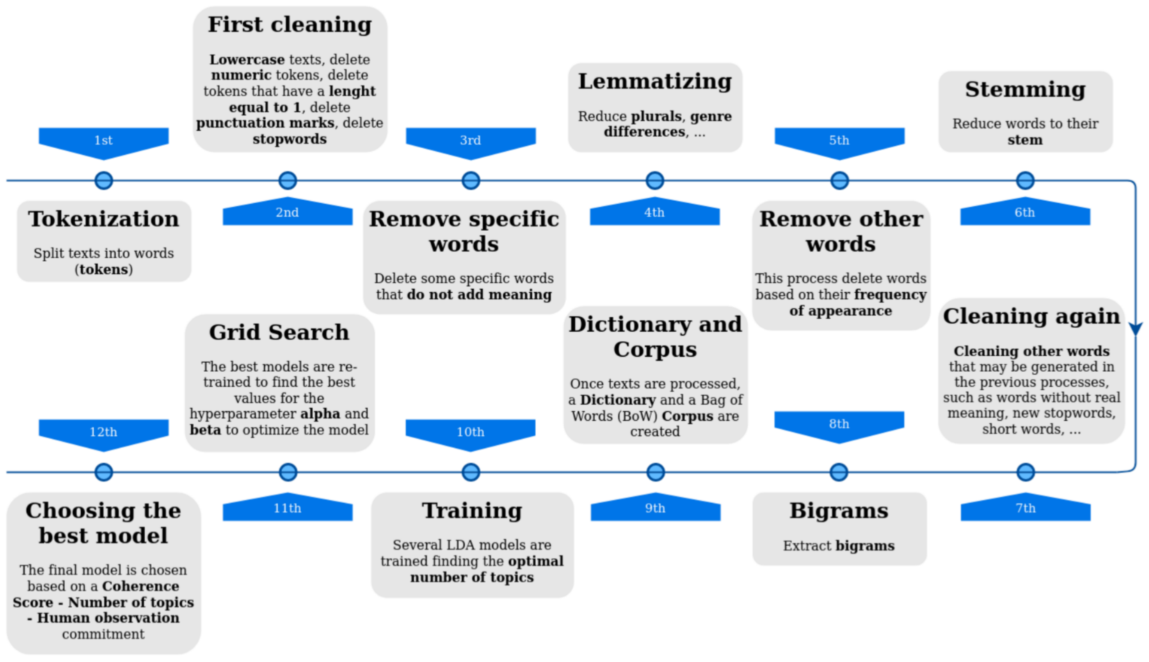

1st
Tokenization
Split texts into words (tokens)
2nd
First cleaning
Lowercase texts, delete numeric tokens, delete tokens that have a lenght equal to 1, delete punctuation marks, delete stopwords
3rd
Remove specific words
Delete some specific words that do not add meaning
4th
Lemmatizing
Reduce plurals, genre differences, ...
5th
Remove other words
This process delete words based on their frequency of appearance
6th
Stemming
Reduce words to their stem
7th
Cleaning again
Cleaning other words that may be generated in the previous processes, such as words without real meaning, new stopwords, short words, ...
8th
Bigrams
Extract bigrams
9th
Dictionary and Corpus
Once texts are processed, a Dictionary and a Bag of Words (BoW) Corpus are created
10th
Training
Several LDA models are trained finding the optimal number of topics
11th
Grid Search
The best models are re-trained to find the best values for the hyperparameter alpha and beta to optimize the model
12th
Choosing the best model
The final model is chosen based on a Coherence Score - Number of topics - Human observation commitment

Figure 3.4: LDA Procedure Flow Chart

As it can be seen, the LDA procedure has 12 steps. Phases 1 to 8 belong to document processing, while phases 9 to 12 belong to training, model generation and model selection. It is important to note that many of these phases have been modified during the development of the process to achieve the best model as possible. This has been carried out using certain techniques, such as direct observation or computational techniques. These techniques will be explained later.

- **Phases 1 to 8 - Processing of documents**

  The first two phases correspond to the preparation of the corpus, carrying out processes such as the tokenization of the documents, the transformation of texts avoiding capital letters, short words (short words are usually prepositions, wrong-written words or words with no interesting meaning), punctuation marks, numbers and stopwords (prepositions, personal pronouns and determiners, among others).

  Once the documents are prepared, a vocabulary is created to remove some specific words that do not have interesting meaning and could add noise to the model. Then, the lemmatization process has been carried out with the aim of reducing data sparsity. This process can generate new words that do not add meaning and they must be removed.

After these processes, the stemming process is applied to each document. After this transformation, documents are cleaned again removing new words because new stop-words or words without real meaning may have been generated.

Finally, once the documents are completely cleaned, the texts are processed searching for bigrams. This process creates new words which are combinations of two words that appear together in some documents. This is an important process that tries to avoid to analyze words that have no meaning if they are analyzed as separated words. For instance, "*united*" and "*states*" have different sense if they are analyzed as separated tokens and the bigrams extraction process joint this two words into a single token, "*united_states*", which has sense and adds meaning. New tokens generated in this process appears as single tokens joint by an underscore character ("_"), as can be observed in the previous example. Bigrams extraction has been carried out with the Gensim *Phrases* package, which allows to search for n-grams within documents and provides some parameters to change the way in which these n-grams are generated, such as window size (the distance between the n words must have to consider them as a n-gram) or the value of n, to search for unigrams (n = 1), bigrams (n = 2), trigrams (n = 3) and more.

- **Phase 9 - Dictionary and Corpus**

After processing the documents with the previous processes, a dictionary composed of every unique token that makes up the documents is created. This dictionary can be filtered to contain only words that appear within the corpus at least determined times. This function is useful to avoid wrong-written tokens that appear a few times and can add noise to the models.

Once the dictionary is created and filtered, the corpus is transformed into a Bag of Words (BoW) so that the model is able to understand the distribution of words in a machine-readable format. This format transforms tokens into tuples, been the first position of the tuple the identifier of the word within the dictionary, and the second position is the number of appearances of this word within the document to which the word being processed belongs. This transformation can be seen in Figure 3.5

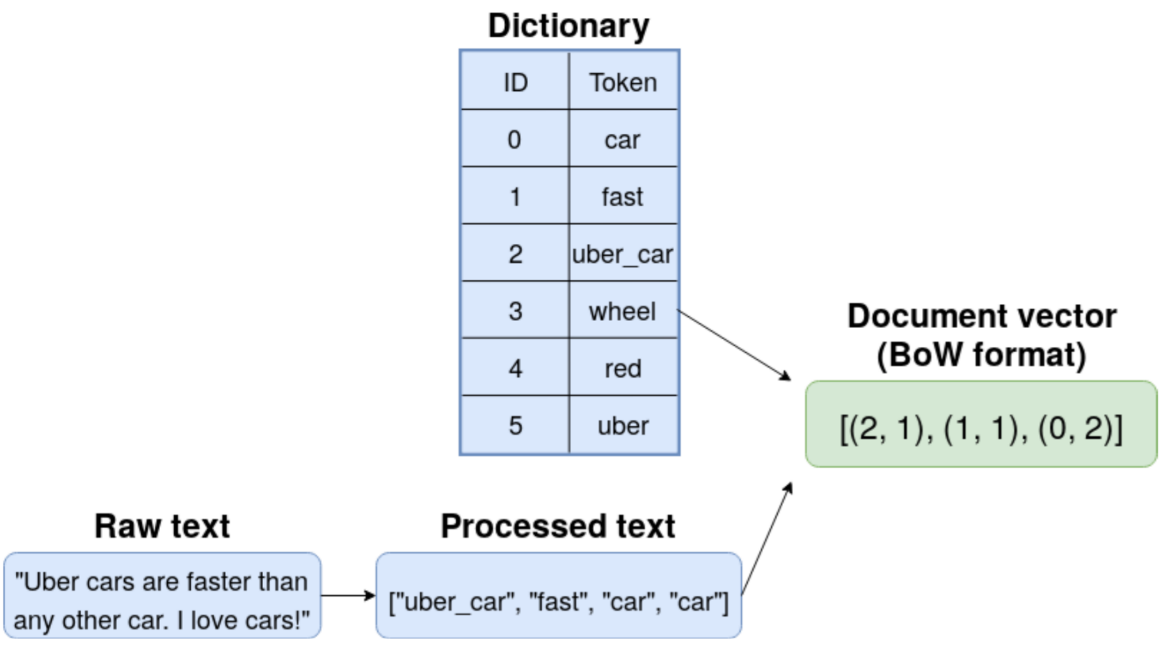

Dictionary
ID
Token
0
car
1
fast
2
uber_car
3
wheel
4
red
5
uber
Document vector (BoW format)
[(2, 1), (1, 1), (0, 2)]
Raw text
"Uber cars are faster than any other car. I love cars!"
Processed text
["uber_car", "fast", "car", "car"]

Figure 3.5: Bag of Words vectors generation example

- **Phases 10 to 11 - Training and Hyperparameter Optimization**

  The next step is to train the model. As said before, the LDA models of this project were generated using the Gensim library. This library provides mainly two versions to generate a LDA model: the LdaModel module and the LdaMulticore module, both available in the *gensim.models* package. The generated models of this project have been generated using the LdaMulticore module with the aim of using the most efficient way to generate these models and thus accelerate the process.

  The LDA algorithm needs some parameters that must be specified:

  - The **number of topics *k*** that will be extracted from documents. This value must be fixed before training and the algorithm will try to classify documents into k topics.
  - The **corpus** in a BoW format, as explained in the previous paragraphs.
  - The **dictionary** containing the tokens and their identifiers.
  - The **chunksize**, which is the number of documents used in each training chunk.
  - The **random_state** parameter, used for reproducibility.
  - The number of **passes** through the corpus during the training process.
  - $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, called *alpha* and *eta*, which are the hyperparameters of the model.

Every LDA model trained in the development of this project was trained with the multicore version, with chunksize = 100, random_state = 100, and passes = 10. In addition, every model was trained with $k \in [2, 50)$ in order to know which is the optimum number of topics for each model.

In the first approach, to get the best models and discard the worsts, α and β hyperparameters were set to their default values. The election of the best models was based on the performance metrics explained before: the Coherence Score and Perplexity.

Perplexity does not provide as much information about the best model and the optimal number of topics as Coherence Score, so the latter has been used mainly to analyze the models, although Perplexity has also been used.

The hyperparameters selection has been mainly based on the application of the Coherence Score to the best models. These best models are those that, with the number of topics already defined, are optimized by selecting the value of the hyperparameters. This optimization process has been carried out with a Grid Search optimization, finding the values that get the highest Coherence Score value.

- **Phase 12 - Choosing the best model**

  The chosen model, in addition to having a high coherence value, must be understandable to human beings and must have the largest number of topics that clearly represent a theme clearly differentiated from the rest of the topics. Therefore, the choice of the best model was based on a **Coherence Score - Number of topics - Human observation** commitment.

  For this reason, the chosen model will be the model with the highest coherence value and with the greater number of topics, but it has to be understood by a human being and make sense beyond the computational techniques used to analyze it. For the observation, the models have been plotted with the pyLDAvis tool, which allows to draw each cluster in a two-dimensional (X and Y) diagram and allows to see the correlation of two or more topics, if there are clusters intersecting with other clusters or if on the opposite they are far apart and therefore are totally independent topics.

  The pyLDAvis tool has a λ parameter that can be set between 0 and 1. The lower the λ value, the more words are selected from each cluster that belong only to that cluster. If a higher value is selected, the words displayed will have more weight in the topic, but may also appear in other topics. For this reason, the pyLDAvis tool has been very useful in the process of improving models, because it allows to see if topics are similar and why, making it easier to add words to a rejection vocabulary to remove these words from the corpus and increase the independence between topics.

After all this process, once the best model is chosen, the model is saved and stored to be used later implemented in the whole system.

3.4.2 Features Selection

The main functionality of the Classification Process is to predict the topic of new unseen texts. For prediction, a multiclass Machine Learning classifier must be trained. The training set of these models must be composed by two types of data: features and target data. Features are the representation of each text which a Machine Learning algorithm will try to learn, while the target data will be the topic information of each text. The objective of the classifier will be to relate the representation of each feature vector to its corresponding topic, and in this way assign topics to texts that the model has not yet seen.

As the objective is to relate texts to topic labels, it is mandatory to obtain good representations of every whole text. For this purpose, several different techniques have been implemented, with different performance results.

3.4.2.1 Word Embeddings Approach

As explained before, Word Embeddings models allow to transform the words of a set of texts into vectors of real numbers. This is fundamental to take into account the semantic similarity between words and to know if a word is strongly related to another word or if it is not related.

To generate these models, the first step of the procedure consists of extracting the texts to be fed into the models and processing them. These texts are directly extracted from the Cleaning Process output. It is important to note that texts are collected before the Validation Process. This is done with the objective of modeling as many texts as possible, without determining whether those texts will be rejected later or not. The more texts there are, the better the models will be able to relate the words correctly.

Texts must be processed before the model is trained. Accordingly, texts are processed in the same way that in the Topic Modeling step (Section 3.4.1). The output of this processing step must be the corpus with the cleaned and tokenized texts.

After that, different FastText and Word2Vec (CBOW approach) models are trained. Both models have been trained using some special parameters according to the Gensim API and both models have been trained using the same values for these parameters. These values are:

- **sentences:** The corpus, in a list of lists format, each list containing a cleaned and tokenized text.

- **vector_size:** The dimensions number of the output vectors. These vectors are the representation of each word within the corpus. Models with 100, 300, and 500 dimensions have been trained.
- **window:** The size of the window around each word. This window covers those words to be taken into account (i.e., the context of the word) when each word is represented as a vector. Every model developed in this project has a window size of 10.
- **min_count:** The number of times that at least a word must appear in the corpus to be taken into account when establishing relationships and making transformations. Every model developed in this project has a min_count = 5.
- **epochs:** Number of iterations around the corpus. Every model developed in this project has been developed with 10 iterations around the corpus.
- **workers:** For multicore computation. This number defines how many cores will be used to train the model.

Once a Word Embeddings model has been generated, every word of each text can be transformed into a real number vector. However, this is not enough, since the vector representation of the complete text is necessary. Therefore, this representation has been done in two different ways, which will be explained below.

- **Averaging vectors:**
  As each word is represented as a real number vector, and each text is composed of different words, the representation of a text can be modelled as the average of the sum of the vectors that compound that text.

  $$text\_vector = \frac{\sum_{i=1}^{N} w_i}{N}, \tag{3.1}$$

  where N is the number of words that forms the text and w_i is each word vector. This approach generates n-dimensional vectors, where n can be 100, 300, or 500, for each text of the corpus.
- **SIMON:**
  In this project, SIMON has been used as an alternative to the more traditional feature extraction based on Word Embeddings. The lexicon, which SIMON needs to work, is composed of a set of words from each topic. The problem is that the higher the number of topics, the larger the lexicon. This imply that the generated features become very larger and the computation tasks can be difficult.

The experiments that have been carried out in this work will be explained in the sections related to the use cases of each domain.

3.4.2.2 N-grams Approach

Another approach to feature extraction that has been used is n-gram extraction. This approach does not depend of Word Embeddings and it has been used mainly to observe if the use of Word Embeddings improves the performance of the models.

The *n-grams* are extracted in two phases which are integrated into a scikit-learn pipeline.

- **Count Vectorizer:** It converts a list of documents into a a matrix of token counts. It provides several parameters, which those that have been used are:
  - **tokenizer:** It sets the way in which texts will be tokenized. A callable function must be passed.
  - **max_df:** When creating the vocabulary, ignores tokens which appear more times than specified in this parameter.
  - **max_features:** Consider only the specified features, ordered from most frequent to least frequent.
  - **ngram_range:** A tuple which indicates the way in which the n-grams will be formed. The first index indicates the minimum and the second the maximum unit. Thus, a *ngram_range* of (1,1) will create unigrams, (1,2) will create unigrams and bigrams and (2,2) will create only bigrams.
- **Tf-idf:** Using a Tf-idf (Term frequency – Inverse document frequency) transformer, features can be converted into a Tf-idf format. The scikit-learn TfidfTransformer provides these functionalities and can be used in several ways using its provided parameters. In this project, some of them have been used:
  - **use_idf:** If *True*, the *idf* is used, and if *False*, only the *tf* part is computed.
  - **norm:** It sets different ways of normalizing the data.

The result of this process will be a matrix filled with the frequency terms of the different tokens, with which Machine Learning algorithms can be fed.

3.4.3 Training and Optimization

Once the features are selected and modelled, the training phase can start. In this phase, two classifier algorithms have been tested and optimized. It is important to note that, as the system will label texts with their respective topic, multiclass classifiers will be needed.

Thus, the Logistic Regression algorithm and the XgBoost ensemble algorithm have been used and tested.

The process of training is strongly related to the process of optimization. The training process has been carried out inside the optimization process and the final models come directly out of this process.

- **Hyperparameter Optimization:** In In machine learning, optimization usually refers to the optimization of the hyperparameters of the algorithms. There are several ways to do this, either by brute force techniques, random techniques, or with specific algorithms. In this project, they have been used different functionalities implemented in the scikit-learn library that allows to carry out these processes.

  – **Grid Search:** It trains models iterating over every possible combination of hyperparameters. It is an implementation of brute force optimization technique. When it finishes, returns the best model based on the score function, which can be selected too.

  – **Halving Grid Search:** It is a new implementation and it is still in an early version, but it works very well. It iterates over the selected parameters but through several global iterations. In each global iteration, the features are divided by the number of global iterations, reducing the computation time of the normal Grid Search implementation. When a global iteration finishes, the features are duplicated and only the best estimators are recalculated, rejecting the worsts. At the end, the estimator which has had the best score in every global iteration is the final model. The best individuals are selected based on a fitness function, which in this case was the measure of the F-Score. Thus, those models who get the best F-Score values will pass to the next generation.

  – **Genetic Algorithms:** A genetic algorithm [31] is a Machine Learning algorithm based on the process of biological evolution. It can be used to train models in the usual way or in optimization tasks [32].

    Genetic algorithms start training with an initial population. This population will be changing through generations. When a generation finishes, the individuals of the population mutate between them and the next generation starts with other populations. Thus, the model evolves until it converges.

    For optimization tasks, the initial population is formed by different implementations of the same algorithm but with different parameters. The mutation process merges the values of the parameters trying to achieve the best model. The best models are selected based on a fitness function, which in this case is based on the F-Score. This implies that those models with higher F-Score values will pass to the next generation.

- **K-Fold Cross Validation:** Is a Machine Learning technique designed to avoid bias due to bad partitions in the train and test data. When K-Fold is applied, the model is trained K times and then the score is measured as the arithmetic mean of the different K scores. The training data vary depending on the fold: the model is trained using K different training data and testing data. With this technique, the model is trained and validated with all available data to avoid overfitting and to obtain more efficient and verifiable results. In this project, K-Fold has been used with $K = 10$ in most of the processes.

The training process has been done partitioning data into equal sets to prevent bias, deleting duplicates, erroneous values, and shuffling the data to avoid mislearning.

3.5 Data Storage

The Data Storage process saves in a server the annotated and analyzed data. This server is provided by the ElasticSearch engine, which is available on port 9200. ElasticSearch is a document-oriented database, and the JSON documents generated in each analysis are posted.

The engine allows to make requests based on simple queries and aggregations, which allow to extract the information grouped by any variable. The results are provided in a JSON format, so the information retrieved is easy to process and analyze.

ElasticSearch saves the information in indexes, through which data can be accessed. The developed system saves data in an index called *posts*.

Data is stored following a Linked Data schema, using the semantic technologies explained at Section 2.2.5.2. Therefore, each uploaded post has an identifier, which is the result of applying a hash function to the post. This process is carried out to prevent the same document from being uploaded several times. A general schema of how each analyzed post is labelled and uploaded is shown in Figure 3.6. The keys of each field of which the data is composed are the following.

- **_id:** The identifier of the post inside ElasticSearch. It is the result of the application of the hash function to the post.
- **@type:** It is the type of the content. In this project, the @type key can be set as *tweet*, *google_play_review* or *reddit*.
- **sioc:name:** The name of the author of the post.

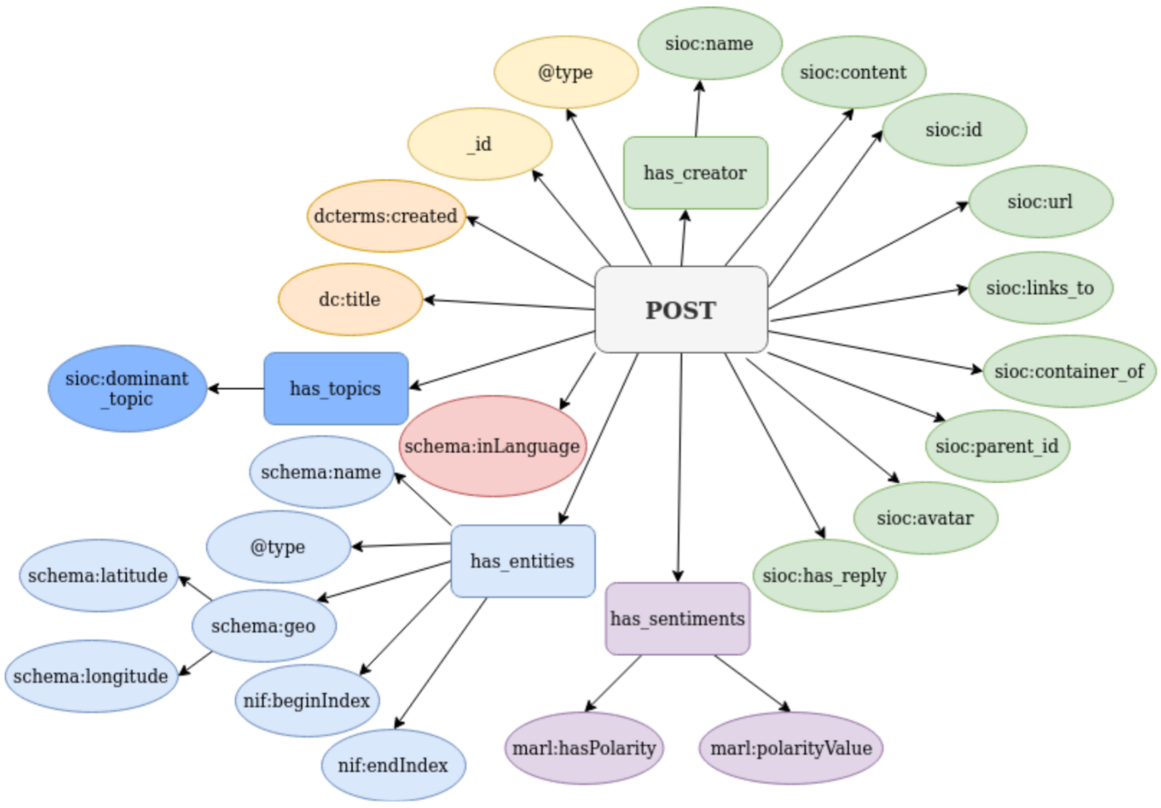


Figure 3.6: Semantic Data Structure

- **sioc:content:** The content of the post, that is, the text.
- **sioc:id:** The identifier of the post inside the social media source.
- **sioc:url:** The url in which that post is accessible.
- **sioc:links_to:** The web links that a post could include inside its content.
- **sioc:container_of:** Its belonging subgroup inside the social media source. It can be a specific app, a subreddit...
- **sioc:parent_id:** The id of the parent post, if exists.
- **sioc:avatar:** The link to the user image.
- **sioc:has_reply:** The possible replies that a post could have.
- **schema:inLanguage:** The language in which the post is written.
- **dc:title:** The title of the post, if any.
- **dcterms:created:** The created date of the post.

- **has_entities:** The entities within the post:
  - **schema:name:** The name of the token / tokens that have been detected as an entity.
  - **@type:** The type of entity: Organization, Time, Person, ...
  - **schema:geo:** The geographic location of the entity if is a place entity. It has two properties: **schema:longitude** and **schema:latitude**.
  - **nif:beginIndex:** The index of the character in which the entity starts.
  - **nif:endIndex:** The index of the character in which the entity ends.
- **has_sentiments:** The analyzed sentiments of the text:
  - **marl:hasPolarity:** It can be *Positive*, *Neutral*, or *Negative*.
  - **marl:polarityValue:** The value of the sentiment, which can be between 0 to 4.
- **has_topics:** The extracted topics of the text:
  - **sioc:dominant_topic:** The dominant topic of the post.

3.6 Orchestration

The system follows a pipeline architecture, where processes extract information from data and transform them, sending the output of each process to the next process. This pipeline is handled by Luigi, which acts as an orchestrator and manages the pipeline.

Luigi organizes the pipeline into tasks, which are each process of the system. In addition, there is a main task that starts the pipeline by calling the first task to be executed.

Tasks are composed of different functions, of which the most important are:

- ***requires() function***, which returns the task which needs to be executed before the current task.
- ***run() function***, which contains the code of the current process.
- ***output() function***, which is used to save the result of the process in an external file. It is very useful for debugging and error handling.

Since each task requires the output of the task that precedes it, the main task starts calling to the last process of the pipeline. When the first task to be executed is required, in this case it would be the task containing the Scraping Process, as this task does not require any other process to be executed, the pipeline starts to run.

This process is executed once every 24 hours, collecting and analyzing data published in the last day. This functionality makes the system a real-time scanning and analysis system.

The architecture schema is shown in Figure 3.7.

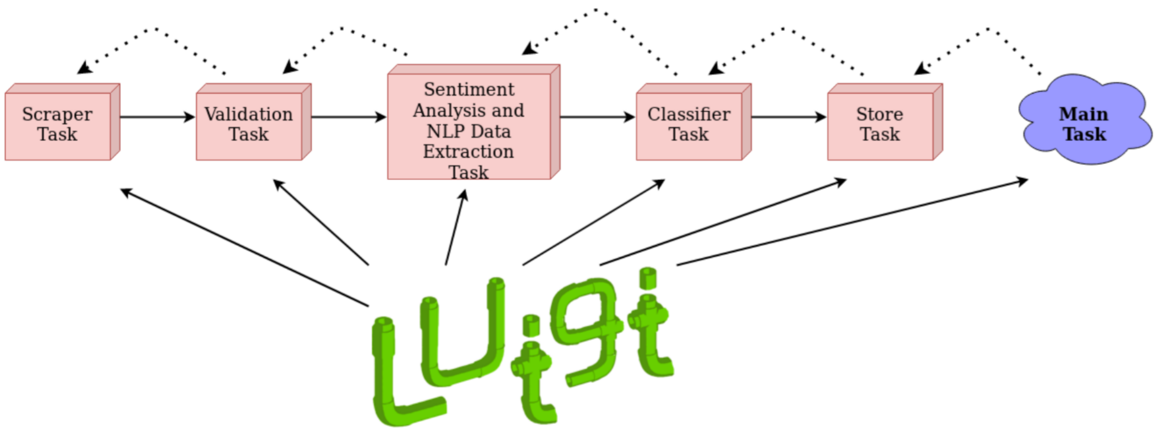


Figure 3.7: Pipeline Architecture

As it can be seen, the system pipeline consists on 6 tasks, being the last one the Main Task. Each task executes its corresponding processes, which are in call order (from end to beginning):

- **Main Task:** Initializes a logger file, in which the results of the different tasks will be stored, and starts the task calls.
- **Store Task:** Uploads analyzed files to the predefined ElasticSearch index.
- **Classifier Task:** Predicts the category of each text file and it annotates each file with its topic.
- **Sentiment Analysis and NLP Data Extraction Task:** It makes a call for each text to the Stanford CoreNLP server and saves the texts with the corresponding annotations.
- **Validation Task:** Executes for each text the Validation Process.
- **Scraper Task:** Run the Scraper Process and the Cleaning Process and save the cleaned files in text files.

Luigi also offers a web interface available on port 8082. In this interface, the state of the processes can be checked, as well as the relations between the different tasks. This interface can be seen in Figure 3.8.

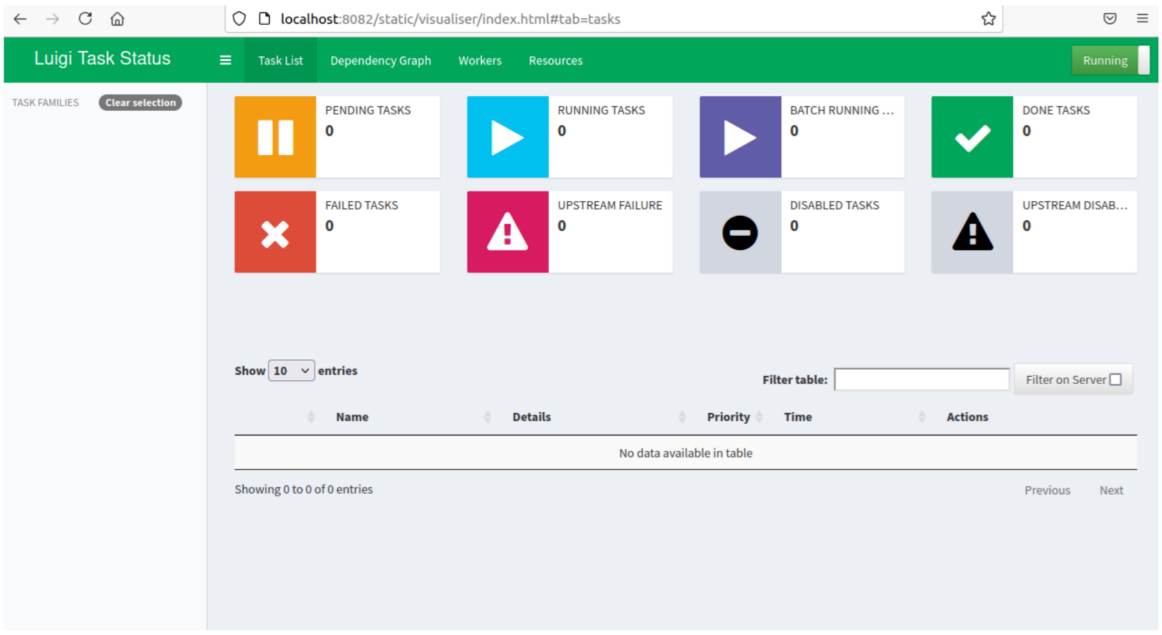


Figure 3.8: Luigi Interface

3.7 Visualization

Once the analysis of the data is finished, this analysis can be seen in the visualization module or Dashboard. This Dashboard is composed of Polymer web components which are graphs in which data can be observed and analyzed in an easy and comfortable way. Data is shown separately or in an aggregated way, depending on the specific analysis. These aggregations are provided by the JavaScript ElasticSearch engine and they allow to combine data or to show several fields at the same time. The Dashboard also provides filters to make the analysis easier and allowing to search for certain queries.

As stated before, the web components follow the Polymer interface and they are based on the D3.js or other common JavaScript libraries. The main web page is also a web component which loads other web components inside it to which it sends the data with new filters or aggregations so that the entire visualization system changes in real time according to the actions of the users. The next are every web component used in the visualization module.

- **Number-chart:** The number chart is used to see how many posts are retrieved and how many posts are shown. For instance, if a filter is applied to the data, this component will show how many posts pass through this filter. Figure 3.9 shows an example of the number chart.

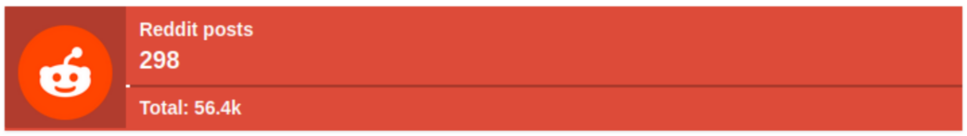
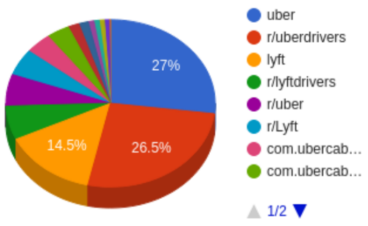
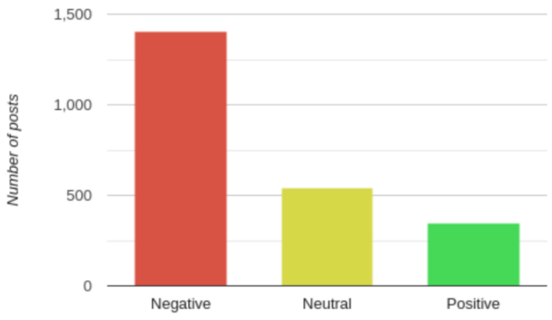


Figure 3.9: Number Chart Example

- **Google-chart:** The Google chart is a web component which allows to represent different graphs such as pie charts, bar graphs, gauges, or simple plot graphs, among others. In this project, different Google charts have been used. The first one is a pie chart which is used to observe how many posts are from each source. In addition, a bar chart has been used to analyze the sentiment distribution over the texts. These examples can be seen in Figure 3.10.



(a) Google Pie Chart Example



(b) Google Bar Chart Example

Figure 3.10: Google Pie Chart and Google Bar Chart Example

Moreover, other Google charts have been used in the visualization module. Specifically, another bar chart has been used for entity analysis and a plot chart has been used to see the dates on which the posts were collected. Figure 3.11 shows the latter.

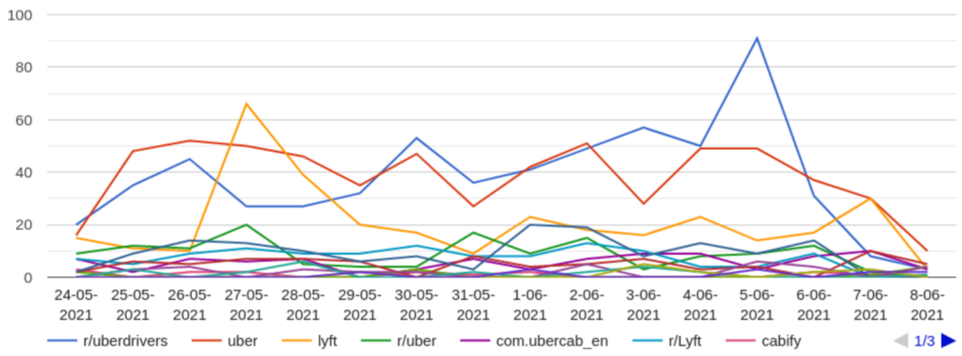


Figure 3.11: Google Chart for Dates Example

- **Radar-chart:** The radar chart has been used to show the distribution of the topics for each collected text. This chart allows to select a topic to see the statistics of texts that belong to this topic. Figure 3.12 shows this web component.

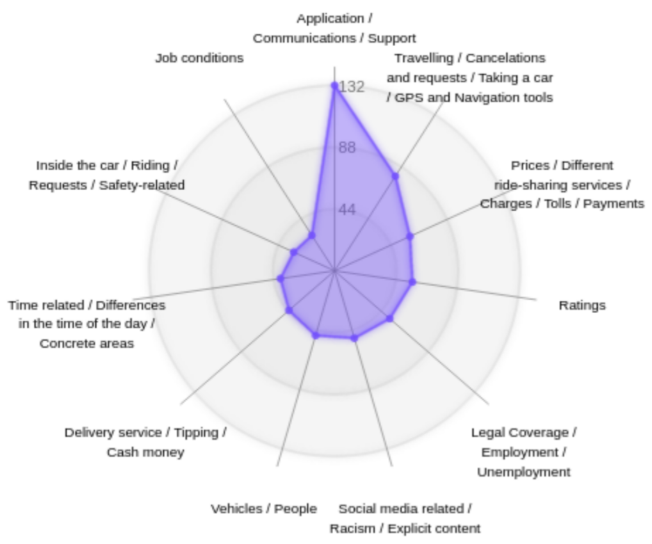


Figure 3.12: Radar Chart Example

- **Entity-chart:** The entity chart is another web component which shows the most common entities in the analyzed texts. As there are so many entities, for reasons of visibility on the Dashboard website, only the six entities with the greatest weight in the texts were included in this component. The contribution of the other entities can be seen in another Google chart available in the Dashboard.

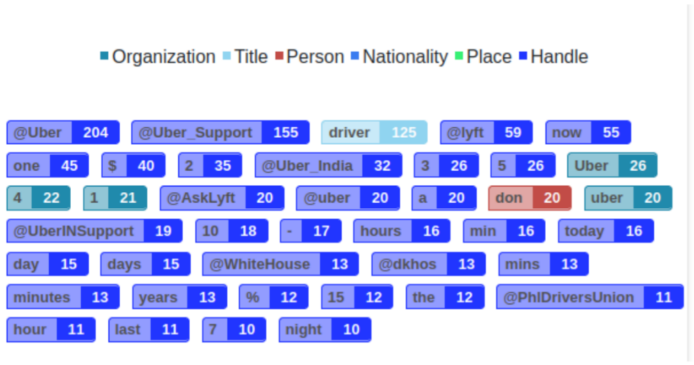


Figure 3.13: Entity Chart Example

- **Heatmap-chart:** The heatmap chart shows the *PLACES_AND_LOCATIONS* entities marking on a map those areas that have been marked as entities.

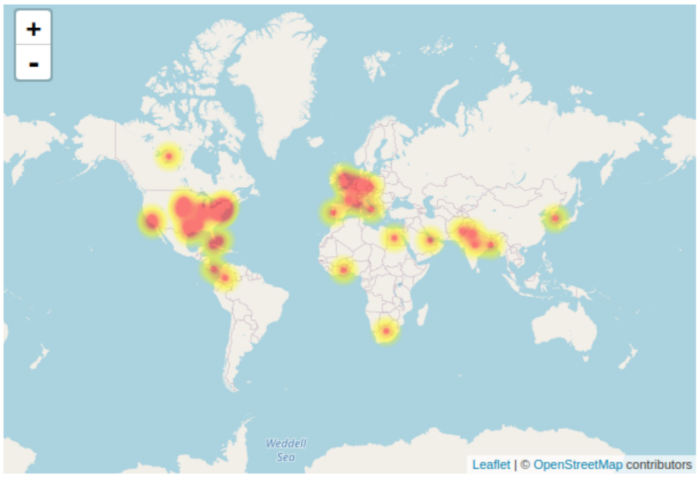
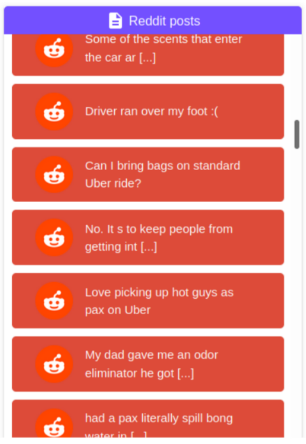


Figure 3.14: Heatmap Example

- **Posts-chart:** The posts chart shows every post from a given source. It also allows to select them and to observe, in addition to the full text, the analysis that has been done on that news. Thus, it shows entities, topics, and feelings in a fast and comfortable way. It also allows to access the website from which the post was extracted. Figure 3.15 shows this chart in two images. The first image is the general chart, where a post can be selected, and the second is the window that opens when clicking on a post.



Reddit posts

Some of the scents that enter the car ar [...]

Driver ran over my foot :(

Can I bring bags on standard Uber ride?

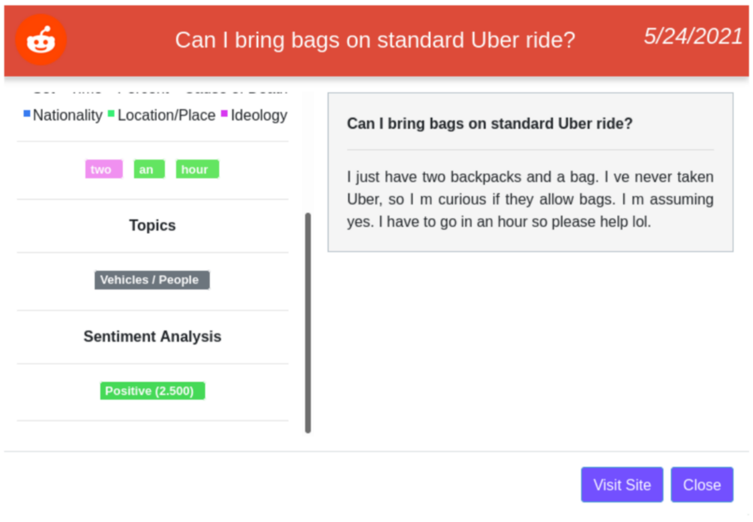
No. It s to keep people from getting int [...]

Love picking up hot guys as pax on Uber

My dad gave me an odor eliminator he got [...]

had a pax literally spill bong

(a) General posts chart



Can I bring bags on standard Uber ride? 5/24/2021

🟦Nationality 🟩Location/Place 🟪Ideology

two an hour

Topics

Vehicles / People

Sentiment Analysis

Positive (2.500)

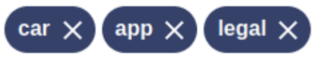
Can I bring bags on standard Uber ride?

I just have two backpacks and a bag. I ve never taken Uber, so I m curious if they allow bags. I m assuming yes. I have to go in an hour so please help lol.

(b) Posts chart when clicking on a post

Figure 3.15: Post Chart Example

- **Filters-chart:** The filters chart consists of a search bar on which the content of the displayed posts can be filtered by specific words. This filter allows to select as many words as required, in addition to being able to remove each word one by one following the desired order.



(a) Filtered words

(b) Search bar

Figure 3.16: Filters Chart Example

CHAPTER 4

Use Case - The Ride-Hailing Domain

4.1 Introduction

The developed system can be applied to several domains in which there is an interest in analyzing and observing what is happening in social media regarding that domain. One of the parties interested in this type of system would be companies that want to analyze other competing companies based on the opinions of their users in social media. The identification of topics and the way in which users talk in social media can become very useful information for companies to improve their services and products, to solve problems and to keep their users satisfied.

A market that can benefit from the use of this system is the Ride-Hailing market. There are several Ride-Hailing companies, which the most known companies are companies such as Uber, Didi, or Lyft. Analyzing the market share over the world of these companies[1], in 2019 the most important company was Uber, followed by Didi, with a similar market share between them (both over the 30%). Then companies such as Lyft, Ola, or Cabify appear, this last with a market share of a 1.46% in 2019.

Moreover, the Ride-Hailing market has been growing each year. This market[2] was valued at $36,450.0 million in 2017 and is projected to reach $126,521.2 million by 2025.

[1]https://www.statista.com/statistics/1156066/leading-ride-hailing-operators-worldwide-by-market-share/

[2]https://www.alliedmarketresearch.com/ride-hailing-service-market

This implies that the Ride-Hailing market is a big market in which companies need to be competitive with other companies for growing.

On the other hand, Ride-Hailing companies rely heavily on their apps and they are a fundamental part of their business model, so their users have easy access to social media where they can share their experiences, complaints or opinions.

The developed system has been designed to collect data from Uber, Lyft, and Cabify social media-related posts. This decision has been based on the availability of these companies when retrieving data from social media. For instance, in the case of Didi, such as this is a Chinese company, the analyzed social media sources did not provide certain information about this company, and for this reason the Didi analysis is not implemented on the system. On the other hand, although the decision of analyzing Uber and Lyft posts has been taken due to the importance of these companies and the availability of their posts on social media, the Cabify implementation has been decided because of its social media availability and for the fact that it is a Spanish company, most of whose posts are written in Spanish and it is interesting to see if the analyses carried out can be extrapolated to companies in other countries and in other languages. This analysis will be explained in Section 6.

This use case has been centered on Uber, which is one of the most popular Ride-Hailing platforms around the world. The information has been extracted from Reddit, in which several subreddits talk about this company in different ways.

The developed system can be a very useful platform for Cabify, as it would allow to observe in real time the complaints of its users or the opinions about different applications and thus be able to improve its services to increase the satisfaction of its users. In addition, it would be able to compare the opinions of its users with those of users of other platforms to ensure the improvement of its services based on improving those of its competitors.

4.2 Collecting and Inspecting Data

As explained before, different subreddits have been analyzed to extract useful information with the aim of developing the system. In particular, two subreddits were analyzed: the ***r/uber*** subreddit, created on October 29, 2011 and with about 22,000 members (as of May 2021) and the ***r/uberdrivers*** subreddit, created on November 5, 2013 with about 181,000 members. The r/uber subreddit is a more generic subreddit, where riders and drivers tell their experiences, and the r/uberdrivers subreddit is a more specific subreddit, which focus on Uber drivers experiences. These are public forums and have not any influence from Uber.

To have as many posts as possible, all texts were extracted from the time of the creation of the subreddits up to the specific time when they started to be collected. Figure 4.1 shows the amount of posts collected by date. In that figure, it can be appreciated that both subreddits have been increasing their popularity over time. In addition, there are more posts in the r/uberdrivers subreddit.

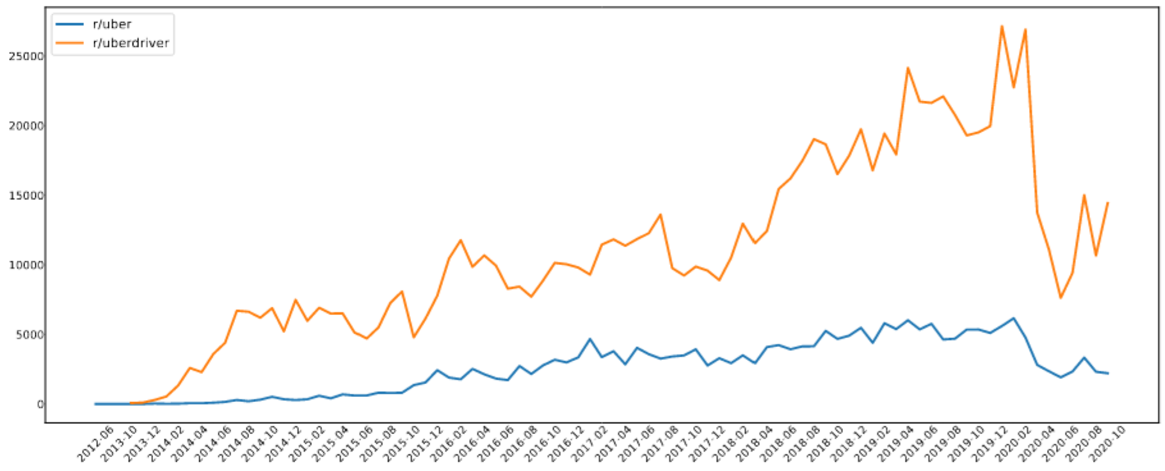


Figure 4.1: r/uber and r/uberdrivers over time

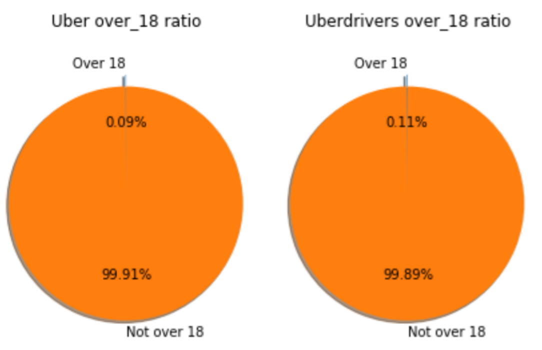
The Pushshift API, with which Reddit data has been collected in this project, also provides other fields such as the author of the post, its id, and other interesting data. Moreover, each type of post has different fields depending on whether it is a submission or a comment.

Unfortunately, almost every field is not complete and has a lot of empty data. Inspecting the collected data, some of the fields that can give useful information about the data, that are almost complete and that can be related to the texts are the *author* field, the *score* field, the *stickied* field or the *over_18* field. Almost all other data are incomplete, so they have not been taken into account. The explanation of every field can be found on the Pushshift official site[3].

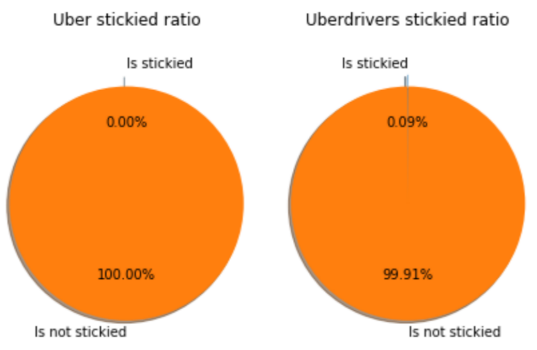
The main problem of this limitation of these fields is that they do not provide useful information since they are are not very relevant. To give an example, Figure 4.2 show the over_18 and the stickied fields in r/uber and r/uberdrivers.

As it can be seen in Figure 4.2, the percentage of texts targeting users over 18 years of age of texts marked as stickied is minimal and similar in both subreddits. This fact, in addition to the fact that most of the data is incomplete, is the reason that only data related to the textual content of the posts has been used for the analysis in this project. In addition, as most of the processes that have been carried out are related to Natural Language Processing, and as one of the objectives is to carry the analysis from some social media to others using the same type of analysis, other fields such as score have been rejected to perform these analyses. Even so, many of these fields are shown and can be observed in

[3]https://pushshift.io/api-parameters/



(a) The *over_18* field in r/uber and r/uberdrivers

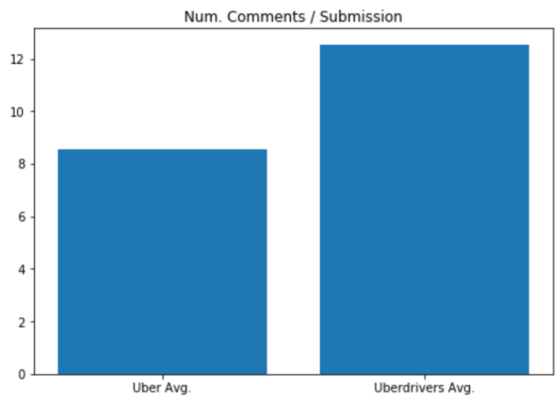


(b) The *stickied* field in r/uber and r/uberdrivers

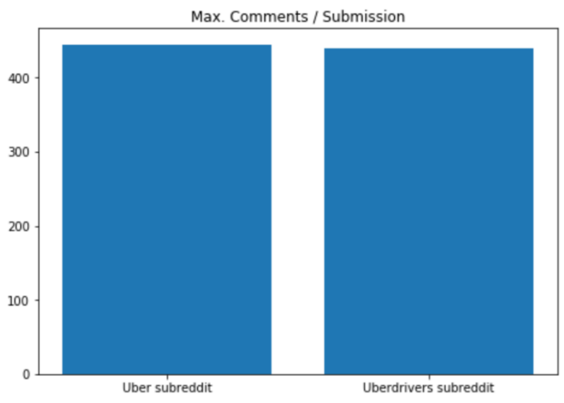
Figure 4.2: Analyzing some fields in r/uber and r/uberdrivers retrieved data

the visualization part of the complete system.

As for the analysis of the texts in this social media platform, some preliminary analyses have been carried out before starting the development. For example, the relationship between comments and submissions can be seen in Figure 4.3.



(a) Comments per Submission in r/uber and r/uberdrivers



(b) Maximum number of Comments per Submission in r/uber and r/uberdrivers

Figure 4.3: Relationship between comments and submissions in r/uber and r/uberdrivers

As it can be seen, the average number of comments per submission is similar in both subreddits, but slightly higher in r/uberdrivers, indicating more activity in this subreddit. In addition, the maximum number of comments per submission is similar in both subreddits too.

Another easy and useful analysis is to see what are the most common words in both subreddits. Figure 4.4 and Figure 4.5 show these most common words for each subreddit, where it can be appreciated that the most frequent words in both subreddits are similar. It should be noted that some contractions such as "*n't*" or "*'ve*" have been removed from these graphs to better appreciate the most common words.

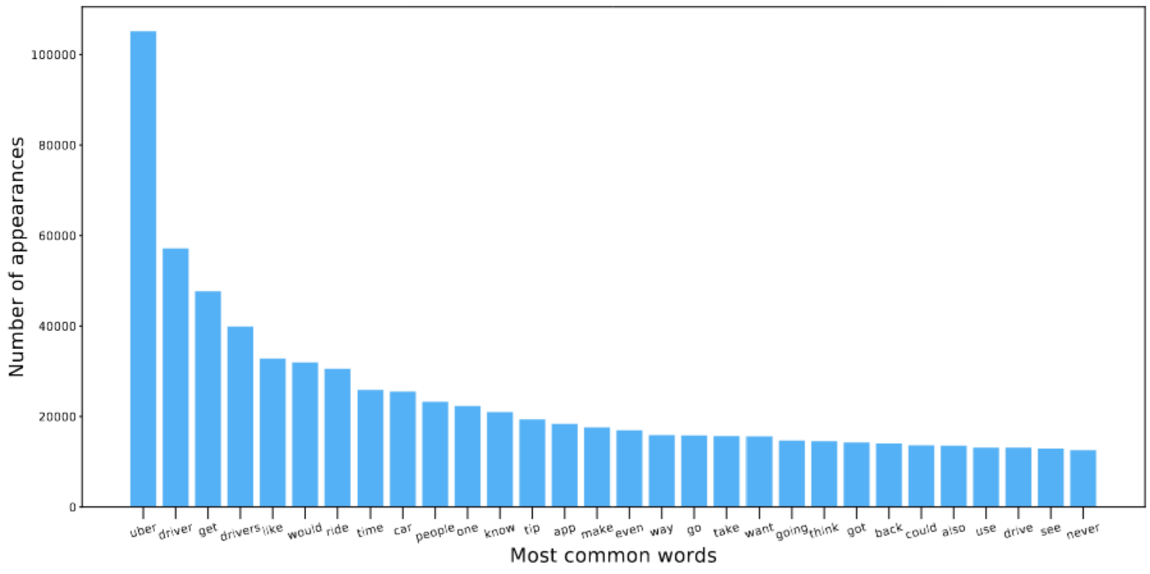


Figure 4.4: Most common words in r/uber

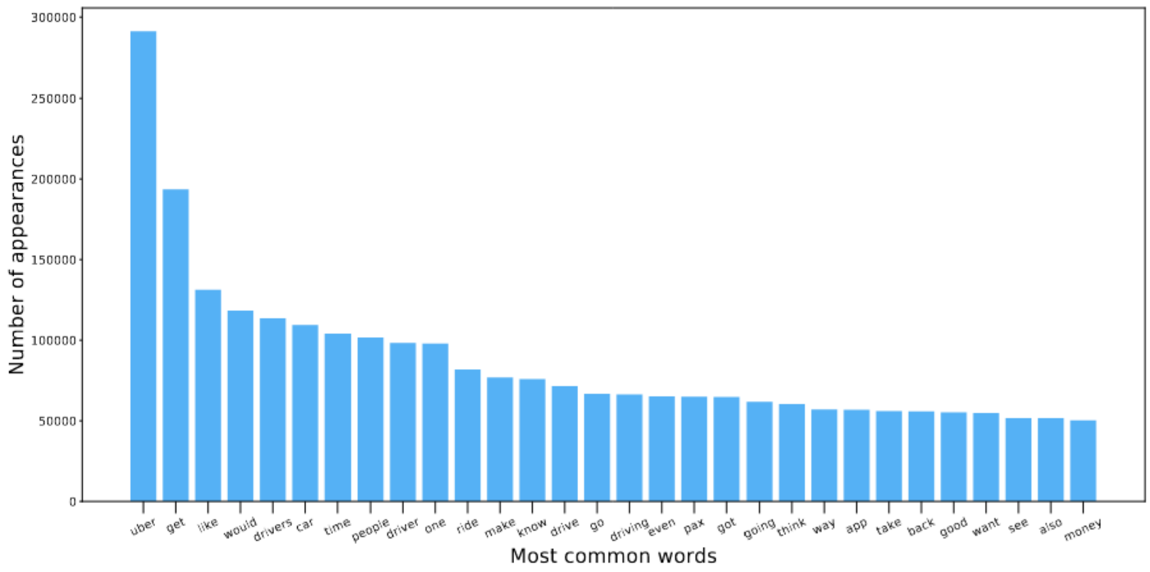


Figure 4.5: Most common words in r/uberdrivers

In both subreddits, the most common word is *uber*, followed by some common English words such as *get*, *like* or *would*. Moreover, words such as *driver/s*, *time*, *people* or *car* have a huge relevance. It can be seen that in r/uber the word *app* seems to be most important than in r/uberdrivers, although in this subreddit is important too. In r/uberdrivers, in contrast to r/uber, words such as *pax* or *money* appear, which indicates that in r/uberdrivers more posts about working conditions or the day-to-day life of the drivers are published.

This first glance at the data indicates a clear similarity in terms of content in both subreddits, although perhaps the topics they talk about are quite different due to the type of users that use these forums. In addition, it can be seen that the metadata associated with each post does not provide much information, so the analysis and the system developed will be based exclusively on the textual data of the posts.

4.3 Topic Modeling

Before starting to train the first models, each collected text is passed through the Cleaning Process explained in Section 3.2.2. This process already rejects some texts, such as empty texts (which may have been removed by the moderators of the subreddit) or texts written in another language, among other cases.

Once the texts are cleaned, the first analysis performed on the texts is Topic Modeling. As stated in Section 2.3.1, a LDA model is trained to extract topics from the corpus. Before training, the preprocessing pipeline explained in Section 3.4.1 is applied to the corpus. This preprocessing pipeline changes from one model to another to try to improve the models.

The performance of the models, as explained before, has been analyzed using mainly the Coherence Score and the visualization of the topics using the pyLDAvis tool. Every model was iterated over around 50 topics (in particular, from 2 to 49 topics) and the model with a number of topics with a higher Coherence Score is taken as the best model. Then, the model is plotted and the distribution of the topics in a two-dimensional plane is analyzed. Thus, mistakes and possible improvements can be taken into account in the next training process, where we attempt to improve the model until we arrive at a coherent and analyzable model.

The first model approach, the baseline model, was trained following a first approach of the final preprocessing pipeline. Some interesting facts about this model is that it was trained without stemming, removing only modal verbs, forming bigrams with the criterion of appearing a minimum of 20 times in the whole corpus, and without using the filter_extremes function when creating the dictionary. The maximum Coherence Score of this model was around 66.17% and the variation of this metric around the number of topics can be seen in Figure 4.6.

As it can be seen in Figure 4.6, the maximum values are obtained for topics between 11 and 15, being 11 topics the more coherent value. As explained before, to improve the model, topics are plotted with the pyLDAvis tool. The representation of the topics relationship can be appreciated in Figure 4.7.

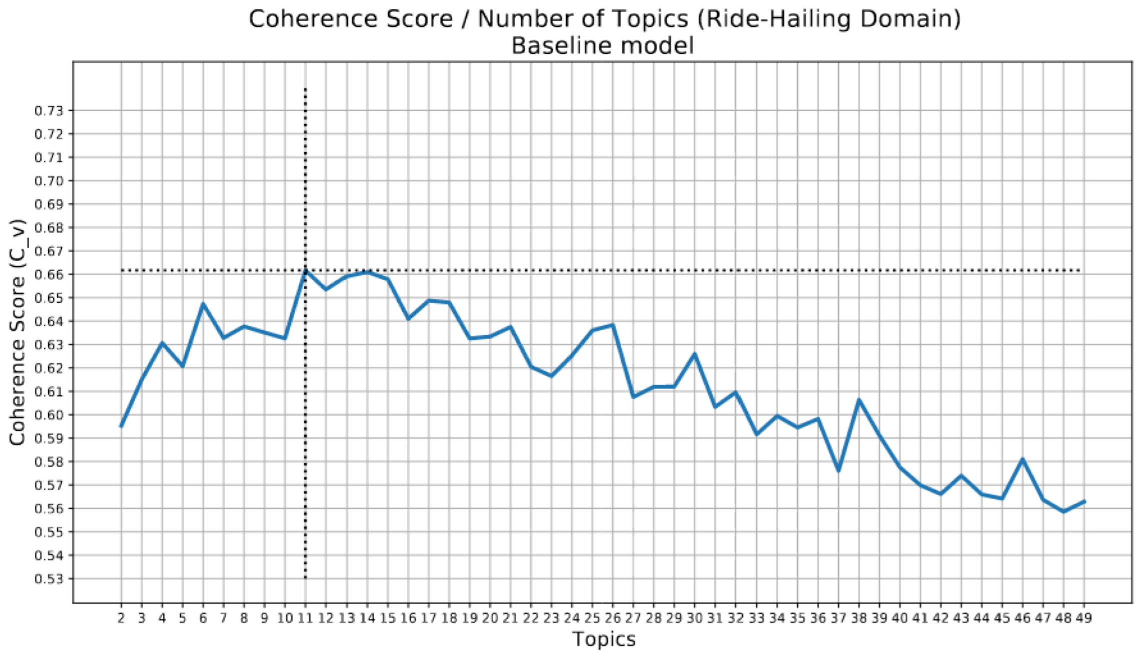


Figure 4.6: Baseline Model - Coherence Score through the Ride-Hailing domain

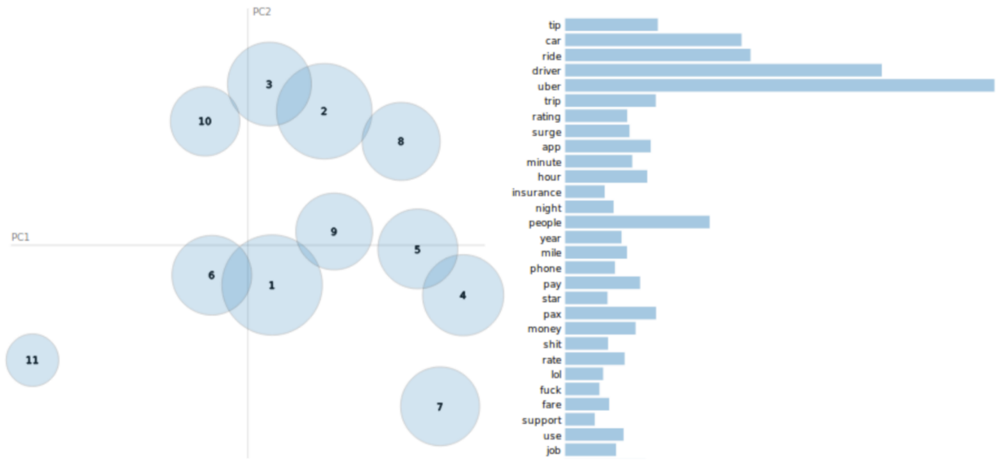


Figure 4.7: Ride-Hailing domain LDA baseline model - Visualization

As it can be seen, almost every topic shares words with other topics. This fact implies that most of them are not independent from each other, and they can be talking about the same thing although the model may identify them as different topics. The aim of the optimization of the LDA model is to achieve the maximum possible independence between these topics. This optimization has been reached refining the process once each model has been trained and analyzed.

Several ideas have been applied to optimize models, such as adding new processes (e.g., the stemming process), improving the cleaning rules, modifying the parameters of some processes (the minimum number of occurrences when doing bigrams or creating the dictionary, among others) and more. One of the most important ideas that has contributed the most to the optimization of the process has been the creation of word bags at the end of certain processes composed of tokens which will be rejected. These words are included based on the POS-Tagging (modal verbs, pronouns without meaning, common verbs with no special meaning, and more) but mainly in the observation: the least common and rarest words that do not provide much information have been observed, both based on the POS and directly observing the topics with the pyLDAvis tool. This tool, in addition to allowing to see the distribution of topics, provides functionalities to select specific words and, thus, it shows the distribution of this word through topics. If a word is common in most topics, it can be deleted or processed in another way to reach the independence between topics. Figure 4.8 shows an example of the weight of the word "*uber*" in the baseline model.

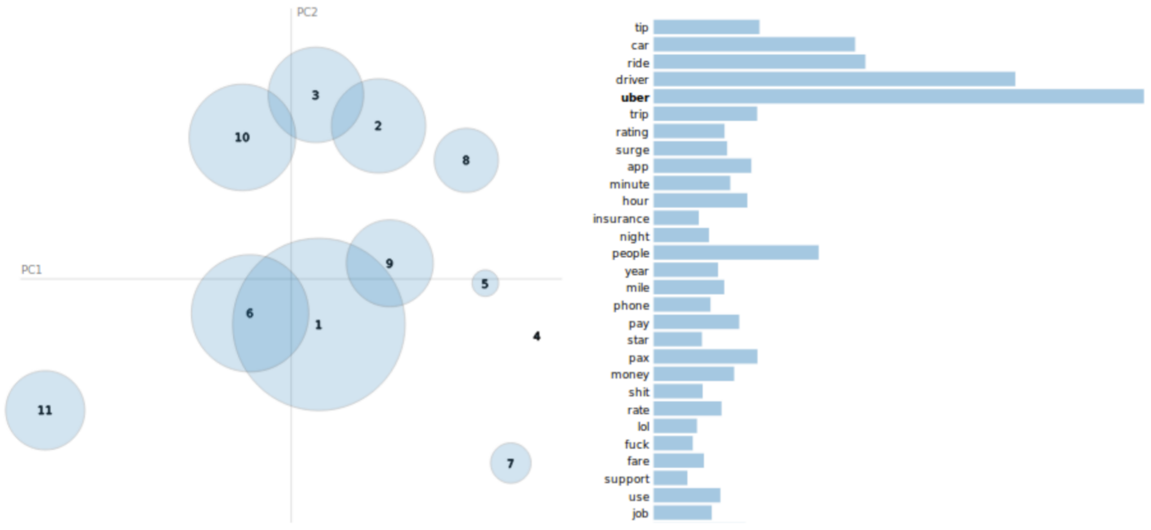


Figure 4.8: pyLDAvis word selection example

As shown in Figure 4.8, when selecting a token in the right panel, the bubbles representing each topic become bigger or smaller depending on the weight of that token in the topic. Thus, it can be appreciated that this word is present in almost every topic, and, consequently the model will identify this word in a wrong way. Therefore, this kind of words must be deleted or processed again to optimize the model.

Following this approach, 12 different preprocessing pipelines were applied to the corpus before training until one was found that the LDA models trained on this corpus identified the topics in the best possible way. It is important to note that the objective is not to find a model with a coherence result of 100%, but to find the one that best identifies the topics, based on the aforementioned Coherence Score - Number of topics - Human observation commitment.

The results of the analysis of the Coherence Score for each trained model are shown in Figure 4.9.

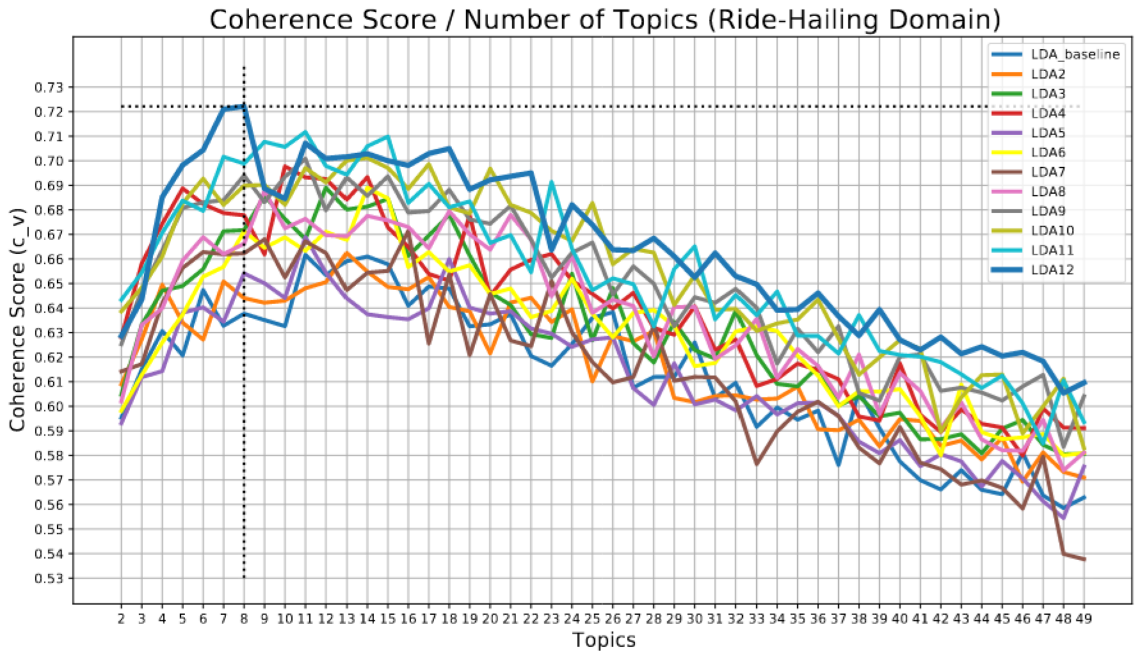


Figure 4.9: Coherence Score through the Ride-Hailing domain

It can be seen that the best model in terms of coherence is the one corresponding to preprocessing LDA12, which is the most elaborate preprocessing pipeline and which was reached after the analysis and observation of the rest of the developed models. This model has a coherence value of 72.21% when the model is trained searching for 8 topics, which implies an improvement of about 6% over the initial model.

Figure 4.9 shows that this pipeline is the best throughout the search for the different sets of topics, as well as being the one that achieves the highest coherence. As the hyperparameters α and β have not been changed, the next process is to try to improve the training process as much as possible. This optimization has been done using a brute-force search.

As this kind of search is costly both in terms of execution time and resources, only a few models were chosen for optimization. This choice was based on both Coherence Score and Perplexity values, which can be seen in Figure 4.10.

As can be seen in the Perplexity figure, for all preprocessing pipelines, the graph decreases steadily from topic 12 or 13 in all cases. This means that the model does not learn too much from that number of topics. Even so, the Coherence Score graph shows that the values do not start to decrease steadily until topic 18. Therefore, to reduce the computa-

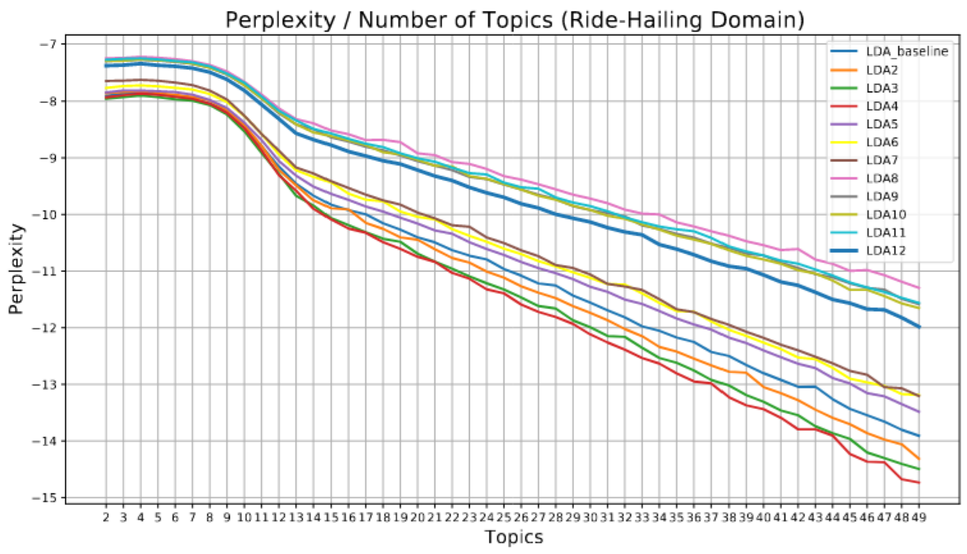


Figure 4.10: Perplexity through the Ride-Hailing Domain

tional load, it was decided to optimize the 4 best models within that range (from 2 to 18 topics). These models are the ones with 7, 8, 11, and 18 topics, with coherence values of 72.10%, 72.21%, 70.71% and 70.49%, respectively. Then different training processes were carried out, iterating over α (with possible values of 0.01, 0.31, 0.61, 0.91, "asymmetric" and "symmetric") and β (with possible values of 0.01, 0.31, 0.61, 0.91 and "symmetric").

The results of this optimization were analyzed in two ways: with Coherence Score and observation. The obtained coherence values and the visualization of the best models for each set of topics are shown in Table 4.1, Figure 4.11 and Figure 4.12.

Topics	**C_v**	α	β
7	72.41%	asymmetric	0.31
8	**72.82%**	**0.31**	**0.9**
11	72.70%	0.9	0.31
18	71.13%	0.9	0.61

Table 4.1: Optimizing LDA Hyperparameters (Ride-Hailing Domain)

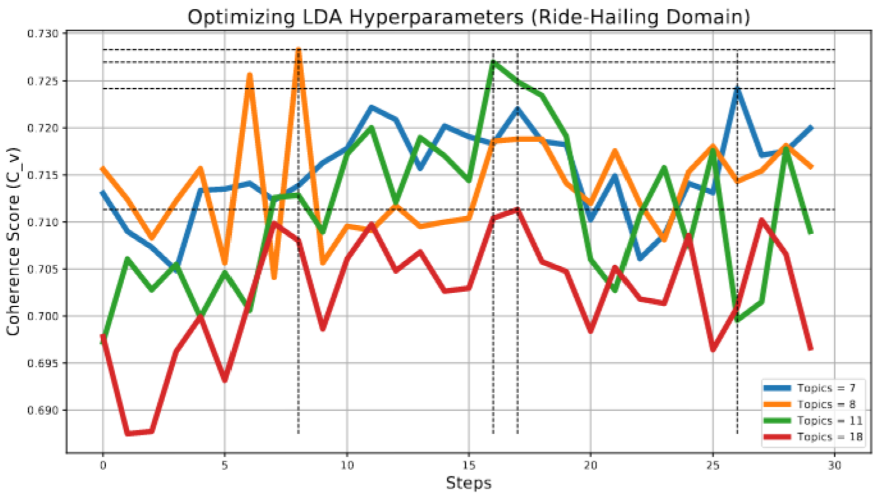
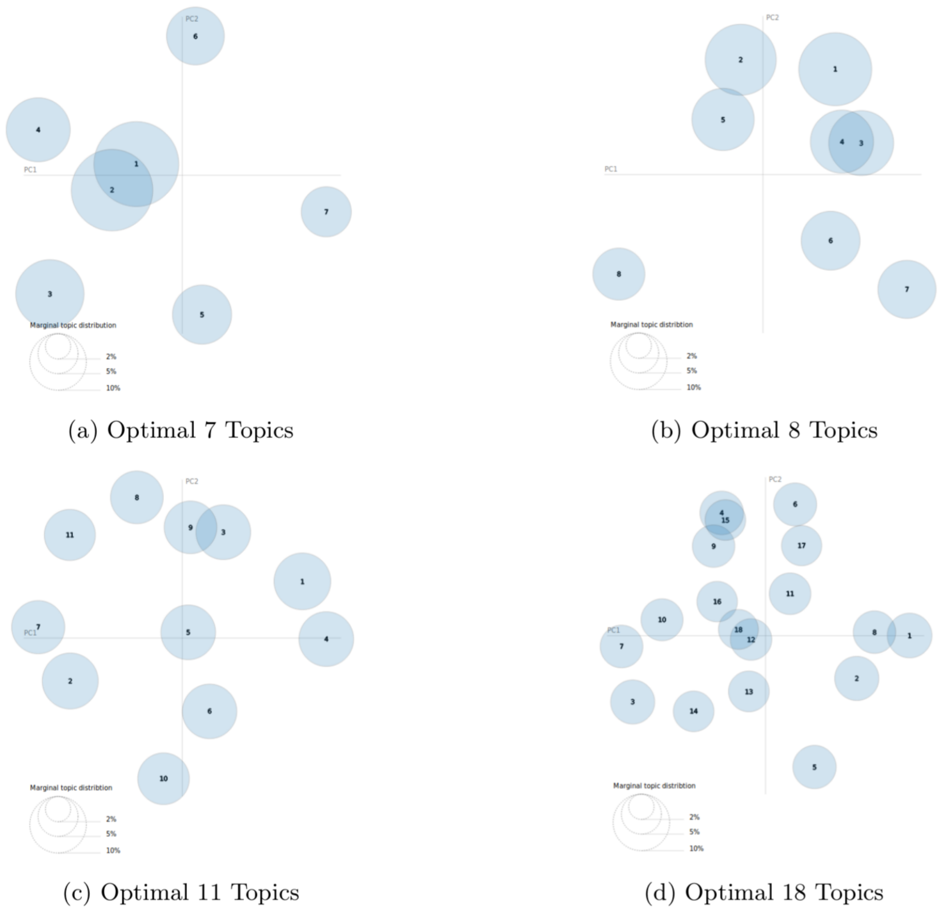


Figure 4.11: Optimizing LDA Hyperparameters (Ride-Hailing Domain)



(a) Optimal 7 Topics

(b) Optimal 8 Topics

(c) Optimal 11 Topics

(d) Optimal 18 Topics

Figure 4.12: Visualization of the best models in the Ride-Hailing domain

Regarding the obtained coherence values in Table 4.1 and Figure 4.11, the bests models seem to be the models with 8, 11, and 7 topics, in this order. On the other hand, the model with 18 topics is by far the worst of all, as was already evident from the Perplexity and Coherence Score graphs. However, it is necessary to see what these topics are like and how many of them are related before being able to choose the best model, since the best coherence values are very similar to each other.

Figure 4.12 shows the distribution of each model with the best hyperparameter combination in coherence terms. As it can be seen, the 7 topics model and the 11 topics model have only two topics related between them and the others seem to be independent. On the other hand, the 8 topics model and the 18 topics model have more than one topic related to others. Moreover, at a glance, it can be seen that the 18 topics model is the worst model.

As the selection of the model is based on the Coherence Score - Number of topics - Human observation commitment, the selected LDA model for this domain is the 11 topics model. This choice is due to the fact that it has the second highest coherence value, 72.70%, only 0.12 away from the best, it has only two topics related, the distribution on the plane is very good and it has as many topics as possible with a good distribution between them.

Once the LDA final model is selected, it is necessary to identify the topics because this algorithm does not name topics and it only calculates the weight of the words for each topic and finds an optimal distribution for them. To perform this task, the words that have more weight in each topic and those words that only appear in that topic were analyzed. It is important to analyze both groups of words since they do not necessarily have to be the same words and sometimes one group gives more information than the other. It is also important to know that the word groups extracted by LDA are preprocessed and therefore do not show their natural form since, among others, the processes of lemmatization, stemming or the creation of bigrams vary the representation of the tokens. For this reason, the name assignation process is not an easy process and the words shown below have been inferred from the extracted words.

After analysis, it has been decided to name the 11 extracted topics in this way:

- **Topic 1 - Inside the car / Riding / Requests / Safety-related:** Conversations, situations occurring inside a car. Important words: *talk*, *accept*, *request*, *ride*, *pool*, *music*, *conversation*, *safe*.

- **Topic 2 - Vehicles / People:** Situations that occur with specific people, such as situations with drunk people, the police or drugs. The characteristics of the car, such as cleanliness or any other characteristic of the vehicle, also fall under this topic. Important words: *car*, *seat*, *clean*, *water*, *door*, *man*, *girl*, *drunk*.

- **Topic 3 - Delivery service / Tipping / Cash money:** Topic related to food home delivery, courier or parcel shipping. It also includes the tipping topic. Important words: *ubereats*, *restaurant*, *food*, *cash*, *delivery*, *order*, *eat*, *tip*.

- **Topic 4 - Time-related / Differences in the time of the day / Concrete areas:** Time-related topic. Situations that occur during the day, night, a particular day, the weekend, a particular time of day, ... It also includes situations in concrete places, cities, and more. Important words: *hour*, *day*, *morning*, *weekend*, *drove*, *today*, *start*.

- **Topic 5 - Prices / Different ride-sharing services / Charges / Tolls / Payments:** Topic related to pricing and also to competing companies. Important words: *lyft*, *market*, *taxi*, *price*, *fare*, *uberx*, *cab*.

- **Topic 6 - Travelling / Cancelations / Taking a car / GPS and Navigation Tools:** Related to the trips, the type of trip, its duration, if a driver was late or took a long time, if he took you on a long trip or on the contrary took a short time, if he arrived on time, ... Important words: *cancel*, *trip*, *traffic*, *waiting*, *destination*, *pickup*, *location*.

- **Topic 7 - Social media related / Racism / Explicit content:** Related to the language used in social media. Important words: *post*, *lol*, *shit*, *troll*, *sub*, *comment*, *idiot*.

- **Topic 8 - Job conditions:** Working conditions, such as how much the company pays, how much money is earned, how much is paid per kilometer, and more. Important words: *money*, *tax*, *gas*, *maintenance*, *income*, *wage*, *mileage*.

- **Topic 9 - Legal Coverage / Employment / Unemployment:** Legal issues, such as robberies, kidnappings, murders, racist issues, questions about laws in certain places, accidents, ... It also includes employment-related texts. Important words: *insurance*, *state*, *law*, *employee*, *legal*, *claim*, *worker*.

- **Topic 10 - Ratings:** Driver, application or general scores. Important words: *driver*, *rate*, *star*, *experience*, *reason*, *system*, *matter*.

- **Topic 11 - Application / Communications / Support:** Related to the performance of the app, of the phone... A technological topic (beta versions, new versions, if something fails in the app, if something else fails because of the app, battery consumption, ...). Important words: *app*, *phone*, *support*, *report*, *account*, *information*, *update*.

4.4 Validation Process, Sentiment Analysis and NER extraction

Once the LDA model is finally generated, it can be used in two tasks: the Validation Process and the Classification Process. The first one, as related in Section 3.2.3, uses a lexicon which is directly extracted from the most common words that define a topic, in addition to other processes discussed above. This process will allow to analyze only those texts which are related to the topics and to reject those texts which do not add information or are not talking about the extracted topics.

The results of the Validation Process over this domain is shown in Table 4.2.

Uber Reddit Data Breakdown					
Subreddit	**Endpoint**	**Text Type**	**Scraped Texts**	**Rejected Texts**	**Rejected Texts (%)**
r/uber	**Comments**	Body	203,825	23,358	11.460
	Submissions	Body	22,850	10,932	47.607
		Title	22,963	2,464	10.730
	Total	**-**	**249,638**	**36,754**	**12.833**
r/uberdrivers	**Comments**	Body	877,013	102,575	11.700
	Submissions	Body	67,753	29,313	43.072
		Title	68,056	8,185	12.027
	Total	**-**	**1,012,822**	**140,073**	**13.830**
			1,262,460	**176,827**	**14.007**

Table 4.2: Analyzing Uber-Related Reddit Data

In relation to rejected texts, it can be seen that the different types of text have similar relationships between the two subreddits. It highlights that comments and titles do not reject so many texts, but submission texts are rejected much more frequently, reaching a rejection rate of over 43% in both subreddits. This is due to there are so many submissions which contain videos or multimedia content and also contain information only in their titles.

The validated texts amount to 1,085,633 texts, containing short and long texts, which is fundamental in the LDA training phase and, above all, for extrapolating this information to other media, as will be explained in Section 6.

To analyze the opinion of the users of these subreddits, texts have been passed through the Sentiment Analysis and NER extraction Process. The results of this analysis are shown in Figure 4.13.

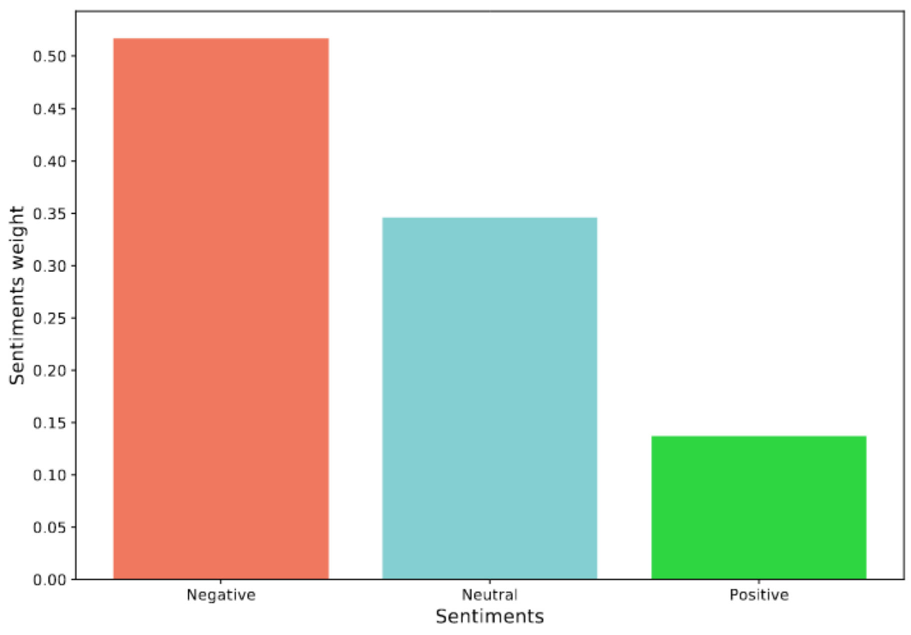


Figure 4.13: Sentiments distribution in r/uber and r/uberdrivers

As the Stanford CoreNLP annotates each sentence with a sentiment (with a value between 0 and 4), the sentiment analysis of the texts has been measured averaging the sentiment values of every sentence of each text. Thus, Table 4.3 shows how this average measure has been taken into account in this project.

Averaged value	**Sentiment**
<2	Negative
2	Neutral
>2	Positive

Table 4.3: Averaged sentiment values

The sentiment analysis over the Reddit texts shows a clear negative distribution, accounting for 51.680% of the total number of texts. Positive texts are the least frequent (13.720%) and are surpassed by texts labeled as neutral (34.600%), which do not show a specific sentiment and talk about general topics. The next step can be to analyze which kind of posts have a larger contribution to a specific sentiment value, as shown in Figure 4.14.

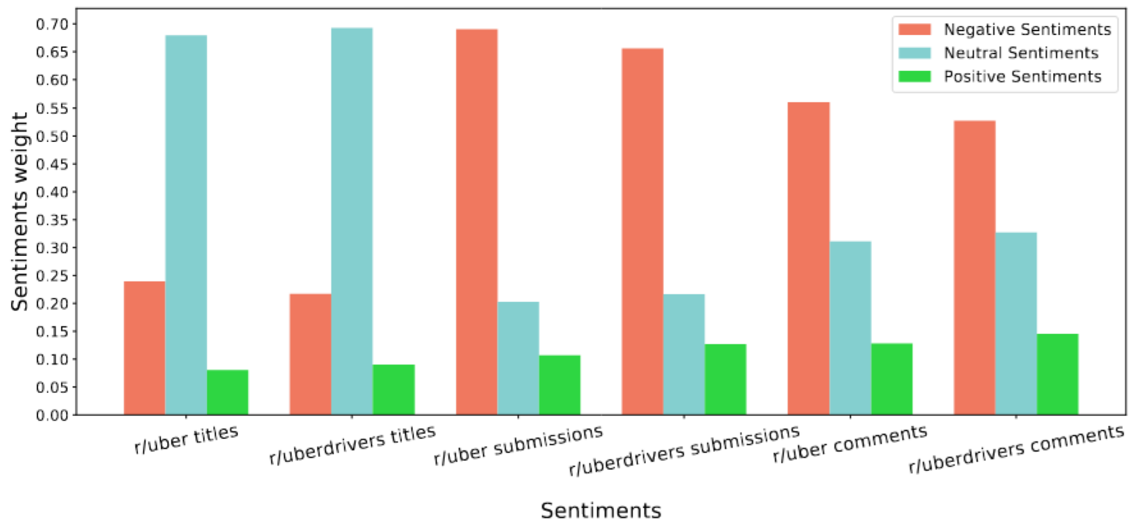


Figure 4.14: Sentiments distribution in r/uber and r/uberdrivers by text type

In the figure above, the analysis shows that titles are mainly neutral posts, rising to values of around 70% in both subreddits. Moreover, in both submission texts and comments, the negative texts are the ones that appear most often, followed by neutral texts. One of the most important results is that that positive texts are in the minority in all text types. In addition, sentiments distribution follows the same pattern in both subreddits, as can be seen also in Figure 4.15.

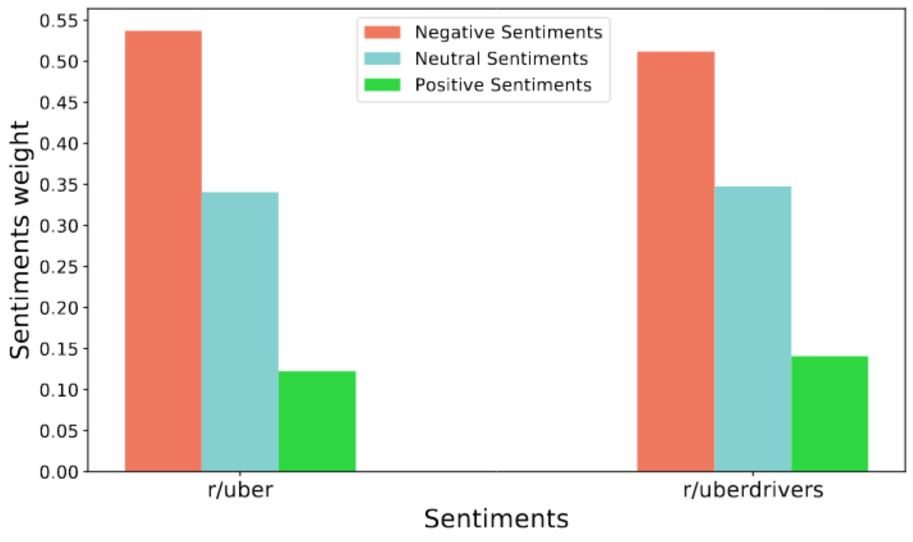


Figure 4.15: Sentiments distribution in r/uber and r/uberdrivers by subreddit

Figure 4.14 and Figure 4.15 show that, although both subreddits have similar sentiment values, the r/uber subreddit has a higher percentage of negative texts and fewer positive texts, indicating that posts on this subreddit are more negative (all types) than on r/uberdrivers. This could indicate that, being a more generic subreddit, users write more complaints than in r/uberdrivers.

On the other hand, entities that appear in some texts have also been annotated in the texts. To reduce the number of tags and to be able to analyze them better, some of them have been eliminated and others have been merged into one. For instance, *URL* and *EMAIL* have been removed because in the Cleaning Process almost every URL and text strings such as emails are removed from texts. In addition, *COUNTRY*, *STATE_OR_PROVINCE*, *CITY* and *LOCATION* have been merged into a tag called *LOCATIONS_AND_PLACES*. Also *NUMBER* and *ORDINAL* have been merged into *NUMBERS*. The distribution of these entities is shown in Figure 4.16.

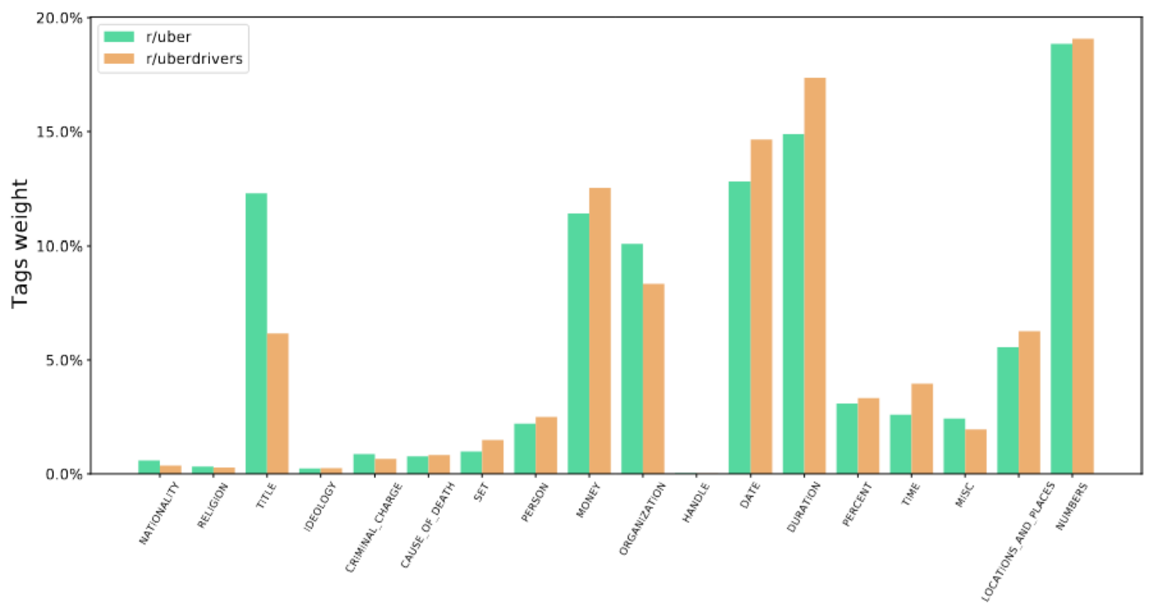


Figure 4.16: NER tags distribution in r/uber and r/uberdrivers

It can be appreciated that both subreddits have similar weights in every tag excepting the *TITLE* entity. This entity comes out more in r/uber as one of the words that usually identifies as this entity is "*driver*" (the tag *TITLE* represents job or study positions, such as *driver*, *student*, *CEO* and more). The most common entity in both subreddits is *NUMBERS*, followed by *DURATION*, *DATE*, *MONEY* and *ORGANIZATION*, which is more common in r/uber.

To analyze more this data, Figure 4.17 shows these results by subreddit and post type.

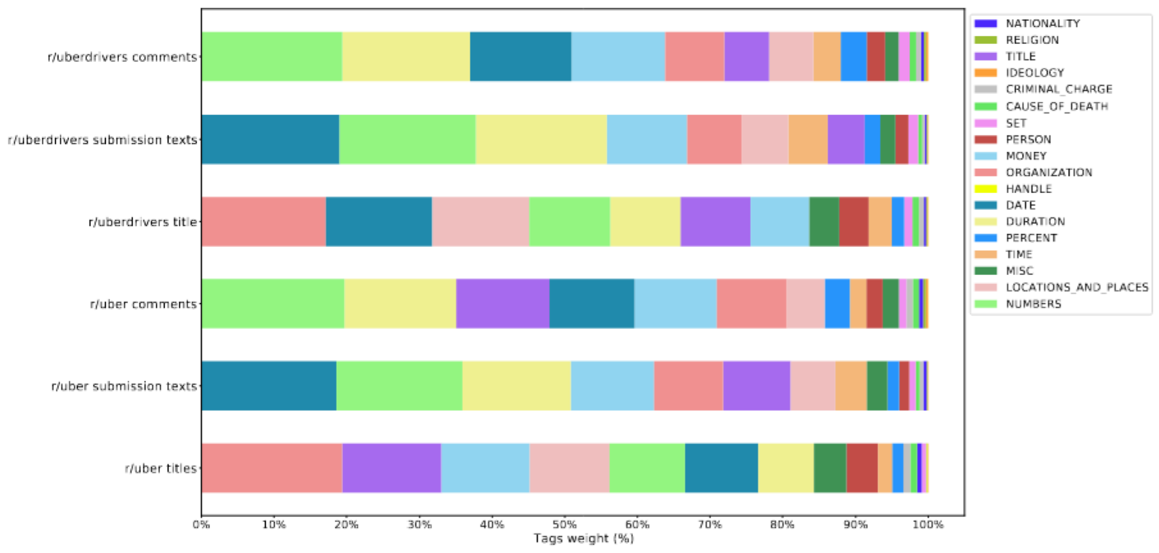


Figure 4.17: NER tags distribution in r/uber and r/uberdrivers by subreddit

According to these data, the entities weights are similar between the same post types even if they are from different subreddits. In titles, it highlights that *ORGANIZATION* entities are dominants. Then, in r/uberdrivers titles *DATE* and *LOCATIONS_AND_PLACES* are very representative, while in r/uber titles *TITLE* and *MONEY* are more important. Submission texts are similar in both subreddits. The most important tags are *DATE*, *NUMBERS*, *DURATION* and *MONEY*, with similar contributions in both subreddits. Finally, comments are similar too and the most important entities in both subreddits are *NUMBERS* and *DURATION*.

In conclusion, it seems that both subreddits are similar and different text types are similar between them too. It highlights that in r/uber, the *TITLE* entity appears with a high frequency. This is because in this subreddits users talk about drivers while talking about their travel experiences. In addition, there are several entities with lower contributions.

4.5 Classification Model

As the developed system is intended to operate in real time, new unseen data will be collected every day. This new data should be classified into the different topics that have been extracted. For this purpose, by making use of the Machine Learning techniques explained in Section 3.4, several classifiers have been built to achieve this objective in the most efficient way possible.

The algorithms were fed with features extracted from the text information of each post. This feature extraction was carried out in different ways, with the aim of obtaining the best possible scores. As related in Section 3.4.2, the final models were trained choosing Tf-idf and Word Embeddings features, extracted in two different ways using the FastText and the Word2Vec approaches.

To improve the models as much as possible, different feature representations have been done. In particular, Word2Vec and FastText models have been generated expecting vectors of 100, 300, and 500 dimensions. These models must be trained with as much data as possible.

As the objective of this project is to receive data from several sources and to be able to analyze them jointly, these models have been trained on data collected from other social media. Specifically, texts from Twitter and Google Play related to the Ride-Hailing domain have been added. For this use case, we are simply going to explain the performance of the embeddings trained on this set of texts in terms of Reddit text classification. The application of the proposed architecture to different data sources is explained in Section 6 (Transfer Learning).

The collected texts for training the embeddings models and their source is shown in Table 4.4.

Word Embeddings Data Collection			
Social Media Source	**Data Source**	**Language**	**Scraped Texts**
Reddit	**r/uber**	English	249,638
	r/uberdrivers	English	1,012,822
	r/Lyft	English	267,235
	r/lyftdrivers	English	373,904
	Total Posts	-	**1,903,599**
Twitter	**Uber**	English	6,223,730
	Lyft	English	1,699,520
	Cabify	Spanish	325,546
	Total Tweets	-	**8,248,796**
Google Play	**com.ubercab**	English	970,778
	com.ubercab	Spanish	551,415
	com.ubercab.driver	English	264,010
	com.ubercab.driver	Spanish	121,297
	me.lyft.android	English	74,342
	com.lyft.android.driver	English	27,139
	com.cabify.rider	Spanish	57,882
	com.cabify.driver	Spanish	21,039
	Total Reviews	-	**2,087,902**
			12,240,297

Table 4.4: Multi-Platform Collected Data

As it can be seen, different data sources have been collected within their respective social media sources. Firstly, in addition to the Uber related subreddits (r/uber and r/uberdrivers), which are the main analyzed data in this use case, data has been collected from two subreddits of the Lyft company, r/Lyft and r/lyftdrivers, which are analogous to those collected from Uber. As for Twitter, tweets discussing Uber, Lyft, and Cabify have been collected. The latter have been collected in Spanish since Cabify is a Spanish company. The parameters specified for the collection of tweets are based on hashtags and usernames, as shown in the following list.

- **Uber search keywords:** *#uber*, *@uber* and *@uber_support*.
- **Lyft search keywords:** *#lyft*, *@lyft* and *@asklyft*.
- **Cabify search keywords:** *#cabify* and *@cabify_espana*.

Lastly, some reviews of different Ride-Hailing applications for both riders and drivers have been collected too. Moreover, some of them are written in Spanish, such as the Cabify and Uber applications. Before training the models, these texts must be translated using the MarianMT transformer.

Figure 4.18 shows more information about the collected data, in particular, the dates in which each retrieved post was posted. As it can be seen, data is mainly retrieved between 2014 and 2020 and has a more or less regular variation over time. This figure can be seen in detail in the Appendix A.

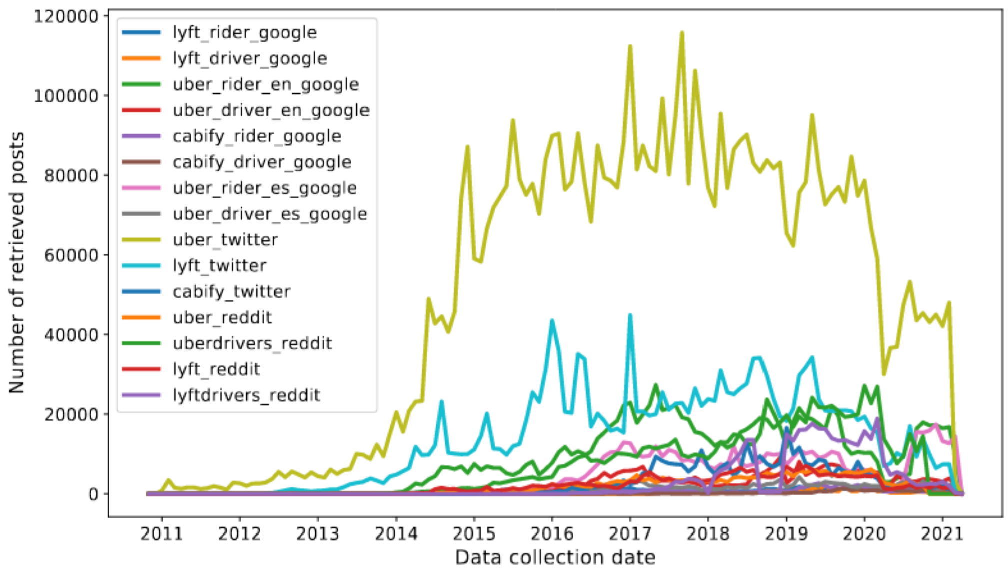


Figure 4.18: Total data retrieved per date

Once texts are collected and cleaned (every text has been passed through the Cleaning Process before training), the different versions of the Word Embeddings models are trained. To see the performance and get a first impression of the models, the 30 more similar words of the 10 most representative words of each topic are obtained for every model. The representation of these words in a two-dimensional space is achieved plotting a PCA (Principal Component Analysis) representation, which reduces multidimensional variables into a two or three dimensional space for observation tasks.

The representations of the Word2Vec and the FastText 100-dimensional models are shown in Figure 4.19 and Figure 4.20, respectively. The representations of the other models can be found in Appendix A, as well as the lexicon used for these representations.

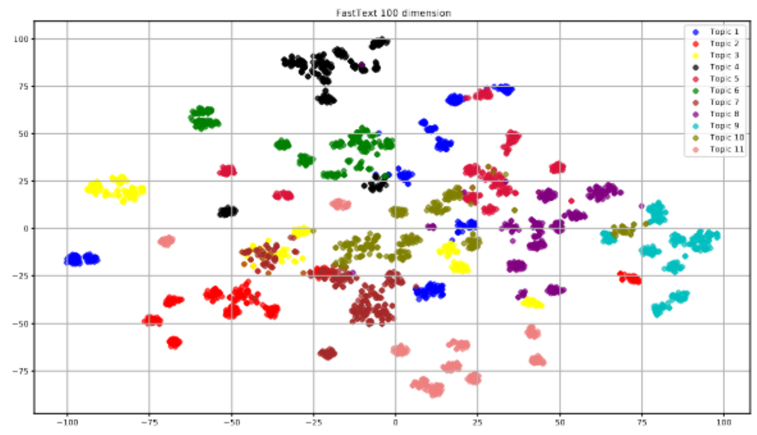


Figure 4.19: FastText 100-dimensional model PCA representation

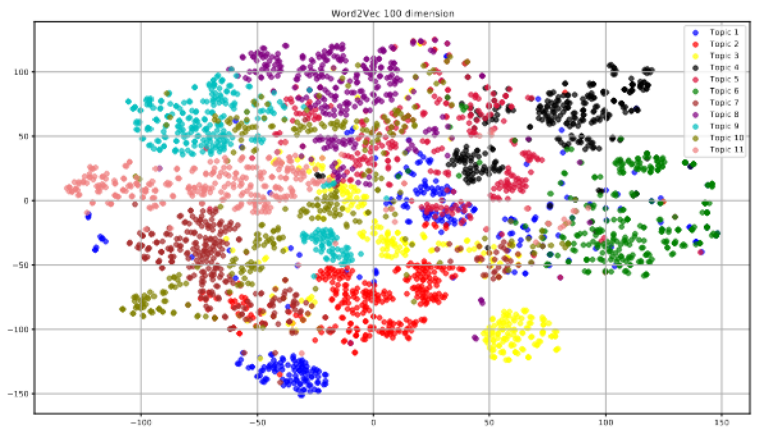


Figure 4.20: Word2Vec 100-dimensional model PCA representation

In the figures above, each color represents each topic, and each point is one of the 30 most similar words for each of the 10 most representative words of each topic. It can be seen that, in both models, the most similar words for each word are very close to the others belonging to the same topic and they form clusters, so it can be deduced that the words of each topic are distributed in the space in an orderly manner according to the topic to which they belong..

It is important to appreciate that, in the FastText model, the represented clusters are smaller and more concentrated than in the case of the Word2Vec model. This may mean that the Word2Vec model has more similarity between its different words and words are more scattered, although the clusters for each topic are perfectly visible. Even so, it is not possible to say at a glance which model is better, since the figures above represent the projection of 100-dimensional matrices onto only 2. Therefore, both models and others with a higher number of dimensions will be used in the classification process and compared.

As explained before, the Word Embeddings features were modeled with the arithmetic mean function of the embeddings of each word of a document and with SIMON. The SIMON approach needs a domain lexicon with the aim of relating document vectors to the specific words of the lexicon. As the classification task intends to classify documents into their respective topics, the lexicon must be composed by the most related words for each topic. These words were extracted based on their weight on each topic, easily accessible through the Gensim API and through the visualization of the LDA model, setting the λ parameter to lower values.

The main problem was that the more words were introduced into the lexicon, the higher the rate of increase of the classification performance. It is important to say that these words were words with essential weights in the topics, so that the performance continued to rise as more words were added.

SIMON adds a feature for each word in the lexicon. The value of this feature in each document will be the similarity of this word with the document. This implies that the more words in the lexicon, the more dimensions the training matrix will have. Having very large arrays in both dimensions implies high computational requirements. This means that the application of SIMON for this particular type of classification becomes unfeasible because of the enormous number of features that must be added to achieve valid results.

Table 4.5 shows the obtained results from the application of SIMON, depending on the number of words. The Word Embeddings model chosen for these tests was generated using the Word2Vec version, as well as the classification algorithm was the Logistic Regression. It should be noted that no further results are shown because from the last one shown in the table, the computational capacity and execution time of the algorithms became so enormous that it was practically impossible to make progress in this regard.

	Lexicon size	Accuracy	Precision	Recall	F-Score
Word2Vec 100 dim	110	58.762	60.636	58.762	59.315
	330	64.675	66.267	64.675	65.172
	573	66.318	67.849	66.318	66.773
	747	**66.858**	**68.413**	**66.858**	**67.319**

Table 4.5: SIMON results

As it can be seen, the performance of the classification task using SIMON increases with the lexicon size. It can also be seen that the more the number of words in the lexicon, the slower the performance grows. It could be possible to achieve better results if the lexicon was bigger, but this implies a very large computational effort, so the use of SIMON has been rejected as a feature extractor.

Thus, the averaged vectors and the n-grams extraction have been used as feature extractors. These features have been used with two different algorithms, the Logistic Regression and a Gradient Boosting algorithm. Different generated models were trained using a K-fold cross-validation approach, in which models were trained on 50,000 texts for each of the 11 topics. The validation set was composed by the rest of the texts.

The number of folds in each of the training processes varied, as well as the type of optimization. All models generated with Logistic Regression used 10 folds and a Grid Search process for optimization, except for the Tf-idf based model, which used 5 folds and the Halving Grid Search process due to its high computational load.

On the other hand, only the 100-dimensional models were optimized in the Gradient Boosting approach because of their extremely high computational cost. These models were tried to be optimized in different ways, concluding that the most efficient in terms of time and capacity was to use a genetic algorithm. To ensure convergence of the algorithm or at least completion at better optimized values, a population of 20 individuals containing different optimization parameters was used. These individuals mutated their parameters over 10 generations concluding with the calculation of the fitness function, which in this case was the calculation of the F-Score, for each individual. The results of this process can be seen in Appendix A.

When the same method was tested for models with 300 or 500 dimensions, the process was extremely time consuming and totally unfeasible. Therefore, models with more dimensions were calculated using the parameters extracted from the optimization of the 100-dimensional models. This process was also adjusted for the calculation of the Tf-idf model. In addition, the results of the Gradient Boosting models have been achieved using 5 folds in the training phase.

The results of the generated models applied to the r/uber and r/uberdrivers data are shown in Table 4.6 and Table 4.7.

	Accuracy	**Precision**	**Recall**	**F-Score**
Word2Vec 100 dim	73.229	74.500	73.229	73.627
Word2Vec 300 dim	77.415	78.399	77.415	77.699
Word2Vec 500 dim	79.462	80.322	79.462	79.704
FastText 100 dim	71.791	73.068	71.791	72.190
FastText 300 dim	75.884	76.966	75.884	76.199
FastText 500 dim	78.192	79.092	78.192	78.444
Bigrams - Tf-idf	**89.719**	**89.744**	**89.719**	**89.651**

Table 4.6: Logistic Regression Results in the Ride-Hailing Domain

	Accuracy	**Precision**	**Recall**	**F-Score**
Word2Vec 100 dim	75.522	76.431	75.522	75.807
Word2Vec 300 dim	77.670	78.496	77.670	77.917
Word2Vec 500 dim	**78.347**	**79.167**	**78.347**	**78.610**
FastText 100 dim	73.891	74.870	73.891	74.205
FastText 300 dim	75.700	76.604	75.700	75.983
FastText 500 dim	76.484	77.378	76.484	76.757
Bigrams - Tf-idf	75.119	76.159	75.119	75.316

Table 4.7: Gradient Boosting Results in the Ride-Hailing Domain

Firstly, one of the most important results is that, for the same number of dimensions, the FastText approaches are always worse than Word2Vec approaches. In addition, it can be appreciated the effects of the hyperparameter optimization. This effect implies that, in the models with 100 dimensions, the Gradient Boosting versions are superior to the Logistic Regression ones, while in the models with more dimensions, these models lower the score, being surpassed in some cases by the Logistic Regression models. In addition, the same is true in both Tf-idf approaches. However, just because they were not optimized does not mean that the results are bad. If it had been possible, it would have been impossible to reproduce these models because the computational time and capacity needed to train a model is an important factor and a direct cause of rejection prior to training, although in this case we wanted to show its consequences.

As for the best model, Logistic Regression with the Tf-idf representation stands out above all others. This result was to be expected, since LDA is based on words and their weights in the different documents that make up the corpus, without taking into account the semantic relationship between the different words. Still, the second model is logistic regression with the 100-dimension Word2Vec representation. The latter achieves an F-Score above 78%, which is a quite acceptable result. In any case, it would be necessary to see the result of the application of these models with different texts coming from other sources or that had not been used for the generation of the LDA model. All this will be discussed and analyzed in Section 6.

4.6 Results

Once the classification procedure is done and implemented, the data can be labeled with the topic to which they belong. In this use case, the Logistic Regression with the Tf-idf representation has been used to model the data and the results shown here have been extracted from the predictions of this model.

Firstly, the distribution of topics in both subreddits can be analyzed. Figure 4.21 shows this distribution for the case of r/uber.

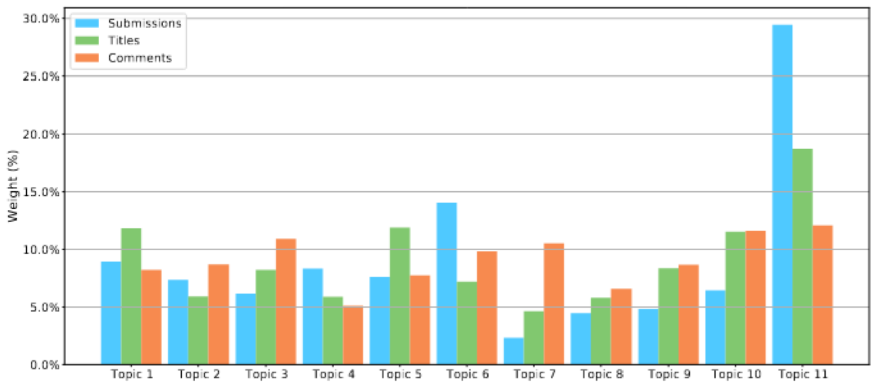


Figure 4.21: Prediction of topics in r/uber

The first thing that stands out is the bar corresponding to the submissions that belong to Topic 11, *Application / Communications / Support*, which clearly stands out above the rest. Moreover, the topic with the highest proportion of titles and comments is Topic 11. For this, it can be said that in r/uber, which is a general-purpose subreddit, the most discussed topic is Topic 11.

In r/uber, the rest of the topics are discussed in similar proportions. The least discussed topics are topic 7, *Job conditions* and topic 8, *Legal coverage / Employment / Unemployment*, although the first one is more represented in the comments.

On the other hand, Figure 4.22 shows the results of the same analysis on r/uberdrivers.

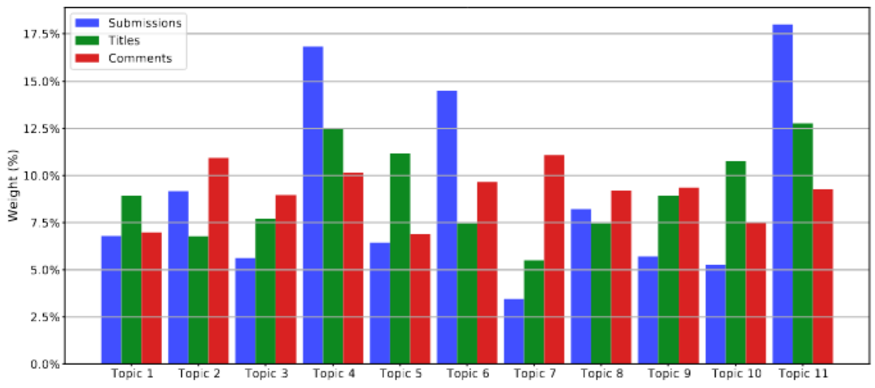


Figure 4.22: Prediction of topics in r/uberdrivers

Firstly, Figure 4.22 shows that Topic 11 has also a high representation on this subreddit, but not as much as in r/uber. It can be appreciated that Topic 4, *Time-related / Differences in the time of the day / Concrete areas*, has also a high representation. The rest of the topics are discussed in a similar way, except for Topic 6, *Travelling / Cancellations / Taking a car / GPS and Navigation Tools*, which in the case of submissions has quite a lot of weight. These topics are more related to the job of drivers, and it is a result in keeping with the fact that r/uberdrivers is a subreddit more focused on Uber drivers.

With the aim of comparing both subreddits more accurately, Figure 4.23 shows both subreddits topic representations.

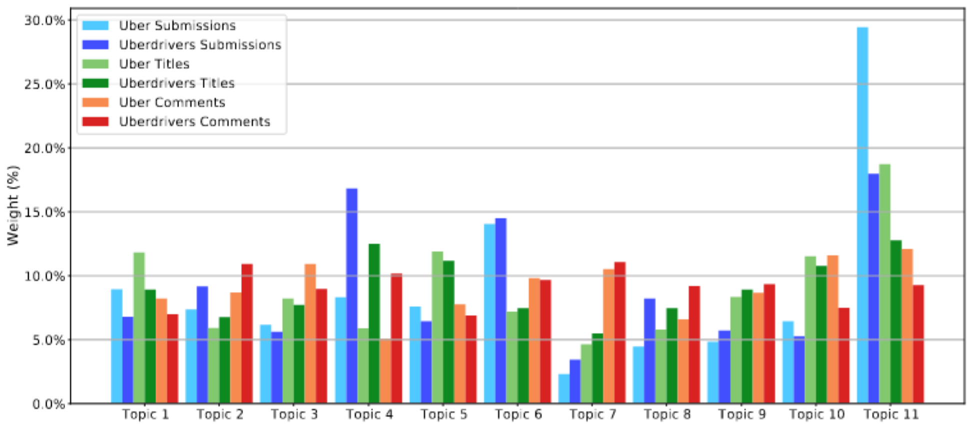


Figure 4.23: Comparison in predictions between r/ubers and r/uberdrivers

The above figure shows that, for each topic, for all text types (titles, submissions, and comments), the ratio between r/uber and r/uberdrivers is the same. That is, in virtually all topics, if one text type is superior in a subreddit, all text types are superior in that subreddit. This is, the predominant topics in r/uber are Topic 1, Topic 3, Topic 5, Topic 10, and Topic 11, which are strongly related to prices, money, and the app, this last one having an extremely importance in this subreddit. On the other hand, the predominant topics in r/uberdrivers are Topic 2, Topic 4, Topic 6, Topic 7, Topic 8, and Topic 9, which are more related with job conditions, legal terms or time-related situations. The latter comes from Topic 4, which, as it can be appreciated, carries a lot of weight in this subreddit. In addition, it seems that social media-related language is most used in r/uberdrivers because Topic 7 has more weight in this subreddit.

These results show that r/uberdrivers talks much more about issues that concern drivers, while r/uber talks more about situations or complaints from riders. Finally, Figure 4.24 shows in an aggregated way which are the most common topics in these subreddits. It can be seen that Topic 11, Topic 7, and Topic 6 are the most common topics, while Topic 1 and Topic 5 are the least.

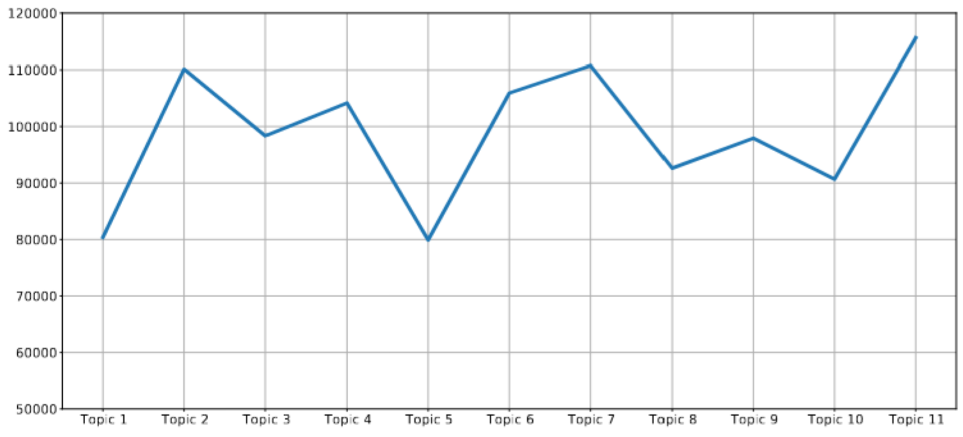


Figure 4.24: Aggregated topic prediction in Reddit

As for entity extraction, Figure 4.25 and Figure 4.26 show the relationship between topics and entities depending on the subreddit. The results are very similar between both subreddits as well as being closely related to the topics. Thus, in Topic 5 and Topic 8, which are related with prices and job conditions, the predominant entity is *MONEY*. Also Topic 3, which includes tipping-related texts, has a lot of *MONEY* entities. Another important observation is that in Topic 4, related to time, entities related to time stand out, such as *DURATION* and *DATE*.

On the other hand, it can be appreciated that many more *TITLE* entities appear in the r/uber subreddit. This is because this subreddit talks more about situations with drivers, while in r/uberdrivers are these drivers who talk about situations. Also, it can be seen that, in both subreddits, the *NUMBERS*, *DURATION* and *DATE* are the main entities in both subreddits. For more information about the NER analysis, some figures showing NER for each kind of text in each subreddit are available in Appendix A.

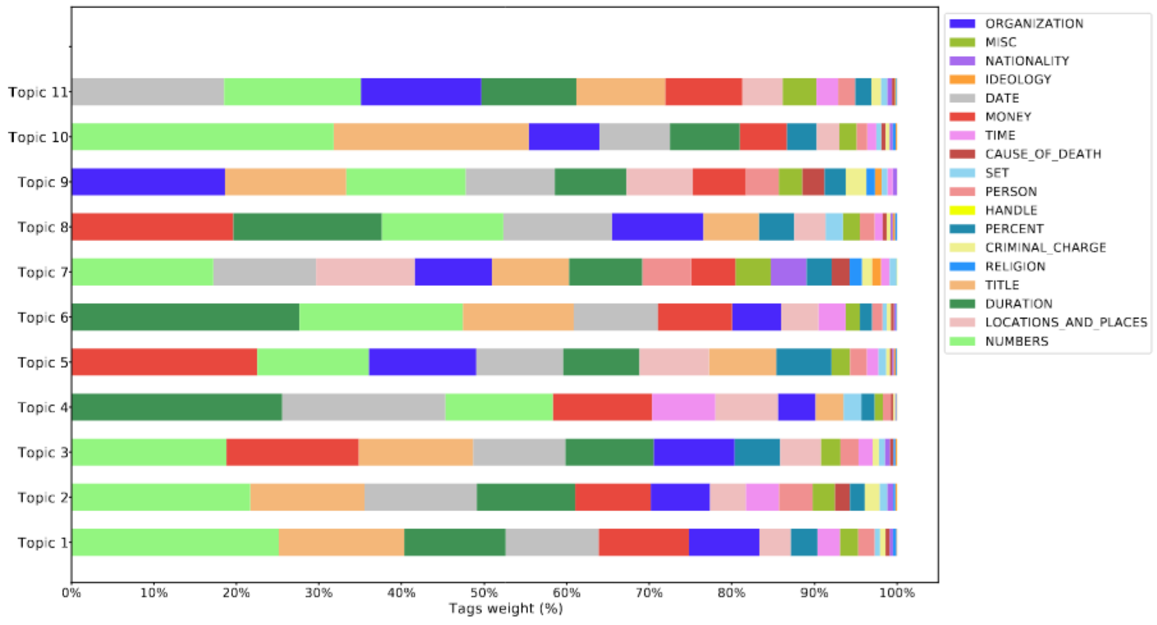


Figure 4.25: NER tags in r/uber by topic

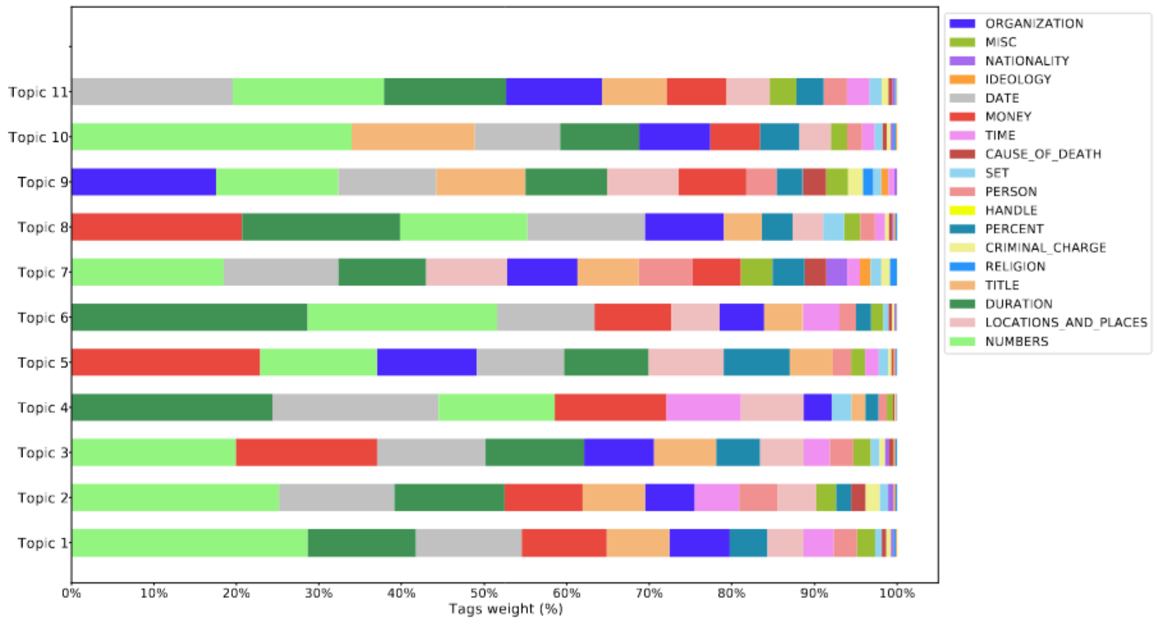


Figure 4.26: NER tags in r/uberdrivers

Finally, although it is already known that both subreddits have negative connotations in general, it remains to be known which of these topics are more negative and, therefore, to be able to discover complaints in the posts. Figure 4.27 and Figure 4.28 show the sentiment analysis per topic in both subreddits.

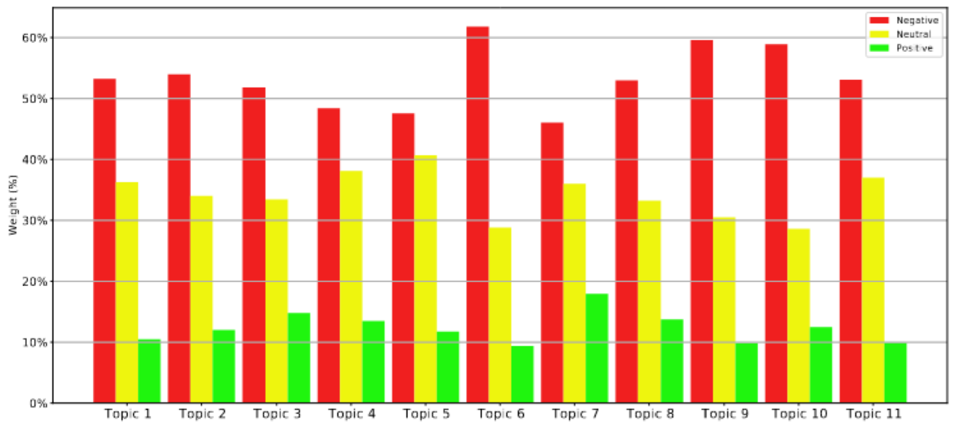


Figure 4.27: Sentiments per topic in r/uber

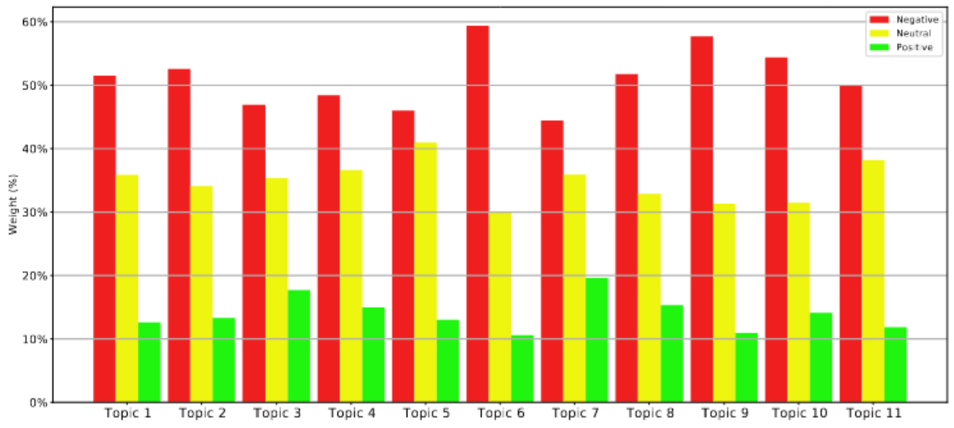


Figure 4.28: Sentiments per topic in r/uberdrivers

As it can be seen, the distribution of sentiments for each topic is highly similar in both subreddits. There are a few differences between the two subreddits. One of them can be seen in Topic 2, related to deliveries, tips, or services such as UberEats, which in r/uber has more negative connotations. On the other hand, Topic 10, related to ratings, is more negative in r/uber than in r/uberdrivers, which could indicate that users complain more about the application than the drivers themselves. Still, as both analyses are very similar, the distribution of sentiments in both subreddits is shown in Figure 4.29.

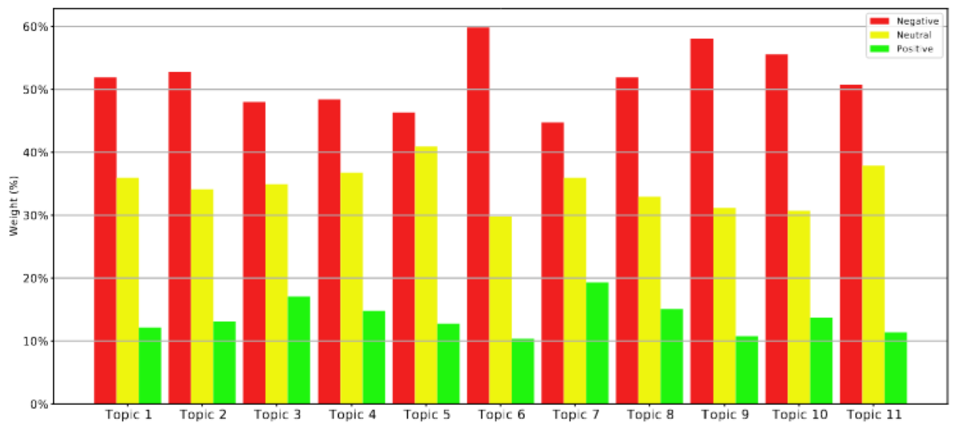


Figure 4.29: Sentiments per topic in both subreddits

Each of the topics has very negative connotations and in none of them do the neutral or positive texts outnumber the negative ones. In addition, positive texts never outperform neutral texts. The most positive posts are related with Topic 3 and Topic 7, while the most negative posts are related with Topic 6, Topic 9 and Topic 10. These results show that riders and drivers complaints are mainly based on travelling (Topic 6) and legal successes (Topic 9), and these complaints have relation with the performance of the application or the experience with the drivers (Topic 10).

4.7 Conclusions

The developed system has demonstrated its capabilities when analyzing this domain. In addition, the above results have shown that the classification process is useful and works well, so that the system would function correctly once it was put into production and collected data every 24 hours. The results of the analysis of the Ride-Hailing domain on Reddit have shown that users talk about different topics of which we have been able to identify 11 different topics. These topics are mainly complaints and they can be modelled with a NER-based analysis, in addition to the topics extraction.

Because of this, we have found that most of the posts from Reddit users are complaints or messages with negative connotations, especially regarding legal issues, pick-up times, and travelling. Therefore, we have developed a tool that allows the analysis of at least Uber users on Reddit, which means that companies can improve based on the feedback of users and easily address the main complaints.

The use of this system in other social media and with other companies will be discussed in Section 6.

CHAPTER 5

Use Case - The Radicalization Domain

5.1 Introduction

This use case has been implemented using parameters and searches provided by the PARTICIPATION project[1].

This use case consists of the analysis of the radicalization domain in social media through the developed system. This application will be based on Twitter data, instead of Reddit data as the previous use case. The used data contains tweets with an anti-Islam and an Islam-related language. The objective of this use case will be to analyze the language used on social media when users talk about radical content.

This application of the developed system can be useful for institutions or governments which are fighting against radical actions, such as extreme ideologies, terrorism, or any other form of radicalism. Moreover, this use case intends to confirm the multidomain application of the system and to observe how the system works with another kind of data.

5.2 Collecting and Inspecting Data

For this use case, several data sets were collected to train the models. The first one was collected in the same way that in the Ride-Hailing use case, using the Twint library for

[1]https://participation-in.eu/

searching for tweets. These tweets were collected between early 2018 to mid-2021 and the searched hashtags were the following:

- *#iraq*, *#syria*, *#muslims*, *#alleyesonisis*, *#BanMuslims* *#islam*, *#is*, *#brotherhood*, *#NoIslam*, *#NoIslamization*, *#afro-Muslims*, *#StopMuslims*, *#Bansharia*, *#banislam*, *#Islamization*, *#nomosques*, *#khilafarestored*, *#stopislam*, *#islamicstate*, *#rapefugees* and *#Europastan.*

These hashtags represent both general terms, which can be expressed in either a radical or non-radical context, and terms with a clear radical content, either pro-Islam or anti-Islam. The main problem of these hashtags is that, due to the Twitter policy, when someone posts a tweet expressing radical content, Twitter removes these tweets from its timeline and they can not be collected. Figure 5.1 and Figure 5.2 show how many tweets were retrieved for each hashtag and the dates in which they were retrieved.

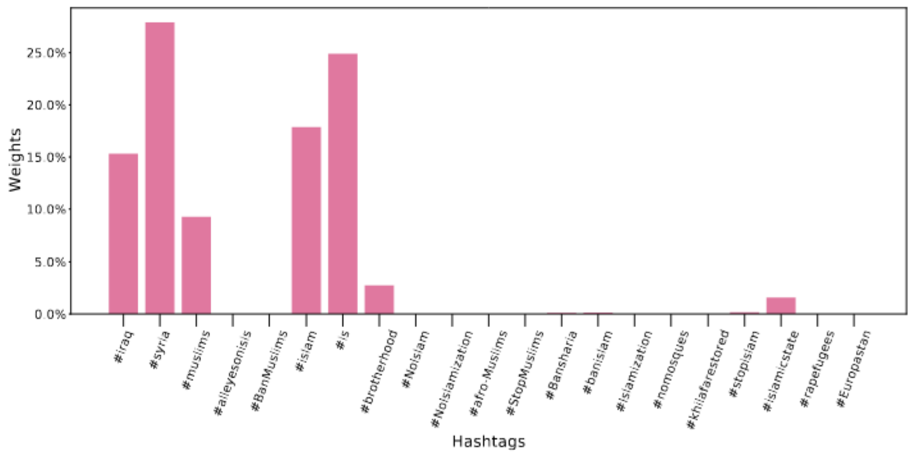


Figure 5.1: Hashtag weights within tweets

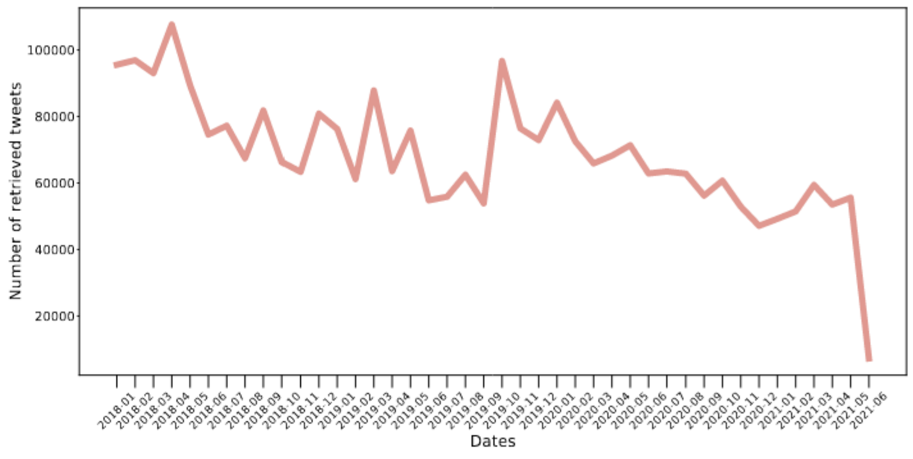


Figure 5.2: Radical tweets collected with Twint over the time

It can be seen that those tweets which contain radical content have practically no presence in the corpus. Overall, 2,876,260 tweets were collected. To solve the problem of the absence of radical tweets, models have been trained using other data sets.

The first one is the Pro-Neu [17] data set. This dataset is composed of two different data sets, the first containing 17,350 tweets extracted from 112 Twitter accounts which have a pro-ISIS context, and the second composed of 197,743 tweets extracted from around 95,000 different accounts which contain ISIS-related texts, both pro-ISIS or neutral.

The second one is called the Pro-Anti [42] data set, which is formed by tweets retrieved from 1,132 different Twitter accounts and which contains 602,511 pro-ISIS tweets and 1,368,827 anti-ISIS tweets. The criteria by which the tweets were collected, both from Pro-Neu and Pro-Anti, is explained in [3].

Figure 5.3 shows the amount of tweets within these data sets as well as the dates in which they were retrieved.

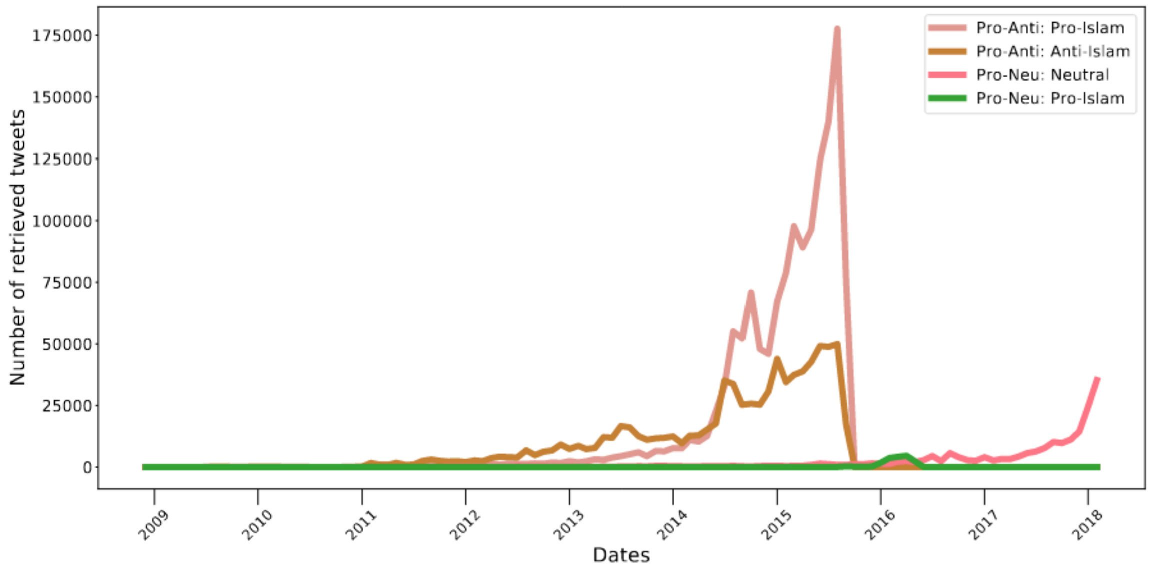


Figure 5.3: Radical tweets from other data sets over the time

As it can be seen, this data fills the gap that was missing in the other dataset, where there were practically no tweets with a clear radical context. In this case, the number of tweets with radical content is very high, unlike the data with neutral content. On the other hand, these data sets are formed by tweets that were collected between 2009 and 2018, being between 2014 and 2016 when more data were obtained.

To get a clearer picture of what these tweets look like, Figure 5.4 shows the 30 most common hashtags in tweets.

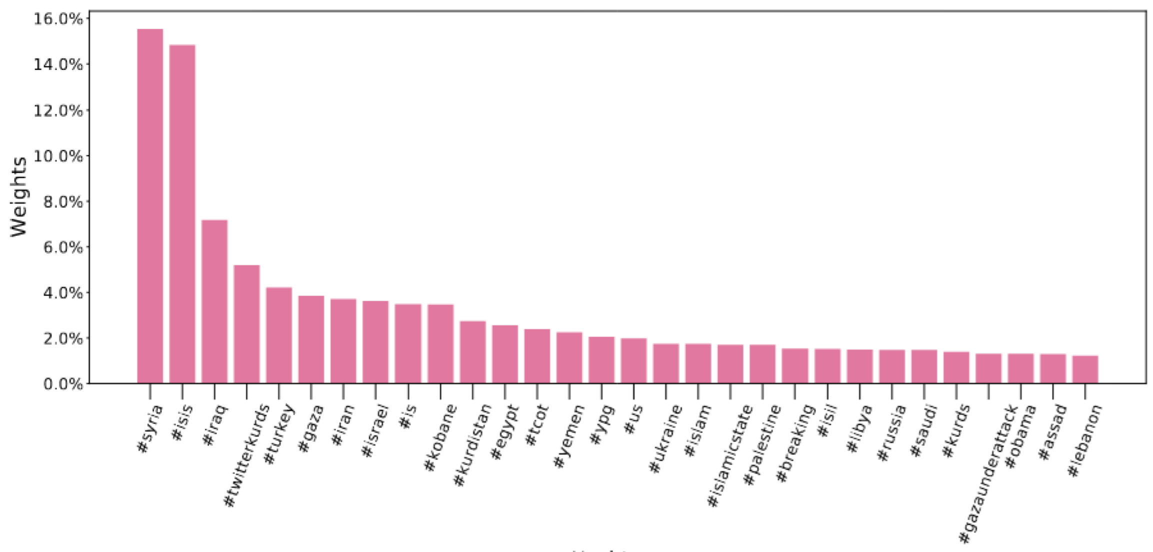


Figure 5.4: Hashtags weights within alternative data sets

It can be appreciated that some of the most common hashtags, such as *#syria* or *#iraq*, among others, also appear within the main data set. In addition, other Islam-related hashtags appear in these additional data sets, so the data is now more complete. Table 5.1 shows the similarities and differences between the different data sets.

Statistic	Original Dataset	Pro-Neu		Pro-Anti		Total
		Pro	Neu	Pro	Anti	
Total no. of tweets	2,876,260	17,350	197,743	1,397,549	831,844	**5,320,746**
Total no. of words	85,353,752	313,770	3,670,477	26,596,093	15,932,109	**131,866,201**
Avg. no. of words per tweet	29.675	18.085	18.562	19.031	19.153	**24.783**
Total no. of different words	3,144,657	35,527	268,737	1,368,391	1,048,333	**5,295,111**
Avg. no. of different words per tweet	1.093	2.048	1.359	0.979	1.260	**0.995**

Table 5.1: Statistics of the used data sets

The original data set has some different characteristics with Pro-Neu and Pro-Anti data sets, which are similar between them. The main difference is that it seems that tweets are larger in the original data set, but the average number of different words per tweet is similar in all data sets, except in the Pro-Neu data set. This is because this data set has much less tweets and it will be easier to find new words in these tweets.

Therefore, among all the data sets, there are 3,074,003 neutral tweets (joining the original data set with the *Pro-Neu: Neu* data set) and 2,246,743 radical tweets, which are divided into pro-Islamists and anti-Islamists. This data will be used in the following steps to create the models that will be later integrated into the final system.

5.3 Topic Modeling

Before training the models, the Cleaning Process is passed through every tweet. After this process, the same procedure that was carried out in the Ride-Hailing domain (which is explained step by step in Section 3.4.1) has been used in the processing of the texts in this domain. As the amount of available texts is very large, it was decided to use 1,000,000 texts for the LDA modeling, leaving the rest of the tweets for future processes, such as the generation of Word Embeddings models or the training of classification algorithms. The criterion used for this data partitioning was to use the same amount of radical and neutral texts, that is, 500,000 radical tweets and 500,000 neutral tweets, chosen in a random way. The same tweets were used in every topic modeling process.

It is important to note that, firstly, a baseline LDA model was generated as well as in the Ride-Hailing domain. This first model was used to try to increase the performance of the models, adding, removing, or changing the different processes that modify the texts. The Coherence Score values of this baseline model are shown in Figure 5.5.

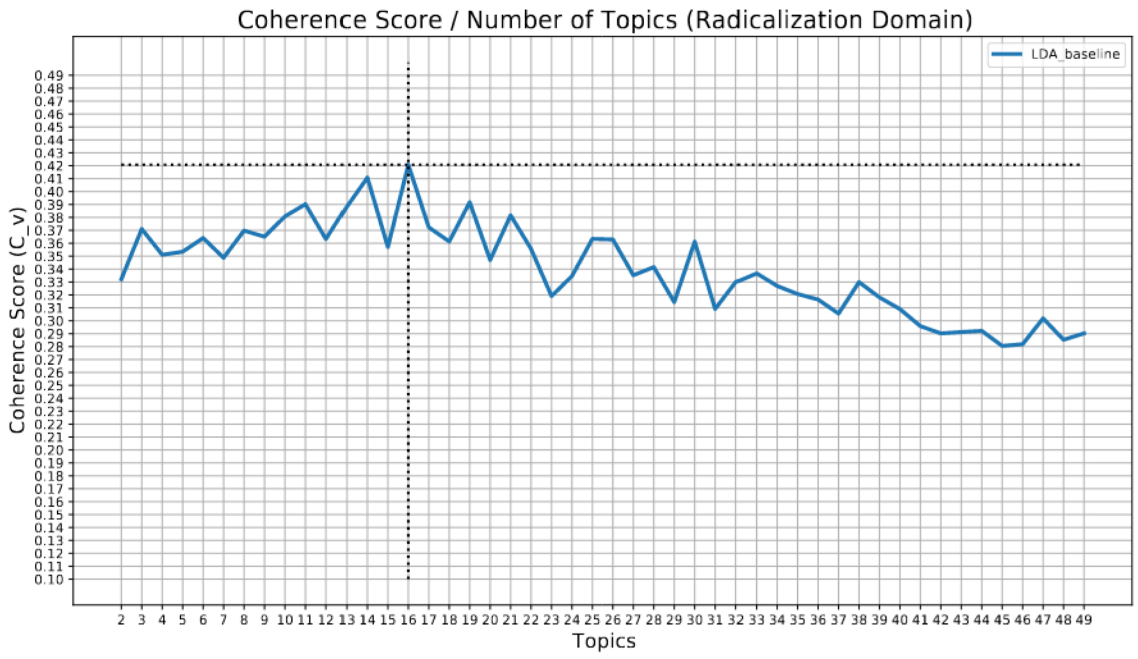


Figure 5.5: Baseline Model - Coherence Score through the Radicalization domain

The optimal number of topics seems to be 16 topics, for which the maximum coherence value is obtained (around 42%). The coherence rises up to the same value of k, from which it falls steadily. As in the Ride-Hailing domain, the goal of LDA optimization will be to improve the coherence values by trying to find the best possible model. The procedure carried out in this domain was the same as in the other use case and 8 different text preprocessing types were used for the creation of the dictionary and the corpus on which the LDA is trained. The number of different preprocessing approaches is lower than the same number in the Ride-Hailing domain. This is because in the Ride-Hailing use case, virtually every new model improved the previous one, whereas in this use case it has cost much more to improve the models. Because of this, the baseline model is one of the best of all those attempted. The result of the Coherence Score for all these models is shown in Figure 5.6.

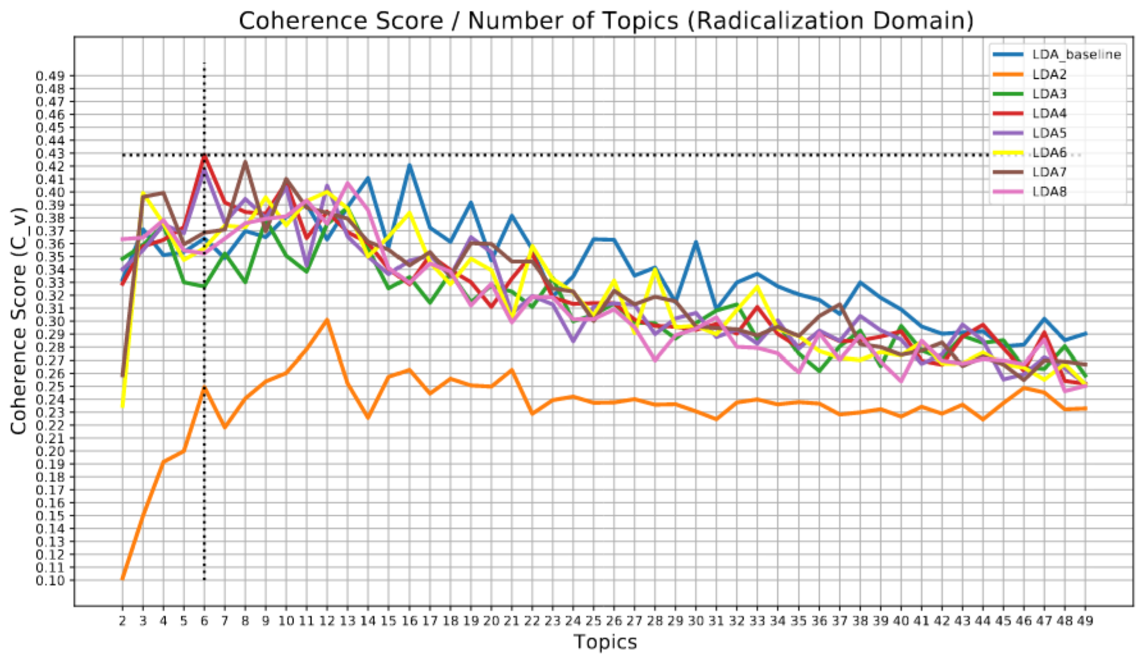


Figure 5.6: Coherence Score through the Radicalization domain

There are several differences with the results in the Ride-Hailing use case. Firstly, the maximum coherence values are lower than in the Ride-Hailing domain, being the maximum being reached with the LDA4 in the 6 topics model. In addition, it can be seen that the maximum values are not so dependent on a specific type of text preprocessing as in the other use case, but that there are several models with values close to the maximum. Finally, the results show a much less marked and flatter curve, where it is not obvious that there is a better number of topics than others, although it is clear that low numbers are better than very high numbers of topics. The latter can also be observed in the Perplexity curve, shown in Figure 5.7.

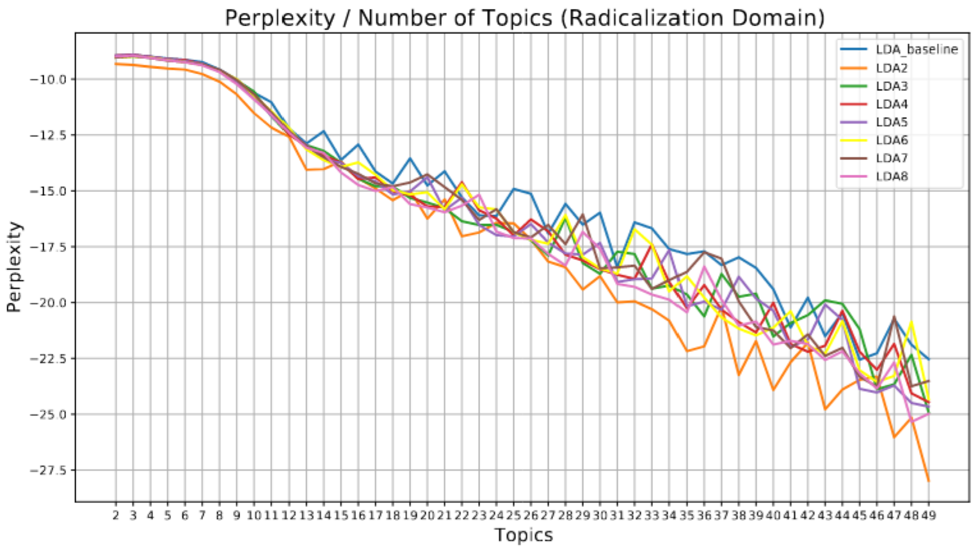


Figure 5.7: Perplexity through the Radicalization domain

As in the Ride-Hailing domain, the more the number of topics increases, the lower the value of perplexity and, therefore, the less surprised the model will be when obtaining new data. Moreover, in this use case, the results of this metric behave more chaotically from topic 9 onwards, after which the values start to drop sharply. This may indicate that a number of topics much higher than 9 may generate a model that is not viable.

For these reasons, and, especially, the coherence values, different models with different text preprocessing types have been optimized. Specifically, the 4 best models have been optimized (in the same way as in the Ride-Hailing use case, except that in that case it was done with the same preprocessing procedure), which are the LDA4 with 6 topics, the LDA7 with 8 topics, the LDA baseline with 16 topics and the LDA7 with 10 topics. The results of this optimization are shown in Table 5.2, Figure 5.8 and Figure 5.9.

Topics	**C_v**	α	β
6	45.327%	0.61	0.61
8	**48.700%**	**0.9**	**0.61**
10	48.548%	0.9	0.9
16	46.544%	0.9	0.61

Table 5.2: Optimizing LDA Hyperparameters (Radicalization Domain)

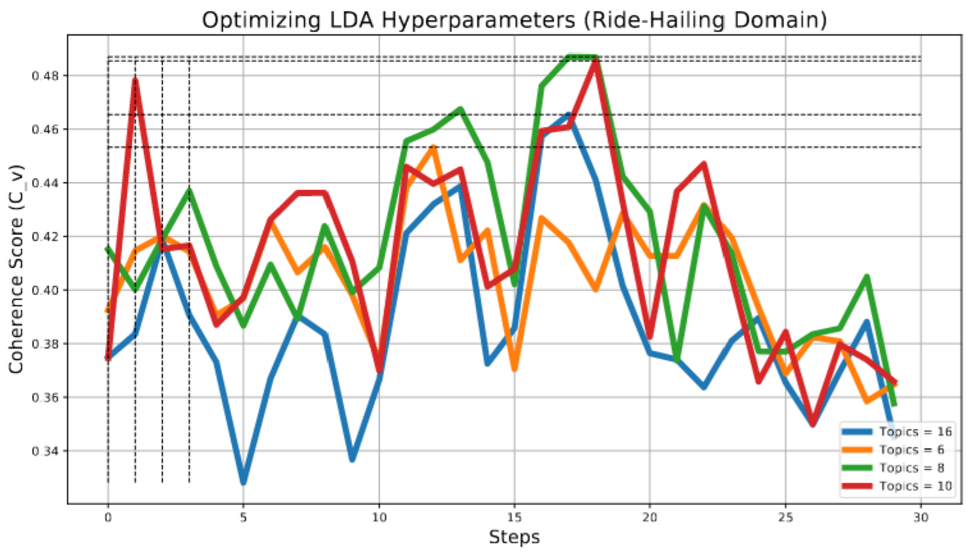
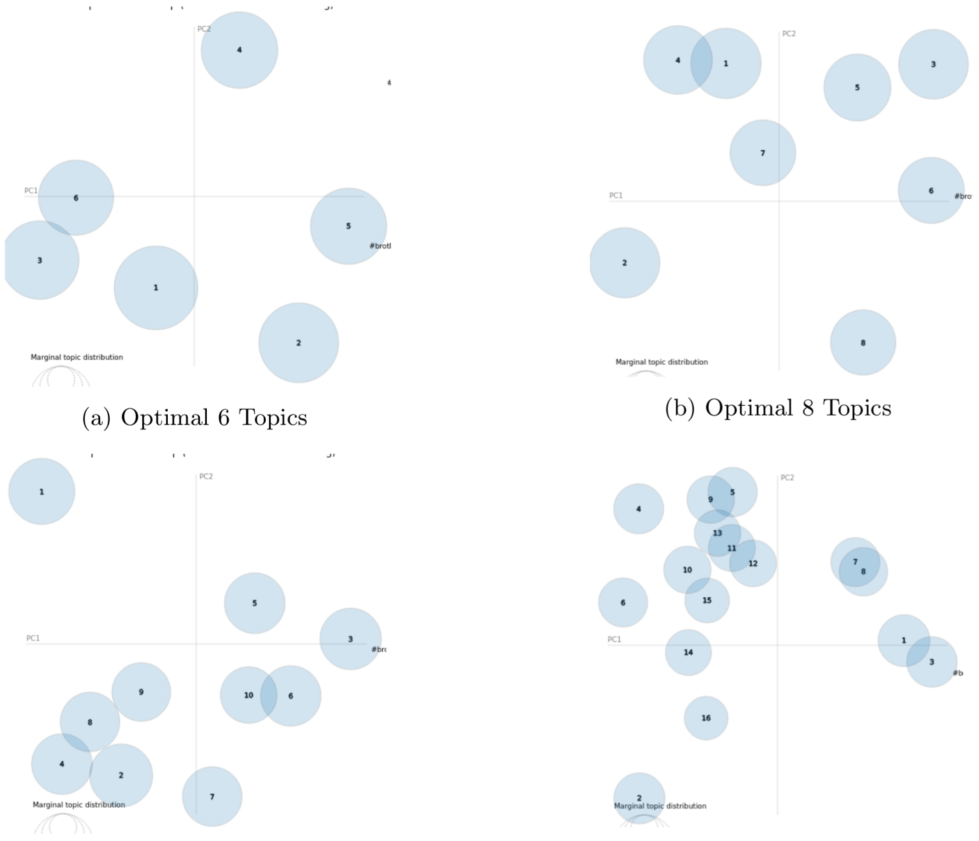


Figure 5.8: Optimizing LDA Hyperparameters (Radicalization Domain)



(a) Optimal 6 Topics

(b) Optimal 8 Topics

(c) Optimal 10 Topics

(d) Optimal 16 Topics

Figure 5.9: Visualization of the best models in the Radicalization domain

As it can be seen, both models with 10 and 16 topics have several related topics that share space in the set of topics. Moreover, the coherence result of the model with 16 topics is one of the lowest, so it is discarded. On the other hand, the model with 10 topics has a good coherence value compared with the others results. Therefore, it will be necessary to analyze whether the information of each topic is good and better than that of the other models, although the independence of the topics must be taken into account. The maximum independence between topics occurs in the model with 8 topics and in the model with 6 topics. Moreover, the model with 8 topics has the best coherence of all the generated models, so this will be the first choice.

As stated before, the election of the final model is strongly based on the Coherence Score - Number of topics - Human observation commitment. The human observation, in addition to the analysis of the correlation of the topics, is given by the analysis of each topic. In this use case, the analysis of the topics of the different models led to the choice of the 8-topic model as the final model, since its topics are easier to interpret than in the other models, where they were slightly overlapping and poorly explained. Thus, the classifiers to be designed later will have to classify tweets into 8 different classes, which are as follows:

- **Topic 1: Middle East situation.** A generic topic in which are included those tweets which talk about Middle East countries, in addition to the actions of the United States in this area. This topic includes words such as *#syria*, *#israel*, *#us*, *#yemen*, *#trump*, *#lybia*, *#turkey*, *#usa*, *#afghanistan*, *#palestin*, *#saudiarabia*.

- **Topic 2: Israeli-Palestinian conflict and religion-related.** Tweets related with the Israeli-Palestinian conflict and with a religious content, with words like *israel*, *american*, *christian*, *palestinian*, *gaza*, *jewish*, *racist*, *#alllivesmatter*, *migrant*, *nazi*, *netanyahu*.

- **Topic 3: Family-related.** It contains family-related words and generic neutral words such as *#brotherhood*, *day*, *today*, *guy*, *girl*, *daughter*, *friend*. It seems that is strongly-related with the *#brotherhood* hashtag, so it is possible that this hashtag is very generic and not solely focused on radicalism.

- **Topic 4: Generic / Social and politics.** This topic includes tweets with a generic social and politics content, with words like *public*, *program*, *social*, *job*, *country*, *state*, *nation*, *politics*, *power*.

- **Topic 5: Syria-related.** Tweets with a Syria-related content, with words like *#syria*, *syria*, *syrian*, *#russia*, *child*, *war*, *bomb*, *#assad*.

- **Topic 6: ISIS attacks.** Related with terrorist attacks and specifically ISIS attacks. It contains words such as *kill*, *attack*, *#isis*, *report*, *force*, *isis*, *#turkey*, *terrorist*, *fight*.

- **Topic 7: Iraq and Iran situation.** Tweets mainly talking about the situation in Iran and Iraq, with words like *#iraq*, *iraq*, *iran*, *govern*, *protest*, *iraqi*, *region*.

- **Topic 8: Islam-related.** Topic mainly related with the Islam in a generic way and without terrorism connotations, with words like *#islam*, *#muslim*, *muslim*, *islam*, *allah*, *#quran*, *religion*, *peace*, *god*, *believe*.

As it can be appreciated, some of the extracted topics in this domain have important differences with the rest of the topics. Specifically, Topic 3, which is related to family relationships, seems to be mainly extracted from tweets with the keyword *#brotherhood*. Tweets extracted with this hashtag seem to be outside of the radicalization domain, but some of them could be inside of it.

In any case, the topics seem to be quite distinct from each other, although the coherence is lower than in the case of Ride-Hailing (in the radicalization domain the best model has a 48.700% of coherence versus the 72.82% of the Ride-Hailing domain). Because of this, it seems that the topic separation of the corpus can be a good approximation to train the classification models, which will be explained later.

5.4 Validation Process, Sentiment Analysis and NER extraction

Once the LDA model is performed, a lexicon with the most important words of each topic is created. This lexicon will be useful to reject texts that do not belong to the domain under study and that may be wrongly included in the data. Every text of the total data set must pass through the Validation Process before the training step. It is important to note that every text is cleaned (Cleaning Process) before the Validation Process.

The Validation Process for this domain has some changes with respect to the other use case. First, no checks are performed on the account posting the tweet, because it is understood that in this domain it is neither necessary nor convenient to do so. On the other hand, the lexicon of words used to check which words each tweet must contain to be added to the data set consists of 714 different words and has been extracted with the same procedure as in the other use case.

The results of the Validation Process were that 137,877 texts were rejected, which implies a 13.877% of rejected texts and, therefore, 862,123 validated texts. Table 5.3 shows these results.

Radical Data Validation Process		
Source	**Rejected Texts**	**Rejected Texts (%)**
Twint Scraped Dataset	80,721	16.144
Pro-Anti and Pro-Neu	57,156	11.431
Total	**137,877**	**13.877**

Table 5.3: Validation Process with the Radicalization domain data

It can be appreciated that those tweets which have been collected in this project have a higher rejection rate than the other data sets. This is to be expected, since the other data sets have been cleaned before and have been used in other projects.

Now, some NLP functions will be applied to the validated texts. Firstly, Figure 5.10 shows the aggregated sentiment analysis for this corpus.

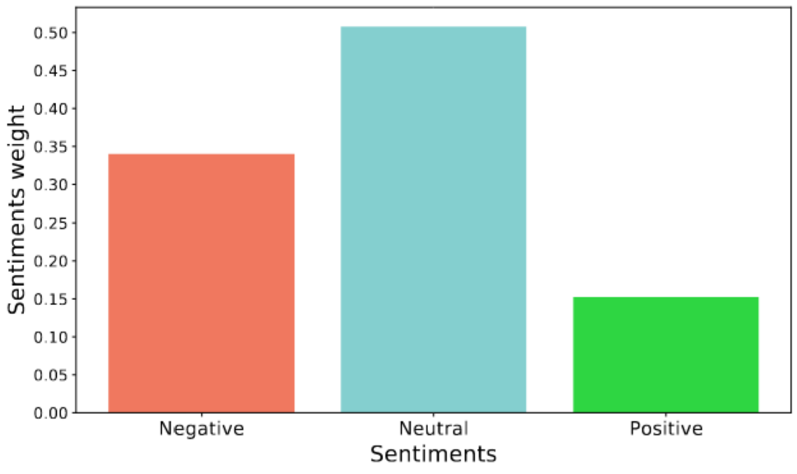


Figure 5.10: Sentiments distribution in the Radicalization domain

The results show that neutral tweets are predominant in all the analyzed texts. In addition, negative tweets are greater than positive tweets. This indicates that there is a clear tendency to write tweets with more negative content about this domain. Figure 5.11 shows the sentiment analysis as a function of tweet content type.

Figure 5.11 shows some interesting results. Firstly, neutral tweets are greater than negative tweets and these are greater than positive tweets in both tweet types. However, several differences can be noted. First, tweets with a neutral content are more negatively trending than radical tweets. Additionally, the corpus of neutral tweets contains more positive tweets than that of radical tweets. Therefore, neutral tweets have fewer tweets that do not express sentiment (neutral sentiment) than radical tweets. The explanation for this fact is given in that those tweets that are radical, although obviously more negative

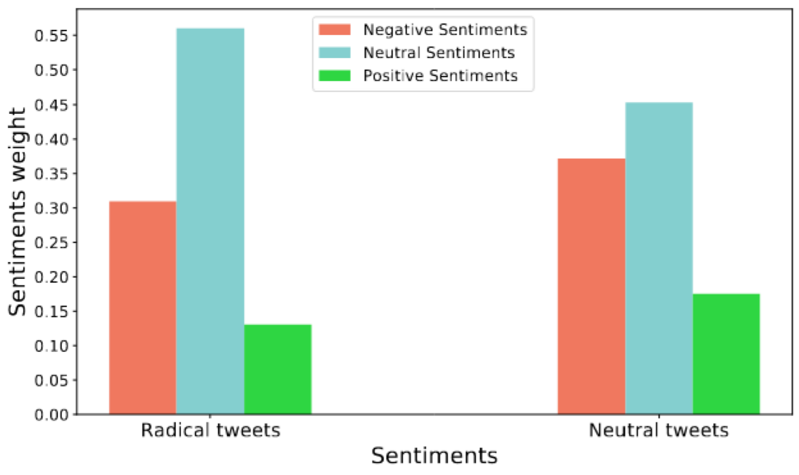


Figure 5.11: Sentiments distribution in the Radicalization domain per ideology

than positive, do not tend to express a marked sentiment beyond their own ideology, which contains clear negative aspects. However, tweets belonging to the corpus of neutral tweets do tend to express more sentiment. This sentiment, being more negative than positive, tends to express rejection or to speak negatively of radicalism, without either the authors or the tweets themselves expressing radical ideas. It is worth remembering that neutral tweets need not express neutral sentiment, but only that they are not radical. These aspects found here can be essential when studying this type of content and give an overview of how these tweets are written, for what purpose they are written, and what are the differences between the two data sets.

Finally, Figure 5.12 shows the results of the NER analysis as a function of the data set.

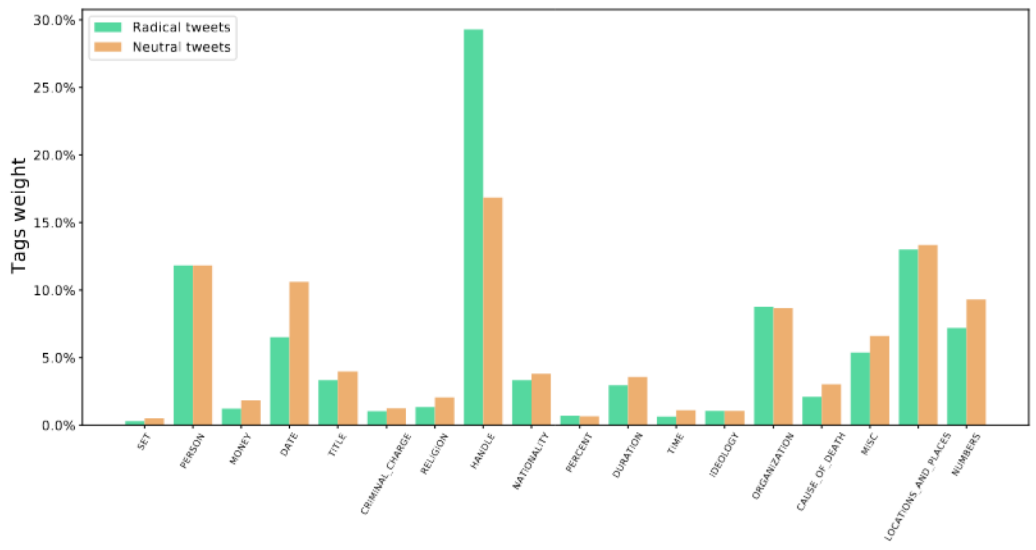


Figure 5.12: NER tags distribution in the Radicalization domain

It can be appreciated that both data sets are similar in terms of labeled entities. *HANDLE* entities stand out, which are usually words such as hashtags or usernames. As the corpus is composed of tweets, this is the predominant entity in both sets. Also, entities such as *PERSON*, *DATE*, *ORGANIZATION*, *LOCATIONS_AND_PLACES* or *NUMBERS* have a lot of influence too. Excepting *HANDLE*, almost every entity has a bigger weight in neutral tweets highlighting *DATE*.

5.5 Classification Model

As well as in the Ride-Hailing use case, new unseen data will be retrieved each day. This data, to be classified, needs to be represented in such a way that it can be introduced as an input into a classification model. As SIMON was discarded in the Ride-Hailing case, representations in the form of Tf-idf and the Word Embeddings, including the Word2Vec and FastText variants, have been realized exactly as in the Ride-Hailing use case.

Word Embeddings were trained using the whole collected corpus before the Validation Process, to train the models with as many texts as possible. Table 5.1 shows that 5,320,746 texts were collected, being 2,876,260 texts collected by Twint and 2,444,486 texts provided by the other data sets. Models were trained using the same parameters that in the Ride-Hailing use case, alternating with 100, 300, and 500 dimensions. The PCA representation of the 100-dimensional models is shown in Figure 5.13 and Figure 5.14. Other images representing the higher-dimensional models, as well as the lexicon used to create the figures, are available in the Appendix B.

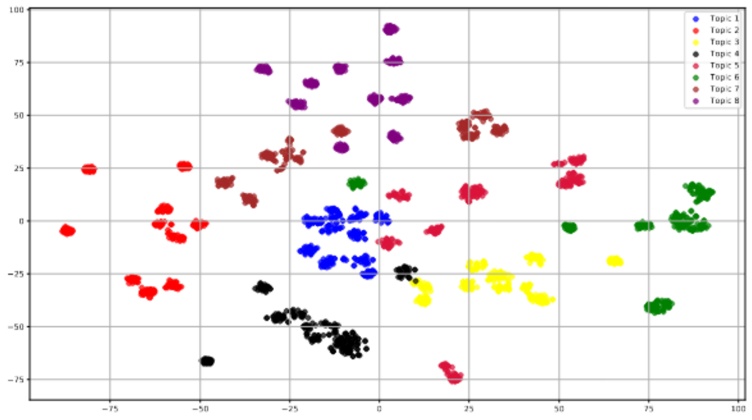


Figure 5.13: FastText 100-dimensional model PCA representation - Radicalization domain

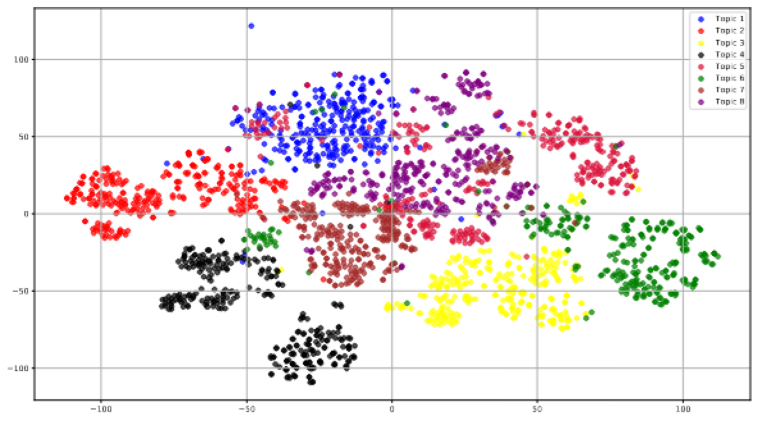


Figure 5.14: Word2Vec 100-dimensional model PCA representation - Radicalization domain

As in the Ride-Hailing domain, cluster formation can be seen in both models. Again, the FastText representations show more scattered clusters, although the similarity between words belonging to the same topic is perfectly visible. On the other hand, the Word2Vec representations show an accumulation of clusters in which the closeness between topics can be perfectly appreciated. These representations, although it cannot be seen a priori which model will perform better, offer a first image that demonstrates the feasibility of this type of representation in terms of training classification models with features extracted from these models.

The training procedure in this domain follows the same procedure that in the Ride-Hailing use case. Firstly, the validated data is labelled with the LDA results, assigning a topic to each tweet. The results of this assignment of topics are shown in Figure 5.15.

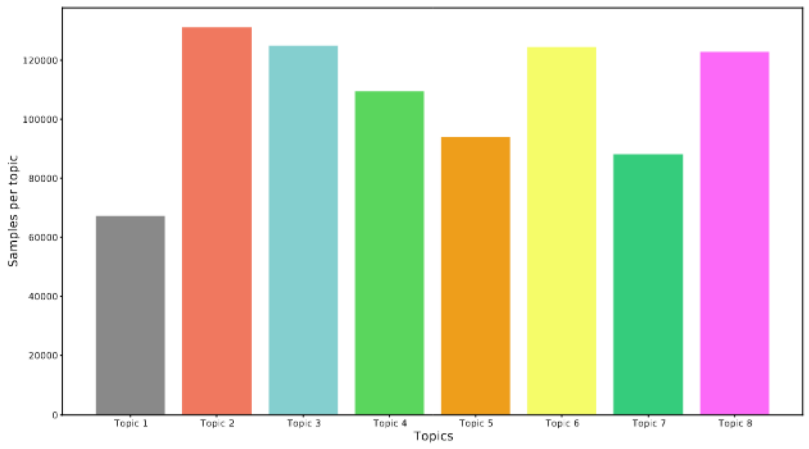


Figure 5.15: Samples per topic in the radical training dataset

As it can be appreciated, Topic 1 has around 60,000 samples, while Topic 1 has over 120,000 samples. For this reason, and to balancing data, the training data sets were limited to 50,000 samples for each topic. The rest of the samples were used as validation data to test models and procedures such as cross-validation processes. This 50,000 samples for each topic were selected randomly and, after this, the training data set was formed by 400,000 samples.

The classification models for this domain followed the same approach that in the Ride-Hailing domain, training models with Tf-idf and Word Embeddings features, but only using the Logistic Regression algorithm. This decision was taken because when the Ride-Hailing domain was analyzed, the Gradient Boosting modeling requires enormous time to arrive at concrete solutions. Due to this, mainly generated by the limitations both in computational capacity and in the time needed for the optimization and development of the models, Gradient Boosting has not been analyzed in this use case. As for model optimization, it was based on a K-fold cross validation, usually with $k = 10$ and optimized with the Grid Search and the Halving Grid Search procedures, carried out in the same way that in the other domain. The results of the final models are shown in Table 5.4.

	Accuracy	**Precision**	**Recall**	**F-Score**
Word2Vec 100 dim	77.287	77.935	77.287	77.442
Word2Vec 300 dim	79.094	79.691	79.094	79.235
Word2Vec 500 dim	80.012	80.597	80.012	80.153
FastText 100 dim	75.070	75.852	75.070	75.256
FastText 300 dim	77.026	77.723	77.026	77.192
FastText 500 dim	77.990	78.669	77.990	78.153
Bigrams - Tf-idf	**83.197**	**84.123**	**83.197**	**83.463**

Table 5.4: Logistic Regression Results in the Radicalization Domain

The results have some similarities with the Ride-Hailing domain results. First of all, the model trained with the features extracted from the Tf-idf is the best of all due to the classification of texts based on the topics generated from the LDA. Secondly, all Word2Vec models are better than FastText models with the same dimensions. Moreover, the results are all good, with even the 500-dimensional Word2Vec model exceeding 80% F-Score.

Therefore, as in the case of Ride-Hailing, it is demonstrated that the system works correctly when classifying topics in this domain and therefore it is confirmed that the system is valid for the analysis of topics and opinions in different domains. This system could be useful in identifying hate speech or radical speech, as well as in tasks of prevention of radicalism that can be carried out by police or intelligence agencies around the world.

5.6 Results

When the final model is performed, the model can predict the topic of each text of the corpus. In this section, the Tf-idf approach will be used to classify texts in their corresponding topics. The election of the Tf-idf is due to its F-Score, which is the highest of all the generated models. However, it is important to note that embeddings models are likely to be better as soon as new data is introduced, that the models have not seen or data that comes from other sources. This will be discussed for the Ride-Hailing domain in Section 6.

Firstly, the distribution of topics around the corpus and by the type of the tweet (radical or neutral) when they are predicted with the Tf-idf model is shown in Figure 5.16 and Figure 5.17.

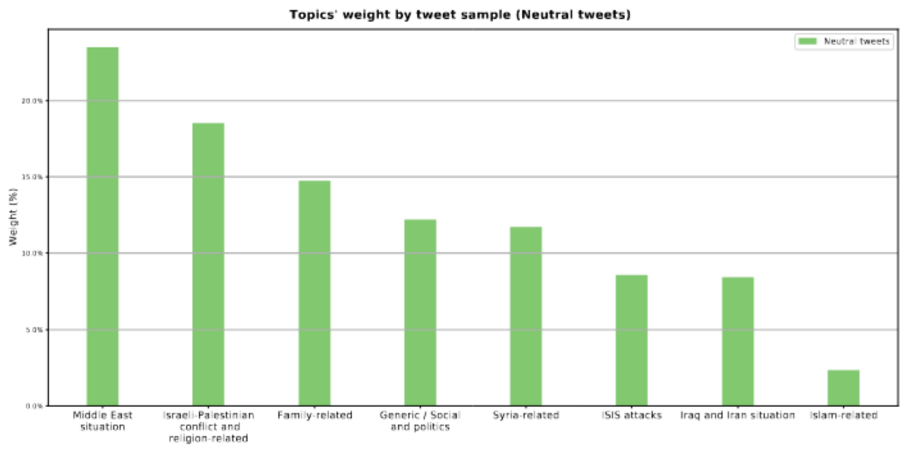


Figure 5.16: Prediction of topics in the neutral tweets corpus

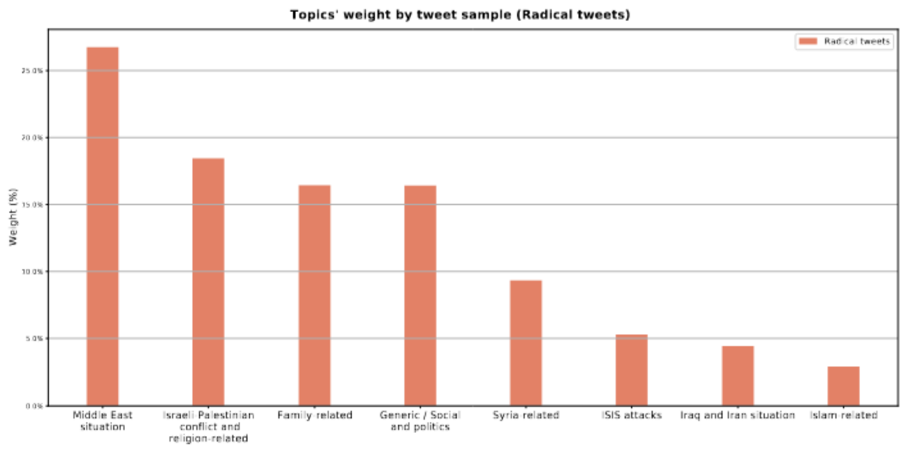


Figure 5.17: Prediction of topics in the radical tweets corpus

It can be seen that Topic 1, Topic 2, and Topic 3 are the dominant topics in both corpus. However, Topic 4 appears more frequently in neutral tweets. On the other hand, Topic 5, Topic 6, Topic 7, and Topic 8 are the least frequent in both corpora, but appear more in neutral tweets. To facilitate the analysis, Figure 5.18 shows the two corpora in the same figure.

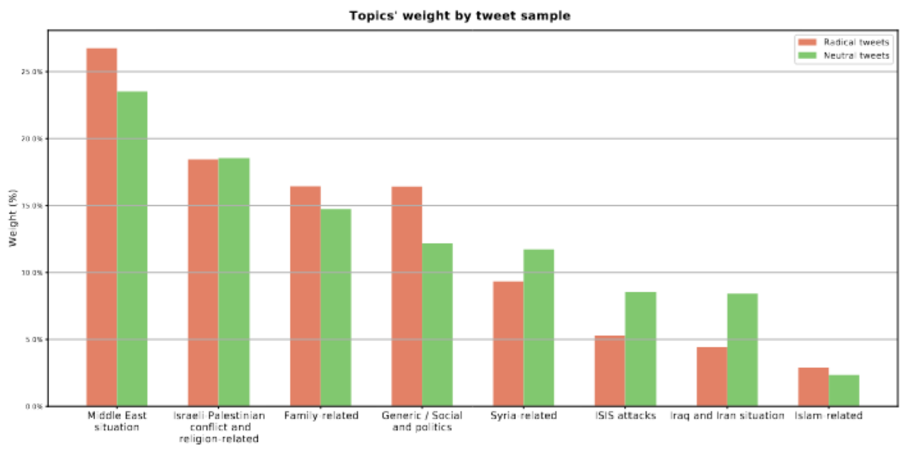


Figure 5.18: Prediction of topics in the whole tweets corpus

As it can be appreciated, Topic 1, Topic 3, Topic 4 and Topic 8 are more common in the radical corpus. On the other hand, Topic 5, Topic 6, and Topic 7 appear with a higher frequency in neutral tweets. In addition, Topic 2 appears approximately with the same weight in both corpus.

With these results, it seems that radical texts talk with a higher frequency about general Middle East countries, family, Islam and society or politics. On the other hand, neutral tweets seem to talk more about the Syria situation, the ISIS, and countries such as Iraq and Iran. Moreover, it is interesting that the Israeli-Palestinian conflict appears with practically the same frequency in both corpora.

These results show that neutral tweets, which are written by people who are not extremists, talk with a higher frequency about terrorism-related topics such as the situation in Syria, in Iraq or the attacks of ISIS. On the other hand, radical people talk about topics which do not talk about radicalism in a direct way.

Another analysis that has been done is the NER analysis, which is shown below in Figure 5.19 and Figure 5.20.

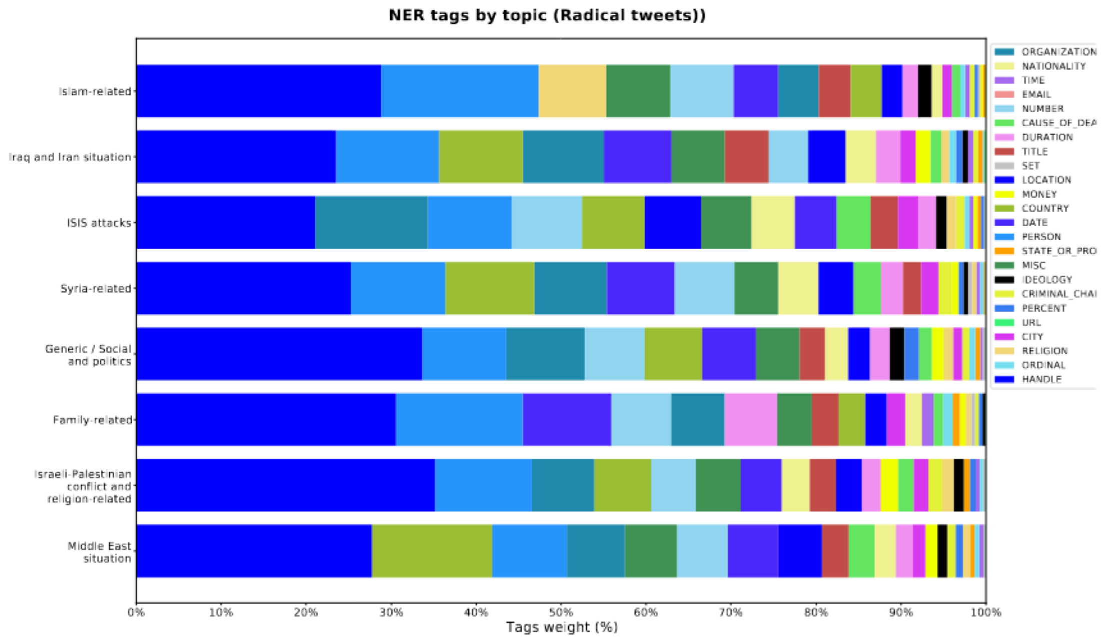


Figure 5.19: NER tags in radical tweets

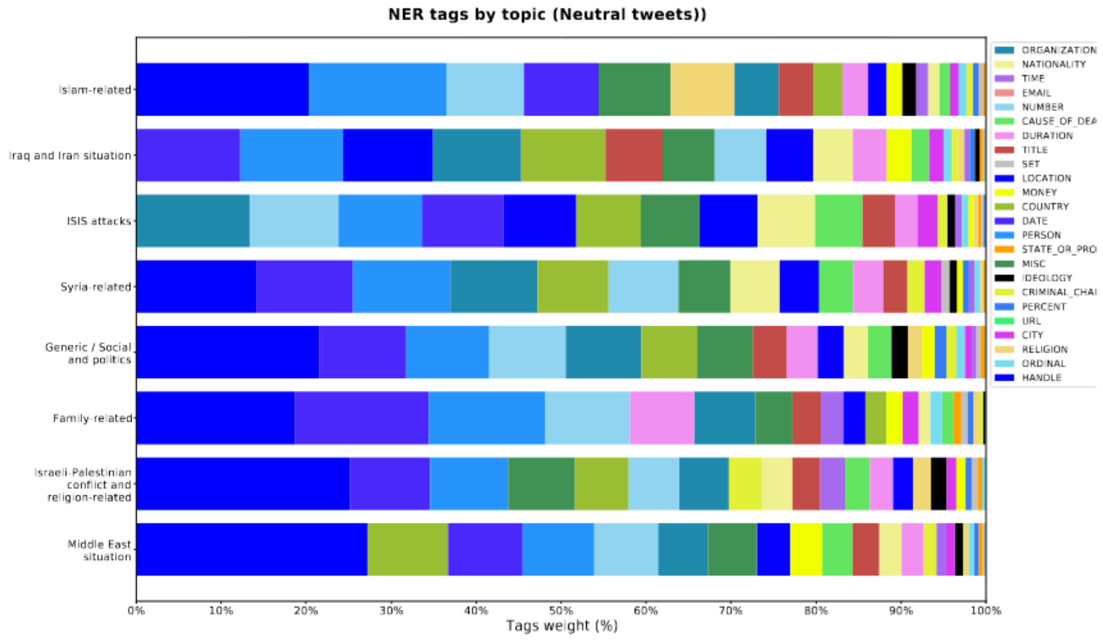


Figure 5.20: NER tags in neutral tweets

As texts are tweets, in almost every topic the dominant entity is *HANDLE*. After this, the *PERSON* entity seems to appear in both corpus with high weights, but more in radical tweets, while in neutral tweets, in some topics, the *DATE* entity outperforms *PERSON*. It seems that both corpus are similar in terms of the entities that appear in the texts of each topic.

Furthermore, it can be seen that some topics have some thematic entities, such as the *ORGANIZATION* entity in the ISIS-related topic, the *COUNTRY* entity in the Middle East-related topic or the *RELIGION* entity in the Islam-related topic. This happens in both corpus, although these entities can have more presence in one of them. Finally, it should be noted that these entities make a lot of sense in the respective topics in which they appear.

The last performed analysis were the sentiment analysis. The results of this analysis for each corpus are shown in Figure 5.21 and Figure 5.22.

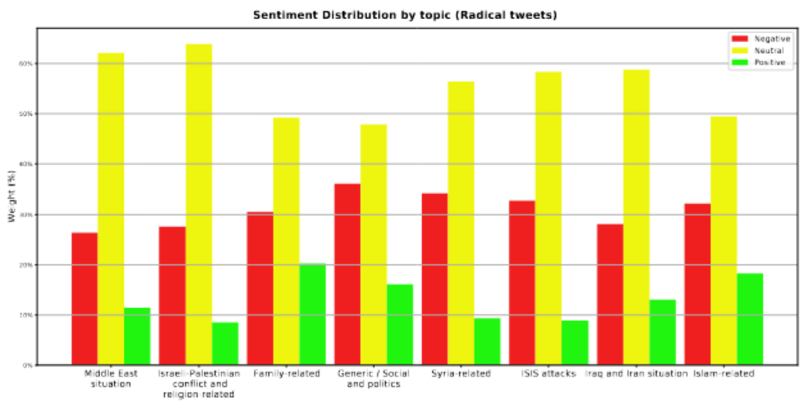


Figure 5.21: Sentiments per topic in the radical tweets corpus

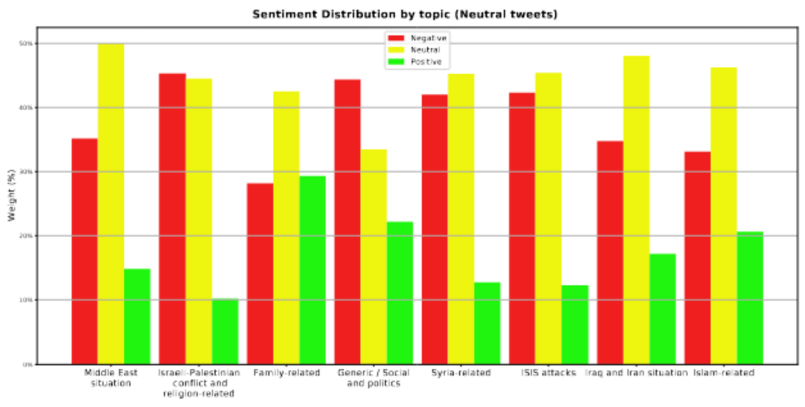


Figure 5.22: Sentiments per topic in the neutral tweets corpus

These figures show very interesting results. First, it can be appreciated that neutral tweets are, by far, much more negative than radical tweets in every topic. An explanation of this fact is that radical tweets, which are extremist tweets, do not talk bad about extremist actions. For instance, a radical Islamist will speak well of an attack, while a non-extremist will speak disparagingly or badly about it. Because of this, neutral tweets are much more negative than radical ones. This is also important, since if there are non-extremist people writing tweets with a lot of negativity, it can lead to these people becoming extremist people as well.

It is important to see that in neutral tweets, these tweets are also more positive than in the radical corpus. Because of this, it can be said that neutral tweets are more polarized than radical tweets, which neutral sentiments are dominant in every topic.

In addition, in radical tweets the more negative topics are Topic 4 (Social and politics) and Topic 5 (Syria-related), while in neutral tweets the more negative topics are Topic 1 (Israel and Palestine) and Topic 4 (Social and politics). On the other hand, the more positive topics in radical tweets are Topic 3 (Family-related) and Topic 8 (Islam-related), while in neutral tweets the more positive topics are Topic 3 (Family-related) and Topic 4 (Social and politics). Therefore, it can be seen that Topic 3 in both corpora has positive connotations, while Topic 4 has more negative connotations. It is also extracted that both corpora speak well of Islam in general terms. Some of these observations and others can be done analyzing Figure 5.23, where the aggregated sentiment analysis in both corpus is shown.

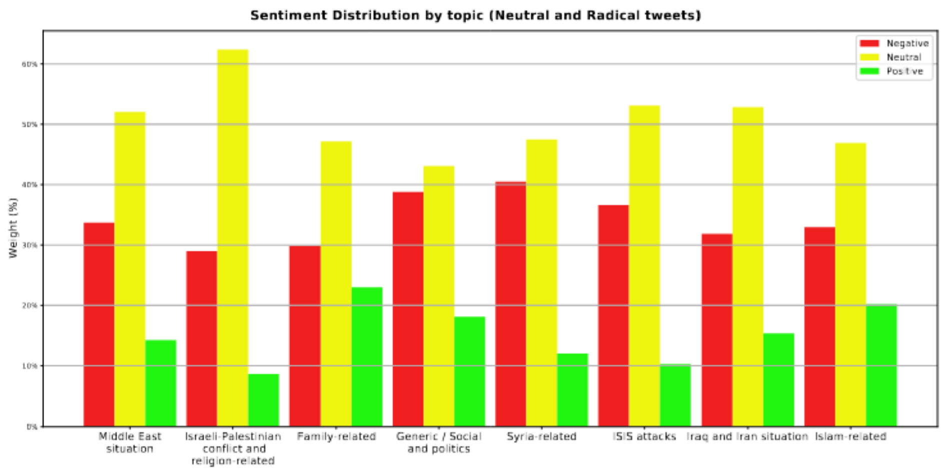


Figure 5.23: Sentiments per topic in radical a neutral tweets corpus

5.7 Conclusions

As well as in the Ride-Hailing domain, the system and the processes that make up the system and its operation have also been shown to be effective when radical content tweets are introduced to the system. The analysis of this domain, in addition to the Ride-Hailing domain, has proven that the developed system can be used to analyze any type of domain. Additionally, it has been demonstrated that the system can be fed by different kinds of texts (in this use case have been tweets) and operate in a correct way.

The results shown here also demonstrate that it is possible to use the system to analyze radicalism and extremism in social networks. It has been shown how these tweets, both of radical and neutral content, can be classified into at least 8 topics. Each of these topics shows a different reality and ranges from religious and family topics to topics of conflicts between different territories or directly terrorist topics.

In addition, entity analysis and sentiment analysis have shown different realities of these topics, such as neutral tweets being more polarized than radical ones. This type of analysis can be fundamental in the study of radicalism prevention.

For all these reasons, it has been demonstrated that the developed system works perfectly in this domain, and therefore the premise of developing a multi-platform and multi-domain tool has been fulfilled.

CHAPTER 6

Cross-Source Validation: Transfer Learning

6.1 Introduction

One of the main objectives of the developed system is to analyze and compare the language and opinion of users on different platforms. Social media sites stand out, among other things, for using different ways of writing depending on the social media, some being more informal than others and having a particular language.

Up to this moment, Twitter, Reddit and Google Play Store data have been collected and treated as the same data type, that is, texts, for instance, when creating the embeddings models. However, the way in which these texts are written highly depends on the social media source. For providing context, Twitter posts contain a lot of misspelled words in addition to hashtags (words starting by the "#" character which work as keywords), usernames (starting by the "@" character) and since 2017 they are limited to 280 characters per tweet (previously 140). On the other hand, Reddit posts, both submissions and comments, and Google Play reviews are not limited by the number of characters and they do not have special keywords beyond the jargon used in these networks [16].

In addition to what has been mentioned and explained throughout this work, where it is clear that the main objective of the developed system is the analysis of language and opinion, it remains to be seen whether this analysis can also be transferred to other sources. This is because, although it is true that certain models have been fed with Word

Embeddings models that have been generated from data coming from all these sources, the categories in which the texts are classified have been extracted only from Reddit. In the field of machine learning, this type of classification, where a task is learned on certain data and is intended to be taken to another type of task without retraining the models, is called Transfer Learning [49].

To perform this analysis, the first step was to collect the data, which are the same as those used for the embeddings. Subsequently, a small set of texts has been manually labeled to evaluate this methodology. Finally, using the classifiers developed for the Ride-Hailing domain, the topic of each labeled text was predicted and the reliability of this method for this particular domain was evaluated.

This chapter will therefore explore this possibility and explain how this analysis has been done and if indeed learning can be transferred to other data sources, at least in the domain of study.

6.2 Inspecting Data

As stated before, the collected data was the same as the Word Embeddings training data set and it has been described in Table 4.4 and in Figure 4.18.

First of all, the data obtained must go through the corresponding Cleaning and Validation Processes before being classified, since these processes directly reject texts that do not comply with the requirements of the system and they must not be annotated before the validation. Table 6.1 shows the results of the Validation Process.

It is important to note that Reddit rejected texts are calculated averaging its rejected titles, submissions, and comments. This is because each type of text on Reddit has been considered an independent element. The breakdown of these calculations can be seen in the Appendix C.

As it can be seen in Table 6.1, the average of all rejected texts was around 13.4%. The most rejected posts were from the Google Play Store, while the least rejected posts were from Twitter. This makes sense, since, in the case of Google Play, many of the reviews are very short, many of them consisting of only one word, with reviews such as "*Great!*" or "*Terrible.*". In the case of Twitter, as the search keywords are very specific, the retrieved posts are more in line with the topics extracted from Reddit.

On the other hand, the rejection rate of Reddit posts is almost constant and similar to the total rejection rate. It also highlights that on Twitter non-Uber tweets are the most rejected, while on Google Play the most rejected texts are those about Uber apps.

Cleaning and Validation Processes					
Social Media Source	**Data Source**	**Language**	**Scraped Texts**	**Rejected Texts**	**Rejected Texts (%)**
Reddit	r/uber	English	249,638	36,754	12.833
	r/uberdrivers	English	1,012,822	140,073	13.830
	r/Lyft	English	267,235	33,895	12.684
	r/lyftdrivers	English	373,904	51,527	13.781
	Total Posts	-	**1,903,599**	**262,249**	**13.776**
Twitter	Uber	English	6,223,730	571,831	9.188
	Lyft	English	1,699,520	193,504	11.386
	Cabify	Spanish	325,546	50,407	15.484
	Total Tweets	-	**8,248,796**	**815,742**	**9.890**
Google Play	com.ubercab	English	970,778	290,462	29.921
	com.ubercab	Spanish	551,415	108,872	19.744
	com.ubercab.driver	English	264,010	103,799	39.316
	com.ubercab.driver	Spanish	121,297	29,683	24.471
	me.lyft.android	English	74,342	10,510	14.137
	com.lyft.android.driver	English	27,139	5,758	21.217
	com.cabify.rider	Spanish	57,882	8,809	15.219
	com.cabify.driver	Spanish	21,039	4,976	23.651
	Total Reviews	-	**2,087,902**	**562,869**	**23.959**
			12,240,297	**1,640,860**	**13.405**

Table 6.1: Scraping, Cleaning and Validation Processes Results: The Ride-Hailing Domain

Finally, there is not too much difference between posts written in English and those written in Spanish, although on Twitter it is true that the most rejected tweets are those from Cabify, written in Spanish.

6.3 Evaluation

Once the data are validated, to evaluate transfer learning, it is necessary to annotate some texts according to the topic to which they belong. As the process of annotation is difficult and requires large amounts of time, a small sample of all texts was taken. This sample contains equal sets of texts from all sources. The annotation process was carried out among a group of 6 people with domain knowledge, including the author of this work. Subsequently, the annotations were checked to avoid having large errors. Before constructing the evaluation data set, the texts were cleaned and, as mentioned above, validated. It is important to note that the Spanish texts were labeled directly in Spanish, avoiding possible confusions with the language and to annotate the texts as faithfully as possible to the original text. The mother tongue of all the people who participated in the annotation process is Spanish.

As the LDA algorithm gives a weighted probability of belonging to each of the topics (in this case there are 11), the process of annotation becomes difficult because it is mandatory to choose only one topic for each text. This problem could be solved by building a multilabel classifier with multiclass classification, but this type of classifiers are more complicated, and because of this, in this project the built classifiers are multiclass classifiers with one label per entry. For these reasons, there may be some errors in the annotations, although they have been checked to avoid them.

The texts to be annotated were chosen randomly, selecting 100 random texts from each platform. Those texts that, after several revisions, were not able to reach a concrete conclusion as to which topic they belonged to, either because of the difficulty of annotating them or because the text was too generic, were discarded.

Thus, 1,158 texts were labeled, belonging to all the sets of texts collected. Figure 6.1 shows the distribution of topics in these texts according to what has been annotated.

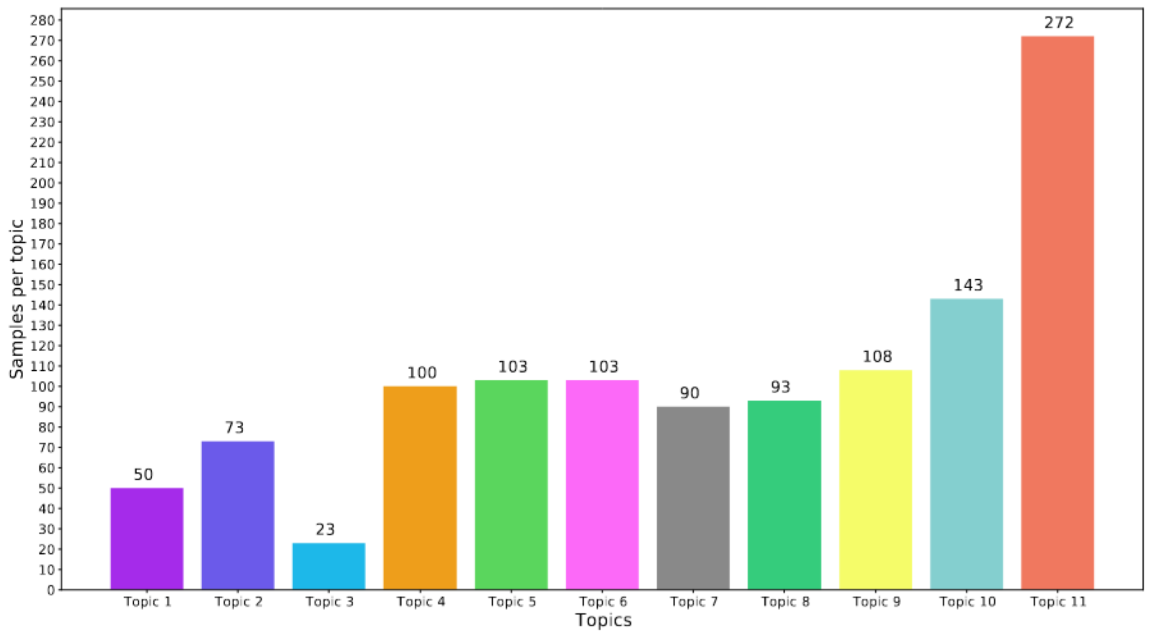


Figure 6.1: Annotated samples per topic

In the above figure highlights that Topic 3 has the lowest number of entries, while Topic 11 has the highest number. Topic 1 and Topic 2 have more than Topic 3 and the other topics have about the same contributions. In addition, Table 6.2 shows the source of the annotated data and the language in which they are written.

	Spanish	English	Total
Reddit	0	437	437
Google Play	256	267	523
Twitter	73	125	198

Table 6.2: Distribution of the annotated data over Source and Language

The evaluation of this transfer learning task consists on analyzing the performance of the Ride-Hailing domain trained models, which were trained on data collected from the subreddits r/uber and r/uberdrivers, when these models predict the topic of new unseen data which can be obtained from other sources.

6.3.1 General Evaluation

The general evaluation consisted of seeing the performance of the models when predicting the topic label for each document on the evaluation data set, without making any distinction by source or language. The performance has been measured using the Precision, the Recall and the F-Score metrics, as well as observing the confusion matrices of every model. The results can be seen in Table 6.3, and a further breakdown of these can be found in the Appendix C.

These results show that there are many differences with the results shown in Table 4.6 and Table 4.7, where the same models were tested with r/uber and r/uberdrivers data and the global performance seemed to be better. These models are trained with data from these subreddits and labeled with the results of the Topic Modeling process, which also used data from these subreddits. Nevertheless, the results shown in Table 6.3 have been obtained testing these models with new unseen data, collected from different platforms, which some of them talk about other companies and which some of them are written in Spanish.

The main difference of these results is that embeddings approaches are much better than Tf-idf approaches, in contrast to Section 4 , where the models based on Tf-idf representations were much better. These results are to be expected since the Tf-idf representation takes into account only the distribution of the n-grams in the different documents, while the embeddings also take into account the semantic similarity of the different words.

It also can be appreciated the importance of the optimization processes in the Gradient Boosting approaches. The Word2Vec 100-dimensional and the FastText 100-dimensional versions are better than the same versions trained with the Logistic Regression algorithm.

		Accuracy	Precission	Recall	F-Score
Logistic Regression	**Word2Vec 100 dim**	63.990	64.857	63.990	64.063
	Word2Vec 300 dim	62.953	64.140	62.953	63.223
	Word2Vec 500 dim	**65.199**	**66.578**	**65.199**	**65.384**
	FastText 100 dim	61.226	62.005	61.226	61.274
	FastText 300 dim	61.572	63.163	61.572	62.020
	FastText 500 dim	60.449	61.506	60.449	60.654
	Bigrams - Tf-idf	55.699	58.185	55.699	56.444
Gradient Boosting	**Word2Vec 100 dim**	64.335	65.614	64.335	64.583
	Word2Vec 300 dim	61.399	62.768	61.399	61.785
	Word2Vec 500 dim	62.781	64.593	62.781	63.265
	FastText 100 dim	61.313	62.661	61.313	61.591
	FastText 300 dim	60.708	62.389	60.708	61.168
	FastText 500 dim	61.054	62.435	61.054	61.355
	Bigrams - Tf-idf	54.922	57.372	54.922	55.362

Table 6.3: Transfer Learning General Evaluation Results

However, this changes when the number of dimensions is increased, where the only Gradient Boosting model that outperforms its same version in Logistic Regression is the FastText 500-dimensional model. However, as stated before, the optimization of these models is not feasible either in time or computational capacity, so these models are simply not good even if they could be optimized.

On the other hand, if the confusion matrices and the results over each topic are taken into account, it can be seen that in every model Topic 3 (Topic 2 in the results shown in the Appendix C because it starts counting from zero), which is strongly related with UberEats, the results are very bad. This happens in all the models, and it shows that this topic is extremely related with the Uber domain and the learning on this topic is hardly transferable to other platforms and companies. In addition, Topic 7 (Topic 6 in Appendix C has also a very high relation with the language used in Reddit. Because of this, models do not predict correctly in other platforms. Nevertheless, Topic 11 and Topic 6 have the higher results in almost every model, and the rest of the topics are classified with acceptable results. This shows that both Topic 3 and Topic 7 are two topics closely related to the original training data and it costs more to adapt them to other sources, but all the other topics, which are more general, are classified in a correct way.

Because of this, the best model is the Logistic Regression Word2Vec 500-dimensional model, with a F-Score of 65.384%, followed by the Gradient Boosting Word2Vec 100-dimensional model, with a F-Score of 64.583%. It can be seen that the Word2Vec versions outperform of their FastText versions in all the models, as was already the case with the results obtained with r/uber and r/uberdrivers.

Thus, it has been proved that, in this domain and with the techniques performed in this project, it is possible to transfer the learning to other platforms, because although the results of the models are lower, the results are acceptable and the performance of the models is correct. It should be noted that there may be errors in the annotations or that the texts have not been chosen as correctly as possible, since only about a thousand texts have been taken for the evaluation. All in all, and although the results could even improve depending on the choice of the evaluation corpus, the results are good and the Transfer Learning in this domain has been proved.

Next, the impact of the language and the different platforms on the Transfer Learning analysis will be analyzed.

6.3.2 Cross-Platform Evaluation

As it can be seen in Table 6.2, there are 437 texts from Reddit, 523 from Google Play, and 198 from Twitter. Google Play and Twitter data contain texts from every application and tweet (hashtags and usernames from the Uber, Lyft, or Cabify domains) collected, but the Reddit data of the annotated corpus has been collected from r/Lyft and r/lyftdrivers, being these texts titles, submissions and comments. To facilitate the understanding of the results, the distribution of the topics of these annotated texts for each platform can be seen in Figure 6.2.

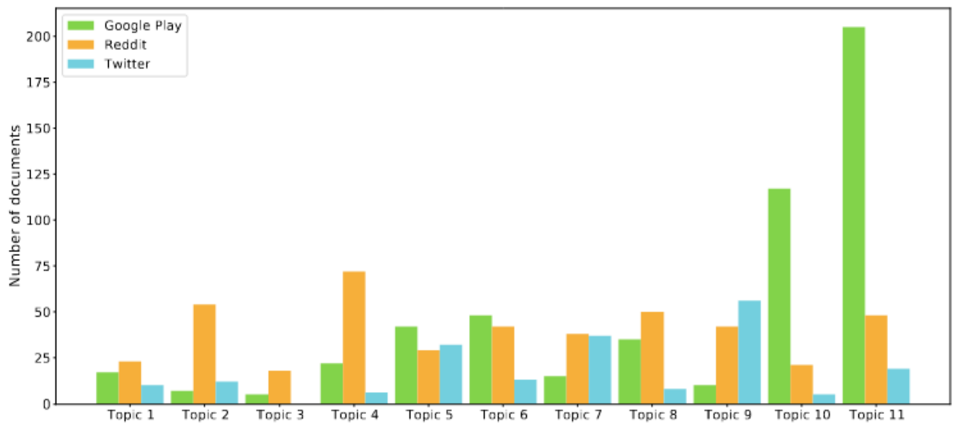


Figure 6.2: Annotated documents per topic and per source

It can be seen that Google Play data set has a lot of texts belonging to Topic 10 and Topic 11, while Reddit data set has more distributed values. In addition, Twitter data set is also distributed, but with less annotated texts.

Table 6.4 shows the results of the Cross-Platform evaluation for each model when the topic of each text of the annotated data set is predicted.

		Google Play	**Twitter**	**Reddit**
Logistic Regression	**Word2Vec 100dim**	68.628	59.028	62.996
	Word2Vec 300dim	68.506	58.633	61.233
	Word2Vec 500dim	**70.944**	61.598	62.603
	FastText 100dim	68.002	54.478	58.717
	FastText 300dim	68.687	53.154	59.664
	FastText 500dim	68.036	51.137	58.301
	Bigrams - Tf-idf	61.819	50.722	55.530
Gradient Boosting	**Word2Vec 100dim**	69.506	59.809	**63.048**
	Word2Vec 300dim	66.445	55.331	61.250
	Word2Vec 500dim	66.795	**62.213**	61.844
	FastText 100dim	65.439	58.217	60.534
	FastText 300dim	64.832	56.483	60.698
	FastText 500dim	67.284	53.360	60.166
	Bigrams - Tf-idf	61.445	46.229	55.655

Table 6.4: F-Score results in every platform data set

The above table shows some interesting results. First of all, Google Play data has the best results in terms of classification. In addition, Twitter data has the worst results. On the other hand, Reddit texts remain above 60% for almost all models but without reaching the very good values of the Google Play review rankings.

The good results of the reviews come from the fact that most of the texts talk about topics related to the application, such as Topic 10 and Topic 11. If we look at the confusion matrices and the performance tables of each of the models developed in this work, available in Appendix C, we see that Topic 11 is one of the topics with the best results in all the models, reaching 80% F-Score in some cases. Therefore, the good results of the Google Play texts are not only a matter of the texts that have been chosen to annotate, but in Google Play, being a mobile app store, topics 10 and 11 will tend to be more assiduously talked about. Therefore, with the developed models, the knowledge learned in Reddit with posts related to Ride-Hailing can be transferred to Google Play texts with good results in general.

In Twitter data, the only models in which F-Score exceeds 60% is in the Word2Vec 500-dimensional versions in both algorithms. Although Twitter results are not very good results, they are similar to the Reddit results.

These results imply that the application of the developed Machine Learning models to other sources is not overly good, but it is applicable and can classify posts reasonably well. However, the results are strongly related to the topics, so those sources that talk more about those topics will tend to get a more efficient knowledge transfer.

In addition, the results show that Tf-idf-based models are worse than the embeddings approach used in this project. This is of course to be expected due to the computation of word similarity in the case of embeddings. Moreover, the FastText approaches have proved to be worse than those of Word2Vec throughout this project, at least for the performance and approach of the developed system.

Furthermore, it is clear that, in addition to the relationship between the social network theme and the ranking performance, posts that are more similar to the originals will perform better. For example, in this case, as Google Play reviews are written more similarly to Reddit posts than tweets, the latter will perform worse.

6.3.3 Cross-Language Evaluation

		Google Play	**Twitter**
Logistic Regression	**Word2Vec 100dim**	65.392	**63.032**
	Word2Vec 300dim	64.202	59.334
	Word2Vec 500dim	**66.997**	61.830
	FastText 100dim	66.733	51.670
	FastText 300dim	66.337	51.960
	FastText 500dim	64.951	54.092
	Bigrams - Tf-idf	55.511	44.314
Gradient Boosting	**Word2Vec 100dim**	64.585	61.536
	Word2Vec 300dim	62.010	57.457
	Word2Vec 500dim	61.448	59.930
	FastText 100dim	62.072	61.181
	FastText 300dim	61.975	49.718
	FastText 500dim	64.143	54.829
	Bigrams - Tf-idf	57.996	41.927

Table 6.5: F-Score results in every platform data set with translated Spanish posts

In this case, the analyzed posts have been translated before the predictions, but, as stated before, these posts were annotated before this translation process by native Spanish people. The results are so similar to those explained in the above section, showing that Google Play predictions are better than Twitter predictions because of the reasons explained before.

There are even models that work better on Twitter for the Spanish texts than for the predictions of all of them, although this claim can not be properly justified since there are not enough texts.

Even so, what this section does show is that the automatic translation of texts from different social media from Spanish to English does not deliberately influence the classification process based on embeddings. Therefore, machine translation is a process that, if used correctly, allows transferring the knowledge learned on English data to other Spanish data without losing relevant information.

6.4 Results

After all the experiments and evaluations, it was decided to implement the Word2Vec 500-dimensional classification model in the overall system architecture for post classification and analysis. To show how the results would look like, some images of some results obtained from the visualization interface of the system will be put here. The analyzed data were collected over 15 days, specifically from May 24, 2021 to June 9, 2021.

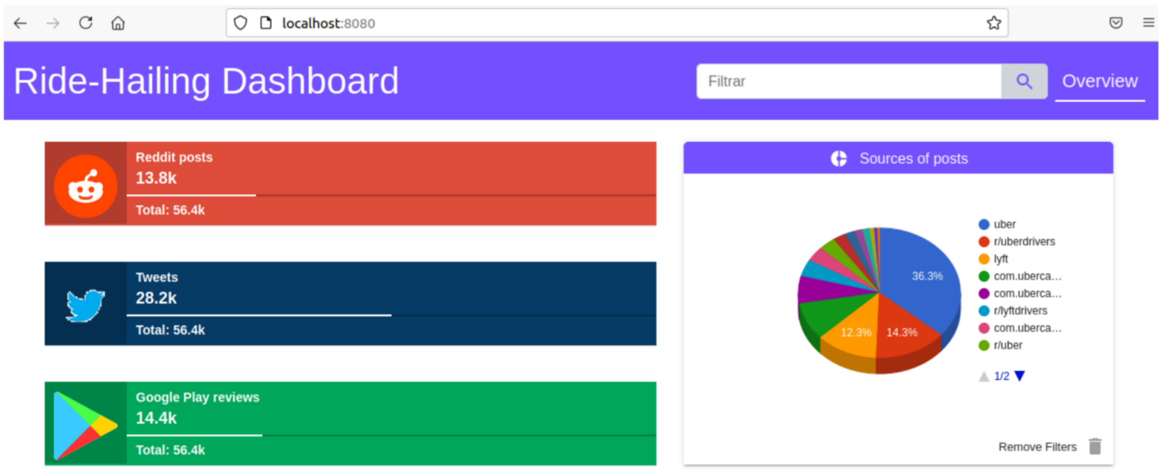


Figure 6.3: General Analysis: Weight of the sources

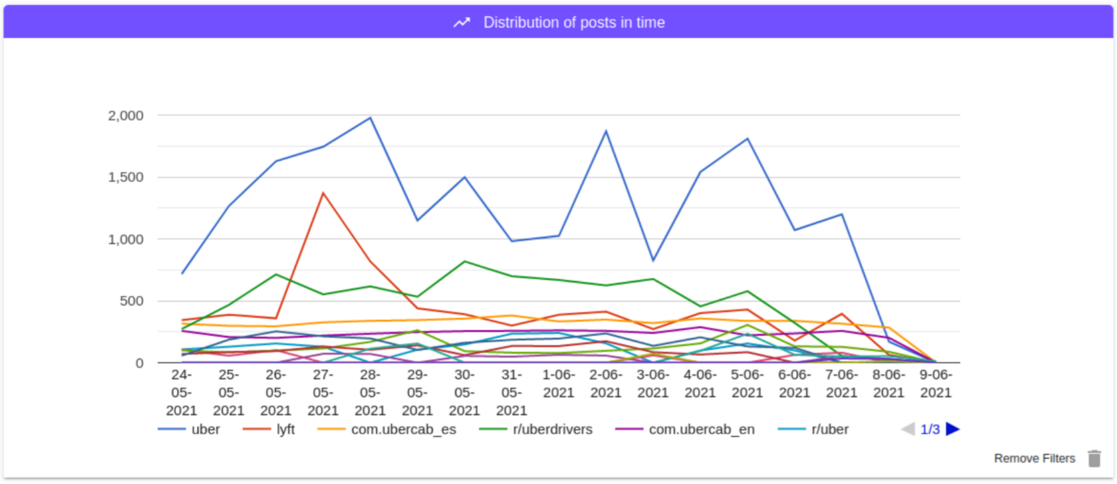


Figure 6.4: General Analysis: Number of posts collected by date and source

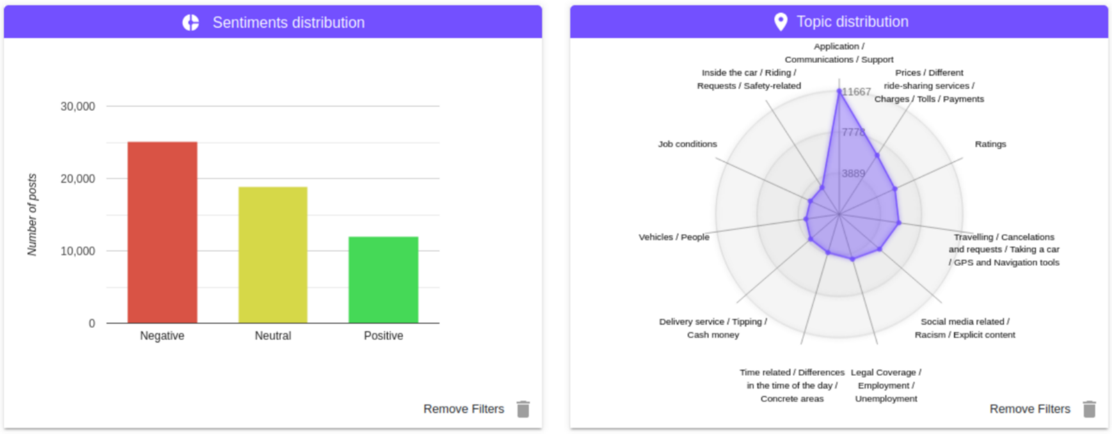


Figure 6.5: General Analysis: Topics and Sentiment Analysis Representation

During these 15 days, posts have been collected from the 3 social networks that have been studied during this project. It can be seen that Twitter has published the most related posts and that tweets of Uber have been the most collected, followed by r/uberdrivers and the tweets of Lyft.

On the other hand, there are no excessively large peaks in terms of data collected during these days, so it does not seem that there has been a particularly interesting event during this time. Most of the posts are negative, as expected. Furthermore, you can see that most of the posts have to do with the application or support or are related to pricing.

These results are to be expected and are consistent with what has been studied so far. Now, the results of the sentiment analysis and the topics for each source will be shown to try to make a more exhaustive analysis.

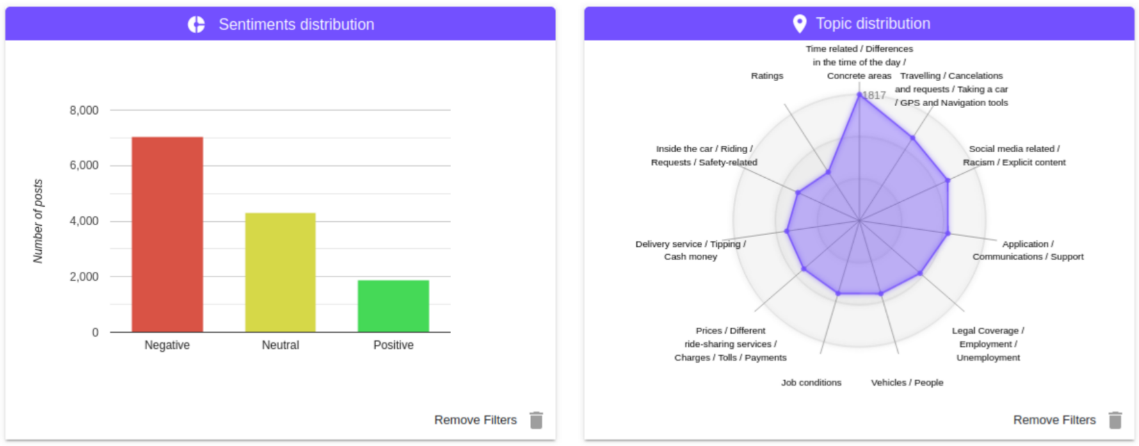


Figure 6.6: Reddit Analysis: Topics and Sentiment Analysis Representation

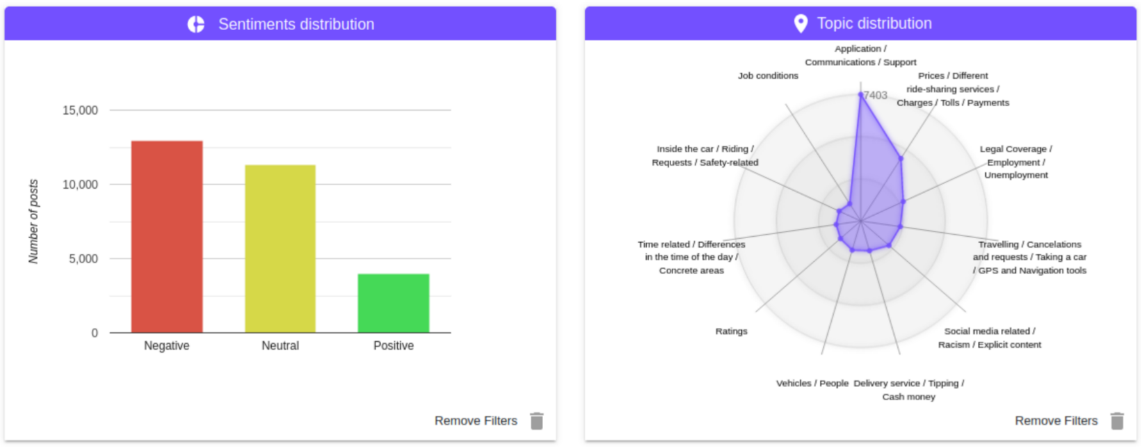


Figure 6.7: Twitter Analysis: Topics and Sentiment Analysis Representation

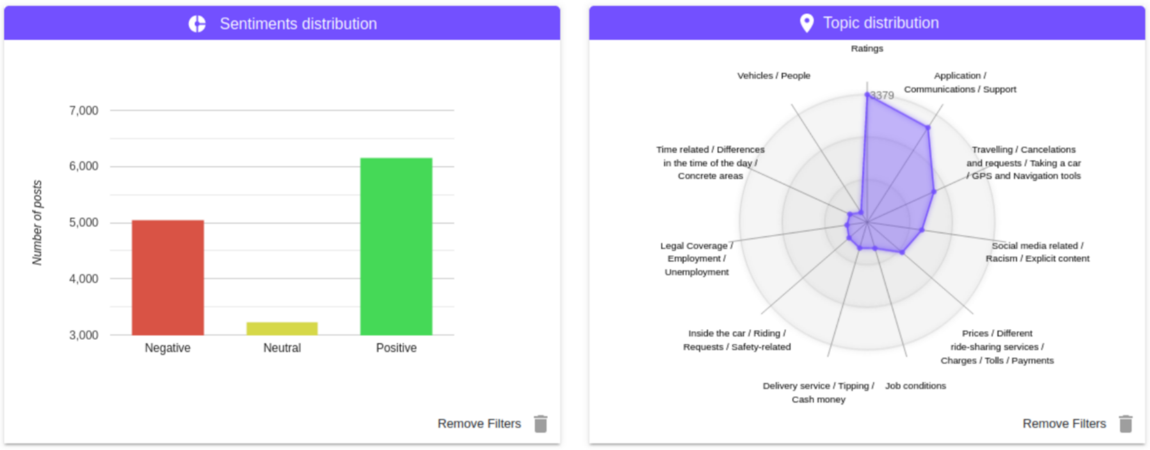


Figure 6.8: Google Play Analysis: Topics and Sentiment Analysis Representation

Firstly, sentiment analysis shows different features for each source. In Reddit and Twitter data, negative posts predominate, while on Google Play it is the positive ones that predominate. Likewise, on Twitter there are more neutral posts than on Reddit. In terms of topics, Reddit is dominated by the time-related topic, the travelling-related topic, and the social media-related topic, while the Ratings topic is the one that appears the least. It is noteworthy that most of the topics seem to have similar contributions. Twitter is clearly dominated by the application-related topic and also posts about the topic related to prices, among others that have lower contributions. Finally, Google Play reviews are clearly dominated by the Rating topic and the application-related topic, as it is a social network for mobile application reviews.

From all this, it can be seen that the system works correctly and shows the expected results according to each type of source. Even so, more exhaustive analysis can be done, searching by specific words, by entities, identifying other topics within each topic and more. All this can be done with the real-time system and the dynamic visualization system implemented in this project.

CHAPTER 7

Conclusions and Future Work

This chapter will describe the achieved goals done by the master thesis, as well as the problems encountered and what is left for future work.

7.1 Conclusions

In this project, an opinion mining system has been developed. This system, as it has been demonstrated throughout the entire work, is a multidomain and multiplatform system, which its performance does not depend on the content of the texts or even the source from which they originate, although this of course means that, for the system to function correctly, it is necessary to design a series of specific models for it.

To prove this, texts from the Ride-Hailing domain and the Radicalization domain and from different sources have been used to feed the system. The analysis of these domains before and after the processing of the texts shows the feasibility and usefulness of applying the developed system to domains such as these. Of course, it has been shown the utilities and benefits that this system can have both in the business world and in more social issues.

Therefore, this system could be useful to Cabify, which can use the system to monitor in a certain way the interactions in social networks related to its application or services, being able to improve its services based on what their users want or analyzing the complaints and opinions of users of competing companies.

7.2 Achieved Goals

The achieved goals for this project are the following ones:

- **Develop of the whole system:** The whole system has been developed, including every process and virtualized module.
- **Develop of the visualization module:** An interactive visualization module web-based has been developed. This module will be the most useful part of the system, because users will be able to interact directly with it to extract customized analyses.
- **Multi-domain nature of the system:** The multidomain nature of the system has been demonstrated, analyzing both the Ride-Hailing and the Radicalization domains, obtaining good results.
- **Multi-platform nature of the system:** It has been proven and explained how this system can work with different social media sources at the same time.
- **Cross-Platform and Cross-Language system:** It has been studied how is the performance of the system when is fed with texts from a specific source and in a specific language and then used with texts from other sources or written in different languages.
- **Analysis of the Ride-Hailing and the Radicalization domain using the developed system:** The analysis of these domains has been done using the processes implemented in the system. It has been seen the differences between both domains and the opinion of users, what they talk about, and how they talk about the different topics extracted from each domain.

7.3 Problems Encountered

Several problems have appeared during the development of the project. The most important problems are explained below.

- **Limited computing capacity:** The generation of the models designed in this project is time-consuming and computationally intensive. The amount of texts used to train the models, over 1,000,000 in many cases, increases the difficult of the training task. However, although these parts of the project have been executed on a dedicated machine with large capacity, the required time of some processes was totally unacceptable, so the decision was made to limit the number of models and not to generate all the models that were originally going to be generated.

- **Large amounts of data:** The performance of the system is based on the training with large amounts of data and the collection of large amounts too. This can be difficult to work with this data and leads to certain decisions, such as limiting this data or splitting it up to work with it separately and then merging it back together again. In addition, the dataset cleaning processes have been exhaustive, both with the texts as well as checking that there are no duplicated or invalid elements, to have the necessary amount of texts and no more.

- **Different sources of data:** As this project has been centered in NLP techniques, the source of data is essential to process it. As in this project different sources of data have been analyzed at the same time and each one needs a specific preprocessing and treatment, some difficulties have arisen and some processes have had to be redesigned or variations have had to be implemented depending on the source or language of the data.

- **Situations arising from the COVID-19 crisis:** Teleworking, the difficulty of being able to go to the School and above all the impossibility of communicating with department colleagues or professors without a telematic meeting has complicated the development of the project.

7.4 Future Work

The development of the project has covered many topics, achieving all the objectives initially set. However, some new branches of research and development have been emerging during the development of the project, for which it was complicated to develop for this project and which is proposed as a continuation of the project.

- **Multi-class and Multi-label classification:** The machine learning models developed in this project have the function of classifying among numerous classes, 11 or 8 classes depending on the domain, which are the topics extracted from the LDA. It is difficult to annotate these texts with only one of these topics, since the output of the LDA is the weight of the text in each of the topics. Therefore, it is proposed to extend the system so that the classification algorithms classify each text, instead of only in one class, in several. Therefore, multiclass (all the topics that can exist) and multilabel (all the topics to which a text can belong) models would be developed.

- **Training models using different algorithms:** Two classification algorithms have been used in this project: Logistic Regression and Gradient Boosting. It is proposed to use others, such as Random Forest or other more modern ensembles, such as AdaBoost.

- **Extend the system to other domains:** Two domains have been studied in this project: the Ride-Hailing and the Radicalization domain. It is proposed to extend these domains to others such as technological companies, economics, or more social issues such as bullying prevention or mistreatment in social media.
- **Adding more processes, such as Emotion Analysis:** The system could analyze more features and thus be more robust. One of these new features could be the Emotion Analysis.
- **Adding more features to the classification models:** The results of the Sentiment Analysis or NER extraction could be introduced as features in the classification algorithms because it has already been seen that, especially the NER, is closely related to the extraction of topics and could improve the results.
- **Extend the system to other languages and other platforms:** Other languages, such as French or Germany, could be introduced to the system and translated with another translation model. In addition, different social media platforms could be used, such as Facebook or the Apple Store.

APPENDIX A

Ride-Hailing use case Appendix

In this appendix, some figures and explanations of the Ride-Hailing use case that did not fit in the original body of this document will be shown here.

- **1. Data retrieved per date and per source.**

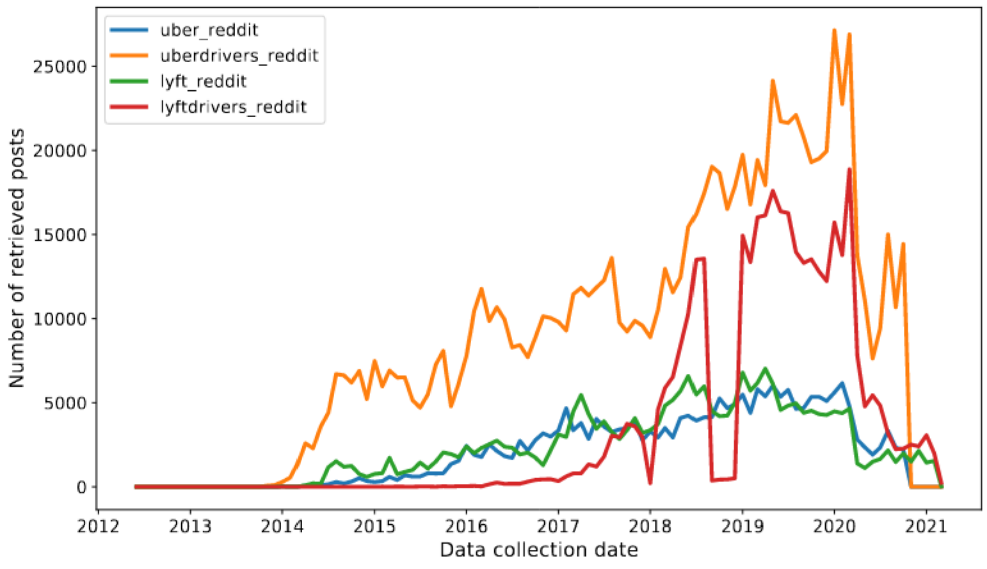


Figure A.1: Total Reddit data retrieved per date

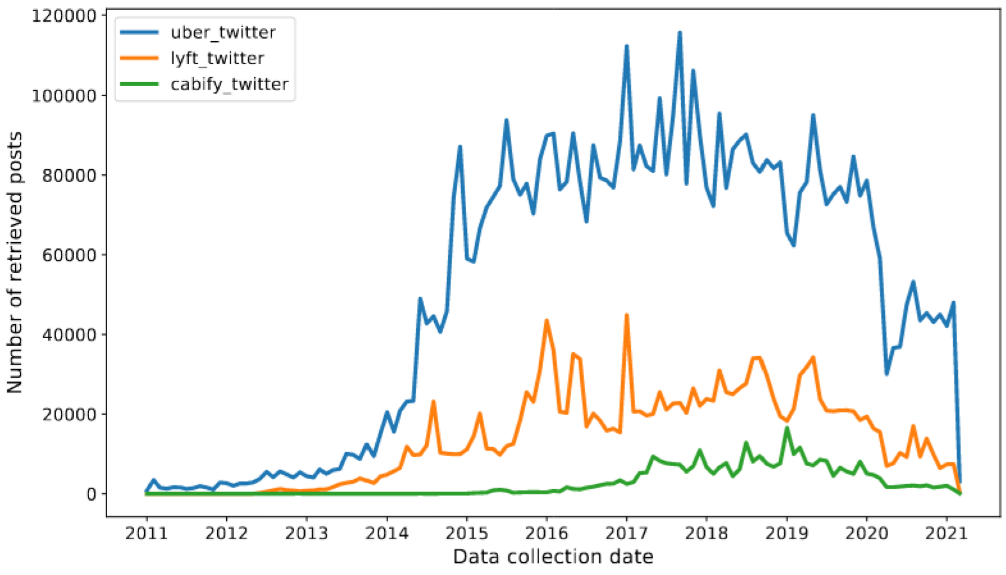


Figure A.2: Total Twitter data retrieved per date

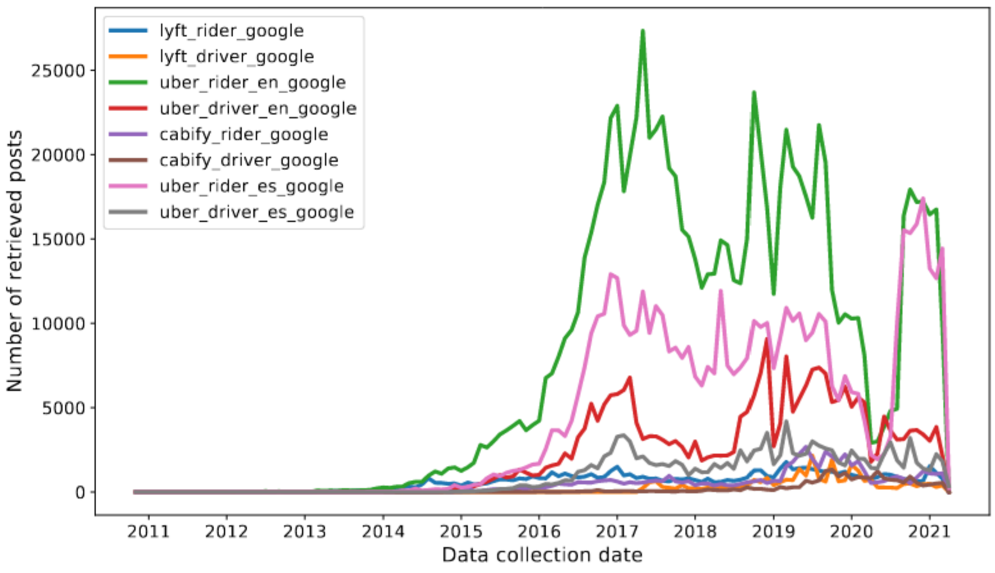


Figure A.3: Total Google Play data retrieved per date

- **2. Word Embeddings representations with 300 and 500 dimensions.**

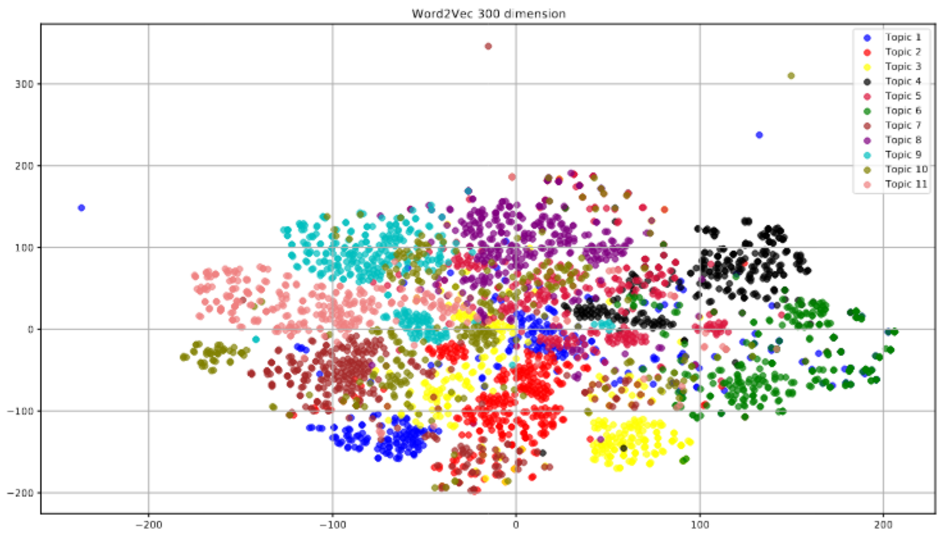


Figure A.4: Word2Vec 300-dimensional model PCA representation

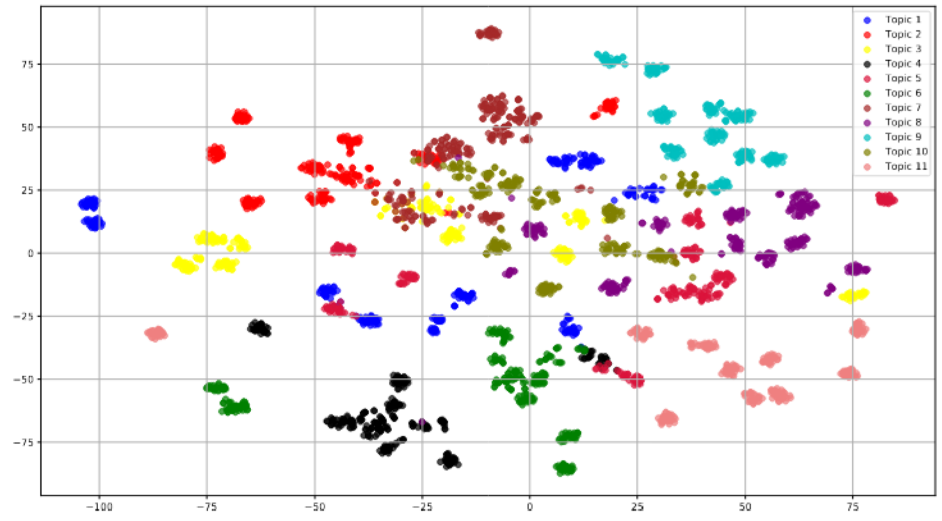


Figure A.5: FastText 300-dimensional model PCA representation

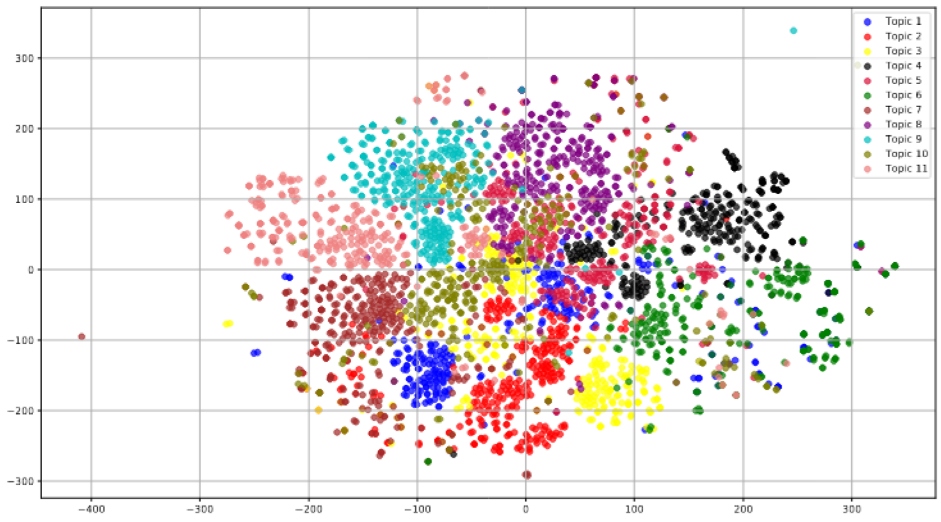


Figure A.6: Word2Vec 500-dimensional model PCA representation

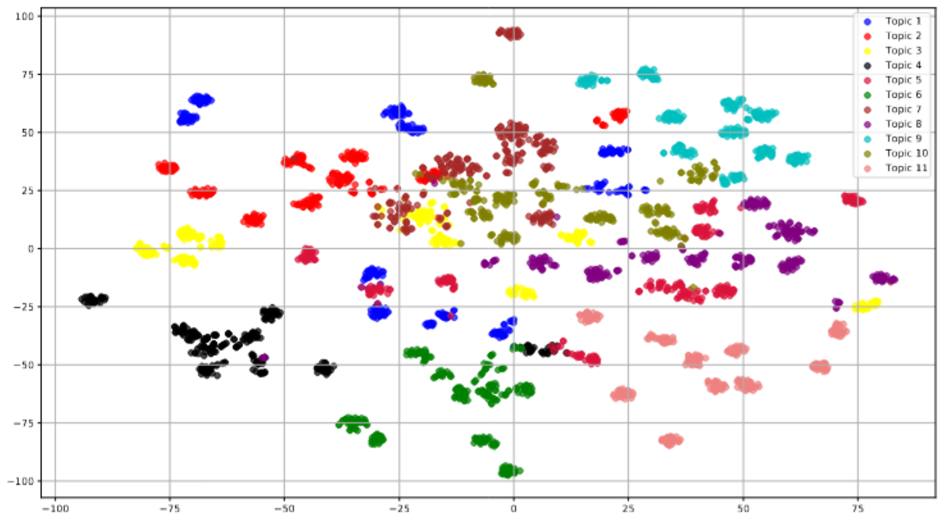


Figure A.7: FastText 500-dimensional model PCA representation

- **3. Used lexicon on the PCA Word Embeddings representation.**
  - **Topic 1:** *pool, ride, convers, passeng , prefer, music, talk, xl, declin, radio*
  - **Topic 2:** *fuck, drunk, clean, dude, kid, girl, seat, woman, cop, smell*
  - **Topic 3:** *tip, yeah, eat, cash, food, deliveri, restaur, appreci, awesom, yep*
  - **Topic 4:** *hour, week, day, weekend, morn, quest, saturday, friday_saturday, summer, vega*
  - **Topic 5:** *fare, market, price, cab, increas, nyc, uberx, promot, incent, adjust*
  - **Topic 6:** *minut, wait, ping, pickup, min, drop, locat, street, arriv, block*
  - **Topic 7:** *lol, post, sub, reddit, idiot, troll, wow, dumb, sound, news*
  - **Topic 8:** *money, expens, tax, gas, buy, deduct, profit, incom, spend, full_time*
  - **Topic 9:** *state, employe, polici, law, legal, rule, accid, compani, coverag, contract*
  - **Topic 10:** *rate, star, low, shitti, poor, entiti, bad, improv, give, deserv*
  - **Topic 11:** *phone, account, email, support, updat, contact, info, card, link, messag*

- **4. Genetic Algorithm for Gradient Boosting optimization (100-dimensional models).**

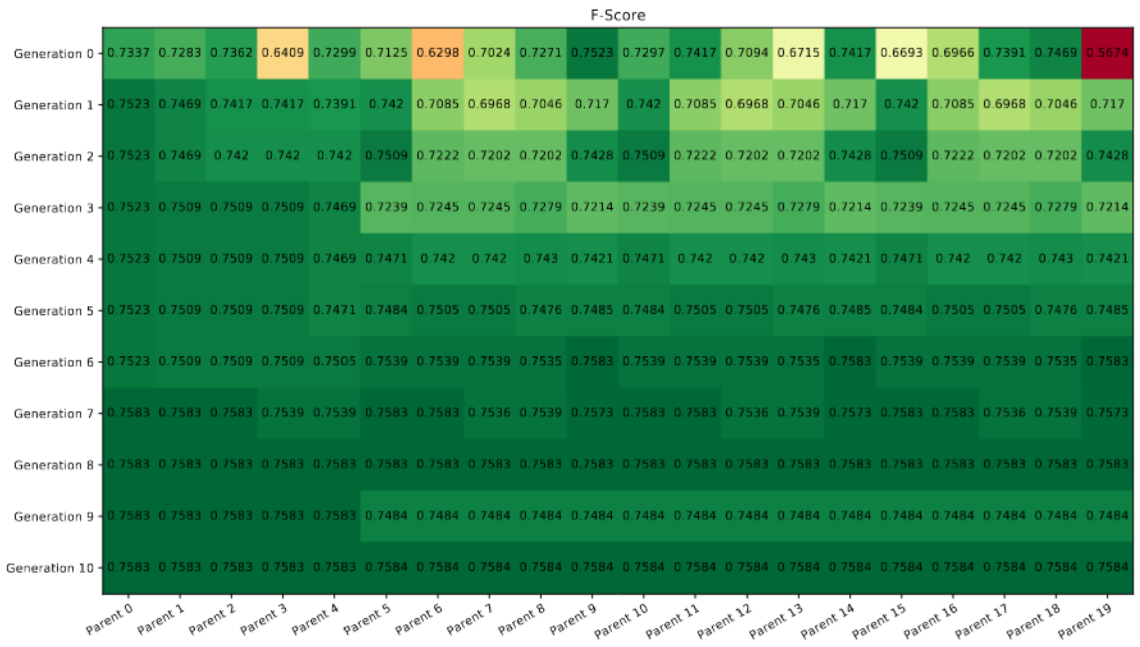
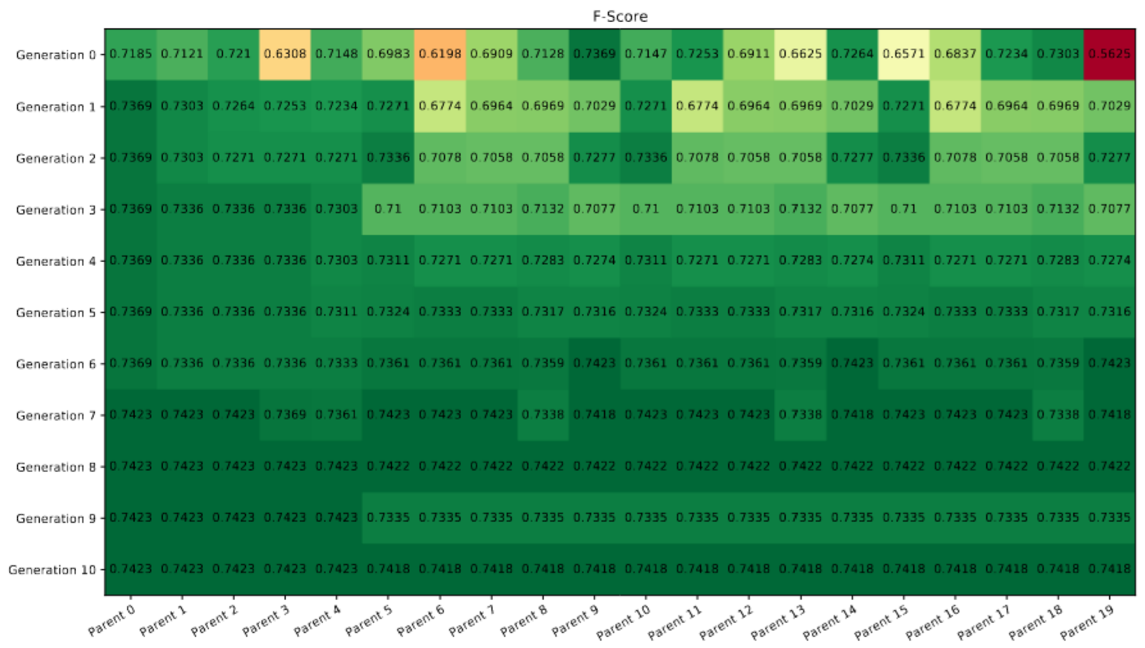


Figure A.8: Genetic Algorithm for XGBoost optimization: Word2Vec - 100-dim



	Parent 0	Parent 1	Parent 2	Parent 3	Parent 4	Parent 5	Parent 6	Parent 7	Parent 8	Parent 9	Parent 10	Parent 11	Parent 12	Parent 13	Parent 14	Parent 15	Parent 16	Parent 17	Parent 18	Parent 19
Generation 0	0.7185	0.7121	0.721	0.6308	0.7148	0.6983	0.6198	0.6909	0.7128	0.7369	0.7147	0.7253	0.6911	0.6625	0.7264	0.6571	0.6837	0.7234	0.7303	0.5625
Generation 1	0.7369	0.7303	0.7264	0.7253	0.7234	0.7271	0.6774	0.6964	0.6969	0.7029	0.7271	0.6774	0.6964	0.6969	0.7029	0.7271	0.6774	0.6964	0.6969	0.7029
Generation 2	0.7369	0.7303	0.7271	0.7271	0.7271	0.7336	0.7078	0.7058	0.7058	0.7277	0.7336	0.7078	0.7058	0.7058	0.7277	0.7336	0.7078	0.7058	0.7058	0.7277
Generation 3	0.7369	0.7336	0.7336	0.7336	0.7303	0.71	0.7103	0.7103	0.7132	0.7077	0.71	0.7103	0.7103	0.7132	0.7077	0.71	0.7103	0.7103	0.7132	0.7077
Generation 4	0.7369	0.7336	0.7336	0.7336	0.7303	0.7311	0.7271	0.7271	0.7283	0.7274	0.7311	0.7271	0.7271	0.7283	0.7274	0.7311	0.7271	0.7271	0.7283	0.7274
Generation 5	0.7369	0.7336	0.7336	0.7336	0.7311	0.7324	0.7333	0.7333	0.7317	0.7316	0.7324	0.7333	0.7333	0.7317	0.7316	0.7324	0.7333	0.7333	0.7317	0.7316
Generation 6	0.7369	0.7336	0.7336	0.7336	0.7333	0.7361	0.7361	0.7361	0.7359	0.7423	0.7361	0.7361	0.7361	0.7359	0.7423	0.7361	0.7361	0.7361	0.7359	0.7423
Generation 7	0.7423	0.7423	0.7423	0.7369	0.7361	0.7423	0.7423	0.7423	0.7338	0.7418	0.7423	0.7423	0.7423	0.7338	0.7418	0.7423	0.7423	0.7423	0.7338	0.7418
Generation 8	0.7423	0.7423	0.7423	0.7423	0.7423	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422	0.7422
Generation 9	0.7423	0.7423	0.7423	0.7423	0.7423	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335	0.7335
Generation 10	0.7423	0.7423	0.7423	0.7423	0.7423	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418	0.7418

Figure A.9: Genetic Algorithm for XGBoost optimization: FastText - 100-dim

- **5. NER Analysis per type of text and per subreddit**

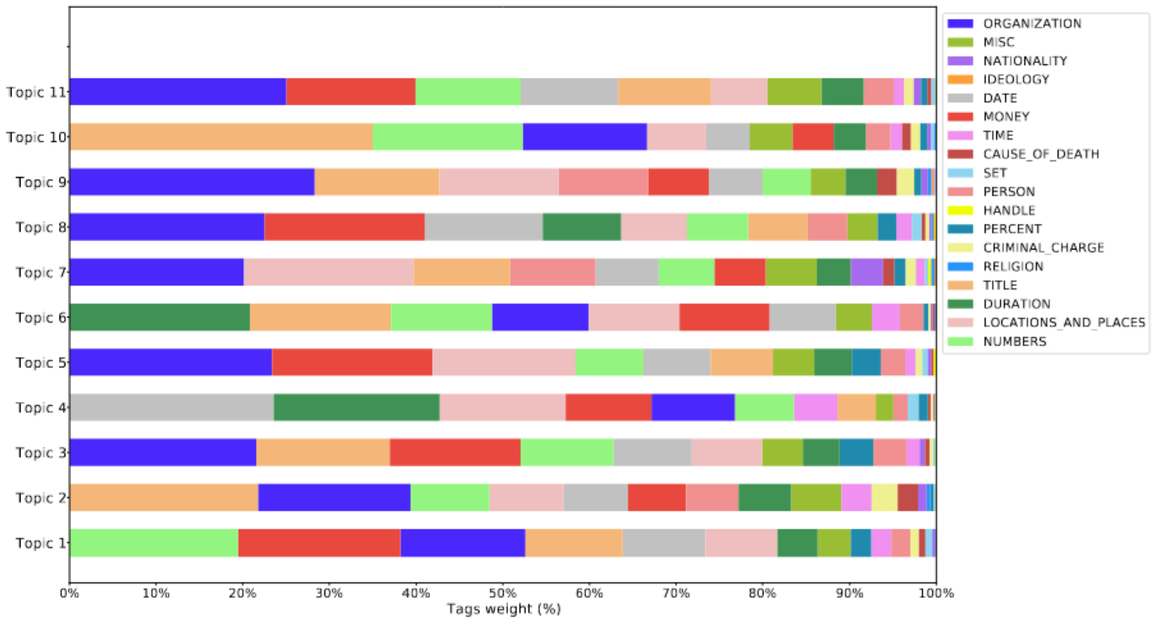


Figure A.10: NER tags in r/uber titles

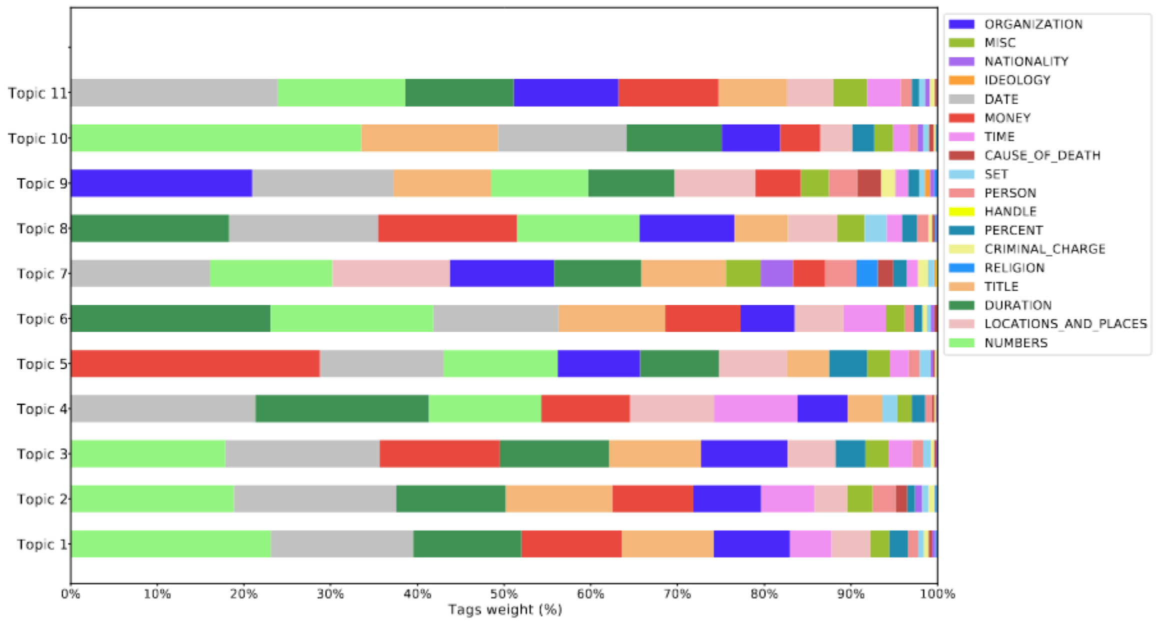


Figure A.11: NER tags in r/uber post texts

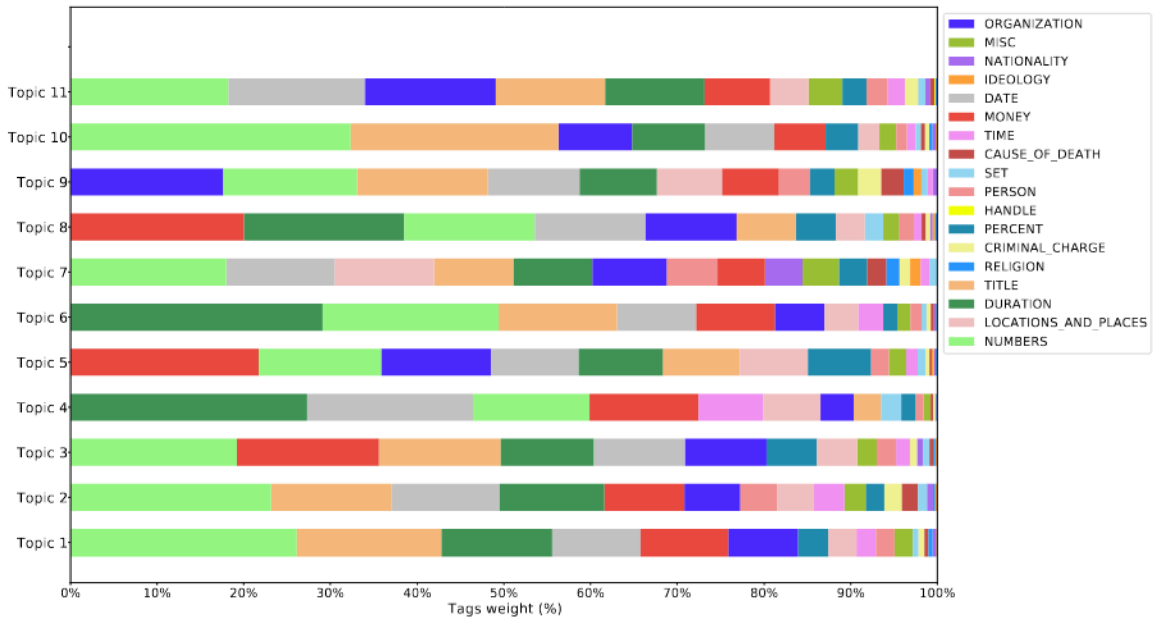


Figure A.12: NER tags in r/uber comments

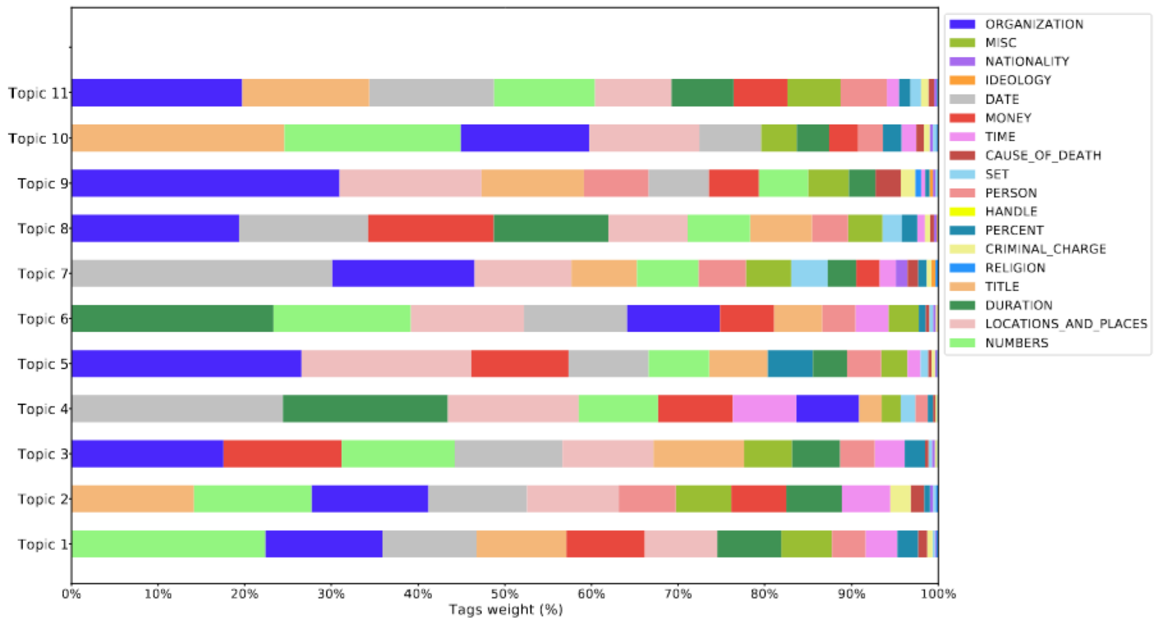


Figure A.13: NER tags in r/uberdrivers titles

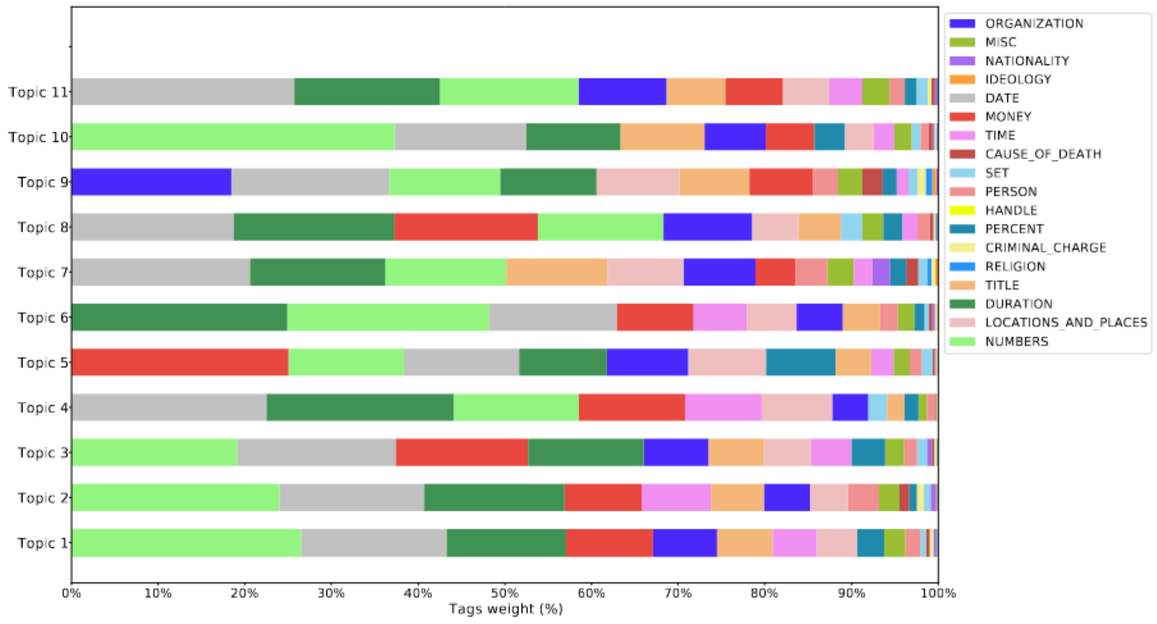


Figure A.14: NER tags in r/uberdrivers post texts

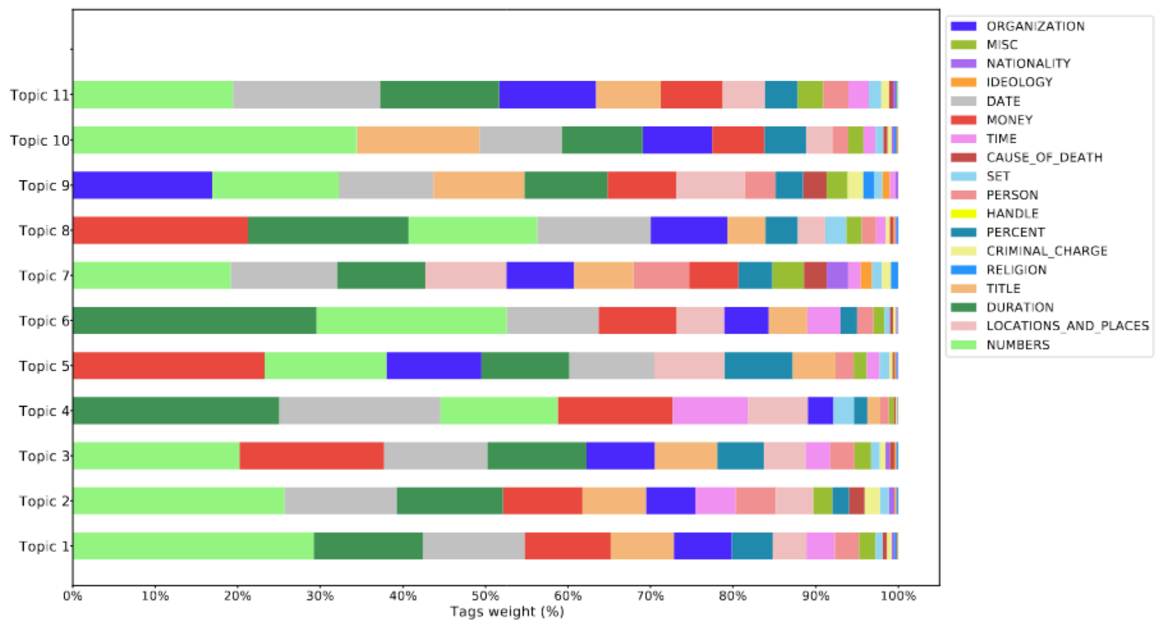


Figure A.15: NER tags in r/uberdrivers comments

APPENDIX B

Radicalization use case Appendix

In this appendix, some figures and explanations of the Radicalization use case that did not fit in the original body of this document will be shown here.

- **1. Word Embeddings representations with 300 and 500 dimensions.**

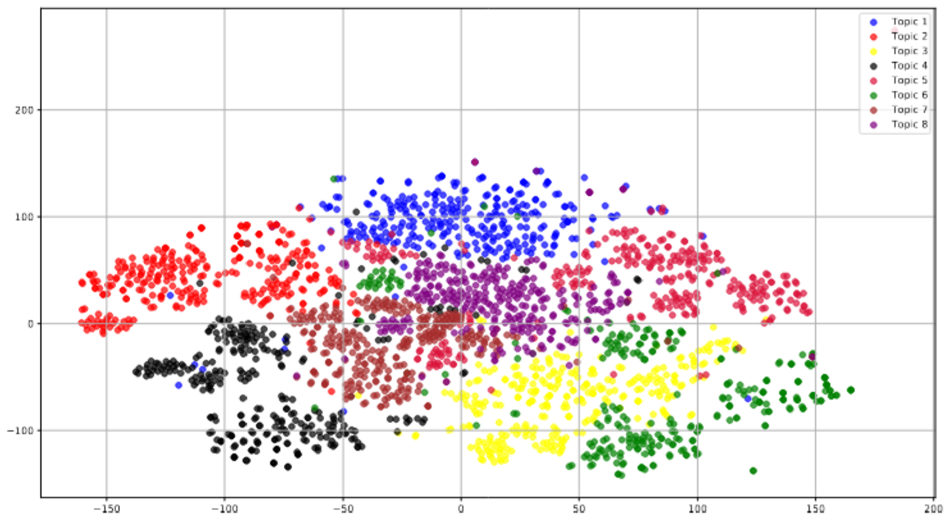


Figure B.1: Word2Vec 300-dimensional model PCA representation - Radicalization domain

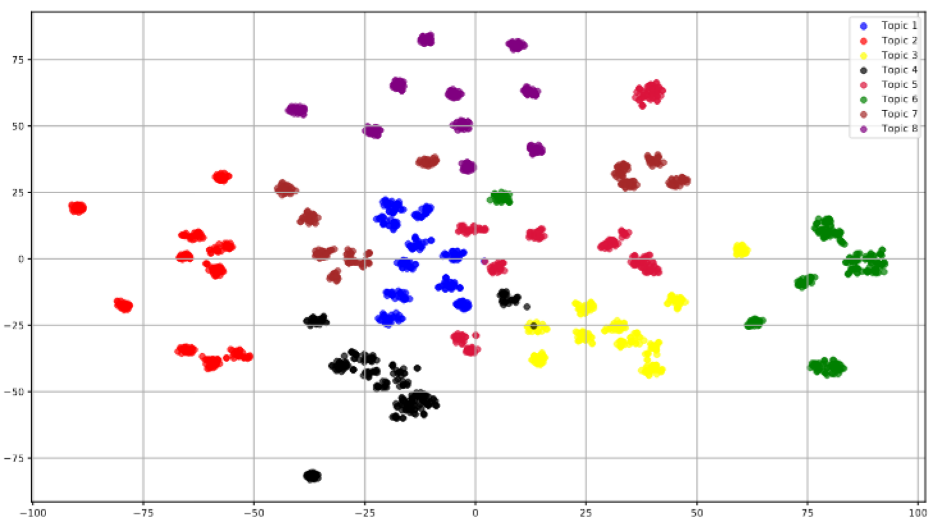


Figure B.2: FastText 300-dimensional model PCA representation - Radicalization domain

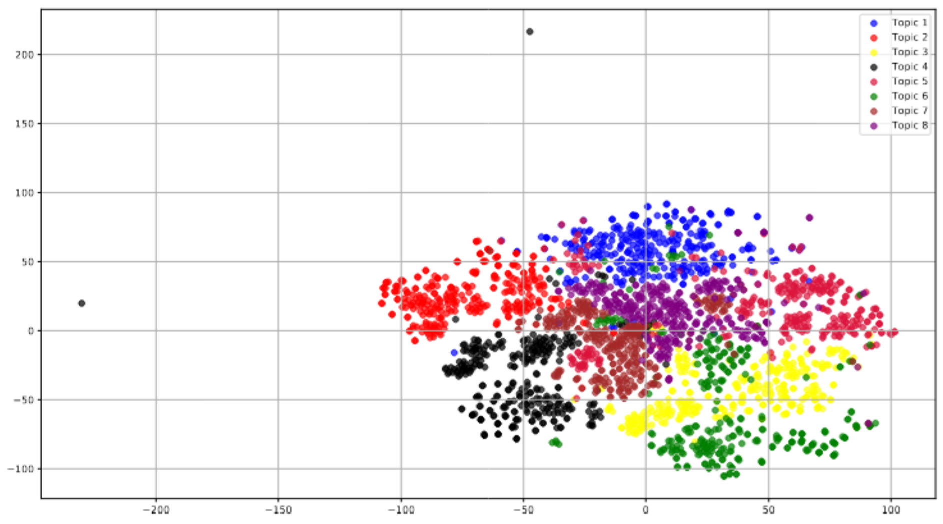


Figure B.3: Word2Vec 500-dimensional model PCA representation - Radicalization domain

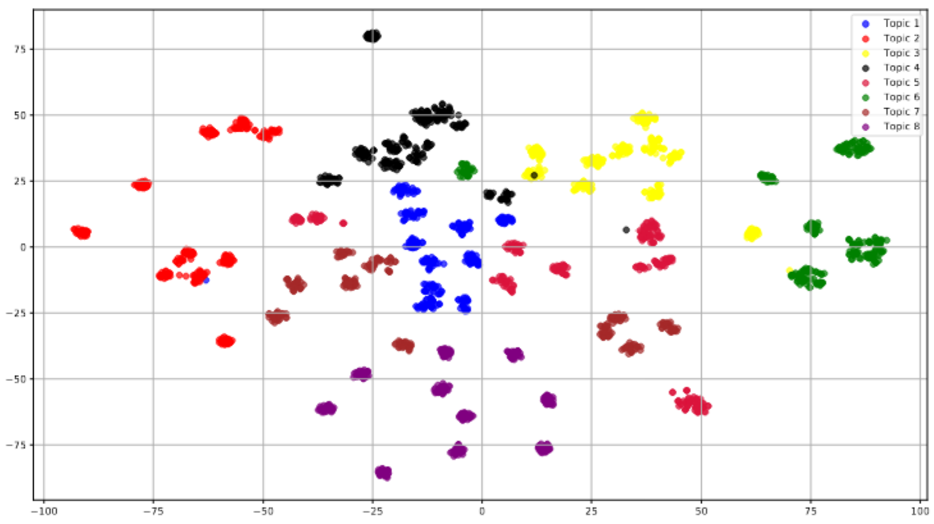


Figure B.4: FastText 500-dimensional model PCA representation - Radicalization domain

- **2. Used lexicon on the PCA Word Embeddings representation.**
  - **Topic 1:** *#libya, #usa, #yemen, afghanistan, #lebanon, #iran, #middleeast, @realdonaldtrump, #saudiarabia, #europ*
  - **Topic 2:** *obama, palestinian, israel, gaza, jewish, racist, palestin, #alllivesmatt, #berniesand_#itsoktobewhit, #femin_#cnn*
  - **Topic 3:** *#brotherhood, game, guy, boy, son, mother, miss, father, fuck, old*
  - **Topic 4:** *public, research, program, busi, social, job, econom, challeng, polit, leadership*
  - **Topic 5:** *#idlib, assad, russia, #assad, idlib, russian, syrian, suffer, assad_regim, #damascus*
  - **Topic 6:** *islam_state, isi, fighter, turkish, milit, near, #sdf, soldier, captur, explos*
  - **Topic 7:** *#kurdistan, muse_#iraq, deal, minist, review, author, kurdistan, demonstr, prime_minist, iraq*
  - **Topic 8:** *#islam, #muslim, islam, allah, #quran, religion, #ramadan, #christian, #allah, god*

APPENDIX C

Transfer Learning Appendix

In this Appendix Section the results of the Transfer Learning approach are shown. These results corresponds to the confusion matrices of the algorithms with the different models, as well as the breakdown of the metrics for each of the topics.

- **1. Breakdown of Reddit data.**

Complete Reddit Data Breakdown					
Subreddit	**Endpoint**	**Text Type**	**Scraped Texts**	**Rejected Texts**	**Rejected Texts (%)**
r/uber	**Comments**	Body	203,825	23,358	11.460
	Submissions	Body	22,850	10,932	47.607
		Title	22,963	2,464	10.730
	Total	-	**249,638**	**36,754**	**12.833**
r/uberdrivers	**Comments**	Body	877,013	102,575	11.700
	Submissions	Body	67,753	29,313	43.072
		Title	68,056	8,185	12.027
	Total	-	**1,012,822**	**140,073**	**13.830**
r/Lyft	**Comments**	Body	226,757	24,349	10.738
	Submissions	Body	20,239	7,313	36.133
		Title	20,239	2,233	11.033
	Total	-	**267,235**	**33,895**	**12.684**
r/lyftdrivers	**Comments**	Body	339,002	41,067	12.114
	Submissions	Body	17,451	8,055	46.158
		Title	17,451	2,405	13.781
	Total	-	**373,904**	**51,527**	**13.781**
			1,903,599	**262,249**	**13.776**

Table C.1: The Validation Process in all Reddit data

- **2. Model results: Classification metrics and Confussion matrices**

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	51.613	64.000	57.143	50
1	-	63.014	63.014	63.014	73
2	-	22.917	47.826	30.986	23
3	-	62.037	67.000	64.423	100
4	-	67.089	51.456	58.242	103
5	-	71.311	84.466	77.333	103
6	-	44.872	38.889	41.667	90
7	-	51.546	53.763	52.632	93
8	-	73.333	61.111	66.667	108
9	-	59.055	52.448	55.556	143
10	-	79.927	80.515	80.220	272
Total	**63.990**	**64.857**	**63.990**	**64.063**	**1158**

$$\begin{pmatrix}
32 & 4 & 2 & 2 & 0 & 3 & 2 & 1 & 0 & 1 & 3 \\
2 & 46 & 2 & 1 & 1 & 5 & 1 & 3 & 3 & 2 & 7 \\
0 & 0 & 11 & 3 & 1 & 0 & 0 & 1 & 1 & 3 & 3 \\
2 & 1 & 0 & 67 & 8 & 8 & 2 & 3 & 1 & 3 & 5 \\
2 & 3 & 0 & 5 & 53 & 3 & 5 & 13 & 3 & 6 & 10 \\
4 & 1 & 1 & 3 & 1 & 87 & 1 & 0 & 0 & 3 & 2 \\
5 & 15 & 8 & 5 & 1 & 2 & 35 & 6 & 3 & 6 & 4 \\
3 & 1 & 6 & 8 & 4 & 1 & 1 & 50 & 2 & 13 & 4 \\
3 & 1 & 2 & 3 & 2 & 2 & 6 & 9 & 66 & 4 & 10 \\
2 & 1 & 12 & 8 & 5 & 4 & 14 & 9 & 6 & 75 & 7 \\
7 & 0 & 4 & 3 & 3 & 7 & 11 & 2 & 5 & 11 & 219
\end{pmatrix} \tag{C.1}$$

Table C.2: Logistic Regression with Word2Vec-100dim model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	51.562	66.000	57.895	50
1	-	60.000	57.534	58.741	73
2	-	22.642	52.174	31.579	23
3	-	60.185	65.000	62.500	100
4	-	62.195	49.515	55.135	103
5	-	70.085	79.612	74.545	103
6	-	42.105	35.556	38.554	90
7	-	49.495	52.688	51.042	93
8	-	64.286	58.333	61.165	108
9	-	55.000	46.154	50.190	143
10	-	78.967	78.676	78.821	272
Total	**61.226**	**62.005**	**61.274**	**61.272**	**1158**

$$\begin{pmatrix}
33 & 3 & 3 & 1 & 0 & 4 & 2 & 1 & 0 & 1 & 2 \\
4 & 42 & 2 & 2 & 0 & 5 & 1 & 0 & 7 & 4 & 6 \\
0 & 0 & 12 & 1 & 1 & 0 & 0 & 2 & 1 & 4 & 2 \\
3 & 1 & 1 & 65 & 9 & 8 & 0 & 1 & 3 & 4 & 5 \\
4 & 3 & 0 & 5 & 51 & 5 & 6 & 11 & 4 & 8 & 6 \\
3 & 1 & 2 & 4 & 1 & 82 & 0 & 0 & 2 & 3 & 5 \\
4 & 12 & 11 & 6 & 2 & 2 & 32 & 6 & 5 & 4 & 6 \\
2 & 3 & 5 & 9 & 5 & 1 & 2 & 49 & 3 & 11 & 3 \\
0 & 1 & 3 & 4 & 3 & 1 & 5 & 13 & 63 & 3 & 12 \\
1 & 0 & 11 & 8 & 8 & 3 & 19 & 12 & 5 & 66 & 10 \\
10 & 4 & 3 & 3 & 2 & 6 & 9 & 4 & 5 & 12 & 214
\end{pmatrix} \tag{C.2}$$

Table C.3: Logistic Regression with FastText-100dim model results

Topic	Accuracy	Precision	Recall	F-Score	Support
0	-	55.714	78.000	65.000	50
1	-	60.759	65.753	63.158	73
2	-	21.154	47.826	29.333	23
3	-	68.000	68.000	68.000	100
4	-	60.638	55.340	57.868	103
5	-	75.862	85.437	80.365	103
6	-	43.182	42.222	42.697	90
7	-	53.261	52.688	52.973	93
8	-	75.904	58.333	65.969	108
9	-	58.824	48.951	53.435	143
10	-	80.755	78.676	79.702	272
Total	**64.335**	**65.614**	**64.335**	**64.583**	**1158**

$$\begin{pmatrix} 39 & 1 & 3 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 3 \\ 1 & 48 & 0 & 2 & 0 & 5 & 2 & 1 & 4 & 1 & 9 \\ 0 & 0 & 11 & 2 & 2 & 0 & 1 & 2 & 0 & 2 & 3 \\ 7 & 2 & 0 & 68 & 5 & 6 & 1 & 2 & 1 & 4 & 4 \\ 4 & 4 & 0 & 4 & 57 & 2 & 5 & 12 & 3 & 6 & 6 \\ 5 & 0 & 1 & 3 & 0 & 88 & 1 & 0 & 0 & 4 & 1 \\ 6 & 16 & 8 & 4 & 1 & 1 & 38 & 3 & 2 & 5 & 6 \\ 2 & 2 & 5 & 7 & 8 & 0 & 2 & 49 & 3 & 12 & 3 \\ 0 & 2 & 2 & 4 & 4 & 2 & 5 & 11 & 63 & 3 & 12 \\ 3 & 0 & 16 & 3 & 11 & 2 & 21 & 9 & 4 & 70 & 4 \\ 3 & 4 & 6 & 2 & 5 & 9 & 11 & 3 & 3 & 12 & 214 \end{pmatrix} \tag{C.3}$$

Table C.4: Gradient Boosting with Word2Vec-100dim model results

Topic	Accuracy	Precision	Recall	F-Score	Support
0	-	51.471	70.000	59.322	50
1	-	61.644	61.644	61.644	73
2	-	22.034	56.522	31.707	23
3	-	64.286	63.000	63.636	100
4	-	57.447	52.427	54.822	103
5	-	71.681	78.641	75.000	103
6	-	40.000	42.222	41.081	90
7	-	51.648	50.538	51.087	93
8	-	71.591	58.333	64.286	108
9	-	53.153	41.259	46.457	143
10	-	79.104	77.941	78.519	272
Total	**61.313**	**62.661**	**61.313**	**61.591**	**1158**

$$\begin{pmatrix} 35 & 2 & 3 & 1 & 1 & 3 & 2 & 1 & 0 & 0 & 2 \\ 4 & 45 & 1 & 2 & 0 & 6 & 2 & 3 & 3 & 1 & 6 \\ 0 & 0 & 13 & 2 & 2 & 0 & 0 & 2 & 0 & 2 & 2 \\ 5 & 1 & 1 & 63 & 9 & 6 & 2 & 3 & 2 & 4 & 4 \\ 3 & 3 & 1 & 4 & 54 & 3 & 4 & 9 & 4 & 10 & 8 \\ 5 & 1 & 2 & 4 & 1 & 81 & 1 & 0 & 1 & 3 & 4 \\ 5 & 14 & 8 & 3 & 2 & 2 & 38 & 3 & 4 & 4 & 7 \\ 2 & 0 & 5 & 9 & 6 & 1 & 3 & 47 & 3 & 13 & 4 \\ 0 & 3 & 3 & 2 & 4 & 2 & 4 & 11 & 63 & 4 & 12 \\ 2 & 1 & 16 & 5 & 11 & 1 & 26 & 10 & 5 & 59 & 7 \\ 7 & 3 & 6 & 3 & 4 & 8 & 13 & 2 & 3 & 11 & 212 \end{pmatrix} \tag{C.4}$$

Table C.5: Gradient Boosting with FastText-100dim model results

Topic	Accuracy	Precision	Recall	F-Score	Support
0	-	53.125	68.000	59.649	50
1	-	59.155	57.534	58.333	73
2	-	21.429	52.174	30.380	23
3	-	60.952	64.000	62.439	100
4	-	60.674	52.427	56.250	103
5	-	71.304	79.612	75.229	103
6	-	46.053	38.889	42.169	90
7	-	54.839	54.839	54.839	93
8	-	74.713	60.185	66.667	108
9	-	56.391	52.448	54.348	143
10	-	79.926	79.044	79.482	272
Total	**62.953**	**64.140**	**62.953**	**63.223**	**1158**

$$\begin{pmatrix} 34 & 2 & 4 & 1 & 0 & 0 & 3 & 0 & 0 & 3 & 3 \\ 4 & 42 & 4 & 2 & 0 & 7 & 0 & 2 & 4 & 1 & 7 \\ 0 & 0 & 12 & 1 & 1 & 0 & 1 & 2 & 0 & 3 & 3 \\ 3 & 2 & 2 & 64 & 7 & 7 & 0 & 4 & 1 & 6 & 4 \\ 5 & 4 & 0 & 6 & 54 & 4 & 5 & 10 & 2 & 7 & 6 \\ 3 & 0 & 4 & 3 & 3 & 82 & 1 & 0 & 0 & 3 & 4 \\ 6 & 15 & 7 & 5 & 3 & 2 & 35 & 4 & 2 & 5 & 6 \\ 2 & 2 & 4 & 9 & 4 & 1 & 3 & 51 & 2 & 11 & 4 \\ 0 & 2 & 3 & 2 & 2 & 2 & 6 & 9 & 65 & 4 & 13 \\ 2 & 1 & 13 & 10 & 9 & 4 & 12 & 9 & 4 & 75 & 4 \\ 5 & 1 & 3 & 2 & 6 & 6 & 10 & 2 & 7 & 15 & 215 \end{pmatrix} \tag{C.5}$$

Table C.6: Logistic Regression with Word2Vec-300dim model results

Topic	Accuracy	Precision	Recall	F-Score	Support
0	-	50.000	68.000	57.627	50
1	-	58.904	58.904	58.904	73
2	-	19.048	52.174	27.907	23
3	-	64.356	65.000	64.677	100
4	-	60.215	54.369	57.143	103
5	-	73.148	76.699	74.882	103
6	-	41.975	37.778	39.766	90
7	-	51.042	52.688	51.852	93
8	-	67.045	54.630	60.204	108
9	-	58.678	49.650	53.788	143
10	-	79.323	77.574	78.439	272
Total	**61.572**	**63.163**	**61.572**	**62.020**	**1158**

$$\begin{pmatrix}
34 & 1 & 4 & 1 & 1 & 1 & 3 & 0 & 0 & 2 & 3 \\
3 & 43 & 3 & 1 & 0 & 6 & 2 & 2 & 5 & 2 & 6 \\
0 & 0 & 12 & 2 & 1 & 0 & 1 & 1 & 0 & 3 & 3 \\
3 & 1 & 2 & 65 & 9 & 5 & 2 & 4 & 1 & 5 & 3 \\
6 & 3 & 0 & 3 & 56 & 3 & 6 & 9 & 4 & 6 & 7 \\
3 & 2 & 4 & 3 & 2 & 79 & 1 & 0 & 1 & 5 & 3 \\
4 & 15 & 11 & 5 & 2 & 2 & 34 & 4 & 1 & 4 & 8 \\
3 & 3 & 5 & 9 & 5 & 0 & 3 & 49 & 4 & 10 & 2 \\
0 & 3 & 3 & 3 & 5 & 1 & 6 & 13 & 59 & 3 & 12 \\
2 & 0 & 11 & 8 & 10 & 4 & 14 & 9 & 6 & 71 & 8 \\
10 & 2 & 8 & 1 & 2 & 7 & 9 & 5 & 7 & 10 & 211
\end{pmatrix} \tag{C.6}$$

Table C.7: Logistic Regression with FastText-300dim model results

Topic	Accuracy	Precision	Recall	F-Score	Support
0	-	52.381	66.000	58.407	50
1	-	57.143	60.274	58.667	73
2	-	17.857	43.478	25.316	23
3	-	62.626	62.000	62.312	100
4	-	54.167	50.485	52.261	103
5	-	73.043	81.553	77.064	103
6	-	43.182	42.222	42.697	90
7	-	52.747	51.613	52.174	93
8	-	71.765	56.481	63.212	108
9	-	57.377	48.951	52.830	143
10	-	78.571	76.838	77.695	272
Total	**61.399**	**62.768**	**61.399**	**61.785**	**1158**

$$\begin{pmatrix}
33 & 2 & 2 & 2 & 1 & 1 & 3 & 1 & 0 & 0 & 5 \\
3 & 44 & 1 & 3 & 0 & 6 & 0 & 3 & 4 & 3 & 7 \\
0 & 0 & 10 & 2 & 2 & 0 & 1 & 2 & 0 & 2 & 4 \\
4 & 3 & 2 & 62 & 8 & 7 & 2 & 2 & 1 & 4 & 5 \\
5 & 3 & 0 & 3 & 52 & 5 & 4 & 10 & 5 & 9 & 7 \\
4 & 1 & 3 & 3 & 0 & 84 & 2 & 0 & 0 & 4 & 2 \\
6 & 15 & 9 & 5 & 2 & 1 & 38 & 3 & 1 & 4 & 6 \\
2 & 1 & 4 & 9 & 6 & 1 & 3 & 48 & 2 & 12 & 5 \\
0 & 3 & 3 & 2 & 5 & 3 & 3 & 12 & 61 & 4 & 12 \\
1 & 1 & 15 & 3 & 12 & 1 & 21 & 8 & 7 & 70 & 4 \\
5 & 4 & 7 & 5 & 8 & 7 & 11 & 2 & 4 & 10 & 209
\end{pmatrix} \tag{C.7}$$

Table C.8: Gradient Boosting with Word2Vec-300dim model results

Topic	Accuracy	Precision	Recall	F-Score	Support
0	-	50.704	72.000	59.504	50
1	-	61.672	60.274	61.111	73
2	-	18.966	47.826	27.160	23
3	-	70.455	62.000	65.957	100
4	-	57.895	53.398	55.556	103
5	-	71.429	77.670	74.419	103
6	-	36.458	38.889	37.634	90
7	-	49.485	51.613	50.526	93
8	-	69.663	57.407	62.944	108
9	-	53.913	43.357	48.062	143
10	-	78.195	76.471	77.323	272
Total	**60.708**	**62.389**	**60.708**	**61.168**	**1158**

$$\begin{pmatrix}
36 & 1 & 3 & 1 & 0 & 1 & 3 & 0 & 0 & 2 & 3 \\
4 & 44 & 1 & 2 & 0 & 5 & 1 & 3 & 4 & 2 & 7 \\
1 & 0 & 11 & 1 & 3 & 0 & 1 & 2 & 0 & 2 & 2 \\
3 & 1 & 3 & 62 & 9 & 6 & 2 & 3 & 2 & 3 & 6 \\
5 & 4 & 0 & 2 & 55 & 3 & 5 & 10 & 4 & 8 & 7 \\
3 & 1 & 2 & 2 & 2 & 80 & 2 & 0 & 1 & 4 & 6 \\
7 & 14 & 7 & 3 & 2 & 3 & 35 & 4 & 3 & 6 & 6 \\
2 & 1 & 5 & 8 & 4 & 1 & 5 & 48 & 6 & 11 & 2 \\
0 & 1 & 4 & 4 & 4 & 3 & 5 & 12 & 62 & 3 & 10 \\
3 & 1 & 16 & 3 & 10 & 2 & 25 & 9 & 3 & 62 & 9 \\
7 & 3 & 6 & 0 & 6 & 8 & 12 & 6 & 4 & 12 & 208
\end{pmatrix} \tag{C.8}$$

Table C.9: Gradient Boosting with FastText-300dim model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	55.696	88.000	68.217	50
1	-	60.526	63.014	61.745	73
2	-	23.077	52.174	32.000	23
3	-	62.500	65.000	63.725	100
4	-	65.476	53.398	58.824	103
5	-	74.359	84.466	79.091	103
6	-	48.718	42.222	45.238	90
7	-	54.167	55.914	55.026	93
8	-	79.070	62.963	70.103	108
9	-	60.976	52.448	56.391	143
10	-	80.989	78.309	79.626	272
Total	**65.199**	**66.578**	**65.199**	**65.384**	**1158**

$$\begin{pmatrix} 44 & 2 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 2 \\ 4 & 46 & 1 & 2 & 0 & 6 & 0 & 2 & 4 & 1 & 7 \\ 0 & 0 & 12 & 1 & 1 & 0 & 1 & 2 & 0 & 3 & 3 \\ 5 & 0 & 3 & 65 & 6 & 7 & 3 & 3 & 1 & 3 & 4 \\ 5 & 4 & 1 & 4 & 55 & 3 & 4 & 11 & 3 & 7 & 6 \\ 2 & 0 & 3 & 3 & 1 & 87 & 2 & 0 & 1 & 3 & 1 \\ 4 & 15 & 9 & 3 & 1 & 3 & 38 & 3 & 1 & 6 & 7 \\ 2 & 2 & 3 & 11 & 4 & 1 & 3 & 52 & 1 & 10 & 4 \\ 1 & 3 & 3 & 2 & 2 & 2 & 3 & 10 & 68 & 3 & 11 \\ 4 & 1 & 11 & 9 & 8 & 3 & 14 & 10 & 3 & 75 & 5 \\ 8 & 3 & 6 & 4 & 6 & 5 & 8 & 3 & 4 & 12 & 213 \end{pmatrix} \tag{C.9}$$

Table C.10: Logistic Regression with Word2Vec-500dim model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	50.000	70.000	58.333	50
1	-	60.870	57.534	59.155	73
2	-	21.053	52.174	30.000	23
3	-	60.000	63.000	61.463	100
4	-	56.180	48.544	52.083	103
5	-	72.727	77.670	75.117	103
6	-	42.667	35.556	38.788	90
7	-	51.613	51.613	51.613	93
8	-	67.033	56.481	61.307	108
9	-	54.615	49.650	52.015	143
10	-	76.580	75.735	76.155	272
Total	**60.449**	**61.506**	**60.449**	**60.654**	**1158**

$$\begin{pmatrix} 35 & 2 & 3 & 1 & 1 & 1 & 3 & 1 & 0 & 2 & 1 \\ 3 & 42 & 1 & 2 & 0 & 5 & 2 & 1 & 5 & 2 & 10 \\ 0 & 0 & 12 & 1 & 1 & 0 & 1 & 2 & 0 & 3 & 3 \\ 3 & 0 & 2 & 63 & 10 & 7 & 2 & 2 & 1 & 5 & 5 \\ 4 & 3 & 2 & 7 & 50 & 3 & 6 & 11 & 4 & 7 & 6 \\ 3 & 0 & 2 & 4 & 0 & 80 & 2 & 0 & 1 & 3 & 8 \\ 6 & 13 & 11 & 3 & 3 & 3 & 32 & 3 & 2 & 7 & 7 \\ 3 & 3 & 5 & 10 & 6 & 1 & 3 & 48 & 3 & 9 & 2 \\ 1 & 3 & 3 & 3 & 3 & 2 & 4 & 11 & 61 & 4 & 13 \\ 1 & 1 & 11 & 8 & 11 & 3 & 16 & 9 & 4 & 71 & 8 \\ 11 & 2 & 5 & 3 & 4 & 5 & 4 & 5 & 10 & 17 & 206 \end{pmatrix} \tag{C.10}$$

Table C.11: Logistic Regression with FastText-500dim model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	50.667	76.000	60.800	50
1	-	64.384	64.384	64.384	73
2	-	17.742	47.826	25.882	23
3	-	64.078	66.000	65.025	100
4	-	61.458	57.282	59.296	103
5	-	73.451	80.583	76.852	103
6	-	43.182	42.222	42.697	90
7	-	55.814	51.613	53.631	93
8	-	72.527	61.111	66.332	108
9	-	57.522	45.455	50.781	143
10	-	79.845	75.735	77.736	272
Total	**62.781**	**64.593**	**62.781**	**63.265**	**1158**

$$\begin{pmatrix} 38 & 1 & 2 & 2 & 0 & 1 & 2 & 0 & 0 & 2 & 2 \\ 3 & 47 & 3 & 1 & 0 & 4 & 0 & 3 & 3 & 1 & 8 \\ 1 & 0 & 11 & 1 & 2 & 0 & 1 & 2 & 0 & 2 & 3 \\ 4 & 3 & 2 & 66 & 5 & 6 & 2 & 3 & 1 & 3 & 5 \\ 5 & 3 & 0 & 4 & 59 & 4 & 3 & 10 & 3 & 7 & 5 \\ 3 & 0 & 3 & 5 & 2 & 83 & 1 & 0 & 1 & 2 & 3 \\ 6 & 14 & 9 & 4 & 1 & 1 & 38 & 3 & 2 & 5 & 7 \\ 2 & 0 & 6 & 8 & 5 & 1 & 2 & 48 & 5 & 12 & 4 \\ 0 & 2 & 3 & 3 & 3 & 3 & 4 & 10 & 66 & 3 & 11 \\ 4 & 0 & 17 & 5 & 13 & 2 & 22 & 5 & 6 & 65 & 4 \\ 9 & 3 & 6 & 4 & 6 & 8 & 13 & 2 & 4 & 11 & 206 \end{pmatrix} \tag{C.11}$$

Table C.12: Gradient Boosting with Word2Vec-500dim model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	47.945	70.000	56.911	50
1	-	59.722	58.904	59.310	73
2	-	22.222	52.174	31.169	23
3	-	70.968	66.000	68.394	100
4	-	56.863	56.311	56.585	103
5	-	71.296	74.757	72.986	103
6	-	35.417	37.778	36.559	90
7	-	52.874	49.462	51.111	93
8	-	67.857	52.778	59.375	108
9	-	55.652	44.755	49.612	143
10	-	78.467	79.044	78.755	272
Total	**61.054**	**62.435**	**61.054**	**61.355**	**1158**

$$\begin{pmatrix} 35 & 1 & 4 & 1 & 1 & 1 & 2 & 0 & 0 & 2 & 3 \\ 2 & 43 & 1 & 2 & 0 & 6 & 3 & 2 & 5 & 2 & 7 \\ 2 & 0 & 12 & 0 & 2 & 0 & 1 & 2 & 0 & 2 & 2 \\ 5 & 2 & 0 & 66 & 10 & 6 & 1 & 1 & 1 & 3 & 5 \\ 5 & 4 & 0 & 2 & 58 & 2 & 5 & 10 & 3 & 9 & 5 \\ 5 & 1 & 2 & 4 & 2 & 77 & 2 & 0 & 2 & 3 & 5 \\ 7 & 13 & 8 & 2 & 3 & 2 & 34 & 3 & 3 & 6 & 9 \\ 3 & 2 & 5 & 8 & 7 & 1 & 4 & 46 & 4 & 11 & 2 \\ 0 & 1 & 3 & 4 & 3 & 3 & 7 & 13 & 57 & 5 & 12 \\ 0 & 1 & 14 & 3 & 12 & 3 & 24 & 8 & 5 & 64 & 9 \\ 9 & 4 & 5 & 1 & 4 & 7 & 13 & 2 & 4 & 8 & 215 \end{pmatrix} \tag{C.12}$$

Table C.13: Gradient Boosting FastText-500dim model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	42.683	70.000	53.030	50
1	-	52.308	46.575	49.275	73
2	-	16.216	52.174	24.742	23
3	-	65.306	64.000	64.646	100
4	-	52.000	50.485	51.232	103
5	-	70.874	70.874	70.874	103
6	-	35.443	31.111	33.136	90
7	-	43.617	44.086	43.850	93
8	-	62.195	47.222	53.684	108
9	-	46.667	44.056	45.324	143
10	-	78.049	70.588	74.131	272
Total	**55.699**	**58.185**	**55.699**	**56.444**	**1158**

$$\begin{pmatrix} 35 & 3 & 3 & 1 & 1 & 1 & 2 & 0 & 0 & 3 & 1 \\ 6 & 34 & 4 & 1 & 0 & 6 & 2 & 3 & 4 & 5 & 8 \\ 1 & 0 & 12 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 3 \\ 3 & 3 & 1 & 64 & 7 & 6 & 0 & 3 & 2 & 5 & 6 \\ 7 & 3 & 3 & 2 & 52 & 5 & 3 & 10 & 4 & 7 & 7 \\ 4 & 1 & 4 & 5 & 3 & 73 & 3 & 0 & 2 & 6 & 2 \\ 9 & 14 & 11 & 5 & 3 & 1 & 28 & 4 & 1 & 6 & 8 \\ 4 & 1 & 8 & 7 & 6 & 1 & 5 & 41 & 2 & 14 & 4 \\ 1 & 2 & 4 & 5 & 8 & 2 & 3 & 11 & 51 & 9 & 12 \\ 3 & 2 & 15 & 4 & 12 & 1 & 21 & 9 & 10 & 63 & 3 \\ 9 & 2 & 9 & 3 & 6 & 7 & 12 & 10 & 6 & 16 & 192 \end{pmatrix} \tag{C.13}$$

Table C.14: Logistic Regression with Tf-idf model results

Topic	**Accuracy**	**Precision**	**Recall**	**F-Score**	**Support**
0	-	37.931	66.000	48.175	50
1	-	48.148	53.425	50.649	73
2	-	20.635	56.522	30.233	23
3	-	51.515	51.000	51.256	100
4	-	58.537	46.602	51.892	103
5	-	68.421	75.728	71.889	103
6	-	35.106	36.667	35.870	90
7	-	50.000	50.538	50.267	93
8	-	67.188	39.815	50.000	108
9	-	46.825	41.259	43.866	143
10	-	75.591	70.588	73.004	272
Total	**54.922**	**57.372**	**54.922**	**55.362**	**1158**

$$\begin{pmatrix} 33 & 2 & 1 & 2 & 1 & 1 & 3 & 0 & 2 & 4 & 1 \\ 4 & 39 & 3 & 1 & 0 & 7 & 0 & 3 & 4 & 2 & 10 \\ 3 & 0 & 13 & 2 & 0 & 0 & 0 & 2 & 0 & 2 & 1 \\ 7 & 4 & 1 & 51 & 7 & 6 & 3 & 2 & 1 & 9 & 9 \\ 9 & 3 & 1 & 5 & 48 & 5 & 2 & 14 & 3 & 5 & 8 \\ 4 & 0 & 4 & 4 & 1 & 78 & 1 & 0 & 0 & 5 & 6 \\ 8 & 16 & 9 & 5 & 2 & 2 & 33 & 3 & 1 & 7 & 4 \\ 3 & 1 & 5 & 10 & 6 & 1 & 5 & 47 & 0 & 11 & 4 \\ 2 & 4 & 4 & 6 & 9 & 4 & 5 & 10 & 43 & 8 & 13 \\ 2 & 9 & 14 & 5 & 5 & 4 & 26 & 6 & 7 & 59 & 6 \\ 12 & 3 & 8 & 8 & 3 & 6 & 16 & 7 & 3 & 14 & 192 \end{pmatrix} \tag{C.14}$$

Table C.15: Gradient Boosting with Tf-idf model results

APPENDIX D

Impact of this Project

D.1 Introduction

Social media are used daily by millions of users around the world and account for a large part of the messages circulating on the Internet. Having a system which can monitor these messages and interactions can greatly facilitate the analysis of users and the exchanged messages, as well as provide a competitive advantage. Knowing what is being talked about, how it is being talked about, and in what way makes it possible to identify and observe situations that would otherwise not be possible.

In this section, the impact of the developed system will be discussed.

D.2 Social Impact

The system is based on the content of posts written in different social media platforms. It can be used to analyze different domains, and in this project the Ride-Hailing domain and the Radicalization domain have been analyzed. In the Radicalization domain, it has to be seen that the system could act as a radicalism identifier, observing people which are talking about these topics and analyzing how they are talking about it. This can be very useful to discover new extremist lines or even to detect radicalization in individuals before a serious situation can occur.

In the specific case of Islamist radicalism, the developed tool can be used to identify possible hate speeches towards or from certain groups and thus prevent messages of a terrorist nature or that can influence society, to prevent some individuals from becoming extremists.

D.3 Economic Impact

Regarding the economic impact, in this project, one of the studied domains was the Ride-Hailing domain. Ride-Hailing companies live in a strongly competitive world, where gaining a foothold in the market can be difficult due to the large number of companies and their strength. But the developed system could be a good tool to analyze the market and the opinion of the users, both its users and those of other companies.

In the case of Cabify, as its main competitors in Spain are Uber and Taxis, the system can give a real-time insight into what its users are complaining about or what users on other platforms like, so it can grow by satisfying its users and gaining new ones.

D.4 Environmental Impact

The development of the system needs large computational requirements. This can cause the system consumption when creating the models to be large. In addition, once the system is brought into production, additional equipment may be needed to back up the data or to mount the system in RAIDs to ensure data persistence and control over possible failures that may occur.

Because of this, it is necessary to carry out an energy transition that uses less polluting energy and more renewable energy sources. On the other hand, it is also necessary to use green computing techniques to reduce the pollution produced by the use and generation of energy, not only with this project, but with so many others.

D.5 Ethical Implications

The ethical implications of this project are largely related to user privacy. As the system uses social media messages, it is mandatory to obtain the consent of the users once the system is put into production. The reasons for this are that the system developed analyzes the opinion of users and the way they speak, so that if it were to be misused, specific users could be individually analyzed for purposes for which this system has not been developed. Therefore, it is always necessary to obtain the consent of the users before performing any kind of analysis.

In addition, in the event that the system is used by a public organization and a web address is provided to access the platform, users should also be asked for their consent to publish their data on that website.

APPENDIX E

Cost of the System

E.1 Introduction

In this section, all project costs, both human and economical, as well as the resources used during development, will be discussed.

E.2 Physical Resources

Two machines have been used to develop the system. Firstly, a personal computer (desktop computer) with the next characteristics has been used:

- **CPU:** Intel Core i5-3350P 3.10GHz, x86_64 architecture.
- **RAM:** 16 GB.
- **Disk:** 500 GB - HDD.
- **Operative System:** Ubuntu 20.04 LTS.
- **Price:** At the moment of the develop of the project, the approximated cost of this computer would be around 400 €.

On the other hand, the second machine was provided by the GSI and consists of a cluster of machines with the next specifications:

- **CPU:** Intel Xeon Silver 4210 CPU 2.20GHz, x86_64 architecture.
- **RAM:** 125 GB.
- **Disk:** 27 TB (only a small part available for the development of this project).
- **Operative System:** Ubuntu 18.04 LTS.
- **Price:** At the moment of the develop of the project, the approximated cost of this machine would be around 1500 €.

The first machine has been used in the deployment of the application and in almost all the development, while the second one has been used in the training and data processing phase.

E.3 Human Resources

This project has been developed within the Cabify Chair (*Cátedra Cabify*) with an hourly net salary of 6.5 €. This project, framed within the Master's Thesis, which in the Master's Degree in Telecommunications Engineering (MUIT) of the ETSIT-UPM consists of 30 ECTS. Being 1 ECTS 30 hours of work, the Master's thesis requires at least a total of 900 hours. The associated cost to one person is 5,850 €.

E.4 Licenses

Every component, library or package used in this project follows an open source philosophy. For this reason, the project has been developed without acquiring any type of license fee.

E.5 Total Costs

With the information explained above, the total economic cost of the project amounts to approximately 7,750 €.

Bibliography

[1] Shakeel Ahmad, Muhammad Zubair Asghar, Fahad M Alotaibi, and Irfanullah Awan. Detection and classification of social media-based extremist affiliations using sentiment analysis techniques. *Human-centric Computing and Information Sciences*, 9(1):24, 2019.

[2] Rubayyi Alghamdi and Khalid Alfalqi. A survey of topic modeling in text mining. *Int. J. Adv. Comput. Sci. Appl.(IJACSA)*, 6(1), 2015.

[3] Oscar Araque and Carlos A Iglesias. An approach for radicalization detection based on emotion signals and semantic similarity. *IEEE Access*, 8:17877–17891, 2020.

[4] Oscar Araque, Ganggao Zhu, Manuel García-Amado, and Carlos A Iglesias. Mining the opinionated web: classification and detection of aspect contexts for aspect based sentiment analysis. In *2016 IEEE 16th international conference on data mining workshops (ICDMW)*, pages 900–907. IEEE, 2016.

[5] Oscar Araque, Ganggao Zhu, and Carlos A Iglesias. A semantic similarity-based perspective of affect lexicons for sentiment analysis. *Knowledge-Based Systems*, 165:346–359, 2019.

[6] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *The journal of machine learning research*, 3:1137–1155, 2003.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[9] John G Breslin, Stefan Decker, Andreas Harth, and Uldis Bojars. Sioc: an approach to connect web-based communities. *International Journal of Web Based Communities*, 2(2):133–142, 2006.

[10] William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*, volume 161175. Citeseer, 1994.

[11] Jonathan Chang, Jordan Boyd-Graber, Chong Wang, Sean Gerrish, and David M Blei. Reading tea leaves: How humans interpret topic models. In *Neural information processing systems*, volume 22, pages 288–296. Citeseer, 2009.

[12] Glen Coppersmith, Ryan Leary, Patrick Crutchley, and Alex Fine. Natural language processing of social media as screening for suicide risk. *Biomedical informatics insights*, 10:1178222618792860, 2018.

[13] Biraj Dahal, Sathish AP Kumar, and Zhenlong Li. Topic modeling and sentiment analysis of global climate change tweets. *Social Network Analysis and Mining*, 9(1):1–20, 2019.

[14] Álvaro de Pablo, Oscar Araque, and Carlos Angel Iglesias. Radical text detection based on stylometry.

[15] Mark Dredze. How social media will change public health. *IEEE Intelligent Systems*, 27(4):81–84, 2012.

[16] Tracie Farrell, Oscar Araque, Miriam Fernandez, and Harith Alani. On the use of jargon and word embeddings to explore subculture within the reddit's manosphere. In *12th ACM Conference on Web Science*, pages 221–230, 2020.

[17] Miriam Fernandez, Moizzah Asif, and Harith Alani. Understanding the roots of radicalisation on twitter. In *Proceedings of the 10th acm conference on web science*, pages 1–10, 2018.

[18] Christian Fürber. Semantic technologies. In *Data Quality Management with Semantic Technologies*, pages 56–68. Springer, 2016.

[19] Liangjie Hong and Brian D Davison. Empirical study of topic modeling in twitter. In *Proceedings of the first workshop on social media analytics*, pages 80–88, 2010.

[20] Ian Horrocks, Peter F Patel-Schneider, and Frank Van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Journal of web semantics*, 1(1):7–26, 2003.

[21] Derek Howard, Marta M Maslej, Justin Lee, Jacob Ritchie, Geoffrey Woollard, and Leon French. Transfer learning for risk classification of social media posts: Model evaluation study. *Journal of medical Internet research*, 22(5):e15371, 2020.

[22] Marcin Junczys-Dowmunt, Roman Grundkiewicz, Tomasz Dwojak, Hieu Hoang, Kenneth Heafield, Tom Neckermann, Frank Seide, Ulrich Germann, Alham Fikri Aji, Nikolay Bogoychev, et al. Marian: Fast neural machine translation in c++. *arXiv preprint arXiv:1804.00344*, 2018.

[23] Ora Lassila, Ralph R Swick, et al. Resource description framework (rdf) model and syntax specification. 1998.

[24] Omer Levy, Yoav Goldberg, and Ido Dagan. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225, 2015.

[25] Bing Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[26] Bing Liu et al. Sentiment analysis and subjectivity. *Handbook of natural language processing*, 2(2010):627–666, 2010.

[27] Edward Loper and Steven Bird. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*, 2002.

[28] Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

[29] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[30] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1310.4546*, 2013.

[31] Seyedali Mirjalili. Genetic algorithm. In *Evolutionary algorithms and neural networks*, pages 43–55. Springer, 2019.

[32] Ahmed Abdul Moiz, Pinaki Pal, Daniel Probst, Yuanjiang Pei, Yu Zhang, Sibendu Som, and Janardhan Kodavasal. A machine learning-genetic algorithm (ml-ga) approach for rapid optimization using high-performance computing. *SAE International Journal of Commercial Vehicles*, 11(2018-01-0190):291–306, 2018.

[33] Stephen Nabareseh, Eric Afful-Dadzie, and Petr Klimek. Leveraging fine-grained sentiment analysis for competitivity. *Journal of Information & Knowledge Management*, 17(02):1850018, 2018.

[34] Thien Hai Nguyen and Kiyoaki Shirai. Topic modeling based sentiment analysis on social media for stock market prediction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1354–1364, 2015.

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

[36] David Ramamonjisoa. Topic modeling on users's comments. In *2014 Third ICT International Student Project Conference (ICT-ISPC)*, pages 177–180. IEEE, 2014.

[37] Juan Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer, 2003.

[38] Radim Rehurek and Petr Sojka. Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2), 2011.

[39] Marian-Andrei Rizoiu, Tianyu Wang, Gabriela Ferraro, and Hanna Suominen. Transfer learning for hate speech detection in social media. *arXiv preprint arXiv:1906.03829*, 2019.

[40] Michael Röder, Andreas Both, and Alexander Hinneburg. Exploring the space of topic coherence measures. In *Proceedings of the eighth ACM international conference on Web search and data mining*, pages 399–408, 2015.

[41] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. *arXiv preprint arXiv:1207.4169*, 2012.

[42] Matthew Rowe and Hassan Saif. Mining pro-isis radicalisation signals from social media users. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 10, 2016.

[43] Anna Schmidt and Michael Wiegand. A survey on hate speech detection using natural language processing. In *Proceedings of the fifth international workshop on natural language processing for social media*, pages 1–10, 2017.

[44] Ainhoa Serna, Jon Kepa Gerrikagoitia, Unai Bernabé, and Tomás Ruiz. Sustainability analysis on urban mobility based on social media content. *Transportation Research Procedia*, 24:1–8, 2017.

[45] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[46] Robyn Speer, Joshua Chin, Andrew Lin, Sara Jewett, and Lance Nathan. Luminosoinsight/-wordfreq: v2.2, October 2018.

[47] Shaheen Syed and Marco Spruit. Full-text or abstract? examining topic coherence scores using latent dirichlet allocation. In *2017 IEEE International conference on data science and advanced analytics (DSAA)*, pages 165–174. IEEE, 2017.

[48] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. The penn treebank: an overview. *Treebanks*, pages 5–22, 2003.

[49] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[50] Kristina Toutanova, Dan Klein, Christopher D Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 252–259, 2003.

[51] Erik Tromp and Mykola Pechenizkiy. Graph-based n-gram language identification on short texts. In *Proc. 20th Machine Learning conference of Belgium and The Netherlands*, pages 27–34, 2011.

[52] Walter JB Van Heuven, Pawel Mandera, Emmanuel Keuleers, and Marc Brysbaert. Subtlex-uk: A new and improved word frequency database for british english. *Quarterly journal of experimental psychology*, 67(6):1176–1190, 2014.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[54] Hongwei Wang, Song Gao, Pei Yin, and James Nga-Kwok Liu. Competitiveness analysis through comparative relation mining. *Industrial Management & Data Systems*, 2017.

[55] Stuart Weibel, John Kunze, Carl Lagoze, and Misha Wolf. Dublin core metadata for resource discovery. *Internet Engineering Task Force RFC*, 2413(222):132, 1998.

[56] Adam Westerski, Carlos Angel Iglesias, and Fernando Tapia Rico. Linked opinions: Describing sentiments on the structured web of data. In *SDoW@ ISWC*, 2011.

[57] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[58] Ming Yan, Jitao Sang, Tao Mei, and Changsheng Xu. Friend transfer: Cold-start friend recommendation with cross-platform transfer learning of social knowledge. In *2013 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2013.