



**Miguel  
Reis Vicente**

**Caraterização de Utilizadores em Redes Sociais  
User Characterization in Social Media**





**Miguel  
Reis Vicente**

**Caraterização de Utilizadores em Redes Sociais**  
**User Characterization in Social Media**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Diogo Nuno Pereira Gomes, Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.



**o júri / the jury**

presidente / president

Prof. Doutora Ana Maria Perfeito Tomé

Professora Associada da Universidade do Aveiro

vogais / examiners committee

Prof. Doutor Carlos Manuel das Neves Santos

Professor Auxiliar da Universidade de Aveiro

Prof. Doutor Diogo Nuno Pereira Gomes

Professor Auxiliar da Universidade de Aveiro (orientador)



## **agradecimentos / acknowledgements**

Começo por agradecer ao Professor Diogo Gomes, pela oportunidade em trabalhar numa área que me apraz, pela liberdade que me deu na realização deste trabalho e pela ajuda disponibilizada durante o mesmo.

Aos meus companheiros e colegas de curso, palavras não me chegam para chegar a todos os que de alguma forma me tornaram uma pessoa melhor e fizeram desta estadia de cinco anos em Aveiro dos melhores anos da minha ainda curta vida.

Aos meus amigos Vasco Santos, Rui Monteiro, José Sequeira e Joel Pinheiro, o meu obrigado por todas as aventuras e desventuras que levamos do nosso primeiro ano e dos outros, pela nossa querida habitação, pelas noites sem dormir, pelas discussões efervescentes e por tudo o que vocês representam neste percurso.

Aos meus colegas de longa data, por se manterem sempre por perto apesar de fisicamente estarem longe.

Por fim quero deixar uma palavra especial à minha família. Ao meu irmão, pai e mãe, por tudo o que passaram para que este momento fosse possível. O percurso até aqui foi acidentado, mas sempre com a vossa constante e total disponibilidade para me apoiar nas minhas decisões e me aturar durante os piores momentos. O meu mais sentido obrigado por fazerem de mim a pessoa que sou hoje.





**Palavras Chave**

Redes Sociais, Twitter, Análise de Grafos, Extração de Dados, Aprendizagem de Máquina, Análise de Dados, Visualização de Dados

**Resumo**

O crescimento acentuado das Redes Sociais que se verificou num passado recente, criou uma nova área de estudo na investigação em análise e extração de dados. A sua disseminação pela sociedade moderna torna-as uma fonte interessante para a aplicação de ciência dos dados, visto que auxiliam a perceção de comportamentos e padrões em dados sociais. Este tipo de informação possui valor estratégico em áreas como a publicidade e o marketing. Nesta dissertação é apresentado um protótipo para uma aplicação web que visa apresentar informação sobre a rede Twitter e os utilizadores que a compõem, através de esquemas de visualização de dados. Esta aplicação adota um modelo de dados de um grafo de propriedades, armazenado numa base de dados de grafos, para permitir uma análise eficiente das relações entre os dados existentes no Twitter. Para além disso, também faz uso de algoritmos de aprendizagem supervisionados e não-supervisionados, assim como análise estatística, para extrair padrões no conteúdo de tweets e prever atributos latentes em utilizadores do Twitter. O objetivo final é permitir a caracterização dos utilizadores Portugueses do Twitter, através da interpretação dos resultados apresentados.



**Keywords**

Social Media, Twitter, Graph Analysis, Data Mining, Machine Learning, Data Analysis, Data Visualization

**Abstract**

The massive growth of Social Media platforms in recent years has created a new area of study for Data Mining research. Its general dissemination in modern society makes it a very interesting data science resource, as it enables the better understanding of social behavior and demographic statistics, information that has strategic value in business areas like marketing and advertising. This dissertation presents a prototype for a web application that provides a number of intuitive and interactive data visualization schemes that present information about the Twitter network and its individual users. This application leverages a property graph data model, modeled from a collection of millions of tweets from the Portuguese community and stored in a state of the art graph database, to enable an efficient analysis of the existent relationships in Twitter data. It also makes use of Supervised and Unsupervised learning algorithms, as well as statistical analysis, to extract meaningful patterns in tweets content and predict latent attributes in Twitter users. The end goal is to allow the characterization of the Portuguese users in Twitter, through the created visual representations of the achieved results.



# CONTENTS

---

CONTENTS . . . . .	i
LIST OF FIGURES . . . . .	v
LIST OF TABLES . . . . .	vii
LISTINGS . . . . .	viii
ACRONYMS . . . . .	xi
1 INTRODUCTION . . . . .	1
1.1 Contextualization . . . . .	1
1.2 Goals . . . . .	2
1.3 Outline . . . . .	2
2 STATE OF THE ART . . . . .	3
2.1 Social Media . . . . .	3
2.1.1 Growth and Relevance . . . . .	3
2.1.2 Social Media Environments . . . . .	4
2.1.3 Microblogging and Twitter . . . . .	5
2.2 Data Mining in Social Media Environments . . . . .	6
2.2.1 Data Mining and the KDD Process . . . . .	6
2.2.2 Data Preprocessing . . . . .	7
2.2.3 Data Mining and Machine Learning . . . . .	10
2.2.4 Data Postprocessing . . . . .	13
2.2.5 Challenges With Social Media Data . . . . .	14
2.2.6 Relevant Research Topics . . . . .	17
2.2.7 Related Work . . . . .	20
2.3 Technology Review . . . . .	21
2.3.1 Graph Databases . . . . .	22
2.3.2 Neo4j . . . . .	23
2.3.3 OrientDB . . . . .	25
2.3.4 Titan . . . . .	27
2.3.5 Graph Databases Review . . . . .	28
2.3.6 Data Mining and Machine Learning Tools . . . . .	30
3 SYSTEM DESCRIPTION AND ARCHITECTURE . . . . .	33

3.1	Description and Requirements . . . . .	33
3.1.1	Network Scope . . . . .	34
3.1.2	User Scope . . . . .	34
3.1.3	Non-functional Requirements . . . . .	35
3.2	Data Model . . . . .	35
3.2.1	Available Data . . . . .	35
3.2.2	Proposed Data Model . . . . .	37
3.2.3	Storage Requirements . . . . .	39
3.3	Client-Server Model . . . . .	39
3.3.1	Backend . . . . .	40
3.3.2	Frontend . . . . .	40
3.3.3	API . . . . .	41
3.4	Data Mining Modules . . . . .	41
3.5	Architecture Overview . . . . .	42
4	IMPLEMENTATION . . . . .	43
4.1	Storage . . . . .	43
4.1.1	Neo4j Configuration . . . . .	43
4.1.2	Data Model . . . . .	45
4.1.3	Data Migration . . . . .	49
4.2	Network Scope Goals . . . . .	50
4.2.1	Network Activity . . . . .	50
4.2.2	Network Gender Distribution . . . . .	54
4.2.3	Network Influence . . . . .	59
4.2.4	Network Content . . . . .	61
4.3	User Scope Goals . . . . .	65
4.3.1	User Activity . . . . .	65
4.3.2	User Network . . . . .	68
4.3.3	User Content . . . . .	70
4.3.4	Topics . . . . .	70
4.3.5	Similarity . . . . .	71
4.4	Web Application . . . . .	71
4.4.1	Django Web Framework . . . . .	72
4.4.2	Celery . . . . .	78
4.4.3	Redis . . . . .	79
4.4.4	Visualization Frameworks and Libraries . . . . .	79
5	RESULTS . . . . .	83
5.1	Prototype . . . . .	83
5.1.1	Home page . . . . .	83
5.1.2	Network Explorer - Activity . . . . .	84
5.1.3	Network Explorer - Influence . . . . .	89
5.1.4	Network Explorer - Content . . . . .	90
5.1.5	User Explorer - Profiling . . . . .	91
5.1.6	Gender Classification . . . . .	95
5.2	API . . . . .	96
5.3	Benchmarking and Performance . . . . .	97
5.4	User Interface Evaluation . . . . .	98
6	CONCLUSION . . . . .	101

6.1	Final Considerations . . . . .	101
6.2	Future Work . . . . .	102
	REFERENCES . . . . .	103
	APPENDIX A: TWITTER API OBJECTS . . . . .	109
	APPENDIX B: GENDER CLASSIFIER . . . . .	115
	APPENDIX C: API DOCUMENTATION . . . . .	123
	APPENDIX D: USABILITY ENQUIRY . . . . .	127





# LIST OF FIGURES

---

2.1	The first tweet ever posted on Twitter. . . . .	6
2.2	Generic KDD process . . . . .	7
2.3	Diagram of a standard Supervised Learning application . . . . .	12
2.4	Google Trends result for the term "big data". . . . .	15
2.5	Example of a property graph of Twitter dynamics . . . . .	22
2.6	Graph view on the Neo4j web interface. . . . .	25
3.1	Graph representation of Status objects . . . . .	38
3.2	Client-Server Model . . . . .	40
3.3	High-level System Architecture . . . . .	42
4.1	Generation of 2-grams at character and word level. (The _ represents a blank space) . . . . .	56
4.2	Training of the gender classifier . . . . .	57
4.3	Creation of the Document model . . . . .	63
4.4	Clustering of Documents . . . . .	65
4.5	Architecture diagram of the web application . . . . .	72
4.6	Loading of the TopicSimilarity class . . . . .	77
5.1	Prototype home page . . . . .	84
5.2	Network activity by month . . . . .	85
5.3	Network activity by hours . . . . .	86
5.4	Network activity by locations . . . . .	87
5.5	Gender distribution . . . . .	88
5.6	Source distribution . . . . .	88
5.7	Filters for the influence rank . . . . .	89
5.8	Influence rank . . . . .	89
5.9	Graphical representation of the uncovered clusters . . . . .	90
5.10	Most relevant terms for cluster 1 . . . . .	90
5.11	Find user to profile form . . . . .	91
5.12	User profile information . . . . .	91
5.13	User activity . . . . .	92
5.14	User network interaction . . . . .	92
5.15	User similarity graph . . . . .	93
5.16	User distance graph . . . . .	93
5.17	User term frequency as a word cloud . . . . .	94
5.18	User topic modeling as a graph layout . . . . .	95
5.19	Find user for classification form . . . . .	96

5.20	Classification results . . . . .	96
5.21	Neo4j Profile Interface . . . . .	97
1	API - Network activity in each month . . . . .	123
2	API - Network activity in locations . . . . .	124
3	API - Influence rank . . . . .	124
4	API - Cluster nodes . . . . .	125
5	API - Cluster terms . . . . .	125
6	API - User similarity . . . . .	126
7	API - User topics . . . . .	126
8	Usability Enquiry - Section 1 . . . . .	128
9	Usability Enquiry - Section 2 . . . . .	129
10	Usability Enquiry - Section 3 . . . . .	130
11	Usability Enquiry - Section 4 . . . . .	131
12	Usability Enquiry - Section 5 . . . . .	132
13	Usability Enquiry - Section 6 . . . . .	133

# LIST OF TABLES

---

2.1	Comparison of Graph Database Management Systems (GDBMSs) . . . . .	29
3.1	TVPulse Data Model . . . . .	36
3.2	Relevant properties in Status Objects . . . . .	37
3.3	Proposed User model . . . . .	38
3.4	Proposed Tweet model . . . . .	39
3.5	Proposed Hashtag model . . . . .	39
3.6	API Specification . . . . .	41
4.1	Neo4j System Requirements . . . . .	44
4.2	Number of nodes in the database . . . . .	50
4.3	Number of relationships in the database . . . . .	50
4.4	Database size . . . . .	50
4.5	Sociological feature model . . . . .	55
4.6	N-gram feature model . . . . .	56
4.7	Classification results for the sociological feature model . . . . .	57
4.8	Classification results for the n-gram model . . . . .	58
4.9	Classification results for the joint models . . . . .	58
5.1	Benchmarking of Neo4j queries . . . . .	98

# LISTINGS

---

1	Example of a Cypher query . . . . .	24
2	Neo4j Network Configurations . . . . .	44
3	Example of a create User query . . . . .	45
4	Example of a create Tweet query . . . . .	46
5	Example of a create Hashtag entity query . . . . .	46
6	Example of a create relationship query . . . . .	46
7	Example of a create user-posts-tweet relationship . . . . .	47
8	User id uniqueness constraint . . . . .	47
9	Screen name uniqueness constraint . . . . .	47
10	Tweet id uniqueness constraint . . . . .	47
11	Hashtag text uniqueness constraint . . . . .	48
12	User id index . . . . .	48
13	User screen name index . . . . .	48
14	Tweet id index . . . . .	48
15	Hashtag text index . . . . .	48
16	Query for the number of tweets in a month . . . . .	51
17	Query for the number of tweets from each day in a month . . . . .	51
18	Query for the number of tweets in a specific day . . . . .	51
19	Query for the number of tweets by hour in a day . . . . .	52
20	Query for obtaining tweet locations . . . . .	52
21	Query for obtaining tweet locations in a single month . . . . .	53
22	Query for obtaining tweet sources . . . . .	54
23	First query developed for calculating the in-degree score of User entities . . . . .	60
24	Query developed to create user-mentions-user relationships . . . . .	60
25	Second query developed for calculating the in-degree score of User entities . . . . .	60
26	Python function that tokenizes a tweet . . . . .	62
27	Python function that tokenizes a Document . . . . .	64
28	Query for user activity over months . . . . .	66
29	Query for user activity on a specific month . . . . .	66
30	Query for user activity for all days in a month . . . . .	66
31	Query for user activity on a specific day . . . . .	66
32	Query for user hourly activity on a specific day . . . . .	66
33	Query needed to calculate which weekday is the user more active . . . . .	67
34	Query for user activity for every hour . . . . .	67
35	Query for user activity sources . . . . .	67
36	Query to obtain the users that more frequently mention other user . . . . .	68
37	Query to obtain the users that more frequently reply to another user . . . . .	69

38	Query to obtain the users that another user more frequently mentions . . . . .	69
39	Query to obtain the users that another user more frequently replies to . . . . .	69
40	Query to obtain the distance between two users . . . . .	70
41	User entity as a Python object . . . . .	73
42	Cassandra Document as a Python object . . . . .	74
43	Class developed to calculate similarity between one user and the rest . . . . .	76
44	Example of an API response . . . . .	77
45	Example of a Twitter API User Object . . . . .	109
46	Example of a Twitter API Status Object . . . . .	112
47	Classifier training script . . . . .	115
48	List of gendered Portuguese nicknames and abbreviations . . . . .	119
49	List of gender-meaningful terms . . . . .	120
50	Regular expressions used to remove URLs hashtags and mentions from tweets	121



# ACRONYMS

---

<b>ACID</b>	Atomicity, Consistency, Isolation, Durability	<b>MTV</b>	Model-Template-View
<b>AI</b>	Artificial Intelligence	<b>MVC</b>	Model-View-Controller
<b>AJAX</b>	Asynchronous JavaScript and XML	<b>NoSQL</b>	Not-only Structured Query Language
<b>API</b>	Application Programming Interface	<b>NLP</b>	Natural Language Processing
<b>CRUD</b>	Create, Read, Update and Delete	<b>NLTK</b>	Natural Language Toolkit
<b>CSS</b>	Cascading Style Sheets	<b>OGM</b>	Object-Graph Mapping
<b>DBMS</b>	Database Management System	<b>ORM</b>	Object-Relational Mapping
<b>DM</b>	Data Mining	<b>PCA</b>	Principal Component Analysis
<b>EPG</b>	Electronic Programming Guide	<b>RDBMS</b>	Relational Database Management System
<b>GDBMS</b>	Graph Database Management System	<b>REST</b>	Representational State Transfer
<b>GUI</b>	Graphical User Interface	<b>RT</b>	Retweet
<b>HTML</b>	HyperText Markup Language	<b>SQL</b>	Structured Query Language
<b>HTTP</b>	HyperText Transfer Protocol	<b>SVM</b>	Support Vector Machines
<b>HTTPS</b>	HyperText Transfer Protocol Secure	<b>TF</b>	Term Frequency
<b>JSON</b>	JavaScript Object Notation	<b>TF-IDF</b>	Term Frequency - Inverse Document Frequency
<b>JVM</b>	Java Virtual Machine	<b>SVG</b>	Scalable Vector Graphics
<b>KDD</b>	Knowledge Discovery in Databases	<b>URL</b>	Uniform Resource Locator
<b>LDA</b>	Latent Dirichlet Allocation	<b>WEKA</b>	Waikato Environment for Knowledge Analysis
<b>ML</b>	Machine Learning		





# INTRODUCTION

---

*This chapter introduces this dissertation's scenario and purpose. It starts by contextualizing the reader on the underlying issues that led to the need for this work. Afterwards it proceeds to explain the goals it aims to achieve and finally it explains how rest of the document is structured.*

## 1.1 CONTEXTUALIZATION

Social Media platforms and services have grown to reach a level of popularity and importance in modern society that has made them an integral part of daily communications and social interaction for millions of people around the world. The swift growth and diffusion of Social Media and its increasing ability to provide its users the tools to create, share and appreciate content with one or many of their peers has led to a massive volume of user-generated data, mainly, but not only, in the form of informal text. Its evolution has continuously affected society and human interaction, with what is said and done online having a greater relevance and contribution to how a person is perceived by its peers.

Being able to understand and extract information from Social Media data has been a research field of interest in recent years[1]. This exposure of one's feelings, opinions and other personal traits drawn the attention of multiple business areas that rely on understanding people, as well as sociological research. On the industry level, brands are evermore present in Social Media and use it as way to engage consumers and provide information about their current activity[2]. In politics, it is common nowadays for political parties and leaders to have an active Social Media presence to promote their views and advertise their work. Both of them leverage Social Media to reach and interact with audience levels deemed impractical through standard media outlets, with significantly less costs[3]. Besides being a vehicle for communication, Social Media also represents a source of information. Extracting useful data from the content available in Social Media platforms can provide insight about people, their interests, preferences, or their friends, which can largely influence business decision making. Trying to identify ways to make profitable use of this platforms has become a top of the agenda topic, mainly in the marketing industry, in the form of directed marketing and consumer analytics[4].

While all of these factors present an opportunity, they also represent a challenge. Numerous problems arise from trying to extract information from Social Media. This kind of data is not readily available in a structured and organized fashion and its accessibility can be limited. This contrasts

with the fact that Social Media is a source of Big Data, due to the vast amount of people that use it to produce new content on a regular basis. Additionally, data generated from Social Media data comes in a wide range of formats, like text, images or video. User-generated data in the form of text distinguishes itself from formal text due to the users disregard for grammar rules and inconsistent choice of words. Consequently, it is difficult most of the times to process it and requires specific techniques so it can be effectively handled. One particular aspect about this new type of data is that it is heavily linked. Social Media tries to emulate real-life social relationships and protocols and this is translated into its data. This causes common storing solutions like relational databases unsuitable to manage Social Media data.

All of these issues have led to an increasing academic and entrepreneurial research in ways to store, process and present Social Media data, which is exactly the focus of this dissertation, developing a system capable of efficiently store, handle and visualize data generated from Social Media.

## 1.2 GOALS

The main purpose of this dissertation is to build an information system that leverages Social Media data to provide subjective and objective characterization traits about the Portuguese users that are a part of the Social Media platform Twitter.

Using a data set consisting of millions of tweets and their meta-data, it is intended to understand potentially hidden information that can indicate certain traits about the Portuguese Twitter community, create a data model that embraces the characteristics of Twitter data and stores it in an appropriate solution. Afterwards, it is aimed to develop a web application that relies on the designed data model to extract and present meaningful information about Portuguese Twitter users, using intuitive and appealing data visualization techniques.

## 1.3 OUTLINE

- **Chapter 2** presents the theoretical foundation behind this effort, some related work made in Social Media research that inspired this dissertation and the technical components necessary to build a system of this kind;
- **Chapter 3** describes the requirements that this system should fulfill, as well as how it was planned and conceptually designed;
- **Chapter 4** provides a detailed overview of how the system was implemented and describes the technologies used to accomplish it;
- **Chapter 5** presents the results achieved by the developed prototype;
- **Chapter 6** discusses the obtained results, how they are relevant and future improvements for this work.

# STATE OF THE ART

---

*This chapter captures essential background knowledge relevant to this work. It focuses on understanding the Social Media phenomenon, then it proceeds to explain the process of extracting knowledge from Social Media sources and reviews the technology necessary to build the proposed system.*

## 2.1 SOCIAL MEDIA

### 2.1.1 GROWTH AND RELEVANCE

Social Media is conceptually defined as a set of computer-mediated tools to create, exchange or share information in a virtual community[5]. This concept by itself is not particularly groundbreaking as one can go as far as 1979 to find examples of online Social Media that fit this description[1]. However, the birth of modern Social Media was one of the consequences of what is commonly referred as Web 2.0, the notion of the World Wide Web as a platform where content is not only published but continuously created and modified by virtual communities, as well as the set of technological advances that supported this transformation[1]. This, along with the growing availability of high-speed Internet access, allowed the tremendous rise in popularity of this new form of media, leading to creation of platforms such as Facebook<sup>1</sup>, the one that arguably brought the phenomenon of Social Media to public debate[1].

Social Media growth in terms of active user base has been tremendous over the past years. In 2016, Facebook counts over 1.2 billion users on a daily basis[6], photo-sharing giant Instagram<sup>2</sup> is estimated to have over 500 million active users[7] monthly and microblogging platform Twitter<sup>3</sup> around 315 million[8]. It can also be observed that these numbers are in continuous growth since these platforms appeared and this pattern is expected to continue.

These numbers are indicators of the relevance these platforms have in modern society and how extracting knowledge from them can be a key factor in today's enterprise decision making.

---

<sup>1</sup><https://www.facebook.com/>

<sup>2</sup><https://www.instagram.com/>

<sup>3</sup><https://www.twitter.com/>

## 2.1.2 SOCIAL MEDIA ENVIRONMENTS

The adaptable aspect of this kind of internet-based applications and the ability to offer different types of multimedia content such as image, video, audio and text under a different set of rules and conditions set the ground for the proliferation of a number of Social Media platforms. As a result, came the need to understand and classify them according to their purpose, functionality, social presence and media richness. Current theories in Social Media research have identified the following categories for classifying Social Media services[1]:

- Collaborative projects;
- Blogs;
- Content Communities;
- Social Networks;
- Virtual Game Worlds;
- Virtual Social Worlds;

For the purpose of this work, it is relevant to address three of the presented types, which are Blogs, Content Communities and Social Networks.

**Blogs** are accounted to be the first official form of online Social Media[9]. Derived from the term "weblog"[10], this form of Social Media is nothing more than a website that is updated by a person or people under the form of submissions (or *posts*). The content available in a blog is defined by its owner(s). It also allows for commentary and feedback on the writer's entries, establishing a form of interaction between the writer and its readers. While in their genesis blogs were mainly long text based, times dictated the introduction of new variants such video blogs, where video replaces text and *microblogs* where long texts are replaced by short messages. Microblogs will be addressed further and in detail in this chapter.

**Content Communities** focus on media content sharing. By media, it is included media formats other than text, with video, photo and audio being the most common. This form of Social Media differs from blogs in the fact that they usually are aggregated in a single platform and blogs are considered personal web pages. They share with blogs the possibility of commentary on posted content. The most popular content communities nowadays include YouTube<sup>4</sup>, Tumblr<sup>5</sup> and Instagram.

**Social Networks** are currently the most popular form of Social Media environments. The main focus of Social Networking sites is connectivity. They enable the creation of personal profiles, where users expose personal information like contact information, interests, or other character defining facts and allow them to connect and network virtually with other users. They also allow the enrichment of your profile page through content sharing and attempt to emulate social relationships and interaction with concepts of friendship or approval. Major Social Networking platforms include Facebook and LinkedIn<sup>6</sup>.

---

<sup>4</sup><https://www.youtube.com/>

<sup>5</sup><https://www.tumblr.com/>

<sup>6</sup><https://www.linkedin.com/>

### 2.1.3 MICROBLOGGING AND TWITTER

Microblogs are a new branch of the traditional blogs presented before. They are characterized by a significant reduction of content length in the author's entries and a sense of real-time, personal commentary on the world's happenings[11]. Microblogging is also accounted to be a broadcast medium, meaning that content is distributed from the author directly to an audience, under a defined set of communication rules.

It is pointless to discuss microblogging without referring to Twitter, the most popular microblogging platform in existence[11]. Twitter is a microblogging service launched in October 2006 which allows the posting of sentences no longer than 140 characters, reminding of a newspaper headline, called *tweets* or, less knowingly, status updates. Tweets can also contain individual images or video links. Combined with this particular form of content, Twitter also has a collection of communication rules for its network. It uses a dynamic of followers and friends, which are reciprocal. However, a user can be a follower of another user without the opposite being obligatory. Being a follower of another user in Twitter means that all his tweets will be broadcasted to you. Friend is the denomination attributed to the users that you follow and follower is the denomination for users that have followed you. Twitter also uses a well-defined markup culture[12] and rules specific to the tweet itself. This includes Retweet (RT), which is the act of broadcasting other users' tweets to your followers and *Mention* which is identified by the character at ("@") preceding the user's screen name and used to forward a tweet directly to a specific user. A RT also allows the possibility of commenting on the original tweet, which denotes a *quote* Tweet. It is possible as well to reply to other users' tweets, which implies starting a tweet with a Mention to the user being replied. The hash character ("#") before a word is called an *hashtag*. This is used to group tweets into a specific topic. You can perform searches in Twitter by hashtag, which will retrieve issued tweets that contain it.

Twitter's format makes it an interesting source for Social Media research, mainly because most of the content of its postings represents some form of an opinion or public statement regarding a certain topic[13], as well as its current popularity and number of active users. Moreover, not only the content of user messages can be put under scrutiny but also the relations that users can establish among themselves and how they interact with their fellow virtual participants. The study of Twitter interaction mechanisms can also provide valuable insight when establishing a user profile[14]. For brands, Twitter is typically used as a mean for interaction with its consumers[15], which makes it an even more relevant research source, since it is in their best interest to understand their underlying virtual user base, whether in terms of behavior, demographics or sentiment.



Figure 2.1: The first tweet ever posted on Twitter.

## 2.2 DATA MINING IN SOCIAL MEDIA ENVIRONMENTS

Data Mining (DM) in Social Media environments (or Social Media Mining) is the interdisciplinary field that results from the confluence of Social Media services and Data Mining processes. It is "the process of representing, analyzing, and extracting actionable patterns from Social Media data"[16]. As discussed before, Social Media represents a medium that shatters the boundaries of the real and the digital world, where billions of people spend a vast amount of time collecting, curating, publishing, sharing and commenting content[17]. Social interaction becomes no longer restricted by time or space and this results in new opportunities for human behavior analysis.

The application of Data Mining techniques in Social Media environments can enhance researchers' capability to understand new phenomena and provide better knowledge for business decision making and intelligence. Data Mining in Social Media can help to identify influential users in a Social Media platform, detect latent communities formed in Social Networks, classify user sentiment towards a brand, subject or product, among other trending research fields[17].

In this section it will be given a general overview of what a Data Mining process consists, the challenges faced when building a system that relies on Data Mining, the problems posed by Social Media data and advances in this areas that are relevant to the developed work.

### 2.2.1 DATA MINING AND THE KDD PROCESS

Data Mining is defined as the process of discovering explicit, implicit, interesting and useful patterns or relationships, which is to say *knowledge*, in large volumes of data[18],[19],[20]. This emerging multidisciplinary field in computer science is a result of recent advances in computing power, storage capacity and technology, as well as the increasing availability of computer connectivity[19]. This is a broad definition of the term and, in all truth, there is no straightforward way to define and categorize this vast area of computer science. Even the name has debate to what it should be, with Data Mining being the most accepted. However, other similar meaning terms, like data analysis, pattern analysis or data dredging[18] are also used.

To address this topic, it is firstly necessary to decide the scope of what is designated as Data Mining. In scientific literature, Data Mining is either addressed as the whole process of extracting

knowledge from large amounts of data, therefore a synonym for the process of Knowledge Discovery in Databases (KDD) or as the core step of this process [21]. This ambiguous notion derived from the rising popularity in industry and media of the term Data Mining led to less usage of the longer KDD. While this is a matter of semantics, in this work we will refer to Data Mining as one step of the KDD process.

In this section it will be presented the theoretical standard steps in a KDD system/research venture. Broadly speaking, it is possible to divide the KDD process in three core steps: **Data Preprocessing**, **Data Mining** and **Data Postprocessing**[17]. This three steps are further divided in sub-steps, which need not be separate or individual tasks, as one can observe in figure 2.2.

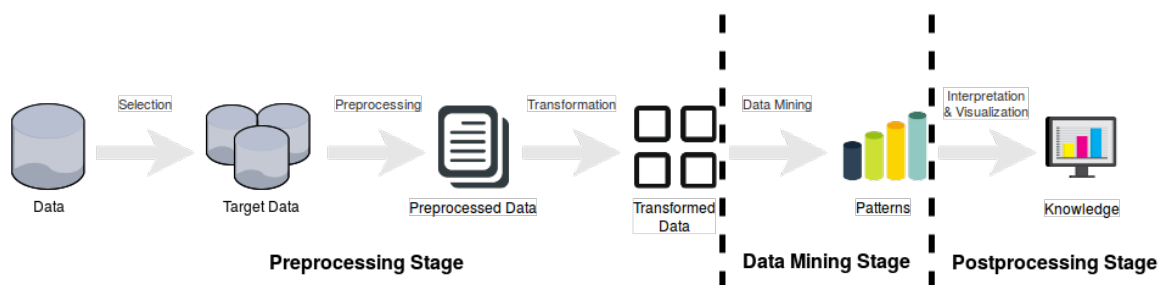


Figure 2.2: Generic KDD process

## 2.2.2 DATA PREPROCESSING

For a Data Mining research to be successful, it is imperative to have a solid comprehension of the domain about to be researched. Since Data Mining applications can be oriented to an extensive array of domains, like market analysis, customer relation, fraud detection, production control or exploratory science [18], it seems ordinary that having consistent knowledge over that same topic will increase the chance of being able to produce positive results. Besides that, it is also vital to have a well defined goal for the application and to work with data aligned with that goal[22]. These are the first two things to take into consideration before actually starting to handle data itself. This is because Data Mining applications work to detect implicit and potentially useful information that is hidden in colossal amounts of data, which most of the times is imperfect and unstructured. To put it in a simple statement, one should know what is going to do and what wishes to accomplish, before actually doing it.

Furthermore, it is necessary to understand the data about to be studied, which is in most cases unsuitable to be directly analyzed. Data can contain errors, unusual values, or inconsistencies and it is necessary to handle these issues for successful results. It is possible to categorize data quality problems by **incompleteness**, **noise** and **inconsistency**[18],[23]. Section 2.2.5 will describe in detail issues encountered when dredging in Social Media data.

Data Preprocessing is a collection of techniques that have as purpose improving the quality of data to make it suitable for the application of Data Mining algorithms. Low quality data is one of the main issues in effective Data Mining. It is directly related to the success of mined results and to the performance of the chosen algorithms[24].

A group of four Data Preprocessing techniques will be addressed, all of them pertinent to the developed work.

## DATA CLEANING

Data cleaning consists of a set of procedures that aim at resolving the issues of inconsistency, incompleteness and noise mentioned above. Frequent data cleaning methods include strategies for the handling of **missing values**[18]. Most common strategies for resolving missing values problems are:

- Ignoring incomplete samples, which is not very effective and usually doesn't result in great improvements in data quality, especially if the percentage of samples with missing values is considerable;
- Filling missing values manually, which produces quite successful results but it is time consuming and nowadays generally infeasible due to the constant growth of available data;
- Filling missing values autonomously, either with a global variable, which can create by itself a pattern in data if it attributes the same value to an appreciable amount of samples;
- Filling missing values by statistic completion, either through a mean or a median;
- Filling missing values by prediction, using the rest of the attributes in the data set.

It is more common to use strategies for autonomous sample completion when dealing with this issue, with the use of prediction being the most popular and desirable, since it is capable to relate existing information from different attributes[18]. Although the strategies presented provide answers to this common problem in preprocessing, it is also important to account for situations where missing values are not an error but a consequence of the type of data that is being dealt with. For example, when handling personal data it is most utterly common that some attributes are missing by choice, either because the person decided not provide them, like age or gender, or because it is not possible for that person to provide them, like a driver's license number in an underage user.

Another concern addressed by data cleaning processes, is *noisy* data. Noise can be defined as a random error or variance in a measurable attribute[18]. This is one of the most difficult problems in inductive Machine Learning (further developed in section 2.2.3)[25]. Methods for handling noise in data include **Binning**, **Regression** and **Clustering**[18].

**Binning** consists in smoothing a sample according to the values that surround it. This method groups values in *bins* and adjusts them in each *bin* according to a defined metric. It can be adjusted by mean, median or by proximity to the bin boundaries.

**Regression** consists in fitting the data to a regression function to be used for value prediction.

**Clustering** is described more extensively in section 2.2.3. This kind of algorithms organize data samples into groups, i.e. *clusters*, according to their values. This allows the detection of outliers, that is to say, samples that don't fall any set of discovered clusters and therefore may not be relevant to the desired task and are creating *noise* in the data set.

## DATA INTEGRATION

Integration of multiple data sources is often a necessity when performing data analysis. Three major concerns rise when addressing data integration. The first is **Entity Identification**. When



merging different data sources that are supposed to represent the same entity in only one data store, it is necessary to maintain coherence and to find a common representation of the same data from its different origins.

The second is **Redundancy**, normally caused from denormalized stores causing duplicate values and inaccurate entries. Redundancy can be detected using correlation analysis techniques.

The third aspect to look out for is **Conflicting Values**. This regards how the same type of information is stored. Data can have multiple representations, scales, or encodings for the same attribute. This frequently happens when storing values for currency, metrics or dates.

Data integration is deeply linked with data transformation techniques, required to transform and consolidate the multiple data sources into appropriate forms for Data Mining algorithms.[20].

## DATA TRANSFORMATION

Data Transformation is the process of altering data into new forms that ease and facilitate its use in Data Mining algorithms. Both data cleaning and data integration make use of data transformation techniques to ensure their goals in the preprocessing pipeline. The following are considered standard techniques for data transformation[18].

**Smoothing**, which was explained in the previous chapter, is the process of removing or correcting noise in data.

**Aggregation** which as the name states aggregates values or quantities into a more dense granularity. To exemplify, having *tweet* counts on a hourly, daily or monthly basis is a form of data aggregation into a different granularity.

**Generalization** is the mapping of a collection of raw attributes to a broader higher-level concept. An example of generalization is replacing municipalities by their corresponding districts.

**Feature construction** which is the act of creating new attributes based on existent ones, such as decomposing a date time stamp into its corresponding year, month, day and hour.

**Normalization** which is adapting the scale of numerical values to smaller ranges, normally to speed up computation.

## DATA REDUCTION

Data reduction is about selecting the data that is relevant for the desired task. It seems reasonable that the amount of data to use on data mining analysis should only be as big as time allows. Performing complex analysis techniques on huge amounts of data can take impractical periods of time. Therefore, data reduction techniques can be applied to obtain a smaller, but relevant data set whose analysis will be much more efficient with the same practical results. To address data reduction, the subsequent techniques are frequently employed:

**Data Cube Aggregation** is a form of aggregation of data into a multidimensional cube. The purpose of constructing data cubes is to create multiple levels of abstraction and hierarchy over data, to provide fast access to precomputed and summarized data, therefore benefiting the analytical process.

**Attribute Subset Selection** cares to reduce the number of existent attributes on a data set into the ones that are relevant to the mining task at hands. As an example, it is fairly obvious that if the goal is to determine the gender in a Social Media profile, the user's chosen name is more relevant to the task than his disclosed location. Irrelevancy and redundancy are usually the causes for discarding

certain attributes in a Data Mining process. Computing on a reduced set of attributes also has the advantage of diminishing the number of attributes appearing in discovered patterns, which makes them easier to understand. The “best” (and “worst”) attributes are typically determined using statistical tests[18].

**Dimensionality Reduction** is the use of transformations with the purpose of obtaining a reduced or “compressed” representation of the original data. This allows a simplified visualization of multi-dimensional data, usually by reducing it into two or three dimensions thus making it humanly interpretable. If the original data can be reconstructed from the compressed data without any loss of information, the data reduction process is called *lossless*. If instead we can reconstruct only an approximation of the original data, it is called *lossy*[18].

A standard dimensionality reduction algorithm is **Principal Component Analysis (PCA)**[26]. Usually used with a feature reduction purpose in Machine Learning classification problems with an elevated number of features (further developed in 2.2.3), PCA aims to achieve an optimal combination of those same features that can be used to better represent the data, despite having a smaller set of variables. It can also be used to reduce data into a humanly understandable dimensional space, such as a two dimensional chart. PCA is computationally inexpensive and versatile as it can be applied to ordered and unordered attributes.

### 2.2.3 DATA MINING AND MACHINE LEARNING

The Data Mining stage is concerned with the algorithmic means by which patterns are extracted and enumerated[21]. To discuss Data Mining algorithms and procedures it is indispensable to introduce the topic of Machine Learning. One of the many branches that compose a Data Mining application, **Machine Learning** is a field in computer science and Artificial Intelligence (AI) that focuses in building algorithms that can learn and make decisions from available data. Learning is discussed here in a practical and objective sense, that is, to be able to observe and compare current and past behaviors and search for improvements in performance when new situations appear. Roughly speaking we want to be able to, within a set of data, have computer algorithms that are autonomously capable of, either describe it and make assumptions about it, or to make predictions about its future. This corresponds to the two main problems that Machine Learning tries to answer in data mining applications: **Description** and **Prediction**[20].

To design a successful Machine Learning application, it is essential that firstly the “machine” understands the conglomerate of concepts on which it will be applied. It must learn a concept, through some form of input, to then produce concept description over that same learning scheme[20].

This poses a series of questions: how much information is there available to learn from? Does it have a direct or indirect relationship with the desired outcomes? Does it represent adequately the distribution of examples on which the final results will be measured? To better understand these issues, the two most influential branches in Machine Learning algorithms will be addressed.

## SUPERVISED LEARNING

Supervised Learning covers Machine Learning models in which the outcome is directly dependent on sample training data. Its a two-step process that comprises training and testing. The training step consists on building the training data set on which the model will learn a predetermined group of

classes, normally referred as *labels*. The training data can be represented by a direct  $x \rightarrow f(x)$  function on which an input (or sample) is *labeled* with a value, either discrete or numerical. Each sample is characterized by a number of *features* which are the values from which the learning algorithm will extract patterns. The training data set is humanly defined, hence the term *supervised*.

The second step consists on evaluating how the built model behaves with new input. It is done with a set of test data so it can measure how accurately it can match unseen examples to the existing labels. The success rate on test data gives an objective measure of how well the concept has been learned.

Supervised Learning problems can either be divided into **Regression** or **Classification**. In Classification problems, the classifier tries to learn a function that maps, i.e. classifies, a certain entry to one of several previously defined classes. Regression problems are actually quite similar to Classification problems, only instead of having a set of discrete values as the target, the predictor is fed a continuous variable as the label and tries to understand what is the next value to appear[20].

## SUPPORT VECTOR MACHINES

Support Vector Machines (SVM)[27] are a popular Supervised Learning algorithm that can be used either for Classification or Regression. It is characterized by its optimal performance on high-dimensional vector-spaces[28] i.e. working with a considerable number of features. Being a Supervised Learning algorithm, the goal of SVMs is to construct a model based on training data that can produce predictions of target values for new data, based on its attributes. Mathematically, given a data set of samples-label tuples  $(\mathbf{x}_i, y_i), i = 1, \dots, l$  where  $\mathbf{x}_i \in R^n$  and  $\mathbf{y} \in \{1, -1\}^l$ , SVMs work the solution for the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^l \xi_i \\ \text{subject to} \quad & y(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned}$$

The equation above states that each training vector  $\mathbf{x}_i$  is mapped into a greater dimensional-space by function  $\phi$  and the SVM will find a separating hyperplane with a maximal margin error within the new space. SVMs come with an associated kernel function, in charge of applying a transformation in data that will determine the new dimensional space for further calculations. Kernel functions are generally described as:

$$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i^T) \phi(\mathbf{x}_j)$$

If the SVM kernel function is of the following type:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$$

it is considered a linear kernel. Support Vector Machines with a linear kernel usually provide good classification performance when the number of features used to build the classification model is greatly superior the number of existent samples in the training data, since a projection to an high-dimensional space is not necessary due to the already high dimension of feature vectors [29].

# VALIDATION

In order to consider a classifier or predictive model useful, it must be verified at what degree the results it produces are accurate. To do this, it is standard procedure to use a testing set of labeled data to measure how accurately it performs, by verifying the percentage of produced labels that match the annotated ones. In practice this consists on dividing training data into two groups and it is a simple and reliable way to get an indicator of how well the model performs.

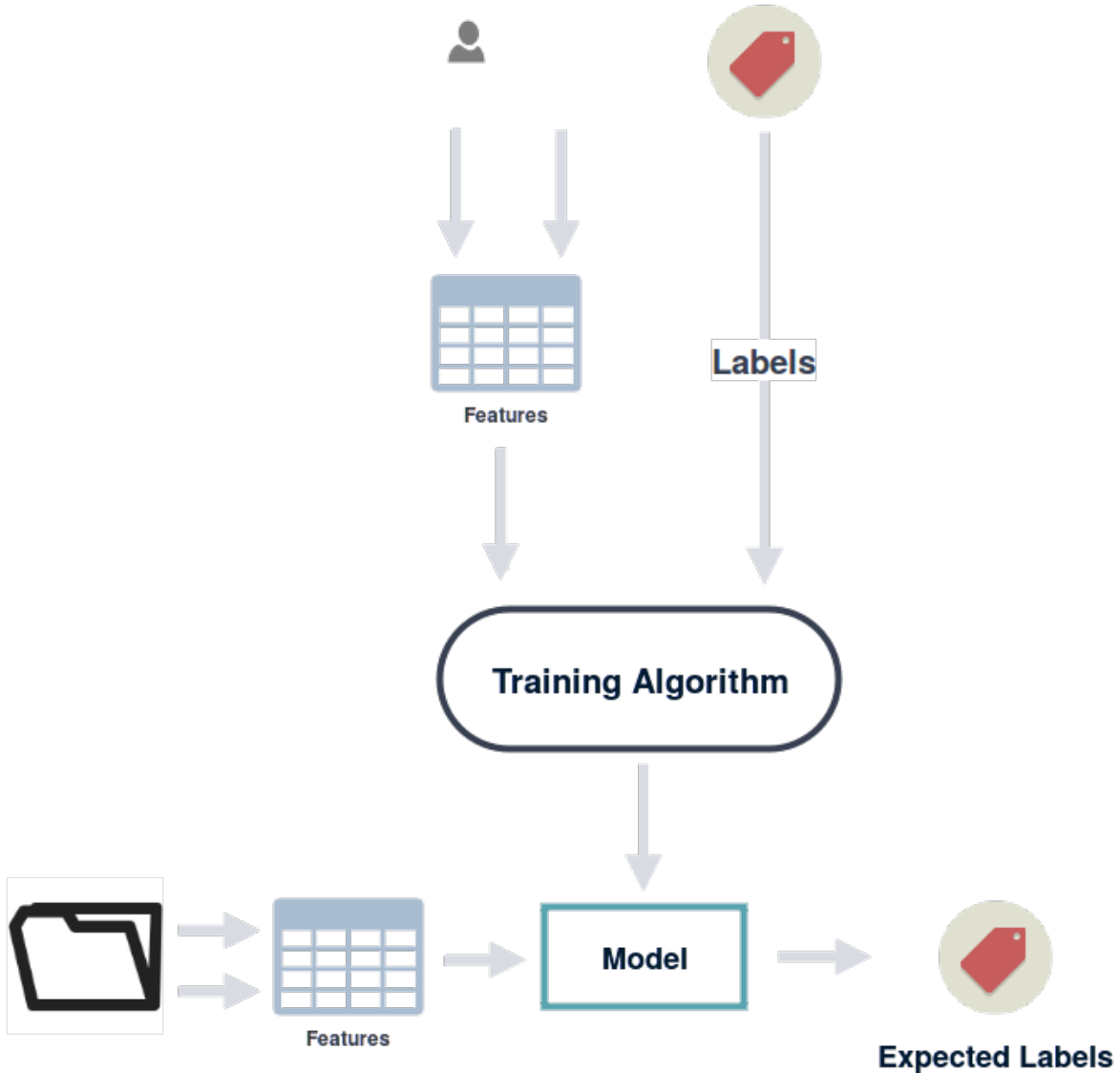


Figure 2.3: Diagram of a standard Supervised Learning application

Another technique employed to verify a model’s accuracy is the **K-Fold Cross Validation** method[28]. It consists on dividing the training set in K equally sized parts and then train the model K times, where the testing set changes every time and find the average accuracy of the K produced models. This measure is more robust than the first and accounts for a more general accuracy of the algorithm, but it is also time consuming.

## UNSUPERVISED LEARNING

Contrarily to Supervised Learning, Unsupervised Learning algorithms are used when dealing with unlabeled data[17]. They build the learning model based on how similar sample data is and they do not depend on training examples. It is a form of learning by observation, rather than by example like in Supervised Learning[18].

Unsupervised Learning can be further divided into the following categories of problems: **Clustering**, **Dimensionality Reduction** (see section 2.2.2), **Anomaly Detection** and **Density problems**[30].

**Clustering** is a term that often overlaps the definition of Unsupervised Learning itself, mostly due to its popularity. It aggregates a set of techniques to divide unlabeled data into *clusters*.

Clustering algorithms usually work to detect some underlying mechanism at work in the domain in which instances are drawn, a source that causes certain instances to be more similar and other more distant. This process results in grouping different objects into clusters where objects within a cluster have high similarities, based on their attribute values[20].

In Data Mining, cluster analysis can be used as a stand-alone tool to gain understanding on the distribution of data, to examine the characteristics of each cluster and to focus on a particular set of clusters for further analysis.

**Density** is another popular category of Unsupervised Learning problems, also known as **Association** problems, which present a series of meta-data that allows the definition of strong relations between different variables of the input dataset[31].

**Anomaly Detection** is a category of problems whose algorithms allow the detection of unexpected values, for instance by checking distances between neighbours (similar to clustering)[32].

Finally, **Summarization** focuses on the creation of meta-data about the input dataset, such as variance, percentiles and so on. This allows a general interpretation of the information that is available. Summarization also includes the **Dimensionality Reduction** addressed in the previous section.

### 2.2.4 DATA POSTPROCESSING

Postprocessing is the last stage in the KDD pipeline. Whatever the goal is in a DM system, if the extracted knowledge is not useful or readable then the effort it took to obtain is meaningless. Therefore, one concern to take into consideration when developing these kind of applications is how the results are going to be presented, i.e. how can the end-user visualize the information that is discovered. There is further processing to be made with the bits and pieces of derived knowledge, whether it is to simplify it, visualize it or document it [21], [33].

Postprocessing techniques are essentially linked with the necessity of **Interpretation**, **Evaluation** or **Integration** of knowledge[33]:

**Interpretation** deals with understanding acquired knowledge and putting it to practical use. Depending on the given goals, information obtained can serve as a base for further research and/or compared with previously obtained knowledge to support it or contradict it. Furthermore, it can be used straight for prediction/classification in Machine Learning tasks or, if it has an end-user as a target, documented or visualized in an appropriate fashion.

**Evaluation** is the act of validating induced knowledge. This area of postprocessing compiles a group of techniques used to assert that the constructed learning models fit the proposed goals of the system. A model can be validated in terms of accuracy, performance, complexity or comprehensibility[33].

**Integration** handles the need for new decision-supporting systems to combine or refine results obtained from several models, instead of depending on a single one. The ability to integrate and blend conceptual models induced from different methods improves the probability of success and accuracy rates[34].

## DATA VISUALIZATION

When a Data Mining system's purpose is to meet an end-user that will make use of it for its own sake, it is of the utmost importance to present data so that it explicitly matches the user's needs in term of accessibility and usability.

Visualization tools must visually stimulate the user and its ability to identify patterns, in other words, these tools should enable cognition [35]. There is research made in the field of Data Visualization that has proposed some ground principles for the development of effective representations of data[36]. These principles are hereby presented:

The **Appropriateness Principle** assumes that the visual representation should provide only the required amount of information that is necessary for the task it was assigned. Any additional information can be distracting and makes the cognitive process harder.

The **Naturalness Principle** states that experimental cognition has more efficiency when the traits of the visual representation are closer matches to the information being represented. When a representation does not match the cognitive model of the information established by the user it may interfere in his understanding.

The **Matching Principle** takes into consideration that visual representations should be suggestive of the action they execute. Representations are more effective when they resemble the task to be performed.

Visualization techniques also have advantages for end-users when compared, for example, with text displaying since it stimulates them to use intuition and recognition processes for understanding, which are more natural than content assimilation through reading[35]. Moreover, they have the capacity to show nuances in data that are rather troublesome to explain in text, like similarity and proximity through shape sizing and colouring, or contextualization through highlighting techniques. It also has conciseness and clarity advantages since they enable the representation of tremendous amounts of data from different sources or backgrounds at the same time.

### 2.2.5 CHALLENGES WITH SOCIAL MEDIA DATA

It is safe to say that Social Media platforms generate a lot of data, though it is a quite particular and distinct form of data when compared to traditional Data Mining sources. Social Media data results from the combination of **User-Generated Content** on Social Media platforms and the social relationships inherent in those same platforms[23].

User-Generated Content possesses characteristics that makes it a unique form of data. It is vast, noisy, distributed among different platforms, unstructured since those same platforms target different user needs and at last dynamic, due to the continuous evolution and transformation of Social Media[17].

Apart from the User-Generated Content, the social interactions that take place in Social Media and its subsequent Social Networks represent a major attraction, because they enable the integration of social theories to computational knowledge, namely the study of how individuals behave and interact

with each other and with other entities like a brand, a political party or an event. Merging these concepts results in an unprecedented new form of data that poses new challenges and opportunities for Data Mining research.

## BIG DATA

Big Data is one the most resounding buzzwords in the computer science. A simple research on Google Trends<sup>7</sup>, shows that the term had a continuous increase in web searches that started to manifest itself in late 2011. This evolution is shown in the following figure.



Figure 2.4: Google Trends result for the term "big data".

The evolution of information systems, the appearance and proliferation of mobile devices and the general increase of access to the Internet and bandwidth rates led to a massive escalation of the amount of data created on a daily basis. Big Data is the area of computer science that deals with these new challenges and opportunities the derive from the need to store and analyze tremendous amounts of data from different scenarios and with different structure (or lack of it), in an practical way and within a reasonable time period[37]. One of the phenomenons that gave meaning to the term Big Data was Social Media, with its ever increasing user base, expansion to multiple platforms and devices and inherent nature to deal with user content.

Big Data is relative - As paradoxical as it seems, Social Media is a common source of Big Data but not a trivial one. What causes this, is the fact that data generated in Social Media has a very high-level scope that continuously loses precision and volume as specific users or topics are targeted, requiring the social characteristics of Social Media to be explored and combined. It is not the data that exists about users as a unit that makes Social Media data "big" but the sum of its relationships between every user. Social Media is then, a unique source of big data[23].

<sup>7</sup><https://www.google.com/trends/>

## LINKED DATA

Social Media data describes the existing social ties and interactions between users in Social Media platforms, as well as the inherent links between users and their corresponding content. The availability of these kind of connections between data points makes Social Media data intrinsically linked. Examples of Social Media data linkage are pretty straightforward when taking Twitter in consideration. "Who posts what" and "who follows who" are two direct examples of connectivity between data in a real-life Social Media scenario[38].

These connections are also statistically dependent and make data non identically distributed which contradicts one of the most enduring assumptions in Machine Learning (ML) methods, that is statistical independence among data instances. This poses a series of difficulties in Machine Learning processes, such as feature selection [39].

When working with Social Media data it is important to take all of this into consideration and work out solutions to overcome the issue of linked data, and embrace it as an advantage for Data Mining research. Graph theory is one of the main methods used in Social Network analysis since the early history of the Social Network concept[40]. This approach is applied to Social Media analysis in order to determine important features of the network through its modeling into nodes and links.

## NOISY DATA

Social Media data contains a considerable portion of noise and spam content[41] and this constitutes a dilemma. There is a balance point in which noise should be removed or not, since removing all noisy data without consideration can lead to the problem described in section 2.2.2. Social relations are also noisy by themselves, as it is difficult to distinguish, for example, the strength of a link between two users.[23] Besides that, noise is by itself difficult to define and dependent on the task at hands.

## UNSTRUCTURED DATA

User-Generated Content is in the majority of situations considerably unstructured. This statement gains even more relevance when addressing text data. The proliferation of the use of mobile devices to publish text content (like *tweets*) has created several distortions in communication rules[23]. A great portion of online text-based content has a minimal appreciation for spelling rules or grammar and poses problems both at a lexical and syntactical levels. Text is usually short-length, informal, contains spelling mistakes, makes frequent use of abbreviations and freeform language like non-existing words, abbreviations or emoticons[42]. These properties require proper text processing techniques to make this kind of content useful for data mining. Twitter data is good example where this situation occurs. The notation used in Twitter dynamics requires specific handling, since it can be present or absent in the tweet and manifests itself in mostly in the form of special characters present in text, that have an underlying meaning that is not necessarily straightforward. As an example, the "@" character can be a mention, a reply or a quote according to its position in the tweet and the content that follows it.

## OTHER CHALLENGES

Access to Social Media data is limited and controlled due to personal privacy considerations[43]. This creates a problem of incompleteness in data sources, which was discussed in section 2.2.2, raising



difficulties in the task of gathering enough sample data that meaningfully represents the target research domain. It also common for Social Media companies to impose limits in the access to their data, sometimes denying it completely. Furthermore, since the content of Social Media is mainly user-generated, it is complicated to establish a truth base from which patterns and knowledge will derive.

## 2.2.6 RELEVANT RESEARCH TOPICS

In this subsection it will be presented some research topics in sub-areas of Data Mining that have a relevant impact when dealing with Social Media environments.

### TOPIC MODELING

**Topic Modeling** is a research area in text mining that works on the ability to analyze large collections of unstructured documents with the purpose of organizing them under different subjects[44].

Currently an emerging area, Topic Modeling algorithms make use of statistical and probabilistic theories to understand the correlation between words in the original text and apprehend how their joint appearance relate to an underlying theme. They do not require human annotation or observation to operate, since their analysis works over text and not discrete values or attributes.

Topic modeling's major purpose is enabling the autonomous summarising of electronic archives at a scale that would be impossible by human standards[45]

### LATENT DIRICHLET ALLOCATION

**Latent Dirichlet Allocation (LDA)** is a generative probabilistic model for collections of discrete data such as text corpora, presented in 2003 by David M. Blei et al [46]. The theoretical foundation behind LDA is that text documents are represented as random mixtures over veiled topics where each of them, which is in practice a latent multinomial variable, is characterized by a distribution over words.

LDA is a three-level hierarchical Bayesian model in which, each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is in turn modeled as an infinite mixture over an underlying set of topic probabilities.[46]

The following is a simplified description of how LDA works for each document  $\mathbf{w}$  in a corpus  $D$ :

---

**Algorithm 1** LDA basic model

---

- 1: Choose  $N \sim \text{Poisson}(\xi)$
  - 2: Choose  $\Theta \sim \text{Dir}(\alpha)$
  - 3: **for all**  $w_n$  words in  $N$  **do**
  - 4:     (a) Choose a topic  $z_n \sim \text{Multinomial}(\Theta)$
  - 5:     (b) Choose a word  $w_n$  from  $p(w_n|z_n, \beta)$  a multinomial probability conditioned on the topic  $z_n$ .
-

## INFLUENCE ANALYSIS

Social Media is dependent on connections. Social interactions are emulated into a virtual platform, by direct and indirect links between users, their characteristics and behaviors. The combination of these three factors establish the network structure. Network connectivity helps observing distinguishable patterns, one of them is social similarity, which states that similar nodes are more likely to be connected to each other. Analogously to the real world, user behavior and relationships, along with network characteristics, shape the network structure and induce changes in other fellow peers.

The phenomenon of social influence, an intuitive and well accepted concept in Social Networks[14], can be defined as the interactions with other peers that cause them to change their behavior[47]. Along with influence, another observable pattern in network analysis is **Homophily**. Homophily is the idea that individuals are naturally more likely to form ties with similar peers and those similarities create correlated effects among their neighbours.[48]

These two aspects of social behavior create what are called assortative patterns in networks[16]. Both of them have the end result of creating networks where similar individuals are more likely to be connected however, it is important, but challenging, to distinguish them since, for example, information diffusion processes differ from homophily-driven networks to influence-driven networks[49].

## INFLUENCE MEASURING

In the context of Social Media there are two approaches to determine influence, by prediction or by observation[18]. The first consists on making use of network centrality measures to define the most important nodes and make predictions based on their attributes and distribution. The second quantifies individual influence by calculating the amount that can be attributed to each node. Being observational, it focuses on palpable phenomenons and is usually connected to real world situations. Measuring parameters can consist of: the size of an audience of a certain media personality, how many people he can affect when spreading information or which value was generated from endorsement of a certain individual[18].

## CENTRALITY MEASURES

As said before, Social Media, more specifically Social Networks can be represented as a graph due to its data being inherently linked (see section 2.2.5). Graph theory states that a graph  $G$  is represented as  $G(V,E)$  where  $V$  is a set of vertexes (or nodes) and  $E$  is a set of edges (or links) connecting vertex pairs in  $V$ [50]. To represent Social Networks under the form of graphs, it is necessary to address graphs more in an abstract sense rather than mathematical. It is intuitive to understand that a graph representation of a Social Network will assume that nodes will represent entities in the network, either users or any form of content, and edges will serve as relationships occurring in the network.

Having a Social Network represented as a graph unlocks the possibility to conduct graph algorithms in that same network. There is a wide spectre of algorithms that can be useful in graph analysis such as finding a node's degree, finding the connectivity between its neighbours, finding a path between two nodes, or finding the shortest path from one node to another. When addressing influence analysis, centrality measures can be quite useful. The standard centrality measures used in graph theory are **Degree Centrality**, **Betweenness Centrality** and **Closeness Centrality**[50].

Degree Centrality is simply counting the number of edges that are linked to a specific network node.

If the graph is directed, that is, its edges have a source and a target, then degree centrality can be further divided into **In-degree** which accounts only for the edges that target a node and **Out-degree** which does opposite, meaning it only accounts for the edges that have specific node as source.

Betweenness Centrality determines the number of times a node is visited when calculating the network's shortest paths. A node with a greater "betweenness" value will be more central in the network as it is a passing point between other nodes.

Closeness Centrality takes into consideration the distance between nodes in the network. In this measure, a node is more central when the sum of its accumulated distances between all other nodes is smaller.

There are many other centrality measures in graph analysis, some used when dealing with weighted graphs where links between nodes have an associated weight, which in this work was not taken in consideration. Others consist of variations of the presented measures.

## INFLUENCE ON TWITTER

Twitter is a Social Media platform concerned with information diffusion. Users choose what users to follow and what information feeds to subscribe. Influence on Twitter can be attributed to the users that have a notable impact in the spread of information and behaviors. Although there are different types of users, they all communicate through the same way, which is by tweeting, i.e. the act of posting a tweet, to their followers. This creates a standard way to compare the different target attributes for influence measurements.

There are some standard metrics to look out for when trying to quantify influence on Twitter. Intuitively, we can think of the number of followers as a measure of influence but this is not the only way to do it. There are three common attributes to evaluate when conducting this kind of investigation, **number of mentions**, **number of retweets** and **number of followers**[16].

The number of mentions is the number of times a user is mentioned in tweets. The mentioning mechanism is explained in section 2.1.3. Mentions denote one's ability to engage others in conversation.

Number of retweets is the number of times that a user's tweets are retweeted. The retweet mechanism is also explained in section 2.1.3. Clearly, the more one's tweets are retweeted the more likely one is influential. The number of retweeted indicates an individual's ability to generate content that is worth being passed along.

The number of users following a person on Twitter is, as discussed, the number of individuals who are interested in someone's tweets. Followers account for an audience measure, meaning it will indicate how many people one person will reach directly.

Although an immensely popular metric, the number of followers has been proven as a meager way to quantify influence in the Twitter network[51]. This measure is more suggestive of popularity, but not as reliable when accounting for the notion of influence. On the other hand, *retweets* and mentions respectively provide insight regarding content value and the user's "name" value[51].

Some research has been made in determining influential users on Twitter using more complex measures. In [52], it was found a relation between the diffusion rate of tweets with Uniform Resource Locators (URLs) and the users that generated them. It was found that a user with more followers and influential in past situations has a tendency to be in origin of the content. An altered version of PageRank[53] as also been used to determine influential users in the Twitter network, although the results do not differ much when compared to using the number of followers as the standard measure[54].

## PROFILING

User profiling is the act of inferring a user’s personal attributes, like gender, location, interests or employment status. In Social Media, this kind of information is available at some level, generally not fully available, since some of it not always shared online due to privacy concerns. There is also the case of it being inconsistent, when a user chooses to provide fictional values [55], [56].

Comprehending and determining user characteristics is a very prized goal in areas like targeted-marketing, online applications that have personalization concerns and recommendation systems.

Establishing a user profile is not a straightforward process, there has been some investigative effort towards predicting attributes like gender, age, location, occupation or interests but some issues arise in the different approaches made. Features like, network structure, the user’s known attributes, generated content or his activity provide some ground data to process but none is quite enough by it’s own, they are required to be combined to improve results.

User profiling approaches fall under the following categories[56]:

**Content-based Profiling** where content is used, whether it is text, a URL or a picture, as the source to identify user attributes. Analysis of content can be made either with low-level characteristics, like word or character counts, keyword identification, generic text-mining algorithms or through more complex high-level models like sentiment analysis or topic modeling algorithms (see chapter 2.2.6).

**Network-based Profiling** studies the Social Network structure and existing links between data points as source for inferring user attributes. The user’s network is commonly employed as a profiling resource when we are able to represent the network as a graph. In [57] there is an example of a predictive model for geo-locations based on Twitter user relationships.

Although these are the most common approaches for extracting latent user attributes, there are other resources that can be taken into consideration in profiling tasks. The visible and accessible part of the user profile can be useful as social context indicator, although the accessibility of this kind of data varies from Social Media platforms as well as the range of attributes that are available. Furthermore, the user’s activity (also referred as user behavior) inside the platform can also have profiling value. As an example, it is common to find users in Social Media platforms who tend to be more pro-active whereas others tend to be more passive which can be a relevant trait to investigate.

### 2.2.7 RELATED WORK

This subsection presents research projects that relate to the proposed dissertation. They have in common the use of Twitter as a data source for their proposed goals.

In the study conducted by Calado et al[55] in 2011, it was evaluated how user activity can be a valuable indicator for user profiling, regardless of the content of his posts. To do this, they modeled user retweet chains and network diffusion chains, with identification of network jumps, which happen when a user diffuses content of another user to which he is not linked. Afterwards, they produced a set of features based on the user’s position in this constructed networks. These features were used on an Expected-Maximization clustering algorithm to group the users in different clusters. The obtained results managed to assign a user to a cluster with 90% probability and denoted some interesting patterns in the generated clusters, such as finding users that are commonly the source of new content, users that propagate content or users with low activity rates.

In the research of Rao et al[58] and Burger et al[59], two approaches were made for latent classification of user attributes. In the former the focus was to develop Supervised Classification

models for the inferring of four user profile attributes: gender, age, regional origin and political orientation, using four data sources from a user profile: his network structure (number of followers, friends and follower-friend ratio), his communication behavior (frequency of response, retweet and post), two models derived from their tweet contents, one based on socio-linguistic attributes like the use of emoticons and other writing behaviors. The other was the counting of generated character and word ngrams from tweets. This study verified that its first two information sources provided little value in inferring user attributes, while the two models had interesting accuracy rates for each of the classification tasks. The latter study had the objective of discriminating the gender of a Twitter user using his profile information and the content of his tweets. It built a classification model based on the presence of character and word ngrams in each of the four fields, separately and joined together to reach a maximum accuracy of around 90%. Both of these studies were made exclusively analyzing English speaking users and content. The first one used a dataset containing 2200 users and the second used around 100000.

TVPulse<sup>8</sup> is a project developed in Instituto de Telecomunicações of Aveiro, in a partnership with PT Inovação<sup>9</sup> that experiments with tweet content recognition and detection of postings that are pertinent to events in television programs.

The proposed goal of TVPulse is to build a system that extracts and stores tweets and information from publicly available Electronic Programming Guides (EPGs) and afterwards performs a group of text processing and data mining techniques to match tweets to television shows based on their semantic similarity. It also accounts for event detection during live emissions[60].

POPmine is an open-source system developed at Universidade do Porto that encompasses data collection, information extraction, opinion mining and visualization of political opinion on the web. This system has been put to use under the website POPSTAR<sup>10</sup> for tracking the political opinion of Portuguese major politic party leaders[61].

In POPmine, opinion tracking was divided regarding buzz and sentiment. Buzz is an indicator of trend, it accounts for the percentage of mentions of a certain figure among the overall mentions of the all figures being tracked, as well as the number of mentions in absolute. Sentiment concerns about tweet mention polarity (positive or negative). It aims to determine whether a certain figure is regarded by the public in a positive sense or the opposite, through the percentage of positive and negative mentions of that same figure. Both of this indicators are presented as an evolution over time, allowing the contextualization between real-word events and eventual changes in the opinion scores.

## 2.3 TECHNOLOGY REVIEW

In the previous sections, Social Media was introduced along the theoretical foundations necessary to create a KDD pipeline focused on Social Media data. It was explained the issues encountered when handling this type of data and the necessary precautions to take to correctly process it. This section aims to evaluate and explain some of the technical components that are necessary to build a system supported on Social Media data and that enable the theoretical knowledge addressed before to be put to practice. It will focus on the storage and processing components of the KDD pipeline.

---

<sup>8</sup><https://www.it.pt/Projects/Index/2049>

<sup>9</sup>PT Inovação was rebranded to Altice Labs after the acquisition of Portugal Telecom by french telecommunications company Altice

<sup>10</sup>[www.popstar.pt](http://www.popstar.pt)

### 2.3.1 GRAPH DATABASES

A graph database is a database system in the family of the Not-only Structured Query Language (NoSQL) ecosystem with Create, Read, Update and Delete (CRUD) methods that exposes a graph data model[62]. Formally, one can define a graph as a set of nodes, which represent entities related to the real world and relationships that connect them.

Graph database models introduce a way of representing data where information about its inter-connectivity and relations has as much importance as its content. Also, data manipulation can be done in a graph oriented fashion, making use of graph properties and traversal algorithms and, since graphs are visual structures, they provide a more natural way of modeling data[63].

Graph data models usually fall under the following categories: **Property Graphs**, **Hypergraphs** and **Triples**.

**Property graphs** are the most common graph data models, in which a graph contains nodes and relationships and both can contain multiple key-value pairs, named as properties, and multiple labels. Relationships are named and always possess a start and an end node.

This type of data model is extremely intuitive and encourages the use of visual representations to achieve it. It can easily represent any domain that can be translated into entities and relationships through the use of whiteboard or paper sketches with relatively little complexity. Figure 2.5 is an example of this approach where it is possible to find a direct mapping of Twitter dynamics into a property graph data model

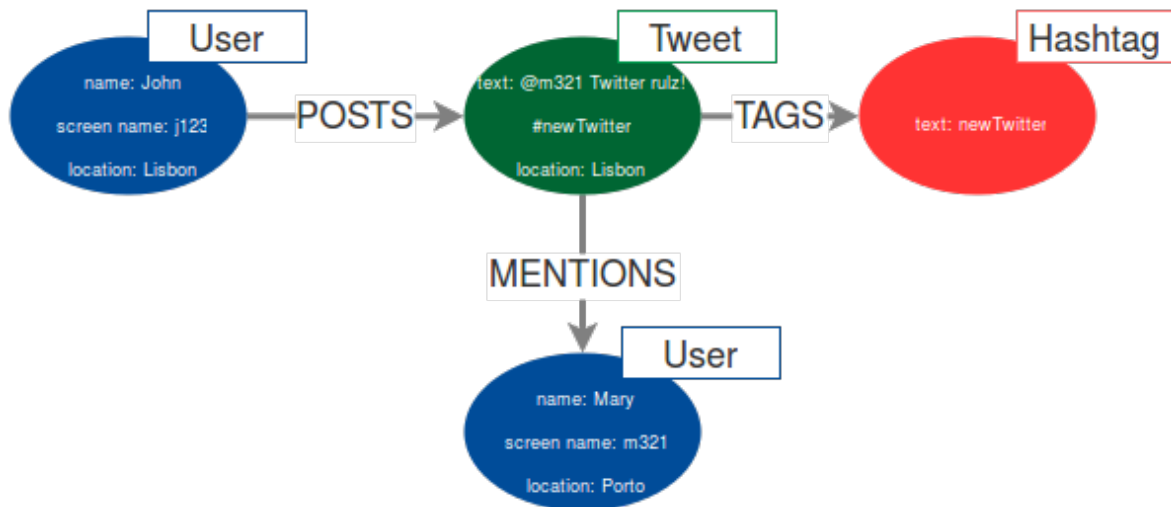


Figure 2.5: Example of a property graph of Twitter dynamics

**Hypergraphs** are similar to property graphs. The Hypergraph model adds a degree of complexity by allowing relationships to have more than a start and an end node, meaning it allows many-to-many relationships.

A **Triple** is a subject-predicate-object structure[64]. This model originated in the semantic web movement with the interest to add semantic markup to the links that connect web resources [64]. It differs from the other two models because it can't be used in a native graph database since each triple is viewed as an independent item that represents a logical link. This implies that it needs an additional layer to connect each independent structure and to allow querying or traversals.

When surveying graph databases, there are two particular aspects to take into consideration, which are its underlying storage and its processing engine. These properties can be classified as native or

non-native. If referring to storage, a native graph storage means the data is stored directly as a graph structure, not having any kind of transformation or serialization for this purpose. When referring to its processing engine a graph database with native graph processing properties means that it leverages on index-free adjacency, that is, with nodes that are connected to each other “physically” and not using meta information[62].

The adoption of graph databases with a property graph data model can bring improvements in terms of:

- **Performance.** Due to its inherent nature of handling heavily linked data, graph databases can have an increased performance when compared with relational databases and NoSQL solutions when dealing with operations on data relationships. This performance is maintained when the amount of data scales because queries are local to the portion of the graph which is necessary to search, rather than the overall data set.
- **Flexibility.** It is rather useful to be able to adapt data to constant changes in the pretended modeling domain instead of having it being imposed upfront of development. This is a feature that the graph data model fundamentally expresses due to its additive nature. It is possible to add new nodes, relationships and substructures to an existing model without threatening it, resulting in a tendency to perform less migrations, reducing maintenance and risks [62]
- **Agility,** since the graph data model is usually schema-free and matched with API and/or a specific query language, which empowers application developers to incrementally evolve the data model with the rest of the application, in the same measure as current iterative software delivery practices.

The advantages of using a graph database for a system based on Social Media data are quite straightforward. Having in section 2.2.5 described the challenges of handling Social Media data, one of them is automatically tackled if the Social Media is modeled through a graph database, which the challenge of linked data. By using a graph database with a property graph data model, it is possible to make a direct projection of the Social Media network into the database without sacrificing almost any data connectivity. Having a graph representation of the network also allows graph theory to be applied to the modeled network. In section 2.2.6, it was discussed the topic of influence detection in a Social Network, as well as defined some graph metrics that can be used to accomplish that goal. Those metrics can all be obtained quite easily with a graph database.

### 2.3.2 NEO4J

Neo4j is arguably the world’s leading graph database system[65] and it is currently used in production with different purposes in several business giants like Ebay, Walmart, HP or Cisco[66]. It had its first release in 2007 and at the date of writing it currently stands first in the DB-Engines Rank for GDBMSs[67]. Neo4j is an open-source<sup>11</sup> disk-based graph database system written in Java that implements a native property graph structure consisting of nodes, relationships and properties. Here are presented some of the features that make it stand out as the most the used graph database in the world.

---

<sup>11</sup>There is also an Enterprise Edition of Neo4j with extended features and capabilities, in this work we will focus only on Community Edition

## SCHEMA-OPTIONAL PROPERTY GRAPH DATA MODEL

As explained in the previous section, property graphs are graphs composed by entities and directed relationships where both of can contain properties and the former can also contain labels. These are the fundamental building blocks of the Neo4j data model. Properties follow a key-value structure where the key must be a string and values can be numerical, String, Boolean or a list of any of them. Labels define roles and meaning to nodes in the model. In Neo4j, the use of Labels is what defines the model's schema. With the use of Labels being optional, Neo4j is considered a schema-optional database. Labels are important because they group all nodes with the same Label into a set and it is frequent in database querying to work only with a subset of the whole data. A node can have any number of Labels.[68]

Neo4j's data model also allows the use of indexes and constraints over node properties when nodes are labeled. The first brings a more efficient node searches and the second is beneficial for data integrity purposes, as it can, for example, define unique properties that won't allow duplicate nodes. All schema modifications in Neo4j can be done without affecting the existing data, which is a major asset in terms of model flexibility and an advantage when dealing with semi-structured data.[69]

## NATIVE GRAPH STORAGE

Neo4j graph storage system is done directly on the file system. It was originally designed to store and manage graphs, without using relational or object-oriented middlewares. Neo4j stores graph data in a number of different store files. Each store file is responsible for a specific part of the graph, meaning that there are files responsible for nodes, others for relationships, others for properties and so on. The division enhances the performance of graph traversals, a key feature in a graph database. A full specification of the internal storage system of Neo4j can be found in[62]. Neo4j is also transactional and Atomicity, Consistency, Isolation, Durability (ACID) compliant, although it does not follow this model as rigidly as traditional relational databases. The Consistency factor in Neo4j is variable due to its schema nature. If we are using it without any schema, the only restraint for consistency is that all relationships have a start and an end node but as we begin to define its schema, restraints in data will become more strict. All this put together turn Neo4j into a multi-purpose database system, with the traditional features of Relational Database Management Systems (RDBMSs), but with an extremely different data model that is suitable for highly connected data[69].

## CYPHER QUERY LANGUAGE

Cypher is a declarative, pattern-matching query language developed for Neo4j. Cypher is built on the basic concepts and clauses of SQL but with added graph-specific functionality, making it simple to work with a rich graph model without being overly verbose.

```
MATCH (u:User) -[:POSTS] ->(t:Tweet) -[:TAGS] ->(h:Hashtag) RETURN u,t,h
```

Listing 1: Example of a Cypher query



The query listed above will retrieve from Neo4j all users, tweets or hashtags and the POST and TAGS relationships that link them.

## LANGUAGE DRIVERS AND HTTP API

The Neo4j Driver API is the preferred means of programmatic interaction with a Neo4j database server. It implements the Bolt protocol, a proprietary binary protocol to communicate with the database, and it is available in C#, .NET, Java, JavaScript, and Python. The API is defined independently of any programming language. This allows a high degree of uniformity across languages.

The Neo4j transactional HTTP endpoint allows the execution of a series of Cypher statements within the scope of a transaction. The transaction may be kept open across multiple HTTP requests, until the client chooses to commit or roll back.

## WEB INTERFACE

Neo4j provides a web interface that can be used to perform Cypher queries and visualize the currently stored graph. It is also possible perform monitoring operations, performance tests and schema modifications.

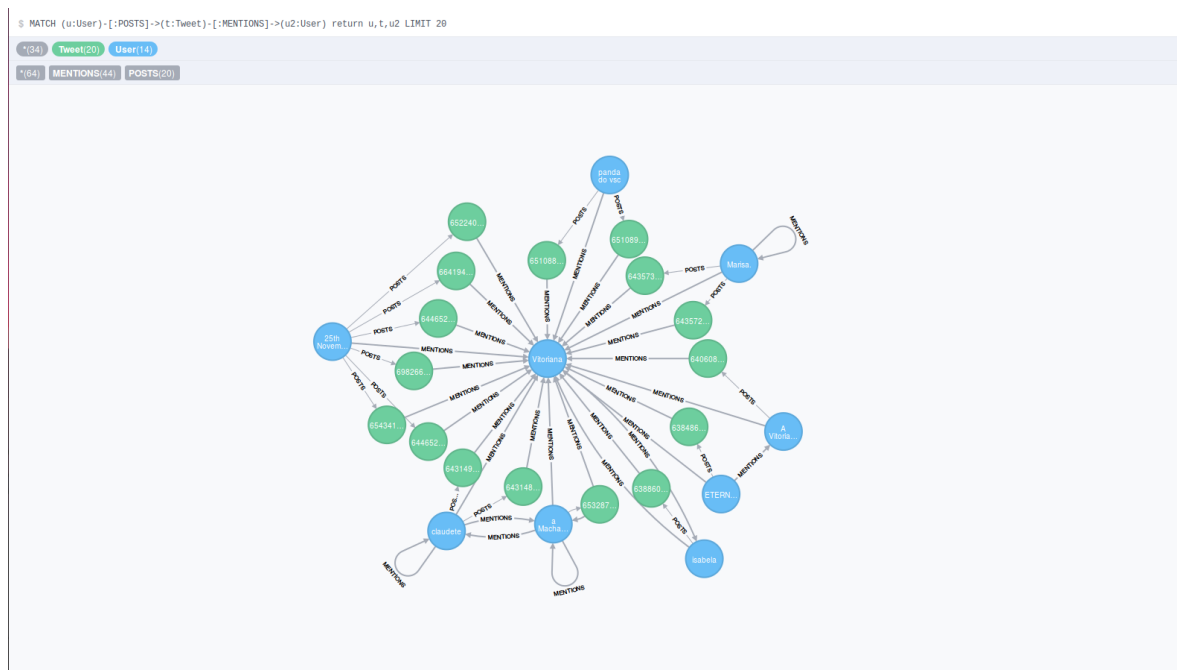


Figure 2.6: Graph view on the Neo4j web interface.

### 2.3.3 ORIENTDB

OrientDB[70] is a multi-model Database Management System (DBMS). As its creators put it "it is a second generation distributed graph-document database". The meaning behind this sentence is

that OrientDB uses a graph data model where each element in the database is a JavaScript Object Notation (JSON) document, but relationships are handled through links between elements thus exposing a graph data model. Both nodes and links can be considered a JSON document. OrientDB had its first release in 2010 and so far stands in second place in the DB-Engines Rank for GDBMS[67]. Being multi-model it is also scored in the rank for key-value stores and document-oriented databases, where it stands at sixth place in both. OrientDB is open-source<sup>12</sup>, written Java and supported in all operating systems with a Java Virtual Machine (JVM). It is used in production environments in companies like Sky, Comcast, Ericsson and Cisco[71].

## SCHEMA-OPTIONAL MULTI-MODEL DATABASE

OrientDB is a versatile DBMS since it offers more than one possible data model. Its data model has as its most basic unit a **Record**, which can consist of a Document, RecordBytes (binary data), a Vertex or an Edge. These four basic units allow that OrientDB can be used either as a key-store or document oriented database, when Vertexes and Edges are discarded or as a graph database, when they are used. It is worth noting that OrientDB processes Vertex and Edge records as documents as well, allowing the attribution of properties to them and therefore enabling a complete property graph data model. Properties data types supported range from the primitives Boolean, String and Integer to complex data structures[72]. All Records in OrientDB are assigned a unique identifier (Record ID) that identifies them among the existing clusters and inside them.

In terms of schema compliance OrientDB is, like Neo4j, schema-optional though the inner workings of its schema definition differ. In Neo4j, schema is defined by the use of labels and constraints while OrientDB can enforce a class level schema. This allows the creation of well defined document classes, much like in the Object Oriented programming paradigm, that can have either a strict property definition where all properties in the class are imperative, hence schema-full, an arbitrary property definition, therefore schema-less or a balance between, where some properties can be mandatory while allowing the creation of new ones. This last model is defined as an hybrid schema[73]. OrientDB also allows the creation of indexes and constraints over properties.

## FAULT-TOLERANT DISTRIBUTED STORES

Unlike Neo4j's Community Edition, OrientDB has a distributed architecture, allowing it to exist spread among different machines. It uses a master-less architecture where each server can read or write records though it also supports read-only servers called REPLICAS[74]. Writing behavior in a distributed deployment of OrientDB follows a logic of ownership. Every defined class relies on a cluster to store records, which in turn is defined by a set of nodes. One of the nodes will be the owner of the cluster and therefore responsible for writing records in that cluster. However, since every class has a cluster with different set of nodes it is possible for every node to write records, but belonging to different clusters. The use of a distributed architecture allows replication, guaranteeing an extra layer of security against data loss. OrientDB is ACID compliant and support transactional operation, even in a distributed environment.

---

<sup>12</sup>Like in Neo4j, there is an enterprise edition that has additional features and tools that will not be focused in this work

## SQL SUPPORT, API AND DRIVERS

The query language in OrientDB is Structured Query Language (SQL) friendly. The developers of OrientDB decided to adopt the SQL syntax to perform database queries, adding some extensions and altering some of its clauses to make work more like a graph query language. For example, the JOIN clause does not exist in OrientDB's SQL, being it possible to perform queries over more than one class at the same time while conditioning results through standard WHERE clauses. There is also support for the Gremlin query language, used in the graph computing framework Apache TinkerPop[75]. OrientDB also provides an Application Programming Interface (API) for a wide range of programming language drivers (Java, Javascript, Scala, PHP, Python, Ruby and more) that communicate either directly through a binary protocol or through HyperText Transfer Protocol (HTTP). There are also Java Wrapped drivers, used for languages that run in the JVM.

### 2.3.4 TITAN

Titan[76] is a distributed graph database written in Java, developed by a startup company called Aurelius<sup>13</sup> that had its first release in 2012 and its currently developed exclusively by the open source community. Currently ranked third in the DB-Engines rank for GDBMSs, Titan's focus was to create a graph database with a property graph data model that scaled in distributed environments and specialized in real-time graph traversals. Unlike the other two presented solutions, Titan has a modular architecture that requires the use of an external storage component for data persistence and allows the use of a search engine to enhance data indexing capabilities.

## SCHEMA-OPTIONAL PROPERTY GRAPH DATA MODEL

The data model and schema used in Titan are quite similar to the ones presented by Neo4j. It uses a logic of labels and properties that can be attributed to both vertexes and edges in the graph and properties are key-value stores. Schema can be explicitly or implicitly defined, modified and extended without sacrificing up time or performance. A difference between Titan and Neo4j's schema is that Titan grants a more strict definition of edge labels, since in Neo4j all relationships are many-to-many and in Titan it is possible to enforce different multiplicity constraints in edges, like one-to-one relationships or one-to-many[77]. Titan also supports a kind variety of data types for its defined properties.

## DISTRIBUTED MODULAR ARCHITECTURE

Titan's architecture has a different to the previous presented solutions. To be precise, Titan should be considered a graph database engine and not as much a GDBMS. Its architecture is modular and targets the interoperability between client applications and different storage and search components, establishing itself as a middleware that exposes a graph database between them[78]. Titan provides out of the box adapter for storage solutions like Apache Cassandra<sup>14</sup>, Apache HBase<sup>15</sup> and Oracle

---

<sup>13</sup>Aurelius was acquired in the beginning of 2015 by DataStax

<sup>14</sup><http://cassandra.apache.org/>

<sup>15</sup><http://hbase.apache.org/>

Berkeley DB<sup>16</sup>. For external indexing, it supports Elasticsearch<sup>17</sup>, Apache Lucene<sup>18</sup> and Apache Solr<sup>19</sup>. For database querying, Titan uses the already mentioned Gremlin query language. It supports ACID compliant transactions like Neo4j and OrientDB and, being distributed, data partitioning among different Titan servers. Unlike the other two solutions, Titan only supports a Java API.

### 2.3.5 GRAPH DATABASES REVIEW

The three presented solutions are fit to embrace a property graph data model based on social media relationships. The major difference that stands out among them, is that Neo4j's Community Edition isn't distributed and therefore must rely on the power of a single machine. However, it is the oldest, most mature and flexible among the three. Performance wise, recent literature in the scientific community tends to pick Neo4j as the best performing solution, though it can vary according to the different operations, whether it is reading, writing or performing graph traversals.

In a study conducted in 2015 by Beis et al[79], Titan, Neo4j and OrientDB were benchmarked and compared in inserting, querying and clustering operations, with the last one referring to their behavior when processing a community detection algorithm. They concluded that Neo4j was the solution that obtained best performance with larger data sets in the inserting and querying operations. However, they found OrientDB to process the successive local queries that their community detection algorithm required. Jouili et al[80] also reviewed the graph database systems mentioned before, among others, in terms of performance doing graph traversals, read operations and write operations. They found Neo4j to be the solution that better performed doing traversals, while their tests in read-only operations didn't find significant differences among the tested systems. On read and write workloads, their setup of Titan with an Apache Cassandra instance for persistence achieved the most interesting results.

The following table presents a summarizes the capabilities and characteristics of the three presented solutions.

---

<sup>16</sup><http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/overview/index.html>

<sup>17</sup><https://www.elastic.co/>

<sup>18</sup><http://lucene.apache.org/>

<sup>19</sup><https://lucene.apache.org/solr/>

	Owner	Language	Platform	Open Source	Integrated Data Storage	Data Model	Transactional	Data Storage	Native Graph Algorithms	Query Language	Communication Protocols
Neo4j	Neo Technology	Java	JVM	X**	X	Property Graph	X	Disk Based	X	Cypher	HTTP; Bolt
OrientDB	NirxolBase Ltd	Java	JVM	X**	X	Multi-model (Document-Graph)	X	Memory Based	-	Extended SQL; Gremlin	HTTP; Binary
Titan	Aurelius*	Java	JVM	X	-	Property Graph	X	Memory Based	-	Gremlin	HTTP

Table 2.1: Comparison of GDBMSs

## 2.3.6 DATA MINING AND MACHINE LEARNING TOOLS

### R

R[81] is a highly capable open-source multi-paradigm programming language, developed specifically for Statistical Computing and Data Mining. R is a highly mathematical influenced language, that shares some similarities with MATLAB, given that it is heavily based on matrix arithmetic. R accomplishes this using developer-friendly data structures such as vectors, matrices, arrays, with the most popular being the DataFrame, a high-level data structure that used to represent data tables. Despite this, R is not strictly a matrix arithmetic tool (although it has similar performance when compared to MATLAB or Octave), as it is highly expandable due to the a repository of user-created packages. Thus, it is possible to extend R's Data Mining capabilities by importing even more ML algorithms, using different plotting libraries, and even adding a Web Framework to the project, among other options. Besides, given that R makes available a huge amount of matrix based operands (which, once again, can be extended with CRAN packages), this means that R can be used as for the whole pipeline of the KDD process, from the preprocessing to the postprocessing stage[82].

### WEKA

The Waikato Environment for Knowledge Analysis (WEKA) is an open source software developed in the University of Waikato with the goal to expedite research in DM and ML, through the unification of algorithms and knowledge analysis tools in a single suite[83]. It also accounted for the need to create new algorithms for data manipulation and model evaluation without resorting to different infrastructures.

The WEKA suite is accessible through a Graphical User Interface (GUI) and a Java API. The user interface can be maneuvered in three different views, the *Explorer*, *Knowledge Flow* and *Experiment* view. The Explorer view is divided in several sections that deal with different kinds of tasks. The first is called Preprocess and, as the name implies, handle the preprocessing stage of KDD pipelines. It allows to load data from different sources and formats or generate it from manufactured sources, as well as provides the tools for data transformation and filtering. The second is the Classify section, which handles the application of supervised learning algorithms over selected data and offers the tools for model validation and visualization. The third section of this view is the Cluster section, which has the same principle but unsupervised learning algorithms the in family of cluster algorithms and the fourth, "Associate" handles association rule methods. WEKA's focus is mainly on supervised learning, namely Classification and Regression algorithms. It has less support when it comes to the unsupervised learning tasks[83]. Besides the mentioned capabilities, the Explorer view also offers dedicated methods to attribute selection and data visualization, namely through the Select Attributes and Visualize sections. The second view, Knowledge Flow, presents a more visual setup of the data processing pipeline. It borrows the concept of data flow diagrams to give the users the possibility to drag, drop and connect nodes that are representative of most the functionality presented the Explorer view. This allows to explore the advantages of algorithms that are incremental, i.e. that do not require to load complete data sets into memory to process them. The last view in the WEKA GUI is the Experimenter view, which eases experimentation and performance comparison of different models in different data sets and allows to distribute the computational effort between different machines.

The data processing capabilities described so far can be achieved through the extensive Java API, that emulates the work flows of the visual interface through its style and makes translating it into

code an accessible task.

## PYTHON DATA SCIENCE STACK

Python is a well-developed, open-source, multi-paradigm language with general purpose usage. However, over the years Python has gained some notoriety in the data science community. Given its high-level interactive nature, Python was an interesting target for the development of scientific libraries. This led to the creation of a set of libraries that are widely used for algorithmic development and data analysis.

NumPy[84] is a low-level library that adds support for n-dimensional arrays, as well as a set of functions to operate on them, thus allowing Python to be used as a matrix arithmetic tool. For more advanced operations, SciPy[85] can be used. It adds support for much more complex operations, such as image or signal manipulation and interpolation operations.

The previously enumerated tools pose the inconvenience of being rather low-level. To fill this gap there is Pandas[86]. The main feature of this library is the addition of the DataFrame object, which acts mainly as a wrapper for NumPy arrays. It is as a two-dimensional tabular data structure, adding slicing, grouping and reshaping operations.

Having addressed the main issues of data manipulation there is still missing the Data Mining component, which is where the meaningful patterns and useful knowledge will be exposed. The most relevant option in the Python ecosystem is Scikit-Learn[87]. This library provides implementations of almost every relevant Machine Learning algorithm, while keeping a consistent interface between them, allowing experimentation in a quite accessible way.

One of the main criticisms to the Python language is its low performance. However, all of the previously described libraries' implementations are extended with C bindings. These bindings are seemingly accessed from Python, therefore increasing the performance of these libraries to nearly compiled languages' level[87].

Python also has an extensive library dedicated to Natural Language Processing (NLP) called Natural Language Toolkit (NLTK). It offers a broad Python API to deal with text representative of the human language, along with a collection of lexical resources and corpus for Text Mining tasks[88]. It is worth noting that NLTK has dedicated resources to deal with the Portuguese language.





# SYSTEM DESCRIPTION AND ARCHITECTURE

---

*This chapter introduces the necessary to requirements to fulfil this dissertations purpose and how its development was conceptualized.*

## 3.1 DESCRIPTION AND REQUIREMENTS

This section provides an in depth description of the functional focus of this work as well as its functional and non-functional requirements

The main objective of this dissertation is to build a web application that incorporates the sociological knowledge present in Social Media platforms, with data mining research in Social Media environments. It aims to present underlying information valuable enough to allow the recognition of traits and characteristics present in users in a determined Social Media platform. Also, this system is directed to the Portuguese community since there isn't, to the best of my knowledge, a platform that evaluates generic user profiles that is solely focused in Portuguese consumers and because restricting the scope of the system target can enhance the depth of knowledge to be gained. Furthermore, a KDD process chance of success is directly correlated with the previous existing knowledge over the target domain. With this being a research made in a Portuguese university by Portuguese researchers, focusing on the Portuguese community brings advantages in terms of awareness of language idiosyncrasies and peculiar traits, background perception of demographic distribution and direct interaction within that same community.

In terms of practical functionality that this system must encompass, the requirements are purely related with the information extracted from the available Twitter data on which it relies. Therefore, it is necessary to summarize what kind of information we're looking to obtain. Due to the vastness possibilities and immense options available to explore the Twitter network, it was decided to divide data mining goals into two different scopes: **Network Scope** and **User Scope**.

### 3.1.1 NETWORK SCOPE

On the Network Scope, this system seeks to evaluate the Twitter network as whole, disregarding user specificity. Although this may seem as a deviation from the main goal of user profiling, examining the network without specifying each individual is also an exercise of profiling, since it will uncover how the community behaves as a unit and provide meaningful information in a more generic sense about the elements that compose it.

To this extent, the following objectives are proposed to be part of this system:

- Determining the time periods in which the Portuguese Twitter community is more active i.e. when are more tweets being posted on an hourly, daily and monthly basis;
- Determining where does the Portuguese Twitter activity concentrate. By "where", it is implicitly meant as the geographical locations where more tweets are coming from;
- Determining the gender distribution of Twitter users. Being an attribute that is not revealed by Twitter users and with great demographic value, it is proposed to find the percentage of male and female Portuguese Twitter users;
- Determining which devices or applications the Portuguese users use to maintain their Twitter activity.
- Determining which users can be considered influential on the Portuguese Twitter network;
- Determining what is talked and discussed inside Twitter.

### 3.1.2 USER SCOPE

On the User Scope, this system aims to be able to identify a Twitter profile and provide detailed information about the user that owns it while contextualizing him within the whole Twitter network. This part of the system will contain information about each target user. Consequently it can be considered the focal point of this work.

The following objectives are proposed for this end of the system:

- Determining the time periods in which the user concentrates his activity on Twitter, i.e in which time slots the user is more active;
- Determining how the user interacts with Twitter in terms of types of devices or applications chosen to maintain his profile;
- Determining which users interact with the user being profiled. This translates in determining which are the users who more frequently make use of the already discussed Twitter dynamics to connect with this user;
- Determining the users which the profiled user frequently interacts. This is the same principle as the one stated above but in a reverse direction;
- Determining the network distance between this user and any other user, i.e. being able to visualize how "close" this user is to other users;
- Determining what are the topics of interest of a user;

- Determining what users in the network have similar topics of interest with the user being profiled.

### 3.1.3 NON-FUNCTIONAL REQUIREMENTS

Given that the final product of this work is a web application, there are common non-functional requirements in which the system can be evaluated like Usability, Reliability, Performance and Supportability.

The web application should provide a functional and appealing interface. Considering that the end goal of the application is to present meaningful information to an end-user, without discriminating his background knowledge to interact with it, the method for interaction and the information presented should be as simple as possible, without sacrificing functionality.

Given that the system has some degree of interaction and that the end-user is working with data he may not be familiarized with, it is harder to achieve complete stability. In case some error occurs when maneuvering the system, a useful message should appear that will allow the rectification of the flaw. Also, none of these errors can cause critical system failure.

Performance is somewhat hard to manage when handling considerable amounts of data. This is not a typical information system, in which most operations take little time to execute. In a system with lots of data being fetched and potentially processed from different sources, some of the operations executed will take time. To minimize these issues, some background data engineering must be done, to allow that information is loaded in acceptable time periods while the application runs smoothly. The client application must also perform asynchronous requests to the server, so it will never block while waiting for some process to be complete.

## 3.2 DATA MODEL

This system will be using data originated from Twitter to meet the set of requirements described in the previous section. As mentioned, Twitter data meets the description of traditional Social Media data since and can be classified as **big**, **noisy** and **linked**. Therefore, handling Twitter data and storing it efficiently without making it lose its beneficial properties is a challenge with significant relevance to fulfill the system's purpose. Above all concerns, it is important to maintain the inherent links between Twitter users and their respective generated content as well as their interactions through Twitter dynamics, as some functional requirements are dependent on this information. This means that the proposed data model for this system must embrace linked data and allow the efficient exploration of those links.

### 3.2.1 AVAILABLE DATA

Twitter data is available through the Twitter API[89]. There are also online data sets used in previous research that were made available by their authors but those are mostly in the English language, which does not fit in this work's purpose.

The Twitter API is a set of APIs supplied by Twitter for application developers. For the purpose of this work it is important to understand two components of the Twitter API ecosystem, the Representational State Transfer (REST) API and the Streaming API<sup>1</sup>.

The REST API offers a group of RESTful Web Services that enable to programatically read or write Twitter data[90]. Through the Twitter REST API, it is possible to perform the functional operations of a normal Twitter account, like posting a tweet or following a user, and to perform searches on the Twitter network. This includes getting information about a specific user, if his profile is public, or his posted status updates. The REST API must be linked to a user account to be operational. However, there are restraints when using Twitter’s REST API, namely it’s rate limiting. The REST API calls have limited number of requests per time slot, depending on the operation, which makes it unsuitable for a network crawling exercise, as it would take an impractical amount of time.

Complementary to the REST API, it is also offered the possibility of streaming Twitter data through the Streaming API[91]. Through this, we are able to access tweets being posted in real-time in the network, by a user or by a group of users. It is also possible to define filters for data streams such as a geographical perimeter, the language of the tweet or the presence of specific words in the content.

All responses from the Twitter API come in the JSON format, with the different available "objects" being well documented in its website. In Appendix-A, section 6.2 there are two examples of Twitter API’s response objects, the first for a User object and the second for a Status object.

The data set used for project TVPulse (see section 2.2.7) was made available for this dissertation. TVPulse incorporated a data collector that made use of the Twitter Streaming API configured to only allow tweets where the language was Portuguese and with a geographic perimeter correspondent to the continental Portuguese territory. The Streaming API responses are objects described in Listing 46.

The project started a partially uninterrupted capture in September 2015. Their captures were subsequently stored in an Apache Cassandra[92] instance with the following data model:

Table 3.1: TVPulse Data Model

Column Name	Data Type	Description
Year	Integer	Year in which the tweet was posted
Month	Integer	Month in which the tweet was posted
Day	Integer	Day in which the tweet was posted
Hour	Integer	Hour in which the tweet was posted
Created_at	Timestamp	Tweet creation time as a 64 bit Timestamp
ID	Varint	Tweet unique identifier
Raw	Text	Original response object in a string representation
Text	Text	The tweet’s text content
User_id	Varint	Unique identifier of the user that posted the tweet

Although there are some flaws (which will be discussed in the next chapter) in the data collected by project TVPulse, in practice it is a fair representation of the Portuguese Twitter activity. It is able to efficiently store tweets, while containing an identifier of the user that posted them, that can be easily retrieved with the time of its posting as a condition. However it has limitations, since that is the only way to get it. Due to Apache Cassandra’s data model it is not possible to make direct associations between the users e.g. who they mention or what hashtags they use, without having a

<sup>1</sup>There is also available an Advertising API which is not relevant for this work

processing layer over tweets raw data. Nevertheless, it is very useful when it comes to fetch data in bulk, like the tweets text in a determined time interval.

Due to the rate limiting restrictions of the REST API, crawling the Twitter network is nearly impossible in an adequate time period, meaning that collecting tweets from the Streaming API is the only possible way of effectively gathering raw, unbiased data from Twitter in considerable amounts.

### 3.2.2 PROPOSED DATA MODEL

Having described how the data is available for this system's purpose, it is necessary to introduce how it will make use of it. The accessible data set does not fulfill the system requirement of linked data. Being stored in a Cassandra instance, the different rows of the Cassandra table represent a single tweet where the only relationship directly available being an identifier of the user that posted it. Also, the data model present in Cassandra databases does not allow to perform queries that join information between the existing rows, making it even more difficult to explore data relationships. However, the original response object is available and through a direct analysis of the properties present in a Status object from the Twitter API it is possible to observe a lot of valuable information. The following table describes with more detail the available information in Status objects that is considered relevant for this system's goals:

Table 3.2: Relevant properties in Status Objects

Property	Data Type	Description
is_quote_status	Boolean	Indicates if this tweet is a Quote
quoted_status	Status Object	The tweet originally quoted
quoted_status_id	Integer	The unique identifier of the quoted tweet
in_reply_to_status_id	Integer	The unique identifier of the replied tweet
in_reply_to_screen_name	String	The screen name of the user which was replied by this tweet
in_reply_to_user_id	Integer	The unique identifier of the user which was replied
retweeted	Boolean	Indicates if this tweet is a Retweet
retweet_count	Integer	The number of times this tweet has been retweeted
entities.user_mentions	Array of Objects	An array containing identifiers of the User's being mentioned in the tweet, among other details
entities.hashtags	Array of Objects	An array containing the hashtags being used in the tweet, among other details
entities.urls	Array of Objects	An array containing the urls being used in the tweet, among other details
place	Places Object	Twitter API object with information regarding the location where the tweet was posted
source	String	The device or application used to post the tweet
user	User Object	The user that posted the tweet

Looking at Table 3.2, it is possible to find meaningful relationships inside a Twitter API Status object. From this objects it is possible to find:

- The user who posted the tweet;
- If this tweet is a Retweet (or Quote) and the original tweet;
- If this tweet is a reply, an identifier of tweet replied and an identifier of the user being replied;
- Identifiers of the user's mentioned in the tweet.

Moreover, other relevant properties are available for the requirements stated in the previous section:

- Hashtags present in the tweet;
- The location at the time of posting;
- The device or application in which the tweet had its source.

The available information in Status objects allows the modeling of a graph, in which nodes represent entities and links the relationships between them. The following graph model is a direct representation of the links available in Status objects:

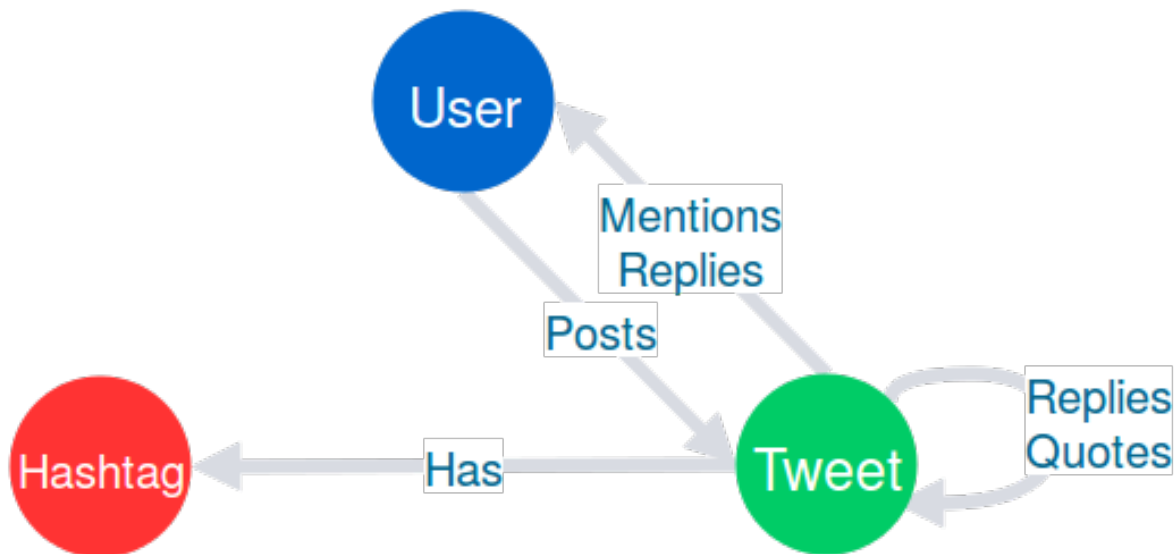


Figure 3.1: Graph representation of Status objects

Although links are valuable it is also important to maintain the individual properties of each entity in the graph and to be able differentiate between them. This graph model can therefore be considered a property graph. The following tables (3.3, 3.4, 3.5) present the attributes proposed to be a part of to each respective entity of the graph, according to the data available in both User and Status API objects.

## USER

Table 3.3: Proposed User model

Field	Data Type
User_id	Integer
Screen_name	String
Name	String
Description	String
Friends_count	Integer
Followers_count	Integer
Favourites_count	Integer
Created_at	Timestamp
Profile_picture	String
Language	String
Url	String
Location	String
Statuses_count	Integer

## TWEET

Table 3.4: Proposed Tweet model

Field	Data Type
Id	Integer
Hour	Integer
Day	Integer
Month	Integer
Created_at	Timestamp
Text	String
Source	String
Language	String
Location	String
Retweet	Boolean

## HASHTAG

Table 3.5: Proposed Hashtag model

Field	Data Type
Text	String

### 3.2.3 STORAGE REQUIREMENTS

This system's storage has as its top priority the ability to mimic the existing relationships in the defined data model without having unnecessary complexity to develop it or to explore it. It must also allow modifications in the model, since there are properties that need to be calculated *a posteriori* and dependable on the collective network data. Performance is also an important issue, since the volume of data is significant and this should have a minimum impact on our ability to retrieve existent information.

The type of database that more assertively fits the described storage requirements is a Graph Database. As described in the previous chapter, Graph Databases expose a graph data model which perfectly adapts to the models presented so far. They embrace data relationships, which is also an intended storage requirement, and usually have a flexible schema definition, which allows to iteratively modify the defined model without sacrificing development time.

## 3.3 CLIENT-SERVER MODEL

The desired architecture for this system complies with a traditional Client-Server model. The Server is responsible for implementing the system's logic and features, while also establishing the

connections with the system's database and handling the Client's requests. The Client is responsible for the visualization capabilities and handling user interaction, resorting to the Server when its necessary.

To further understand the system's underlying behavior, it was decided to segment it into three distinct components: Backend, API and Frontend. The following figure illustrates a standard Client-Server model and places the defined system components in each part of the model.

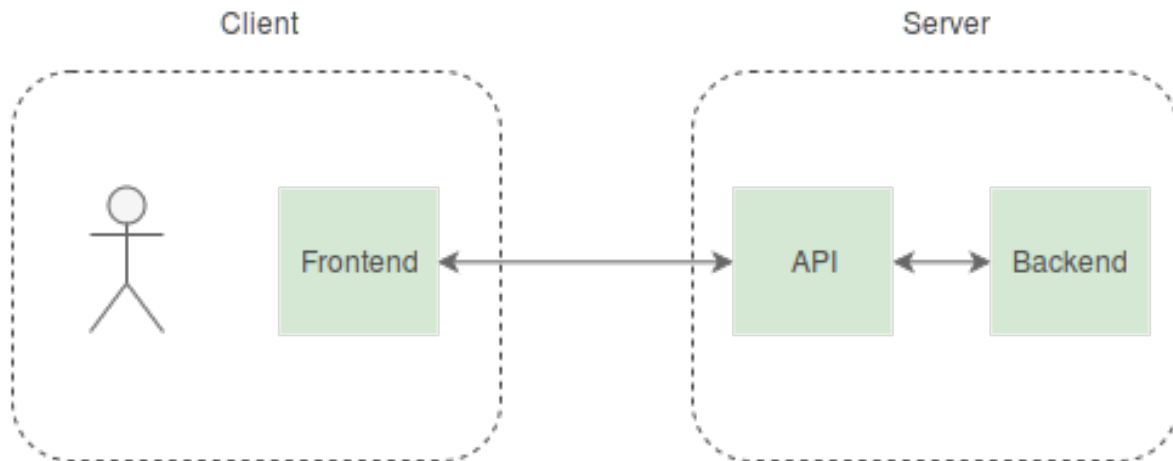


Figure 3.2: Client-Server Model

### 3.3.1 BACKEND

The Backend component is responsible for implementing the system's functional logic. It interacts directly with the storage endpoint and must be able efficiently make use of the developed data model to satisfy the proposed requirements and serve them to the API component. It must be able to retrieve the necessary data, meaning it must understand the underlying data model, process it and deliver it in an adequate format.

The Backend component is where most of the expensive computational effort lies and therefore it is anticipated that it will heavily influence the overall system performance. Besides having important computational tasks, this is also where operations are requested from the DBMS. Deficient handling of the DBMS capabilities can also result in performance penalties.

### 3.3.2 FRONTEND

The system's Frontend is the visible part of the application. It is, as said, responsible for data visualization and user interaction. It must present a clean and visually appealing interface, while taking into consideration the variables on the end-user side like screen size or browser. It must also display data representations, that resemble the designed data model and the proposed goals of the system.



### 3.3.3 API

In order to have some separation of concerns and modularity, the communication between the Frontend and the Backend is made through an independent API. This way, there is a division between the systems logic and the communication between components, which eases the development process and results in a cleaner application. The pattern chosen for the system was a RESTful API. RESTful services are the standard for communication nowadays and they are a reasonable choice for the public exposure of web applications logic.

#### API SPECIFICATION

The communication interface between our Frontend and Backend components can be established based on the goals of this work. Since those goals are information related, it is possible to forecast which methods will be necessary for the Frontend and what kind of information they will need to respond. Hereby is presented the specification of this system's API.

Table 3.6: API Specification

	API Method	Parameters	Purpose
Network Scope	network_day_activity	Date	Getting the hourly counts of tweets in a day
	network_month_activity	Month	Getting the daily counts of tweets in a month
	network_activity	None	Getting the global tweet counts per month
	network_day_locations	Date	Getting the tweet counts per district in a day
	network_month_locations	Month	Getting the tweet counts per district in a month
	network_locations	None	Getting the global tweet counts per district
	network_genderdist	None	Getting the gender distribution of the whole network
	network_sources	None	Getting the distribution of sources used to post tweets
	network_influence_ank	Range	Getting the <range>most influential users in the network
	network_topics	None	Getting a representation of the modeled topics present in Twitter
User Scope	user_day_activity	User_id	Getting a user's hourly counts of tweets in day
	user_month_activity	User_id	Getting a user's daily counts of tweets in a month
	user_activity	User_id	Getting a user's global tweet counts per month
	user_sources	User_id	Getting a user's distribution of sources used to post tweets
	user	User_id	Getting a user model
	user_interactions	User_id, range	Getting the list of the <range>users that more frequently interact with user <User_id>
	user_topics	User_id	Getting a representation of the user's topics of interest
	user_similar	User_id, range	Getting the list of the <range>users that are more similar by topics of interest to user <User_id>

## 3.4 DATA MINING MODULES

This system will rely on information extracted from the available set of Twitter data. Although most of the system's goal rely on information that is directly available through the designed property graph data model, some of it will require analysis made externally to the web application. This is particularly predictable when it is necessary to analyze text content, since it is a kind of process that relies minimally on data relationships and that a graph data model does not have any particular advantage other than to retrieve subsets of data that fit a certain relationship pattern, like for example retrieving all the tweets that mention a determined user.

The Data Mining Modules must handle all the necessary operations of the KDD pipeline, from preprocessing to postprocessing, having as source the data in the storage component of this system and as an end it can either update the data model stored or produce relevant pieces of information to be used by the system on their own.

### 3.5 ARCHITECTURE OVERVIEW

Hereby is presented the high-level projected architecture of the system, with all the components referred throughout his section.

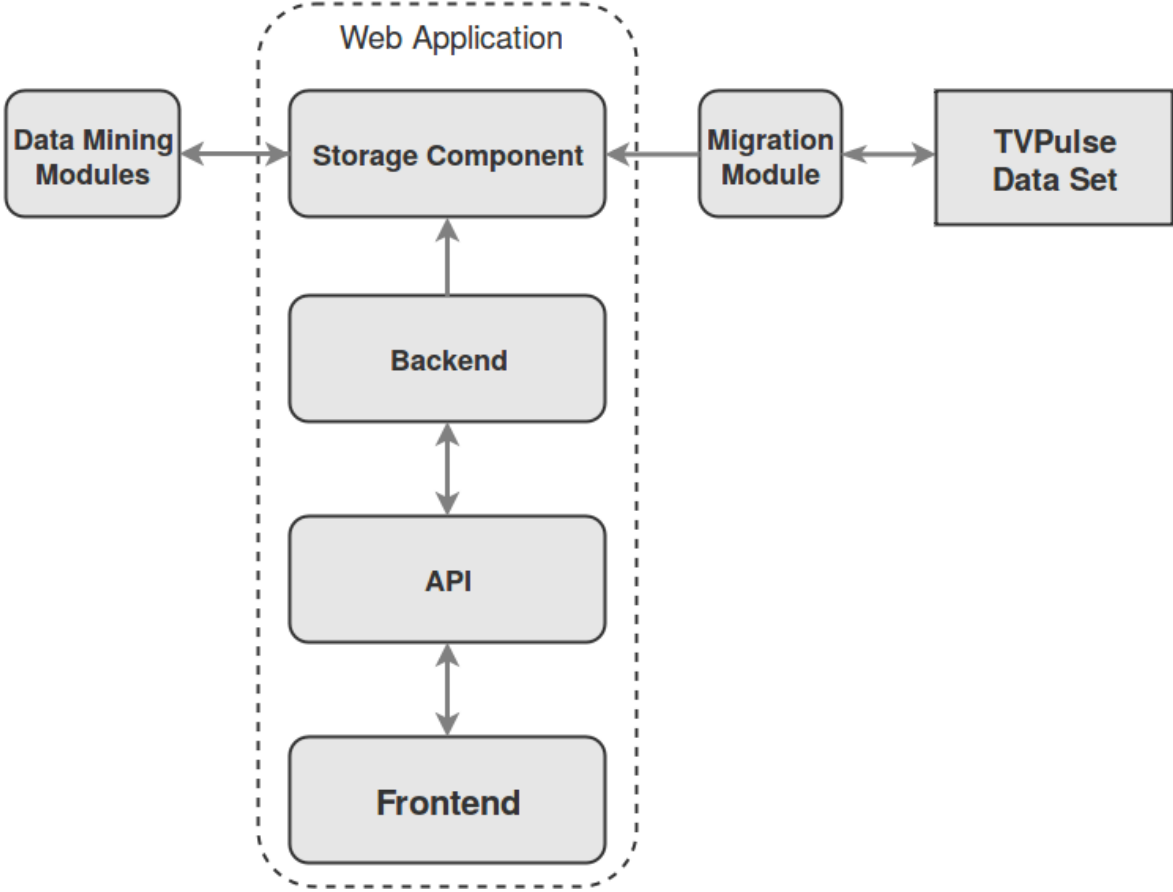


Figure 3.3: High-level System Architecture

# IMPLEMENTATION

---

*In this chapter it will be detailed how the proposed system was implemented, how each of the system's requirements were met and how each component was orchestrated to communicate with each other.*

## 4.1 STORAGE

Having decided that a property graph would better suit the needs of the defined data model, it was a matter of deciding which of the evaluated graph database solutions was the most reliable and offered better performance to the system. In section 2.3, three GDBMSs were presented and reviewed: Neo4j, OrientDB and Titan. The results found in [79] and [80] along with its maturity led to the decision to use Neo4j as this system's core storage component. Requirement wise, Neo4j not only allows to design a data model that exactly matches the proposed one as well as allows for incremental changes, if such is necessary, without sacrificing any up or development time. In terms of integrity and performance, Neo4j tries to join the best of both worlds as it is ACID compliant and uses native graph storage. Neo4j also possesses its own query-language, *Cypher*, which is rather simple and quite developer-friendly. This grants the possibility to explore the defined data model programatically without much effort.

### 4.1.1 NEO4J CONFIGURATION

To make the best use out of Neo4j storage capabilities it was necessary to study its architecture, system requirements and work the best possible configuration to fit the systems needs.

The edition of Neo4j used for this project was the Community Edition 3.0.0, released in March 2016. This version possesses all the features of the Neo4j property graph data model and DBMS, although it misses some redundancy, scalability and monitoring features when compared to the Enterprise Edition.

The minimum and recommended system requirements to work with Neo4j are presented in the next table:

Table 4.1: Neo4j System Requirements

Component	Minimum Requirements	Recommended Requirements
CPU	Intel Core i3	Intel Core i7
Memory	2GB	16-32 GB
Disk	10GB HDD SATA	SSD w/ SATA
Software	Java 8 (Oracle or OpenJDK)	-
Operating System	Linux (Debian, Ubuntu) or Windows Server 2012	-

In terms of configuration parameters, Neo4j uses two configuration files *neo4j.conf* and *neo4j-wrapper.conf*. The first deals with all the configurations of the database system related to its internal functioning, network connectors and security, performance and other miscellaneous parameters. The second is where parameters related to the JVM can be set.

This configuration of Neo4j was mostly network-based, since it was decided to have a dedicated machine to run it and therefore it was necessary to make it accessible by the other components of the system. It was also made some performance tuning, due to the massive amount of data that needed to be stored.

---

```
# Bolt connector
dbms.connector.bolt.type=BOLT
dbms.connector.bolt.enabled=true
dbms.connector.bolt.tls_level=OPTIONAL
# To have Bolt accept non-local connections, uncomment this line
dbms.connector.bolt.address=10.5.40.29:7687

# HTTP Connector
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
#dbms.connector.http.encryption=NONE
# To have HTTP accept non-local connections, uncomment this line
dbms.connector.http.address=10.5.40.29:7474

# HTTPS Connector
dbms.connector.https.type=HTTP
dbms.connector.https.enabled=true
dbms.connector.https.encryption=TLS
dbms.connector.https.address=localhost:7473
```

---

Listing 2: Neo4j Network Configurations

This listing contains the configured properties in Neo4j related to making it remotely accessible as a component in the system's architecture. The database instance was deployed in a virtual machine existing in an OpenStack<sup>1</sup> instance accessible at Instituto de Telecomunicações. These configurations are mostly related with defining the public addresses of the connectors for HTTP, HyperText Transfer Protocol Secure (HTTPS) and Bolt, Neo4j's proprietary communication protocol.

<sup>1</sup><https://www.openstack.org/>

In terms of performance the only measure taken in terms of system configuration was to change the maximum number of open files allowed in the operation system where the Neo4j instance is deployed. This is a practice encouraged by the developers of Neo4j when the database used possesses a lot of indexes and is expected to receive a lot of connections. Therefore the file system limit of open files was set to 40 000, which is the recommended amount by the Neo4j Operations Manual<sup>2</sup>.

## 4.1.2 DATA MODEL

The proposed data model for this system was explained on the previous chapter, in section 3.2.2. It was based on the available information in Twitter API User and Status objects, as well as in the data set existent in project TVPulse. Here it will be explained how the data model was fully achieved using Neo4j's schema-optional property graph data model. The Cypher query language allows the creation of entities and relationships in the graph model through its keyword **CREATE**, analogously to SQL based relational databases.

The following listings show the Cypher statements used to create the individual entities and relationships in the developed property graph. Since Neo4j is schema-optional, each entity is created independently and it is the use of labels and property constraints what makes them both belonging to a set and unique at the same time.

### ENTITIES

The presented Entities assertively match the ones defined in section 3.2.2. Neo4j's schema allowed to create an exact representation of the proposed graph entities and its available property types also have a direct to correspondence to the modeled ones. Indexes and unique constraints are applied to all the graph Entities, to enforce a schema and enhance data consistency and integrity. These topics will be further addressed in the next subsection.

#### User

---

```
CREATE (n:User {user_id: 12345678,
              name:"user name",
              screen_name:"@example",
              description:"I'm an example of how to create a user
                           entity",
              friends_count: 0,
              followers_count: 0,
              favourites_count: 0,
              statuses_count: 1,
              created_at: "2016-08-20T18:43:00",
              profile_pic: "http://twitter/pic/example.png",
              language: "pt",
              url: "http://example.com",
```

---

<sup>2</sup><https://neo4j.com/docs/operations-manual/current/deployment/#post-installation-tasks>

```
})
```

---

Listing 3: Example of a create User query

**Tweet**

---

```
CREATE (n:Tweet {tweet_id: 87654321,
                hour: 18,
                day: 20,
                month: 8,
                year: 2016,
                created_at: "2016-08-20T18:43:00",
                source: "",
                language: "pt",
                location: "Aveiro, Portugal",
                retweet: false,
                text: "Yeah I'm a tweet! @example"
                })
```

---

Listing 4: Example of a create Tweet query

**Hashtag**

---

```
CREATE (n:Hashtag {text: "summer"})
```

---

Listing 5: Example of a create Hashtag entity query

## RELATIONSHIPS

The create Relationship query has the same syntax for all the relationships, since relationships do not possess properties. The following listing presents a generic query used for the creation of relationships.

---

```
CREATE (e1:<Label1>)-[r:<RelationshipLabel>]->(e2:<Label2>)
```

---

Listing 6: Example of a create relationship query

So if the goal is to create the relationship "User with user\_id 12345678 posted Tweet with tweet\_id 8654321", the used query would be:

---

```
CREATE (e1:User {user_id: 12345678})-[r:POSTS]->(e2:Tweet {tweet_id:
87654321})
```

---

Listing 7: Example of a create user-posts-tweet relationship

The same principle applies to all entities and relationships proposed in section 3.2.2.

## INDEXES AND CONSTRAINTS

The use of a defined database schema has benefits in terms of performance and data organization. The use of indexes greatly improve the performance of search queries and data retrieval. For these reasons, property constraints were implemented in the proposed data model as well as property indexes.

The necessity of property constraints in data is fairly obvious. If a user posts  $n$  tweets, it is not required a repeated user entity per POST relationship but one user entity with  $n$  post relationships. This principle can be applied to all the entities in data. Therefore, there must be a uniqueness constraint on each existing entity of our dataset.

The candidate properties for a uniqueness constraint are the ones that uniquely identify an entity among the others. Both User and Tweet entities have a unique id, provided by Twitter, on which a uniqueness constraint was applied. Twitter users also have a screen name, which is also unique among them. The hashtag entities are unique according to their text, meaning that every hashtag with the same text refers to the same entity.

Having explained the properties of chosen to use as unique identifiers, it is now presented the Cypher queries that apply this schema definition.

---

```
CREATE CONSTRAINT ON (n:User) ASSERT n.user_id IS UNIQUE
```

---

Listing 8: User id uniqueness constraint

---

```
CREATE CONSTRAINT ON (n:User) ASSERT n.screen_name IS UNIQUE
```

---

Listing 9: Screen name uniqueness constraint

---

```
CREATE CONSTRAINT ON (n:Tweet) ASSERT n.tweet_id IS UNIQUE
```

---

---

Listing 10: Tweet id uniqueness constraint

---

```
CREATE CONSTRAINT ON (n:Hashtag) ASSERT n.text IS UNIQUE
```

---

Listing 11: Hashtag text uniqueness constraint

Indexes have a huge impact on database performance. Indexed entities will be much faster to search, retrieve and operate upon. Ideal properties to be indexed in an entity are those that are more frequently used in query conditions. To put it in other words, the best properties to index in database are those that are more frequently searched. Coincidentally, these properties match the ones used as unique constraints. Below are listed the Cypher queries used to apply indexes to database entities.

---

```
CREATE INDEX ON :User(user_id)
```

---

Listing 12: User id index

---

```
CREATE INDEX ON :User(screen_name)
```

---

Listing 13: User screen name index

---

```
CREATE INDEX ON :Tweet(tweet_id)
```

---

Listing 14: Tweet id index

---

```
CREATE INDEX ON :Hashtag(text)
```

---

Listing 15: Hashtag text index



### 4.1.3 DATA MIGRATION

The first task to pursue after defining the data model was to populate the database with the existing data from project TVPulse. As said before, project TVPulse had a continuous listener that made use of the Twitter Streaming API to capture tweets posted in the Portuguese territory and in the Portuguese language. The captured tweets were stored in a Cassandra database with the original Status object and some related meta-data and the captured started in September 2015 and progressed until mid 2016. It was decided to use a seven month period (from September 2015 to March 2016) as this system's available data. Since real-time data analysis was out of the current scope and in this period of time there were some events worldwide and in Portugal (two political elections, terrorist attacks and a continuous "war-like" climate between two major Portuguese football clubs) that could have some repercussion in the Twitter community, it seemed an interesting time period to analyze in depth.

To make this data migration from a column-store based data model (present in Cassandra) to the defined property graph, Neo4j offers an import tool made exactly with the purpose of importing data from Cassandra databases to Neo4j. However, this tool is still in development, it is very rudimentary and supports a limited number of use cases, therefore it was implemented a custom migration scheme to have complete control over the entities and relationships created.

The migration scheme consisted on the development of a Python script that used the designated Python Driver for Apache Cassandra<sup>3</sup> and Py2neo<sup>4</sup>, which is a Python library and toolkit for working with Neo4j databases. The work flow of the script is explained in the following listing:

---

**Algorithm 2** Database migration workflow

---

```
1: for all days in time period do
2:   Get tweets for current day
3:   for all tweets do
4:     Extract User and Tweet
5:     Extract Mentions and Replies
6:     Extract Quotes
7:     Extract Hashtags
8:     Commit new entities
```

---

Since a schema for Neo4j was specified, CREATE operations must be schema-compliant. This means that if it is necessary to create an entity that already exists in the database, the operation will fail. To handle this problem, Cypher provides the keyword MERGE which can be translated as a "match or create" operation. The MERGE operation works exactly the same way as CREATE, except that if it matches a node that already exists, it will update the properties defined in the MERGE query. In the situation where properties match each other, it will not alter the matched node. This type of operation also has the advantage of simplifying development, by merging two operations (match and create) in one, without losing functional logic. This is particularly useful for node entities, since each row in the Cassandra represents new relationships in the dataset, although it may not represent new entities (e.g. a tweet that mentions a user that is already stored represents a new MENTION relationship but not a new User entity).

By the end of the migration the Neo4j database had the following numbers:

---

<sup>3</sup><https://github.com/datastax/python-driver>

<sup>4</sup><http://py2neo.org/v3/>

Table 4.2: Number of nodes in the database

Node	Number
User	284701
Tweet	11445772
Hashtags	150942

Table 4.3: Number of relationships in the database

Relationship	Number
User-POSTS->Tweet	11445772
Tweet-MENTIONS->User	2514705
Tweet-REPLIES->Tweet	1688219
Tweet-QUOTES->Tweet	504747
Tweet-HAS->Hashtag	892820

Table 4.4: Database size

Size in GBs	5,76
-------------	------

## 4.2 NETWORK SCOPE GOALS

In this section, it will be introduced how the functional requirements for this system were met. Previously it was defined the two domains in which the information goals were segmented, Network and User Scope, and what goals each domain comprised in the final prototype. This section will be focused the requirements of the Network Scope. The properties of the defined data model facilitated some the proposed objectives, while others required some workarounds and a more complex approach.

The objectives for the network explorer can be divided into four categories **Activity**, **Gender**, **Influence** and **Topics**. Each category aims to answer one or more of the requirements expressed in section 3.1.1.

### 4.2.1 NETWORK ACTIVITY

The Activity category is explored under three parameters: time, location and source.

#### NETWORK ACTIVITY OVER TIME

*Determining the time periods in which the Portuguese Twitter community is more active i.e. when more tweets are being posted on an hourly, daily and monthly basis*

Since all tweets that were posted have a time stamp of the time of their creation and their different time attributes stored as properties (hour, day, month and year) monitoring network activity over time was not expected to be an astoundingly difficult task, since the time properties of tweets can be used as parameters for querying the database. However, after examining the stored data it were found Tweet objects that contained a time stamp but did not discriminate its time attributes. This

was because the time attributes were collected directly from the TVPulse meta-data and tweets that were derived from Status objects, namely quote tweets, did not have these attributes explicitly defined. This led to the need of iterating over all the stored tweets that had a time stamp but did not have the required time properties and unfold their time stamp into year, month, day and hour attributes and update the Tweet object properties. To accomplish this, a Python script with Py2neo and the date-util library<sup>5</sup> was developed and it is explained in the following listing.

---

**Algorithm 3** Unfolding of time stamps

---

- 1: **for all** tweets with a time stamp **do**
  - 2:     *Parse the time stamp with dateutil parser*
  - 3:     *Set tweet properties*
  - 4:     *Update tweet entity*
- 

After overcoming this issue, it was necessary to develop a set of Cypher queries that were able to gather all tweets under different time periods.

To gather all the tweets posted on a per month interval the following Cypher query was developed:

---

```
MATCH (t:Tweet {month: {<month>}})
RETURN count(t) AS count
```

---

Listing 16: Query for the number of tweets in a month

This query takes as parameters the desired month to obtain the number of tweets posted and returns the number of tweets posted in that month.

To obtain all the tweets posted on a daily interval the following Cypher queries were developed:

---

```
MATCH (t:Tweet {month: {<month>}})
RETURN t.day AS day, count(t) AS count ORDER BY day
```

---

Listing 17: Query for the number of tweets from each day in a month

This query takes as argument a month and returns all tweets posted in that month but grouped on a daily granularity. It will retrieve the same number of tweets as the query above but they will be grouped by each day, sparing the necessity to further process the returned data subset.

Since there was a chance of also needing the tweets posted on one day of a specific month the following query was also developed:

---

```
MATCH (t:Tweet {month: {<month>}}) WHERE t.day=<day>
RETURN count(t) AS count
```

---

<sup>5</sup><https://pypi.python.org/pypi/python-dateutil>

---

Listing 18: Query for the number of tweets in a specific day

This query takes two parameters, a month and a day, and returns the number of tweets posted in the specified date. It allows for a more flexible development of visualization schemes and it is less computationally expensive than the query above.

To finish the goals for network activity measures, it was necessary to develop a query that returned all the tweets posted in day but with the counts grouped by hours:

---

```
MATCH (t:Tweet {month: {<month>}})
WHERE t.day=<day>
RETURN t.hour AS hour, count(t) AS count ORDER BY hour
```

---

Listing 19: Query for the number of tweets by hour in a day

The use of the keywords AS and ORDER BY is purely organizational and with the future use of the returned query set in mind. The keyword AS defines an alias for a specific attribute making the query set easier to understand and ORDER BY, as the name implies, orders the set over the values of a chosen property.

## NETWORK ACTIVITY OVER LOCATIONS

*Determining where does the Portuguese Twitter activity concentrate. By where it is implicitly meant the geographical locations where more tweets are coming from*

Similar to analyzing activity over time, activity over locations is enhanced by the existing properties in Tweet entities. The location property is provided by the Twitter API under the Places object. This JSON object contains several attributes regarding the user location at the time of posting, attributes that may be available or not according to his account privacy settings. In the migration scheme, the attribute chosen to represent the tweet location was its name, which accounts for a humanly readable representation of the place's name at the most specific detail available. For Portuguese users this can go from simply "Portugal" as far as the town where the tweet was posted. However, for this representation of activity it was considered that the minimum administrative division to focus was at a district level. This required some auxiliary processing, since the existing location in Tweet objects could be at a layer below.

The Cypher query used for this task is quite similar to the ones presented previously:

---

```
MATCH (t:Tweet)
RETURN t.location AS location, count(t) AS count
```

---

Listing 20: Query for obtaining tweet locations

Since this query is not very flexible and can have a slow execution time due to the amount of existing tweets, it was also developed the same query but with month as parameter for filtering only tweets in a determined month.

---

```
MATCH (t:Tweet {month:<month>})
RETURN t.location AS location, count(t) AS count
```

---

Listing 21: Query for obtaining tweet locations in a single month

To support the counting of tweets posted per district it was resorted to a data package provided by Central de Dados<sup>6</sup> that contained two structured JSON documents, one with district information (name and code) and the other with town information (name, code and district code). This allowed the direct mapping of the location present in a tweet to the corresponding district.

---

**Algorithm 4** Counting tweets by district

---

- 1: *Load district and town data*
  - 2: **for all** Locations and their respective counts **do**
  - 3:     *Find the corresponding district code to the location*
  - 4:     *Find the corresponding district name to the matched code*
  - 5:     *Update total counts for that district*
- 

## NETWORK ACTIVITY SOURCE

The property *source* of Twitter API objects contains information about the origin of the Tweet. The interest here is to see the devices used to post a tweet, although this field also accounts for registered online applications that use the Twitter API. To get only a reference for device usage, the following sources were chosen to be discriminated on the counts:

- Web Client
- Mobile Web Client
- Android App
- iPhone App
- iPad App
- Blackberry App

The rest of the sources were considered as Others.

To get global network usage it was needed to iterate over all the tweets that contained the property *source*, extract the corresponding device and update its counts.

---

<sup>6</sup>[http://centraldedados.pt/codigos\\_postais/](http://centraldedados.pt/codigos_postais/)

---

```
MATCH (t:Tweet) RETURN t.source AS source
```

---

Listing 22: Query for obtaining tweet sources

## 4.2.2 NETWORK GENDER DISTRIBUTION

*Determining the gender distribution of Twitter users. Being an attribute that is undisclosed by Twitter users and with great demographic value, it is intended to estimate the percentage of male and female Portuguese Twitter users.*

Determining the gender distribution of the Portuguese community is a complex task. Since gender is an undisclosed attribute in Twitter profiles the only possibility is to predict, with some degree of certainty, if a user is female or male based on the data associated to them as a Twitter user. To overcome this challenge, a Machine Learning approach was made as a Supervised Classification problem. It was necessary to model a set of features that characterized Twitter profiles and use them to build a classifier. Generic Supervised Learning algorithms and work flows were discussed in section 2.2.3. Here it will be presented the methodology used in this dissertation to build a gender classifier for Portuguese Twitter profiles.

## LEARNING DATA

Approaching this issue as a Supervised Classification problem required the manual annotation of a set of Twitter profiles. For this matter, 1400 distinct Twitter profiles were randomly picked from the whole set of users and manually labeled as Male or Female, totalling the training set for the classifier. The label was attributed through direct observation of the Twitter profile. If the profile was inconclusive or belonged to an entity like a sports club or a news media, it was discarded and replaced by another. The two classification labels are equally distributed, meaning the training set has 50% males and 50% females and for further investigation it was made available online (see section 5.1.6).

## FEATURE SELECTION

Feature selection plays a huge role in the success of classification problems. Deciding what information in a Twitter profile has a decisive impact in determining the gender of its user is a rather subjective matter.

The features extracted from each user can be divided in two distinct models: the first is a sociological feature model that relies on recognition of gender defining terms and identified behaviors on the user's writing and the second a n-gram feature model based on character and word occurrence on available profile information and tweets. The information used as source for feature extraction was the user's name, screen name, description and the last 10 tweets he posted.

## SOCIOLOGICAL FEATURE MODEL

The first feature model is based on recognising gender-meaningful terms and on the hypothesis that there are certain writing behaviors, as well as the way a user completes his profile fields that can differentiate gender. For this matter it was built a dictionary of Portuguese gender defining first names, based on the list of admitted and prohibited vocables as first names by the Portuguese Institute of Notaries and Names<sup>7</sup>. It was also built a list of common Portuguese nicknames and abbreviations for a wide range of first names and a list of gender-meaningful words, mostly composed of honorifics. The second two built dictionaries are available in Appendix-B listings 48 and 49, while the first is one is too big to be presented in this document, but is possible to find it in the provided URL. These three dictionaries cover the term recognition part of this model.

The second part of this model is based on the assumption that the analysis of a user’s lexical and linguistic content can help determining its gender. This hypothesis has been addressed for the English language in the previous work of Rao et Al with considerable success[58]. Inspired by this work, the same is attempted for Portuguese users. Since several profile fields and tweets are being used as sources for feature extraction, it makes sense to search for different socio-linguistic expressions according to the field being explored. The following table shows the features accounted for this model along with the profile information that they are extracted from. The absence of features for the user’s screen name is explained by the lack of linguistic expression present in this field.

Table 4.5: Sociological feature model

Feature	Description	Used on
Presence of male first name	A male name is present in text	Username, Description
Presence of female first name	A female name is present in text	Username, Description
Presence of male nickname	A male nickname is present in text	Username, Description
Presence of female nickame	A female nickname is present in text	Username, Description
Presence of male key word	A male key word is present in text	Username, Description
Presence of female key word	A female key name is present in text	Username, Description
Starts with “o”	The text starts with the “o” character as a word	Username
Starts with “a”	The text starts with the “a” character as a word	Username
Repeated alphabet	The text contains words with repeated characters (E.g. “Joanaaaa”)	Username, Description, Tweets
Capitalized	The text is capitalized	Username, Description, Tweets
Presence of possessive bigrams	The text expresses some form of possession (E.g. “O meu dia foi...”)	Description, Tweets
Link to Snapchat account	The text contains an url to a Snapchat account	Description
Link to Instagram account	The text contains an url to a Instagram account	Description
Link to Tumblr account	The text contains an url to a Tumblr account	Description
Ellipses	The text contains ellipses (E.g. “Enfim...”)	Tweets
Self mentioning	The text contains the “Eu” word	Tweets
Affirmation	The text contains a word that indicates a positive statement (E.g. “Yeaahhh!”)	Tweets
Laughter	The text has a word that expresses laughter (E.g. “ahahahh”)	Tweets
Exclaim	The text contains a sequence of exclamation marks	Tweets
Question	The text contains a sequence of question marks	Tweets

## N-GRAM FEATURE MODEL

N-gram representation of a word or sentence consists in decomposing it into a sequence of tokens with n items. In linguistics, n-grams are usually generated at a character or word level. The following figure illustrates the generation of 2-grams (or bigrams) at a character and word level.

<sup>7</sup>[http://www.irn.mj.pt/sections/irn/a\\_registral/registos-centrais/docs-da-nacionalidade/vocabulos-admitidos-e/](http://www.irn.mj.pt/sections/irn/a_registral/registos-centrais/docs-da-nacionalidade/vocabulos-admitidos-e/)

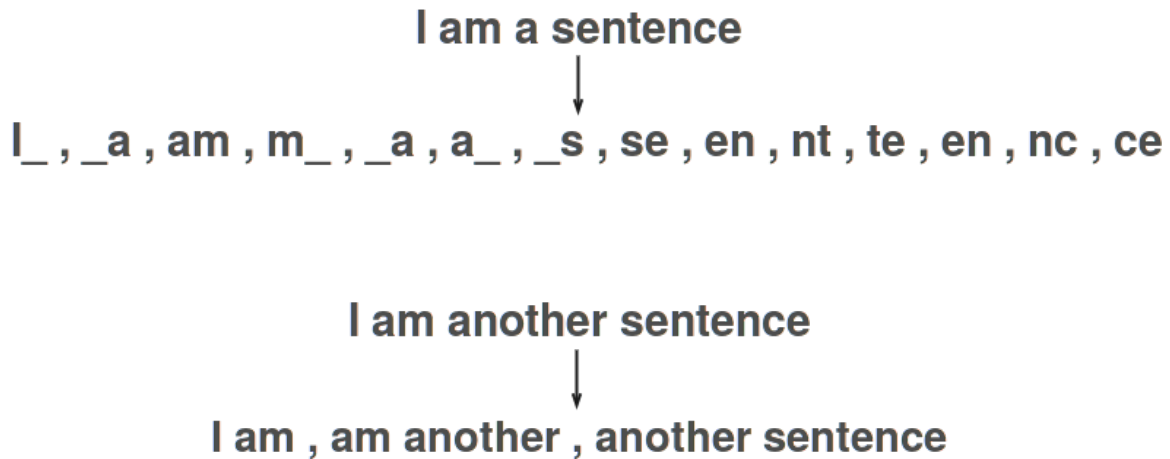


Figure 4.1: Generation of 2-grams at character and word level. (The \_ represents a blank space)

The second feature model relies on the identification of n-gram tokens on a character and word level on each of the available profile fields and user’s tweets. The assumption on n-gram feature models is that the repetition or occurrence of certain tokens can help identifying hidden patterns in text, in this case it is hoped that it can identify gender-defining patterns. This model has been used for latent attribute detection in the work referred in section 2.2.7 with relevant accuracy. For this matter, it was built a dictionary of character and word n-grams for each of the profile information sources, limited to a thousand tokens (or features), with the  $n$  varying according the n-gram level (character or word) and with the information source. This dictionary was built from a thousand random users on the dataset, with the condition that they must have ten posted tweets. The following table describes with detail the chosen properties of the n-gram model.

Table 4.6: N-gram feature model

<b>Field</b>	<b>Character range</b>	<b>Word range</b>
Name	1-5	1
Screen name	1-5	-
Description	1-5	1-2
Tweets	1-5	1-2

After building the n-gram dictionaries, this feature model consists on counting the binary occurrence of n-grams on each of the user’s profile field and tweets, on their corresponding dictionary.

## TRAINING THE CLASSIFIER

The classifier used for this task was a Support Vector Machine with a linear kernel implementation from the Scikit-learn library. As explained in section 2.2.3, SVMs with a linear kernel behave very positively when handling more features than samples, which is the case. The features generated in the n-gram model by itself easily exceed the number of available samples.

To understand the impact of the two feature models, they were tested separately and combined with each other. Understanding the source for features that produces more relevant results is important



information for social media research, therefore the user profile fields used for classification were also tested separately and combined. To increase the chances of greater accuracy, the number of features used in the model was also taken into account. To achieve this, a Python script was developed that allowed the configuration of all of this parameters as well as model specific parameters. It can be found in Appendix-B section 6.2.

The following figure illustrates the training of gender classifier model.

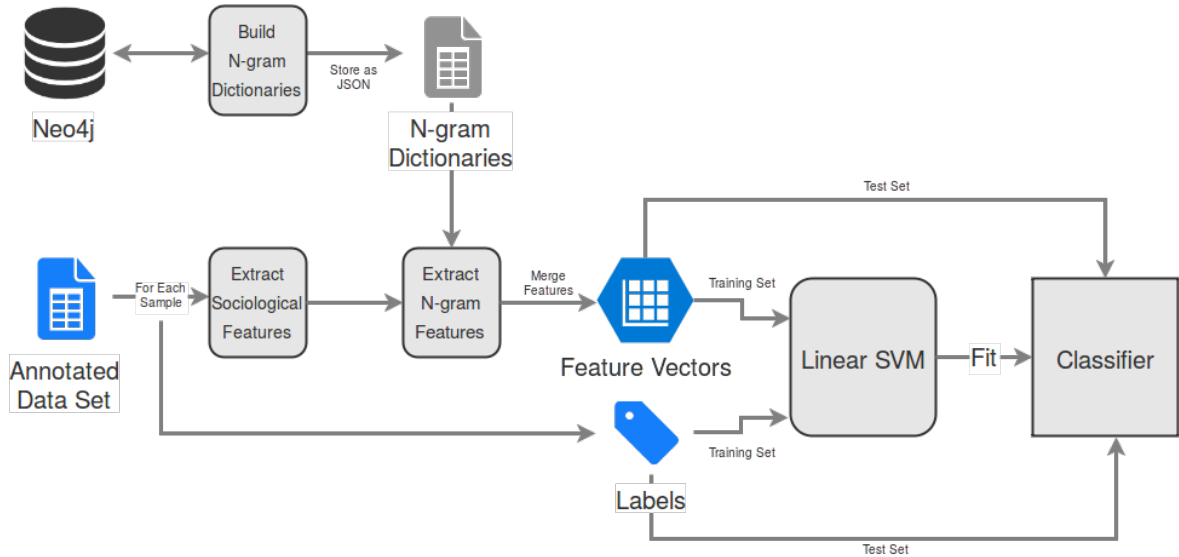


Figure 4.2: Training of the gender classifier

To validate the developed model, it was used a testing data set made of 400 of the manually annotated profiles. The procedure for annotation was the same as in the training set and the label distribution is also equally distributed. To obtain the model with the best accuracy, each of the parameter combinations were used on this testing dataset. The following tables show the results obtained for the gender classification task.

Table 4.7: Classification results for the sociological feature model

Field(s)	Accuracy
Username	0.725
Description	0.525
Tweets	0.595
Username + Description	0.775
Username + Tweets	0.740
Description + Tweets	0.595
All	0.755

Table 4.8: Classification results for the n-gram model

N-grams Scope	Fields	Number of Features			
		2000	5000	10000	20000
Char	Username	0.750	0.745	0.745	0.745
Word	Username	0.655	0.655	0.655	0.655
Char+Word	Username	0.745	0.735	0.745	0.740
Char	Screen name	0.710	0.71	0.715	0.735
Char	Description	0.640	0.685	0.675	0.680
Word	Description	0.530	0.535	0.545	0.545
Char+Word	Description	0.645	0.650	0.665	0.685
Char	Tweets	0.640	0.635	0.640	0.645
Word	Tweets	0.645	0.650	0.655	0.655
Char+Word	Tweets	0.620	0.625	0.635	0.635
Char	Username + Screen name	0.765	0.760	0.760	0.770
Char+Word	Username (word) + Screen name	0.705	0.715	0.720	0.735
Char	Username + Description	0.765	0.745	0.745	0.770
Char+Word	Username (word) + Description	0.670	0.685	0.695	0.695
Char	Username + Tweets	0.670	0.665	0.665	0.665
Char+Word	Username(word) + Tweets	0.640	0.635	0.640	0.645
Char+Word	Username(all) + Screen name	0.760	0.755	0.760	0.770

Table 4.9: Classification results for the joint models

Soc. Model	N-gram Model	Number of Features			
		2000	5000	10000	20000
Username + Description	Username (char)	0.785	0.785	0.785	0.790
Username	Username (char)	0.810	0.8	0.8	0.805
Username + Tweet	Username (char)	0.805	0.81	0.805	0.810
Username + Description + Tweet	Username (char)	0.810	0.830	0.830	0.830
Username	Username (char + word)	0.8	0.790	0.785	0.790
Username + Description	Username (char + word)	0.815	0.800	0.800	0.800
Username + Tweets	Username (char + word)	0.810	0.805	0.800	0.800
Username + Description + Tweet	Username (char + word)	0.810	0.830	0.830	0.830
<b>Username + Description</b>	<b>Username (char) + Screen name</b>	<b>0.835</b>	<b>0.845</b>	<b>0.845</b>	<b>0.860</b>
Username + Description + Tweets	Username (char) + Screen name	0.830	0.835	0.835	0.845
Username + Description	Username (char + word) + Screen name	0.830	0.845	0.850	0.855
Username + Description + Tweet	Username (char + word) + Screen name	0.830	0.835	0.835	0.845

Looking at the obtained results, it is notable that the dictionary of first names and their corresponding gender has a tremendous impact on the classifier’s accuracy when using the sociological feature model. Besides that, it is possible to assume that tweets are a better source for gender distinction than the user’s description, as joining username features and tweet features gives a slight boost in accuracy. Regarding the n-gram model, the user’s screen name is the profile field that generated the most interesting result, since it was unexpected that it would have such a big impact in the classification results. The user’s tweets and description once again fail to achieve considerable accuracy rates, either with character or word n-grams. When the two models are stacked together character n-grams from the username and screen name, combined with sociological features extracted from the username and description proved to be the most effective combination for the classification task.

To further validate the results, the model that obtained the best accuracy percentage with the 1000 train samples and 400 test samples was submitted to a k-fold cross validation (referred in section 2.2.3) over the full 1400 manually annotated samples. The chosen k was seven, which meant the data set was divided into seven portions of 200 users and trained seven times, with each time having a different portion serving as test. The average accuracy obtained with this validation method was 82% with an error margin of more or less 0.07%.

## CLASSIFYING THE DATA SET

Having accomplished the task of building a classifier with relevant precision, it was necessary to classify the remainder of the dataset to make the information permanently and directly available to our system. The optional schema of Neo4j databases allows us to modify the properties of existing nodes without affecting the database integrity and consistency. Therefore, to classify all the User nodes it was required to iterate each and everyone of them, extract its features, send them to the classifier, create a new property in the node with the resulting value from the classifier and update the node so that the changes become permanent. Since the classifier with best results discards the use of tweets it was only necessary to retrieve the user's name, screen name and description.

### 4.2.3 NETWORK INFLUENCE

#### *Determining which users can be considered influential on the Portuguese Twitter network*

For influence detection, it was decided to only use graph centrality measures as the indicator. The convenience of the property graph data model has been thoroughly explained throughout this document and the use of graph centrality measures for influence detection is explained in section 2.2.6. Influence measuring relies on quantifying the interactions a certain user does in the network. To the extent of this work, this means conditionally quantifying the edges that are related to a node in the graph.

There were a few adequate and available metrics for quantifying influence, which were the user's mentions, the number of times his tweets are retweeted, the user's replies and/or quotes. When choosing which ones would be used to measure influence an anomaly was detected in the dataset. There is not a single tweet in the data set with the retweet flag set to true, even though there are tweets that are quoting another tweet. The explanation for this issue could not be found but it is most likely internally related with the Twitter API. The other explanation is that Portuguese Twitter user's do not use the retweet mechanism properly, which is highly unlikely.

Having eliminated retweets as a metric for influence there were still available mentions, replies and quotes. The number of mentions was chosen as the influence metric, since it has been proven a reliable indicator [51] and was the one with greater presence in the data set (see table 4.3).

The centrality measure used to quantify the user's influence was in-degree centrality which translates to counting the number of edges that represent a MENTION that are directed a node. "Betweenness" centrality could also be used however this had extremely high computational cost to be done to every user node in the database as it required to calculate all the shortest paths between each user. The same applies for "closeness" centrality.

Cypher was used to calculate the in-degree score for each user. On a first instance the following query was developed:

---

```
MATCH (u1:User)-[p:POSTS]->(t:Tweet)-[m:MENTIONS]->(u2:User)
WITH u2 as target, count(m) as DegreeScore
SET target.degree_score = DegreeScore
```

---

Listing 23: First query developed for calculating the in-degree score of User entities

As it can be observed, this query counts each tweet that mentions a user. This can be extremely biased since it is only necessary one user making a lot of MENTIONS to elevate the influence score. Moreover, it ignores how many users this user has reached. To put it in less formal language, this method is susceptible to the "celebrity fever" effect, where an obsessed fan can spend a day mentioning him/her. It also fails to address conversations between users that generate a lot of mentions, without any influence value. A more fair metric would be to quantify the number of distinct users that mention a single user. This could be done by counting the users instead of the MENTION relationships, but in this case the query would be inefficiently searching every user's POSTS without it being necessary. To overcome this issue, it was necessary to create a user to user relationship that represented a mention. Intuitively, this means that if a user posts a tweet that mentions another user, then this user mentions the other. To achieve this, the next Cypher query was developed.

---

```
MATCH (u1:User)-[p:POSTS]->(t:Tweet)-[m:MENTIONS]->(u2:User)
WITH u1 as source, u2 as target
MERGE (source)-[:MENTIONS]->(target)
```

---

Listing 24: Query developed to create user-mentions-user relationships

Once again, this schema alteration has no effect on the database consistency, though it does change the initially proposed property graph data model by creating a new user to user relationship.

Subsequently, the calculation of the degree-score is in all manner similar to the query showed in listing 23, though it only has a one level relationship which makes it significantly less computationally expensive.

---

```
MATCH (u1:User)-[m:MENTIONS]->(u2:User)
WITH u2 as target, count(m) as DegreeScore
SET target.degree_score = DegreeScore
```

---

Listing 25: Second query developed for calculating the in-degree score of User entities

This query also creates the new property *degree\_score*, making it a permanent attribute in user nodes that can be used for further querying purposes like retrieving the User entities ordered by their *degree\_score*.

#### 4.2.4 NETWORK CONTENT

##### *Determining what is talked and discussed inside Twitter.*

To understand the content discussed in the Twitter network it was required a different approach to be accomplished as querying the users tweets is not enough to extract meaningful information from them. On another point of view, as much as Neo4j property graph data model is useful to analyze data relationships, it not as suitable to store text data in bulk. Therefore it was necessary to develop a complementary storage system, that allowed to store and retrieve with greater performance large amounts of text data.

#### USER AS A DOCUMENT

Since this part of the system differs from the rest, it was created a new data model representative of a User that will be addressed as Document. A Document consists of the following data:

- User ID;
- Number of tweets it has posted;
- All of the user's tweets;
- The Term Frequency (TF) of the user's tweets;
- The Term Frequency - Inverse Document Frequency (TF-IDF) of the user's tweets;
- The most relevant terms divided by user topics.

Only users that have at least 50 tweets posted can be represented as a Document. Ideally, the more tweets the more likely it is possible to achieve better results and a more valid analysis. However, the number of users that match this criteria abruptly decreases as the number of minimum tweets increases. This number represented a fine balance between the number of tweets necessary for a valid content analysis and the resulting number of users to analyze. To create the Documents, the set of Users stored in Neo4j that had at least 50 tweets was iterated, their tweets were retrieved, processed, converted to a Document and then stored in a Cassandra instance deployed for this purpose. Apache Cassandra was used since its properties assertively fit these requirements and it had already been used in project TVPulse.

To obtain the aforementioned data, it was required to process all the users tweets, since that is where the proposed data model derives from.

First of all, tweets were joined in a single string, each of them delimited by a line break. To achieve this, all tweets were stripped of all existing tabs and line breaks through the use of a regular expression and afterwards merged. The purpose of this preprocessing phase is to obtain a representation of a single user's tweets in a way that was able to store in a single Cassandra value store.

Secondly, it was necessary to preprocess tweets to make them suitable for TF, TF-IDF and LDA calculations. TF and TF-IDF are two statistical measures that account for the relevancy of terms in a set of text documents[93]. The first consists on counting how many times a term occurs in each of the available documents. This measure can be deceiving as it will not differentiate terms that have a frequent appearance in text documents but little meaning in the text content, like pronouns and adverbs, from relevant ones. To overcome this the Inverse Document Frequency (IDF) can be multiplied with TF, hence originating TF-IDF. This results in less weight to terms that have a broader appearance throughout the collection of documents.

To perform these calculations, it was used Scikit-learn's CountVectorizer and TfidfVectorizer. These two classes from the Scikit library can transform a collection of strings to a document-term matrix representation. They offer a number of customizable options for the transformation process but the relevant ones for this task were the tokenizer and the stopword\_filter. To have more control over the terms that were accepted, a custom tokenizer method was developed and it is presented in the next listing.

```
def tokenize_tweet(text):
    tokens = []

    #token must have a letter pattern
    word_pat = re.compile(r'[A-Za-z]')

    # Remove hashtags and mentions and urls
    processed_tweet = normalize(url_matcher.sub('',
        hashtag_mention_matcher.sub('', text)).lower().strip())

    # Extract tokens
    for token in wordpunct_tokenize(processed_tweet):
        # Assure that tokens contain at least three characters
        if word_pat.search(token) and len(token) > 2:
            tokens.append(token)
    return tokens
```

Listing 26: Python function that tokenizes a tweet

As shown, the tokenizing process comprises the following steps: normalization, which replaces punctuated characters by their natural form (e.g à to a), removal of urls, hashtags and mentions through the use of regular expressions (see Appendix-B listing 50), lower casing and white space stripping. URLs, hashtags and mentions were removed because they possess little or no value in exploring user topics and content. The stopword\_filter consists on a list of terms that have meaningless value and therefore are filtered out of the term matrix. These include terms from grammatical classes that offer no insight over the text topic (e.g. adverbs or determinants), nouns that are exceedingly common and swear words. To have an even more refined selection, only terms with more than two characters were accepted.

After fitting the collection of tweets, both the CountVectorizer and the TfidfVectorizer produce

a resulting document-term, i.e. tweet-term, sparse matrix<sup>8</sup>. On the first, each value of the matrix corresponds to the number of times that term occurs in the document and the second its TF-IDF score.

For LDA, it was used the Python library `lda`<sup>9</sup>. This implementation of `lda` associates words to a topic through a array-like structure that allowed to retrieve an arbitrary number of the most important words in each topic.

As features for the Latent Dirichlet Allocation algorithm, the term frequency matrix generated by the `CountVectorizer` was used. It has the number of times a term occurs in each tweet which is adequate for the correlation based inference expected in LDA. The LDA library also has a number of customizable parameters. In this task, the essential ones are the number of topics and the number of iterations. The number of topics was assumed five for each user and number iterations used was 1500. The number of predetermined topics is speculative, since it would not be a feasible task to manually analyze for each of the users which number returned the most interesting results. In result, given that each user holds at least 50 posted tweets it estimated that these tweets cover five different thematics. To support it results were manually analyzed for five different users. The number of iterations increases the correlation factor between identified terms. After running the topics modeling task with 500, 1000 and 1500 iterations it was observed that only when 1500 were used that the correlation factor stabilized for each user the algorithm was ran against.

The final stage was to store the obtained results of the three operations, along the id of the user and the number of tweets he had posted, in Cassandra. Storing the resulting matrices directly in the database was not possible and converting them to string representation that could be parsed is to my best knowledge impossible. The arranged solution was to to convert the matrices into a key-value dictionary. This allowed the conversion to JSON objects and therefore transformed to a string representation, that could afterwards be parsed into the original dictionary.

The next figure presents the overall flow of the creation of the Document models.

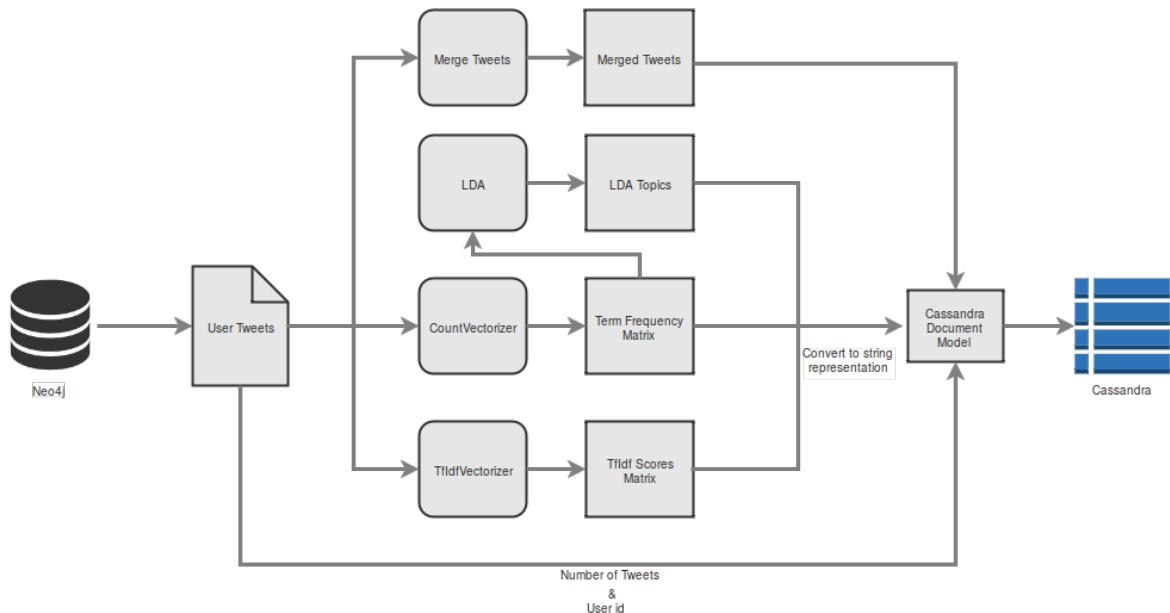


Figure 4.3: Creation of the Document model

<sup>8</sup>[http://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr\\_matrix.html#scipy.sparse.csr\\_matrix](http://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html#scipy.sparse.csr_matrix)

<sup>9</sup><http://pythonhosted.org/lda/>

## CLUSTERING THE DOCUMENTS

This problem was addressed with an Unsupervised Classification approach. It comprised of a two step solution that encompassed labeling all the Documents into clusters, with the purpose of grouping the users that have similarities in their content and checking the most relevant terms used in each cluster. The second step consisted of projecting those users in a two dimensional plane to see how the clusters were formed.

To accomplish the first task, it was used the Scikit's implementation of the K-means clustering algorithm, trained with TF-IDF features. Since each user's tweets were condensed into a single string and stored, it is possible to fit them in Scikit's TfidfVectorizer to create a user-term matrix, in which a column represents a term, a row represents a user and consequently the intersection contains the TF-IDF score for that user. This posed a challenge in the preprocessing stage of the TfidfVectorizer, since the tokenizer developed before was made to process a single tweet. To solve it, a tokenizer that understands that it is before a collection of tweets delimited by a line break was developed, and it is presented in the next listing.

```
def tokenize_document(doc):
    # Split document into original tweets
    tweets = doc.split('\n')

    tokens = []

    #token must have a letter pattern
    word_pat = re.compile(r'[A-Za-z]')

    for tweet in tweets:
        # Remove hashtags and mentions and urls
        processed_tweet = normalize(url_matcher.sub('',
            hashtag_mention_matcher.sub('', tweet)).lower().strip())

        # Extract tokens
        for token in wordpunct_tokenize(processed_tweet):
            if word_pat.search(token) and len(token) > 2:
                tokens.append(token)

    return tokens
```

Listing 27: Python function that tokenizes a Document

After producing the TF-IDF matrix for all the Documents, the K-means model was fitted with it. K-means requires the number of clusters to be specified before being run. The best k was found to be three, after trying with the values from two to six. The result of the K-means algorithm was a list of the labels for each user, i.e. which cluster it belonged.

To be able to visualize the resulting labels, it was necessary to find an humanly understandable representation of the Documents clustered, in this case in a bi-dimensional space. To do this, it was



necessary to find a measure that could evaluate how "distant" the Documents were to one another and afterwards represent that distance in a two dimensional plane.

To find out how close the Documents were related it was calculated the cosine similarity their TF-IDF feature vectors. The cosine similarity implementation of Scikit-Learn works with matrix structures, which allowed it to be directly calculated over the produced TF-IDF matrix. However, it was still necessary to project these results into the desired dimensional space, which was accomplished using Scikit's implementation of PCA (explained in section 2.2.2). Due to the considerable size of the cosine similarity score matrix, the implementation of PCA that was used was the IncrementalPCA, since it scales considerably better than the standard implementation, while obtaining the same results. The process of discovering the network content clusters described through this section is illustrated in listing 4.4.

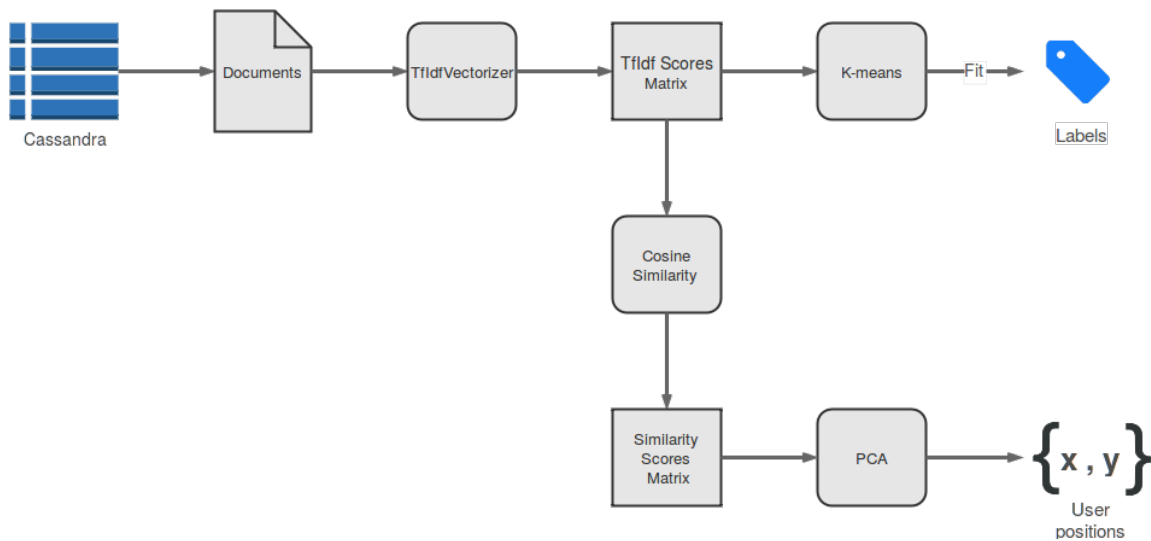


Figure 4.4: Clustering of Documents

## 4.3 USER SCOPE GOALS

In this section it will be explained how the user explorer was developed and its functional requirements were achieved. This part of the system focuses on providing detailed information about the owner of a specific Twitter profile and contextualize him within the network. The objectives for the user explorer (see section 3.1.2) can be divided into three categories **Activity**, **Network** and **Content**.

### 4.3.1 USER ACTIVITY

Much like in Network Activity, this part of the system aims to present information about Twitter activity, but regarding only a single user. In this scope it was decided to divide the activity analysis over time and source.

## USER ACTIVITY OVER TIME

*Determining the time periods in which the user concentrates his activity on Twitter i.e the time slots in which the user is more active*

After uncovering the Twitter global network activity over time, doing the same on a user basis is straightforward as the process is the same but restricted to only the tweets posted by the user being profiled. To accomplish this the following Cypher queries were developed:

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
RETURN t.month as month, count(t) as counts
```

---

Listing 28: Query for user activity over months

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet {month:
  <month>})
RETURN count(t) as counts
```

---

Listing 29: Query for user activity on a specific month

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
WHERE t.month = <month>
RETURN t.day as day, count(t) as counts
```

---

Listing 30: Query for user activity for all days in a month

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
WHERE t.month = <month> AND t.day = <day>
RETURN count(t) as counts
```

---

Listing 31: Query for user activity on a specific day

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
```

---

```
WHERE t.month = <month> AND t.day = <day>
RETURN t.hour as hour, count(t) as counts
```

---

Listing 32: Query for user hourly activity on a specific day

It was also decided to obtain which day of the week is a user more active and the hour of the day in which his activity is concentrated, as they are both interesting facts about user activity. To accomplish it, the following queries were developed.

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
RETURN t.created_at as timestamp
```

---

Listing 33: Query needed to calculate which weekday is the user more active

To get the counts for each week day it was required the time stamp of the Tweet, parsing the date and get the corresponding weekday.

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
RETURN t.hour as hour, count(t) as counts
ORDER BY counts DESC
```

---

Listing 34: Query for user activity for every hour

## USER ACTIVITY OVER SOURCE

*Determining how the user interacts with Twitter in terms of types of devices chosen to maintain his profile*

Like in activity over time, activity over source has the same process as in the Network Explorer but restricted to the tweets posted by a single user. To meet this goal the next Cypher query was developed, which is the same as the one used to get tweet sources for the network but restricted to a single user.

---

```
MATCH (u1:User {user_id: <user_id>})-[:POSTS]->(t:Tweet)
RETURN t.source as source
```

---

Listing 35: Query for user activity sources

### 4.3.2 USER NETWORK

This section meets the necessity of discovering relevant information about the user's interactions and place within the network. The proposed goals dealt with finding which users are the ones that are more frequently a target of Twitter interactions (mentions and replies), along with a way to determine the distance between this user and another.

#### INTERACTIONS

*Determining which users interact with the user being profiled*

*Determining which are the users which the user being profiled frequently interacts*

Once again, Cypher is the support to obtain the desired the results. This is one of the goals in which the use of a graph database is particularly helpful since it is directly related to data relationships. It is meant to calculate which are the users that most frequently mention or reply the designated user and oppositely, the users that this user most frequently mentions and replies. This can be obtained using four different Cypher queries which are presented as follows.

---

```
MATCH (u2:User)-[p:POSTS]->(t:Tweet)-[m:MENTIONS]->(u:User {user_id:
    <user_id>})
RETURN u2 as user, count(p) as n_mentions
ORDER BY n_mentions DESC
LIMIT <limit>
```

---

Listing 36: Query to obtain the users that more frequently mention other user

---

```

MATCH
    (u2:User)-[p:POSTS]->(t:Tweet)-[r:REPLIES_TO]->(t2:Tweet)<-[:POSTS]-(u:User
    {user_id: {id}})
RETURN u2 as user, count(r) as n_replies
ORDER BY count(r) DESC
LIMIT {limit}

```

---

Listing 37: Query to obtain the users that more frequently reply to another user

---

```

MATCH (u:User {user_id:
    <user_id>})-[p:POSTS]->(t:Tweet)-[m:MENTIONS]->(u2:User)
RETURN u2 as user, count(p) as n_mentions
ORDER BY n_mentions DESC
LIMIT <limit>

```

---

Listing 38: Query to obtain the users that another user more frequently mentions

---

```

MATCH (u:User {user_id:
    <user_id>})-[p:POSTS]->(t:Tweet)-[r:REPLIES_TO]->(t2:Tweet)<-[:POSTS]-(u2:User)
RETURN u2 as user, count(r) as n_replies
ORDER BY n_replies DESC
LIMIT {limit}

```

---

Listing 39: Query to obtain the users that another user more frequently replies to

With the use of Cypher it is possible to obtain for each of the requirements, the list of the users that most frequently make use of each of the interaction patterns presented in a clean and ordered subset.

## DISTANCE

*Determining the network distance between this user and any other user i.e. being able to visualize how "close" this user is to other users*

The purpose of this goal is to be able to understand how "close" in the network two users are. Closeness is defined here as the shortest amount of links one has to traverse to go from one user node to another. This is an interesting measure to the extent that it allows to extrapolate how likely two users

are acquaintances in real life or through which users that is possible happen. Cypher offers the built-in pattern *shortestPath* which, as the name implies, returns the shortest path between two entities in the graph. Making use of this pattern, the following Cypher query was developed.

---

```
MATCH n = shortestPath((u1:User {user_id: <user_id1>})-[*]->(n2:User
    {user_id: <user_id2>}))
RETURN n as path
```

---

Listing 40: Query to obtain the distance between two users

This query differs from the rest because it does not return a list of entities or an absolute value but instead it returns a path consisting of the existing User nodes that were encountered between the two users and the relationships that connect them.

### 4.3.3 USER CONTENT

#### *Determining what are the topics of interest of this user*

#### *Determining what users in the network have similar topics of interest with the user being profiled*

This section of the user explorer is designed to show what the user is talking about on Twitter. Like uncovering network topics, it differs from all the other proposed goals since it mostly relies on the analysis of text content and not on data connections. This kind of analysis is only possible if there is enough content, meaning that it can only be performed on users with a minimum number of tweets posted, otherwise it is meaningless to try to perceive their posted content. This requirement was taken into consideration when defining User Documents, as explained in section 4.2.4. As it was defined in the User Scope requirements (section 3.1.2), this section aims to correctly identify which are the topics of interest of a determined user, as well as identifying other users that relate to those same topics.

### 4.3.4 TOPICS

The process for modeling user topics is explained in complete detail the section 4.2.4. Using Python lda over the document-term frequency matrix of a user, it was possible gain insight over the topics of interest of a user through the co-occurrence of words in tweets. This process was completed when creating the User Document database, having the results of the LDA algorithm being stored for every possible user. However, this is only an indicator of the possible topics of interest of a user and requires human interpretation to actually understand what the correlated terms refer to. An adequate visualization scheme is required so that it is possible to intuitively understand what are the relevant terms that represent the user's topics of interest, which will be addressed further in this document.

### 4.3.5 SIMILARITY

This requirement consists on determining which users post similar content when compared to the profiled user. It is a problem of text similarity. We want to be able to pick the collection of tweets posted by two users and verify if the topics they address are similar to one another. The method chosen to verify if two users can be considered similar was calculating the cosine similarity of their TF-IDF scores. This allowed to understand how close two TF-IDF scores of the users are, indicating they have a similar choice of words. By calculating it over TF-IDF, the relevancy of terms is also taken into consideration, making it a suitable measure for a similarity measure.

Comparing the TF-IDF scores of a user with every other user, one at a time, would be an extremely slow process. However, since Scikit's implementation is being used, it is possible to calculate cosine similarity between two matrices, which turns out to be a huge benefit.

Since all the user's term frequencies are stored in a single matrix and Scikit's implementation of cosine similarity works with matrix structures, to perform this calculation between a user and the others it was retrieved the matrix row corresponding to the desired user and afterwards compared to the complete matrix. As it is a standard in Scikit, the result of this operation was Sparse Matrix, with the similarity scores to each user. To get the more similar ones, it was necessary to retrieve the indexes with the highest scores.

Note that by using matrix structures to perform this operation, the only indicator available to determine which row of the matrix corresponds to a user is the row's index. When creating the matrix with all the users term frequencies it was also necessary to store in an auxiliary list each of the user's ids in an orderly fashion, so that the indexes of the matrix rows match the indexes in the list and it is possible to trace back a matrix row to a user.

## 4.4 WEB APPLICATION

After detailing what consists of the supporting storage system and the functional logic behind the proposed goals, it was developed a web application that incorporated both and added a visual representation of the obtained results. The following subsections will address the technological components used to conjugate the created storage system with the functional logic needed to tackle the designed requirements and how the visualization layer that feeds on the generated information was developed and integrated.

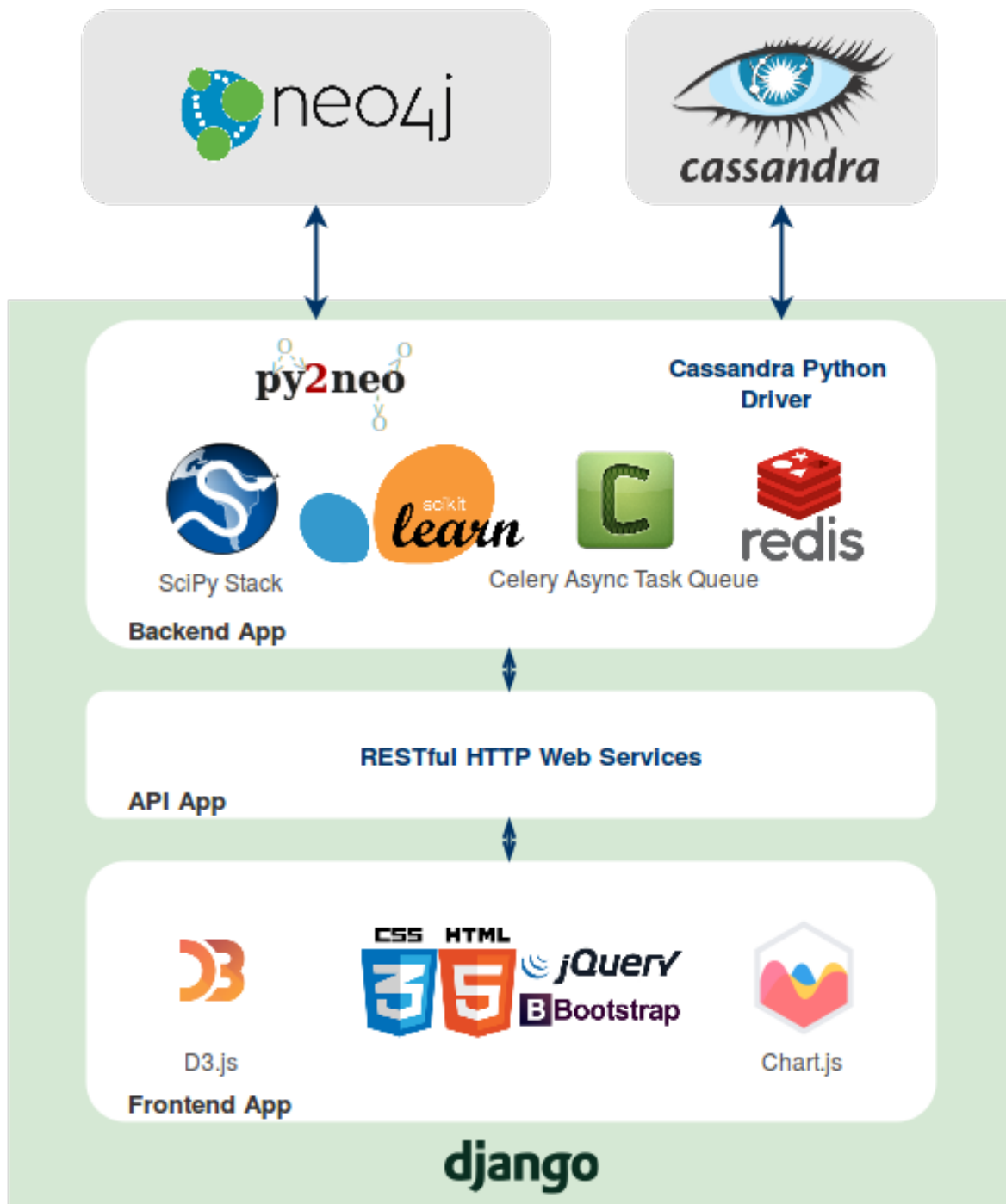


Figure 4.5: Architecture diagram of the web application

#### 4.4.1 DJANGO WEB FRAMEWORK

Considering that the most of the developed work is based on Python libraries, especially ML and data mining tasks, it made sense to build the web application with a Python web framework, as it would greatly facilitate their integration. On top of that, the database migration was made using Py2neo which was found to be an easily understandable and usable library for Neo4j. Taking these requirements into consideration, the framework chosen to implement the web application for this



dissertation was the Django Web Framework<sup>10</sup>, which is regarded as the most reliable and mature Python web framework. Other alternatives consisted of Flask<sup>11</sup> and CherryPy<sup>12</sup>. Although these frameworks are considered more *pythonic* and flexible, they lack the out of the box features that Django has, like a page templating system, an object-relational mapper or a caching framework and require this sort of functionality to be either developed or delegated to third-party libraries. Using Django allowed for a development focused on the system's functional logic.

Django possesses a great set of out of the box functionality that eases the development process of web applications. Features like authentication, administration, database connectors and cache are an integral part of the Django Web Framework and are in place and ready to use if necessary.

Typically, a Django project consists of a collection of modular individual apps. The project directory contains the configuration file that defines the whole system settings and the URL mappings to all the existing apps. Being modular, developed apps can be reused in other Django projects.

With the purpose of developing a web application in mind, rendering of HyperText Markup Language (HTML) files is an important feature as it will be what enables the visualization of pretended results. Django defines itself as a Model-Template-View (MTV) framework where Models are responsible to make a relational mapping between database objects and application objects, Templates define how the data is presented and Views decide what data is to be presented through the use of Python code for logic. This design pattern is pretty similar to Model-View-Controller (MVC), the nuances that distinguish them can be found in [94]. To make use of Django templates, it is necessary to add to the app a template directory where the HTML pages will reside.

As defined in architecture section, the proposed web application consists of three core modules: Backend, Frontend and API. This architecture can directly mapped into three different Django apps and this was the chosen path for the application development.

## BACKEND

The Backend app is the one responsible for establishing the connection with the graph data model in Neo4j and implement the functional logic of the system. One of the defining components of Django its his Object-Relational Mapping (ORM). The Django ORM is the component that enacts the mapping of Relational Database tables into Python objects allowing an object-oriented development. The utmost advantage of working with an ORM is that it provides an additional abstraction layer over the database, making possible to automate a series of SQL commands like SELECT, INSERT, UPDATE or DELETE without working directly with SQL. Since this system doesn't make use of a Relational Database, ORM would seem an irrelevant component to use in Django, but Neo4j, while not being a Relational Database, allows the implementation of a Object-Graph Mapping (OGM) since entities in the graph data model can also be directly mapped to Python objects. An OGM is by all means analogous to an ORM, with the difference being that it maps graph entities and even relationships to objects, instead of Relational Database's tables.

Py2neo has an integrated OGM, that was used to map graph entities in the data model into Python objects. Listing 41 shows how the User entity was mapped as a Python object.

---

<sup>10</sup><https://www.djangoproject.com>

<sup>11</sup><http://flask.pocoo.org/>

<sup>12</sup><http://cherrypy.org/>

```

from py2neo.ogm import GraphObject, Property, RelatedFrom, RelatedTo

class User(GraphObject):
    __primarykey__ = "user_id"

    user_id = Property()
    screen_name = Property()
    name = Property()
    description = Property()
    friends_count = Property()
    followers_count = Property()
    favourites_count = Property()
    created_at = Property()
    degree_score = Property()
    gender = Property()
    predicted_gender = Property()
    profile_pic = Property()
    lang = Property()
    url = Property()
    location = Property()
    statuses_count = Property()

    posts = RelatedTo('Tweet', 'POSTS')
    mentions = RelatedTo('User', 'MENTIONS')

```

Listing 41: User entity as a Python object

To make the models recognize that they are mapping an entity from a Neo4j instance it is necessary to set up a connector object with Py2neo in one of the Django configuration classes, so that it knows the HTTP or Bolt endpoint through which the Neo4j database is accessible.

The Backend app is also responsible for implementing all the logic behind the requirements described in the previous sections. Those that rely on querying can be either mapped directly with a filter on the defined model or, if they require more complex querying, the Py2neo connector can be used to run of custom Cypher queries.

The goals that dealt with topic modeling and similarity required the use of a complementary storage component, Cassandra, which is also handled in this app. Although there is a Django mapper for Cassandra<sup>13</sup> it is not a mature solution so it was decided to use the more robust Cassandra Python-Driver<sup>14</sup> and integrate it in the system. This full fledged library for Cassandra manipulation in Python, possesses an object mapper from Cassandra tables to Python objects, which was used to create the following model.

<sup>13</sup><https://pypi.python.org/pypi/django-cassandra-engine/>

<sup>14</sup><http://datastax.github.io/python-driver/index.html>

```

from cassandra.cqlengine.models import Model
from cassandra.cqlengine import columns

class Document(Model):
    __keyspace__ = 'twitterx'
    __table_name__ = 'user'

    userid = columns.VarInt(primary_key=True)
    num_tweets = columns.Integer()
    document = columns.Text()
    tf = columns.Text()
    tfidf = columns.Text()
    topics = columns.Text()

```

Listing 42: Cassandra Document as a Python object

Through this model, it was possible to perpetrate operations over the stored Documents in an object-oriented fashion and easily retrieve each user's data. For topic modeling, term frequency, TF-IDF scores and LDA topics are stored as "stringified" dictionaries in Cassandra meaning that their retrieval is a matter of parsing them. As for similarity, the solution was to create a Python class called UserSimilarity that is instantiated on application start up and that contains the list of user's ids and the term-frequency matrix of all users. To be able to reuse this matrix, it was used Scikit's class joblib to persist it as a Python object then to load it and make it a part of this Python class. The goal of this class is to, given a user's id, provide the user's ids of its n more similar users. It can be seen in listing 43.

```

import json
import numpy as np
from sklearn.externals import joblib
from sklearn.metrics.pairwise import cosine_similarity
from os import path
import sys

# Necessary to unpickle the persisted object
current_path = path.dirname(__file__)
sys.path.append(current_path)

class UserSimilarity:
    def __init__(self):
        # Necessary for tracing user_ids by index
        self.users =
            json.loads(open(current_path+'/filtered_users.json',
                'rb').read())

        # user|term matrix
        self.tfidf_matrix =
            joblib.load(current_path+'/tfidf_matrix.pkl')

    def get_similarity(self, userid, n_users):
        # Get desired user data
        matrix1 = self.tfidf_matrix.getrow(self.users.index(userid))

        # Calculate distance between this user and the rest of the
        dataset
        distances = 1 - cosine_similarity(matrix1, self.tfidf_matrix)

        # Get the indices of the n most similar users
        # First user will be the original user
        indexes = np.argsort(distances[0])[:n_users]

        # Get the similarity scores of the users
        scores = distances[0][indexes]

        # Get user ids
        users = [self.users[index] for index in indexes]

        return dict(zip(users, scores))

```

Listing 43: Class developed to calculate similarity between one user and the rest

The next figure shows how the TopicSimilarity class loads the necessary term-matrix to perform its calculations.

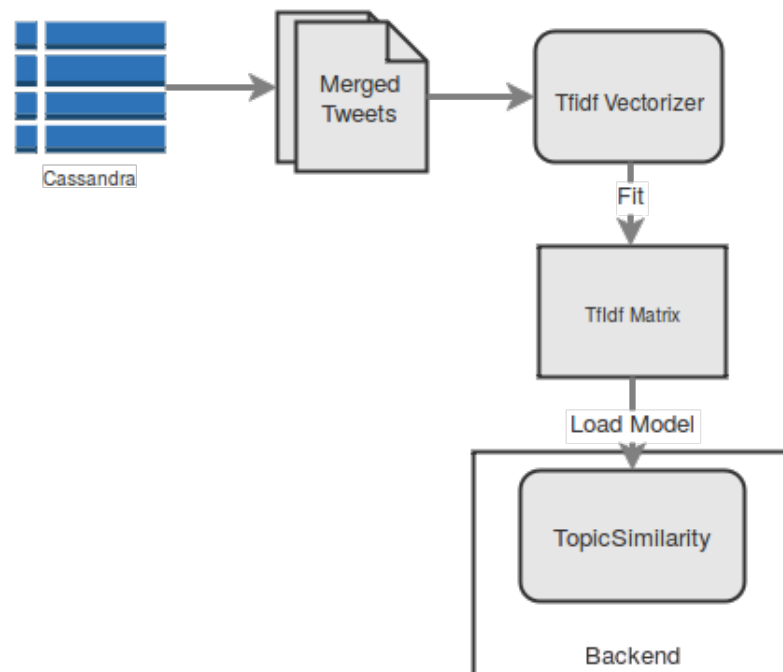


Figure 4.6: Loading of the TopicSimilarity class

## API

The API app is responsible for implementing the API methods described in the section 3.3.3 of the previous chapter. All the developed methods are exposed through an URL and therefore are implemented through the Django Views paradigm. They follow a logic of either fetching results stored in cache or require the Backend to calculate/retrieve them again. If this happens, the API methods will cache the results. The system's cache mechanism explained in the Redis section. The response of all the developed API methods is a JSON object. The following represents a response to the API method that retrieves the tweet counts by district in a given month.

```
{
  "PT.SA": 51863,
  "PT.FA": 50392,
  "PT.CO": 72608,
  "PT.AV": 116084,
  "PT.VR": 8942,
  "PT.BA": 4832,
  "PT.BE": 8908,
  "PT.PO": 229262,
  "PT.LI": 386759,
```

```
"PT.GU": 4260 ,
"PT.PA": 8143 ,
"PT.VC": 3379 ,
"PT.CB": 8165 ,
"PT.BR": 52496 ,
"PT.VI": 17248 ,
"PT.EV": 16564 ,
"PT.LE": 67671 ,
"PT.SE": 153375
}
```

Listing 44: Example of an API response

## FRONTEND

The Frontend app is the one that encompasses the visible part of the system. It mostly consists of the HTML pages that will be displayed to the end user and the Javascript and Cascading Style Sheets (CSS) frameworks used for visualization that will be explained in detail further. This app makes use of the Django template paradigm for HTML rendering and in some cases the Template API<sup>15</sup> for data display, although most of that work is done using Javascript visualization libraries. To accelerate the development, it was used an open-source HTML & CSS web administrator dashboard template built on top of the Bootstrap<sup>16</sup> framework.

### 4.4.2 CELERY

Celery<sup>17</sup> is an asynchronous task queue based on distributed message passing. These queues represent a mechanism to distribute work (tasks) across threads or machines (workers). For accomplishing this, Celery relies on communicating through messages, using a broker to mediate interactions between clients and workers. The most common brokers used with Celery are message brokers such as Redis<sup>18</sup> or RabbitMQ<sup>19</sup>, given their asynchronous capabilities. However, some Database engines are also partially supported.

As it stands, the developed system makes use of Celery's capabilities, using Redis as a broker, by launching tasks at the application start up. This will act as a pre-loader for certain for some visualization schemes that involve processing amounts of data that are too big for real-time loading and therefore done before the application starts running and stored in cache. Celery is used in this system to enhance user experience with the application. Without having data preloaded it would be necessary for the end user to wait uncomfortable amounts of time to obtain some of the displayed data.

<sup>15</sup><https://docs.djangoproject.com/en/1.10/topics/templates/>

<sup>16</sup><http://getbootstrap.com/>

<sup>17</sup><http://www.celeryproject.org/>

<sup>18</sup><http://redis.io>

<sup>19</sup><https://www.rabbitmq.com>

Even with Asynchronous JavaScript and XML (AJAX) requests, which cause the application not to stall, he would still have to wait to see the data which is what is trying to be avoided.

### 4.4.3 REDIS

Redis is an in-memory data structure store, used as database, cache and message broker. Its main purpose is to serve as a reliable and fast key-value database, which is a simple yet very powerful model, particularly when considering that Redis supports data types such as strings, lists or sets.

This system makes use of Redis with two different purposes. The first is as a message broker for Celery tasks and the second as a cache. The caching mechanism is very important in the system in terms of performance. Some of the queries listed in the previous sections are very time consuming and impracticable to make in real-time, therefore they must be executed before the system loads and their results must be accessible through the system's cache. Using Redis as cache allow for the query sets to be stored as they are, due to its flexible data type support.

### 4.4.4 VISUALIZATION FRAMEWORKS AND LIBRARIES

In this section, it will be presented some of the development frameworks used to construct web based, visually appealing representation schemes. It consists of modern state of the art HTML, CSS and Javascript frameworks that are frequently used for this purpose.

## ADMINLTE AND BOOTSTRAP

The AdminLTE<sup>20</sup> Control Panel template is an open-source web administrator and dashboard template. It is a responsive HTML template that is based on the Bootstrap 3 framework.

This template utilizes all of the Bootstrap components in its design and re-styles many of them to create a consistent design that can be used as a user interface for backend applications. It was also designed with modularity in mind, which allows it to be easily customized and built upon.

AdminLTE comes with a wide range of Javascript libraries and plugins used to create charts, interactive components and animations. This template was used as a mean to delegate some of the work required to develop a visually appealing user interface that is consistent and customizable.

## D3.JS

D3.js<sup>21</sup> is a Javascript library created by Mike M. Bostock built for the manipulation of data based documents and to present them in virtually any format. It is framework agnostic, meaning that it can be used along with every web development framework without losing any feature. D3.js operates over HTML, CSS and Scalable Vector Graphics (SVG) elements and offers a wide set of features to manipulate them, that go from simply displaying them in a specific format, to creating animations or highly-interactive representations.

---

<sup>20</sup><https://almsaeedstudio.com/>

<sup>21</sup><https://d3js.org/>

This framework is purely client-side, meaning that it runs exclusively on the user's browser. Data processing can and should be done outside the framework, for the sake of making the representations more lightweight.

In this system, D3.js was used to create animated graph-layouts, although it also uses other representations that were made by the open-source community based on D3.js. This system's graph-layouts are mostly based on D3.js' force-layout. Force-layout is manipulation property that enables the creation of directed forces in graphical animations. These forces can analogously compare to real life physical mechanics like repulsion or gravity. In D3.js, repulsion charges are made to make graph nodes repel each other and gravity is pseudo-representation of the gravitational force that pushes the graph nodes the center of the area of representation. It is also used a weight on node links to make them more or less close to each other.

D3.js has a fairly steep learning curve, due to its vast set of features and their high degree of configurability. One can easily lose grasp over the enormously big variety of possibilities that exist when creating graphical representations, meaning it was necessary to study existing examples, filter the relevant and appropriate existing features and adapt them to the system's requirements.

## CHART.JS

Chart.js is Javascript library focused on creating clean, appealing and interactive charts. Although it does not have such a vast offer of chart types, the ones that does have are fully featured and easily configurable. This library is included in the AdminLTE template that was used for the web application and this template has some built in visual integrations for charts created using it. This made it the obvious choice to create the necessary charts for requirements on information over a time period or distributions. The charts used in the framework for this web application were Line Charts and Pie Charts.

## OTHERS

Since Chart.js has somewhat a limited offer of chart types and D3.js' a slow development process due to its complexity, there were used other third-party open-source libraries to create specific data representations. The plugins used are Cal-Heatmap<sup>22</sup>, Datamaps<sup>23</sup> and D3-Wordcloud<sup>24</sup>.

Cal-heatmap is a Javascript module made to create a calendar heat map, like the ones used in Github<sup>25</sup> to show user contributions. A calendar heat map is a suitable way to represent time-series data over large time spans. In this system it was chosen to show the network activity over the available months in the dataset.

Datamaps is a Javascript library made to create customizable and interactive data visualizations over geographical data. It comes with map topologies for a great part of the globe. Datamaps was used to visualize network activity over the different Portuguese districts.

D3-Wordcloud is a D3.js plugin made by to create word clouds. Word clouds are a visualization technique used to highlight relevant words in a set. It was used to represent the user's tweets term-frequency.

---

<sup>22</sup><http://cal-heatmap.com/>

<sup>23</sup><http://datamaps.github.io/>

<sup>24</sup><https://github.com/jasondavies/d3-cloud>

<sup>25</sup><https://github.com/>



All of this Javascript modules are dependent on the D3.js library, which shows how flexible and powerful it can be when it comes to data visualization.



# RESULTS

---

*In this chapter it will be presented the end result of the developed prototype and the visual results obtained for each of the defined system requirements.*

## 5.1 PROTOTYPE

The developed prototype features visual and interactive representations of the proposed goals allowing the end user to:

- Check Twitter activity on the network scope under three different categories (time, location and source) through chart representations while being able to browse through different time spans (monthly, daily, hourly);
- Check Twitter global posting contents in a visually appealing way;
- Browse through a ranked list of the most influential users in the Twitter network and filter that list under defined properties;
- Choose a user to analyze and obtain information over his profile data, activity, network and posted content. All this information is presented graphically in the form of charts or intuitive data representations;
- Classify any Twitter user on its gender;

### 5.1.1 HOME PAGE

The home page of the prototype is simply a descriptive page of the existing contents in the web application. It resumes the concept behind the application, what the user will find in each of the navigation tabs, the existing features and some information about it was developed.

The Network Explorer contains the sections where the user can explore the Twitter network. For the sake of user experience the Network Explorer was divided into three categorical subsections, one

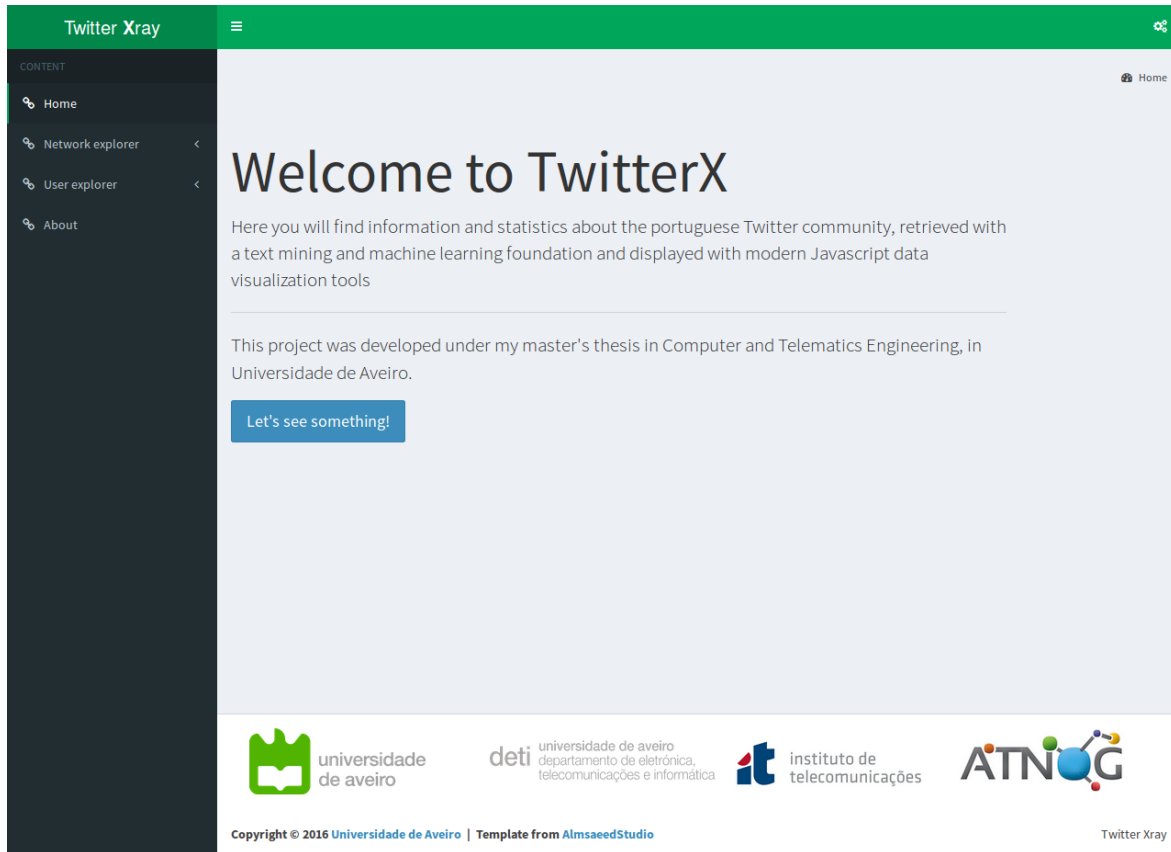


Figure 5.1: Prototype home page

that deals with activity, one with influence and another with content. The User Explorer encompasses two different tabs, the profiling section where resides all the information about a singular user and the gender classification where it is possible to test the developed gender classifier has a single component.

The end user can navigate through these subsections using the sidebar on the left.

## 5.1.2 NETWORK EXPLORER - ACTIVITY

### TWITTER ACTIVITY BY MONTH

By opening the Activity page, the user is confronted at first with a calendar heat map of the Twitter activity. This representation has the advantage of enhancing intuitive visual comparisons over a different time periods, through the use of color shades.

Having decided to use a calendar heat map to represent Twitter activity, two of the goals for the Network Scope information are met, since it enables the user to compare Twitter activity on monthly and daily basis at the same time. Note that there certain days that appear blank as if no activity occurred in that period. This happens due to anomalies in the dataset, most probably caused by errors in the TVPulse's data collector.

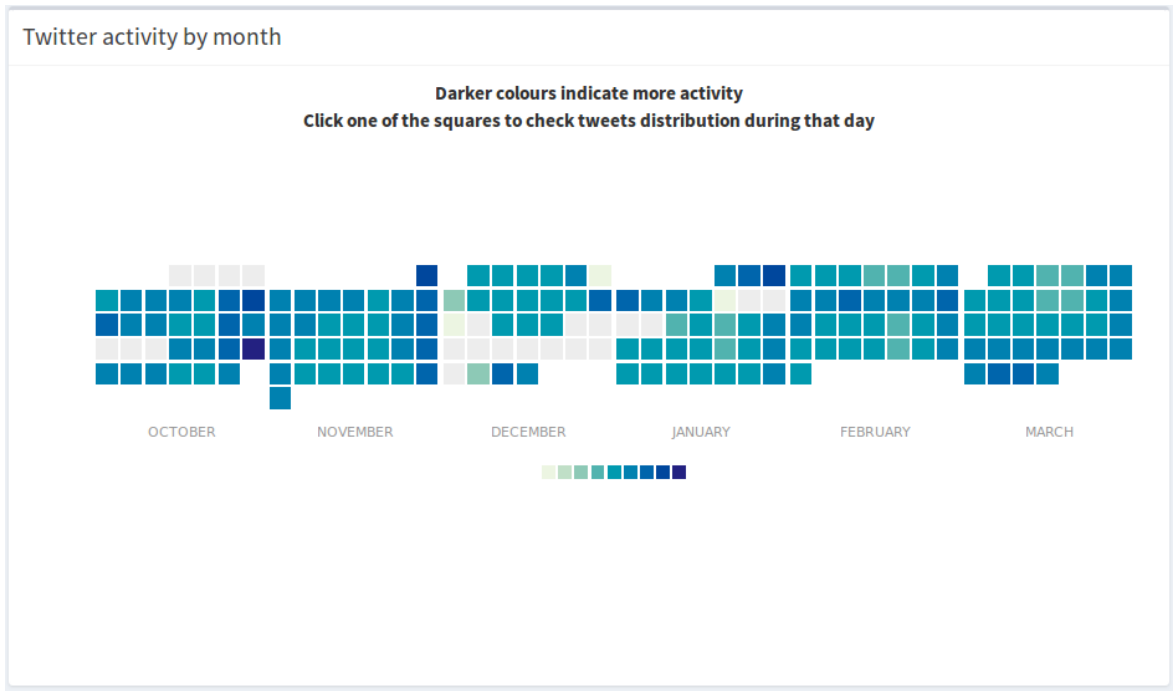


Figure 5.2: Network activity by month

#### TWITTER ACTIVITY BY HOUR

The calendar heat map has a label indicating that by clicking on the square of a determined day it is possible to check the activity in that day in detail. To display the Twitter activity on an hour granularity, it was used a line chart where the Y-axis is the number of tweets and the X-axis the hours of the day. For context, it was also calculated the mean counts each hour has for the whole days in the dataset, to be able to identify unusual Twitter activity.

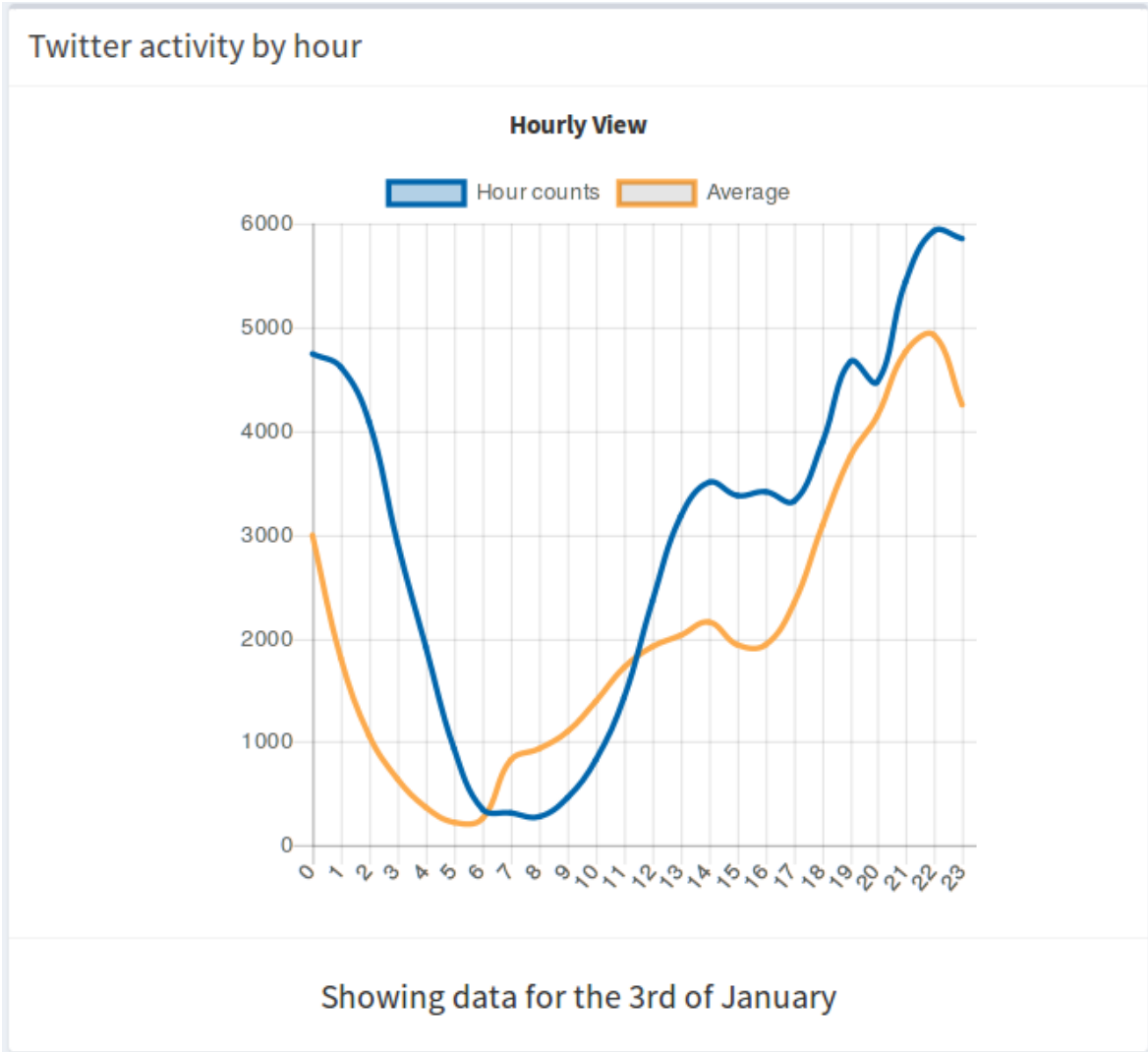


Figure 5.3: Network activity by hours

TWITTER ACTIVITY BY LOCATION

To represent how Twitter activity is distributed over the Portuguese territory it was decided to use an interactive map that discriminates activity on each Portuguese district using color shades. This metric can be browsed through every month available in the data set. To complete this section it was calculated which are the most globally active and inactive districts.

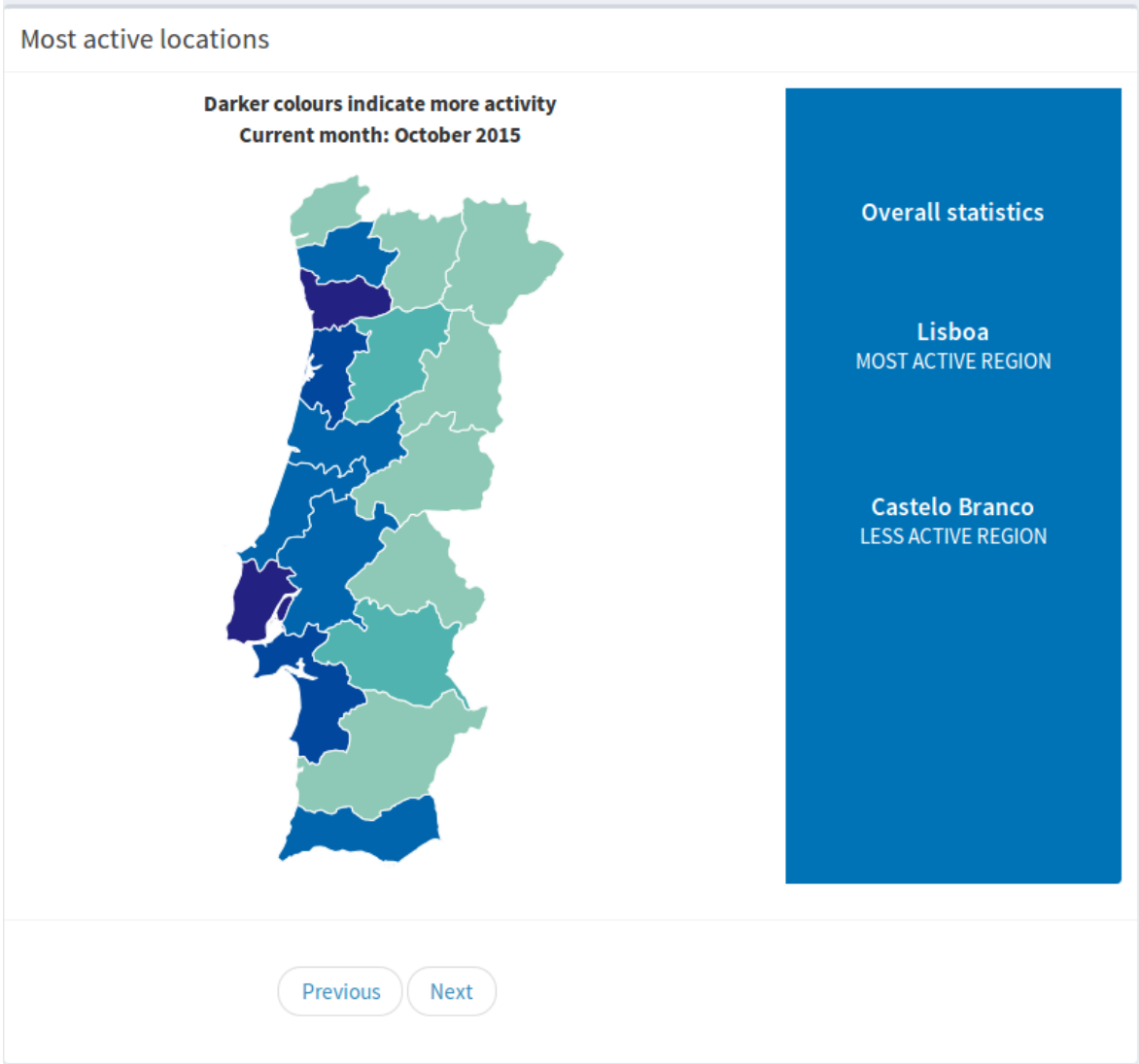


Figure 5.4: Network activity by locations

## TWITTER GENDER AND SOURCE DISTRIBUTIONS

Pie charts were the chosen visualization method to represent distributions throughout the system. In the Network Activity they are used to show how many users in the data set, according to results provided by the developed classifier, are male or female. Along with this, it is also used to show how the tools used to post tweets are distributed in percentage.

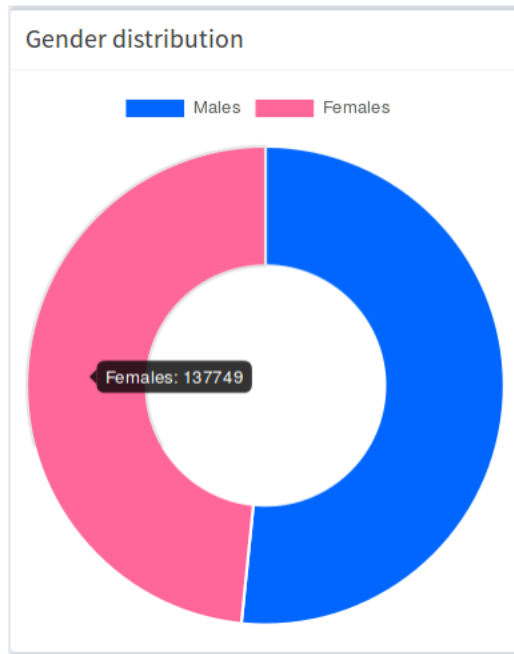


Figure 5.5: Gender distribution

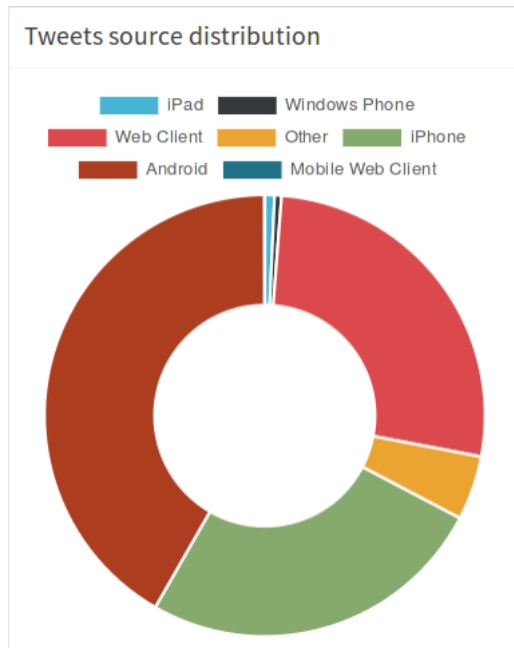


Figure 5.6: Source distribution



### 5.1.3 NETWORK EXPLORER - INFLUENCE

The Influence tab displays a ranked list of the most influential Twitter users in the data set. The process for measuring influence is explained in the previous chapter. This list is paginated, contains ten elements per page and it was developed using the Django Template API.

#### FILTERS

To provide the end-user some freedom to explore this list, two filters are available. It is possible to filter the users in list by native language, to make it exclusively Portuguese, and by gender.

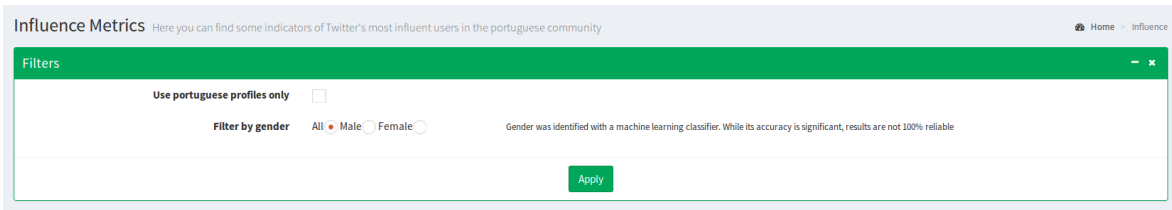


Figure 5.7: Filters for the influence rank

#### DETAILS VIEW

While consulting the Influence page it is possible to access some details about the user profile by clicking the "View Details" button. This detailed view presents some information about the user's profile data and an hyperlink to his User Explorer view.

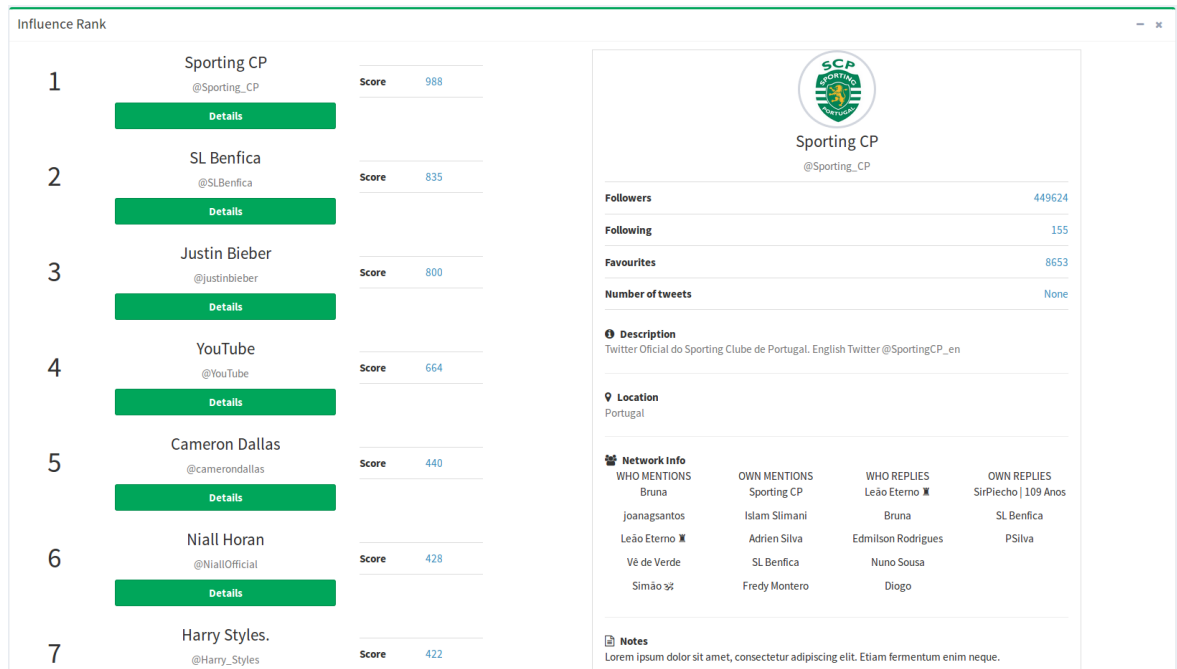


Figure 5.8: Influence rank

## 5.1.4 NETWORK EXPLORER - CONTENT

The content section of the Network Explorer displays a scatter plot of the projected two dimensional cosine similarity distances, where each point in the plot can be one of three colours, each representative of the cluster the K-means algorithm assigned it. The points in the plot can be hovered, displaying basic information about the user they represent. It is also possible to visualize the most relevant terms in each cluster.

Twitter Network Clusters

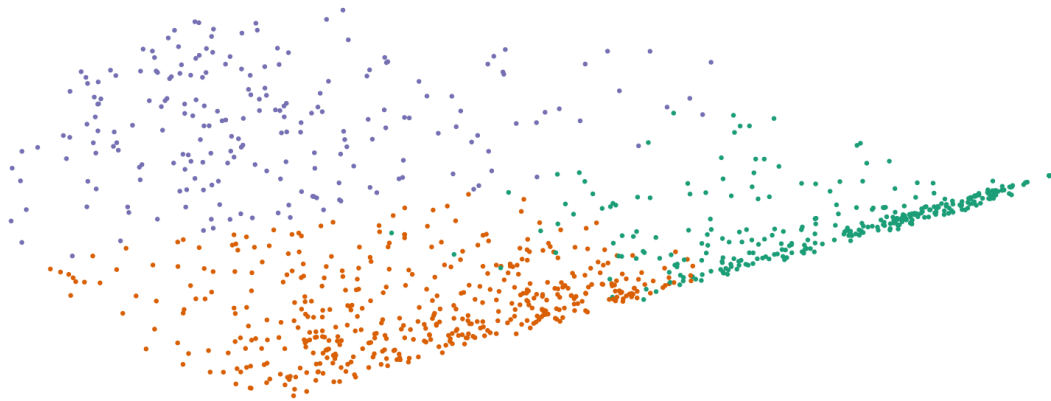


Figure 5.9: Graphical representation of the uncovered clusters

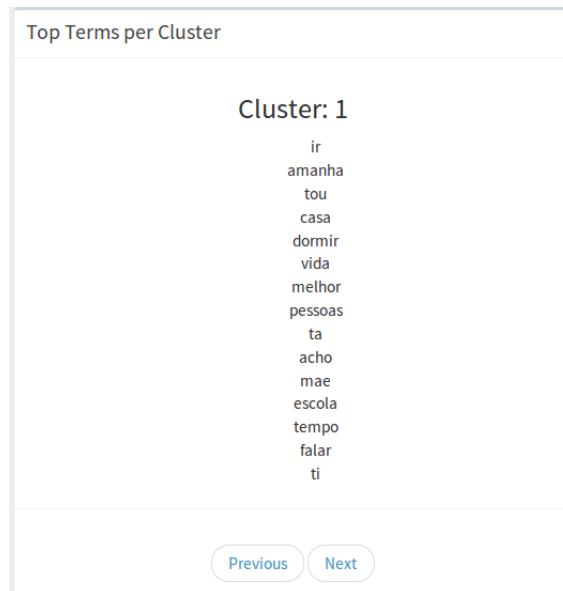
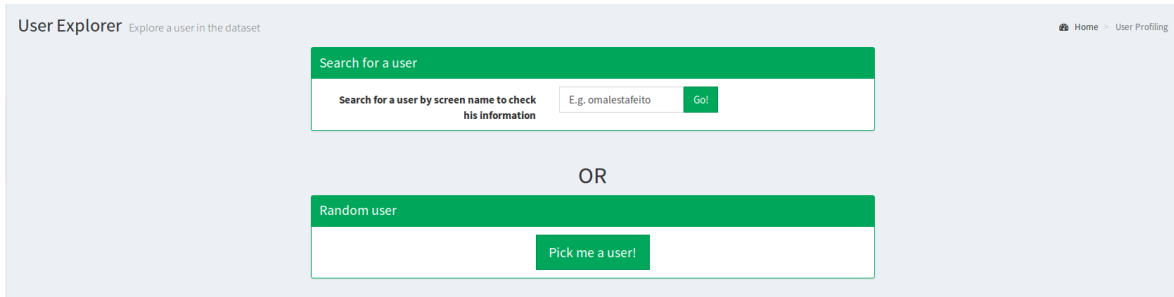


Figure 5.10: Most relevant terms for cluster 1

## 5.1.5 USER EXPLORER - PROFILING

The profiling section contains all the individual information that has been gathered about a single user in the data set. When the application end user navigates to this section, it is presented with a page where he can choose the user he wants to consult by writing his screen name or ask the application to present one for him randomly. He can also navigate to the profiling section through the influence list available on the Network Explorer section.



The screenshot shows the 'User Explorer' interface. At the top left, it says 'User Explorer Explore a user in the dataset'. On the right, there are navigation links for 'Home' and 'User Profiling'. The main content area has two sections. The first is 'Search for a user' with a sub-header 'Search for a user by screen name to check his information'. It features a text input field containing 'E.g. omalestafeito' and a green 'Go!' button. Below this is an 'OR' separator. The second section is 'Random user' with a green 'Pick me a user!' button.

Figure 5.11: Find user to profile form

## PROFILE INFORMATION

The first information presented when in the profiling section is the profile information extracted directly from the user's account. This metrics are available in the database as they were extracted as a part of the migration process. However, there are cases while migrating where this information was not available (see section 4.1.3). To overcome this, the Twitter API was used to retrieve this information in real-time. The user profile is then updated in the database with this new data. This is also used as an update method to all users in system, since this information is volatile and can change with the user's recent activity on Twitter.

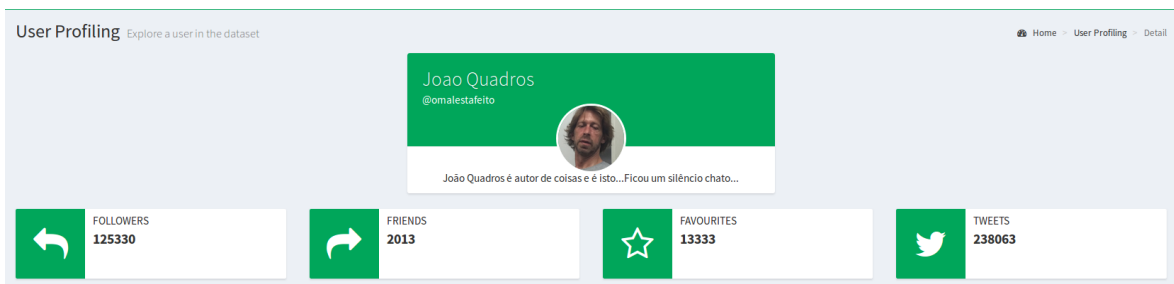


Figure 5.12: User profile information

## USER ACTIVITY

The second content section in this page is related to user activity. To illustrate how active the user has been over the time period that is present in this data set it was used a Line Chart. At a first instance, it was considered using a calendar heat map representation again but since not every Twitter user is active on a daily basis, that would lead to a lot of cases where existed a considerable amount of

empty slots in the map which could make it not very appealing. Through a Line Chart, those cases are mitigated while still providing an intuitive illustration of the evolution of user activity over time. To complement this information, it was calculated which day of the week concentrates most of activity and the opposite. As part of this section it can also be found the source distribution of posted tweets for this user.

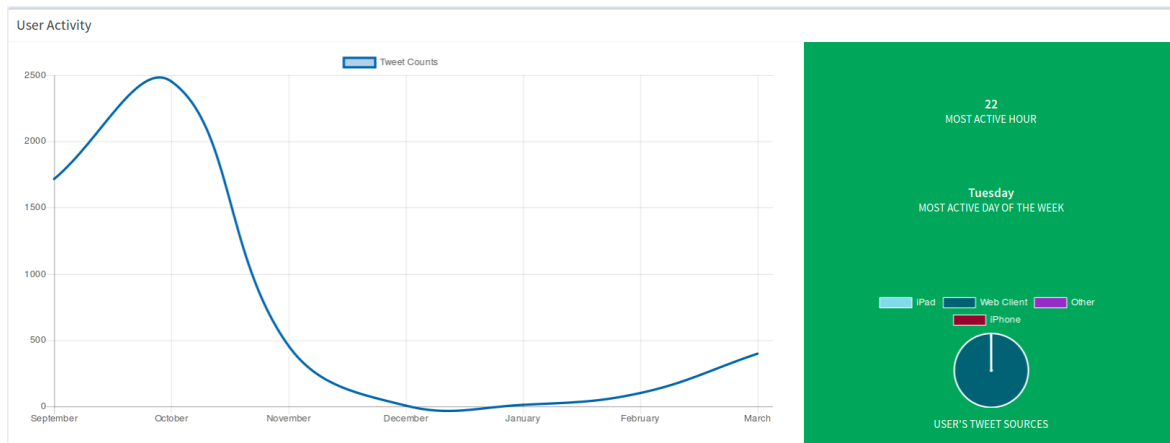


Figure 5.13: User activity

## USER NETWORK

The following section concentrates information about the user's interactions in the network. The first piece of information presented is a table consisting of the most interacted users in the profiled user's network. It is possible to find the top five user's that: mention this user more times, reply to this user more times and the reverse.

WHO MENTIONS	OWN MENTIONS	WHO REPLIES	OWN REPLIES
O@K	bifeahcasa	Rui	O@K
Rui	Rui	Gajo Desempregado	Rui
Madalena Martins God	Gajo Desempregado	bifeahcasa	Madalena Martins God
maria alves	Pedro Goulão	JoãoMiranda	Pureza
Pureza	Filipe	Pedro Goulão	João Luz 236

Figure 5.14: User network interaction

Also, there are two different interactive boxes as part of this section. The first is a graph layout with the ten user's that are considered the most similar to the target user in terms of posted content. Although this could be considered content information, visually it makes more sense to be included in the network section, as it contains information about other user's in the network. When hovered, the nodes in the graph layout provide some basic profile information about the user they represent.

The second box was created to represent distance between users. It starts as an empty box with an input bar where it is asked to insert the screen name of a user at choice. After the user submits the desired screen name, if that user is present in the data set and a path in the graph exists from the target user to the searched user, a graph is constructed in the interface with the nodes of the users present in the identified path. Like in in the previous box, the nodes present information about the entities they symbolize when hovered.

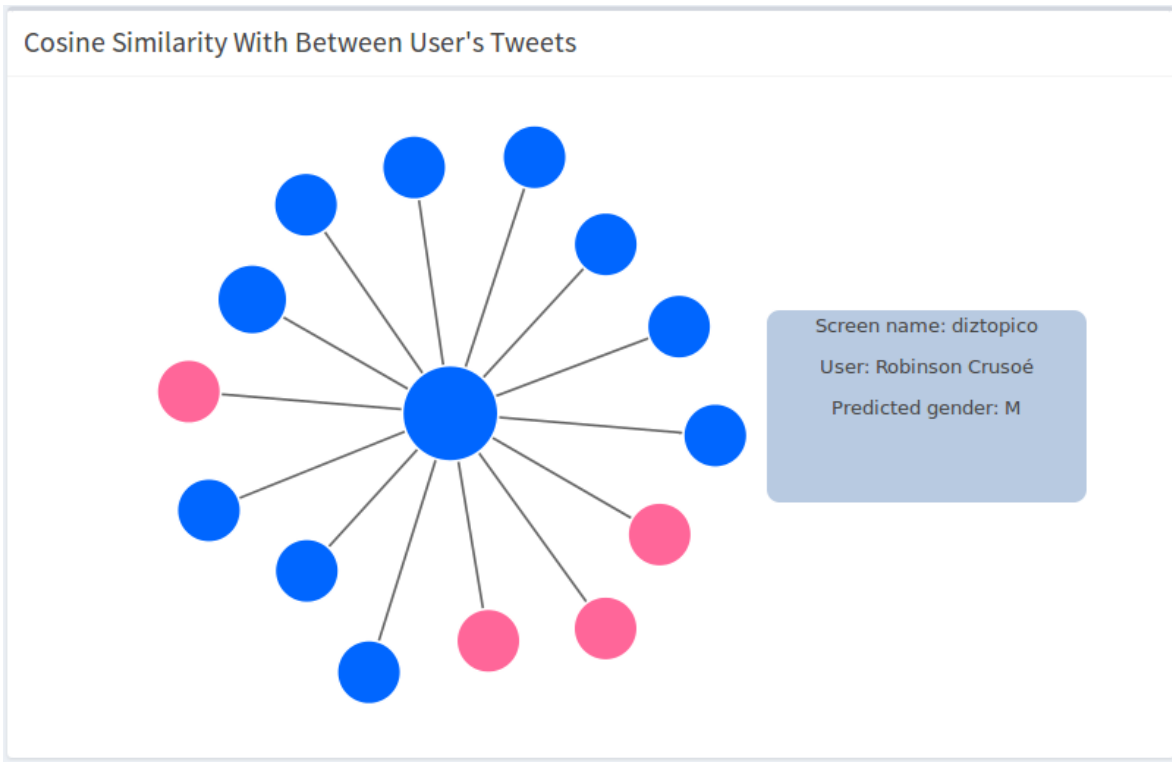


Figure 5.15: User similarity graph

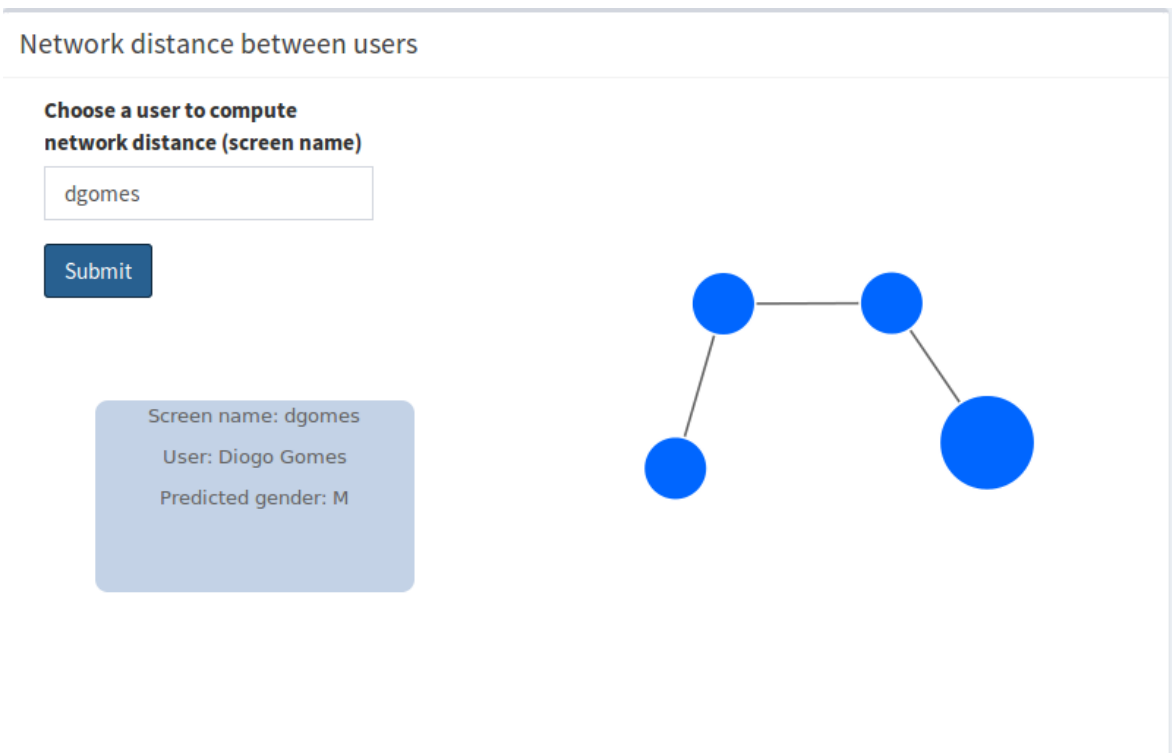


Figure 5.16: User distance graph

## USER CONTENT

The last section in this page is dedicated to user content. In it are found two visual representations of text data, that are based on two different text analysis algorithms. This first one is word cloud of the most frequent terms used by the user. The more frequent the terms are, the more relevancy they have in the portrayed term cloud. Here's an example of the generated term cloud for a user.

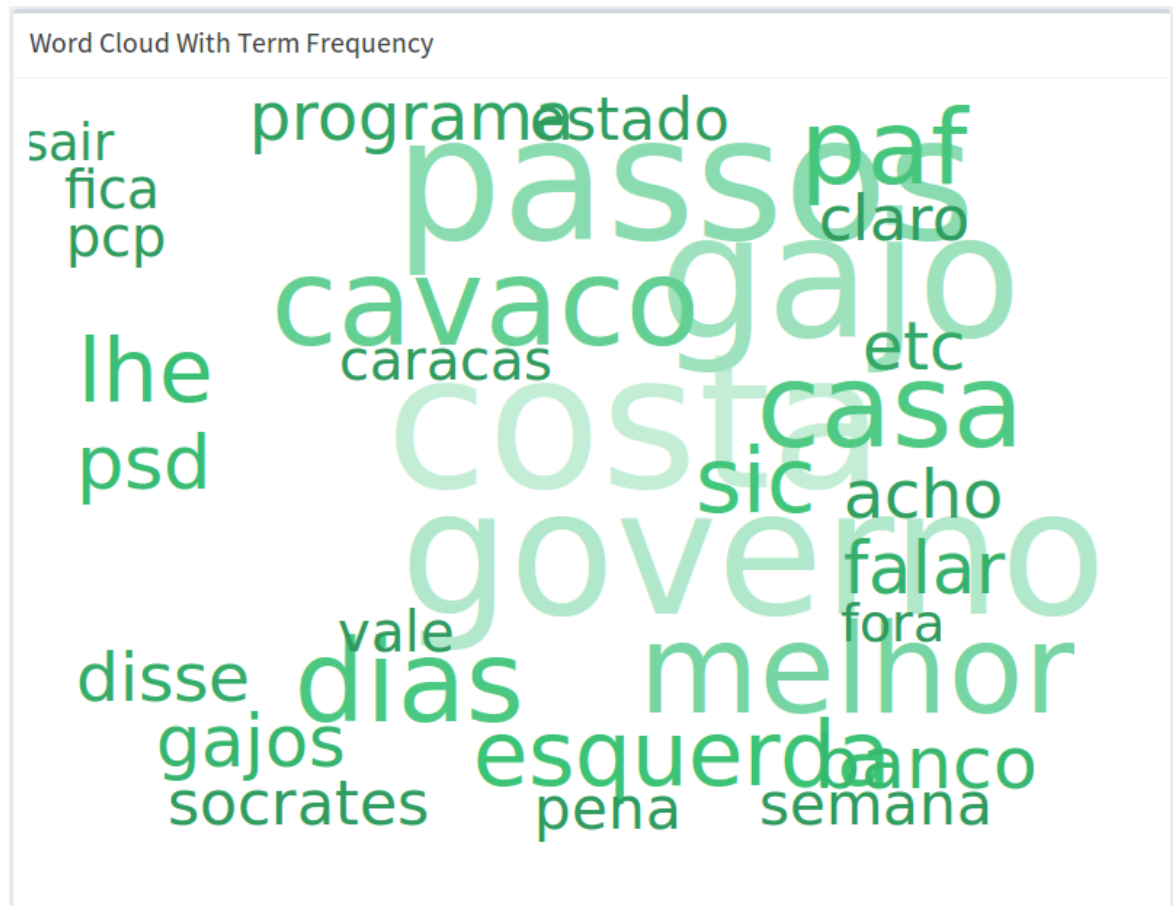


Figure 5.17: User term frequency as a word cloud

The second representation intends to be a visual example of the results obtained by applying LDA to the user's tweets, hence obtaining an indications of his topics of interest. To accomplish this, the most important 15 terms in each of the five topics, calculated when generating each user's Document, form five conglomerates of nodes with the corresponding term displayed over them. The cluster nodes are also distinguished by color, making it more intuitive that they represent a topic of interest.

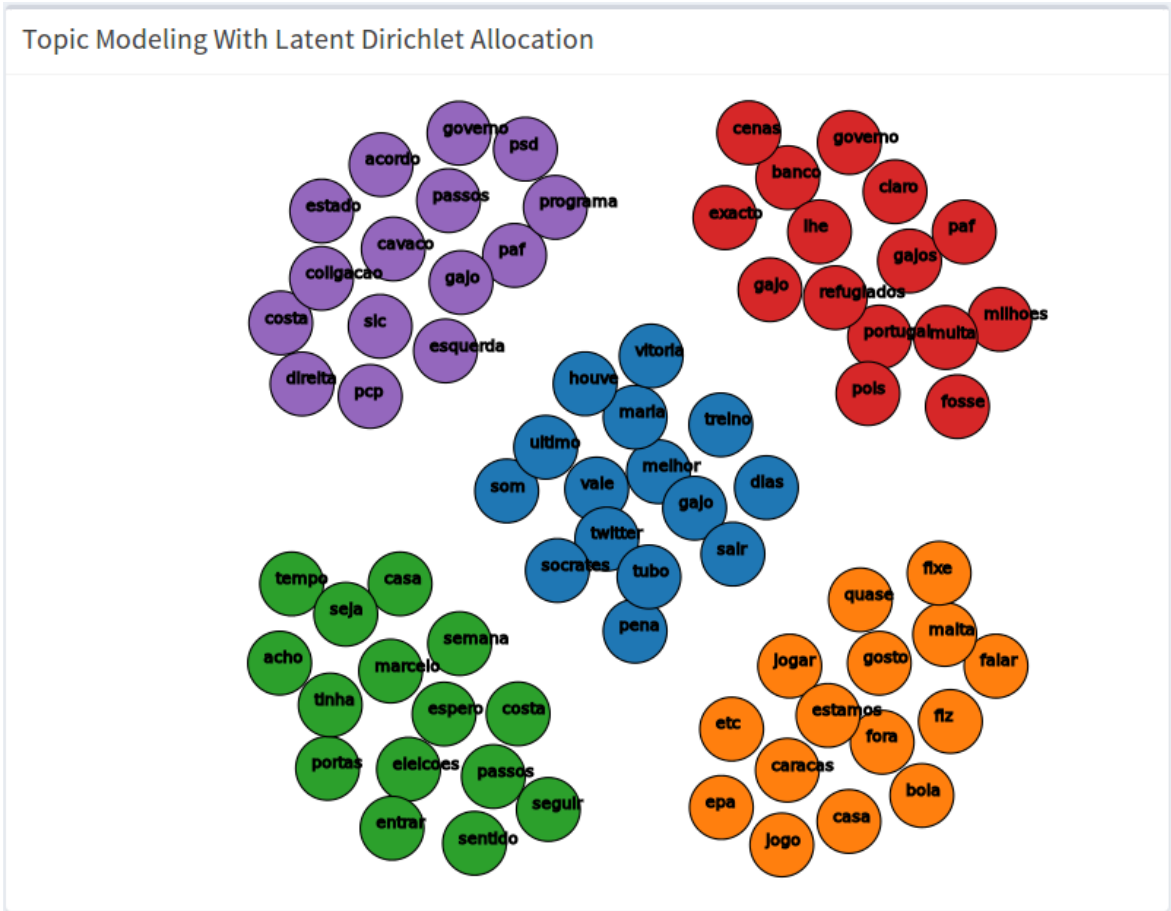


Figure 5.18: User topic modeling as a graph layout

### 5.1.6 GENDER CLASSIFICATION

The last tab of the User Explorer was developed exclusively to demonstrate the gender classifier used to classify the data set as male or female. Since this is an external component to the system and can be used in other data analysis tasks, this page was created so that it could be seen working and tested. To do this, it is given the possibility to search for any user in Twitter by screen name. Using the Twitter API, information about the user profile is retrieved and displayed on the page. After confirming that the retrieved user is the one aimed to classify, the user's name, description and screen name are sent to the server, processed and the result of classification is displayed on the page.

**Classify User**

**User ID**

123123123

Please fill at least one of the fields

**Screen name**

Migvicente

**Find user**

Figure 5.19: Find user for classification form

Miguel Vicente  
Migvicente

Is this the user you we're looking for?

Yes! Nope

Result

This user is

**Male**

Figure 5.20: Classification results

## GITHUB REPOSITORY

The developed classifier was made available at Github on the domain <https://github.com/mvicente93/twitter-gen-classifier-pt>. There it is possible to find the training algorithm used and configure it with new specifications. It is also possible find a serialized model of the classifier that can be loaded using Scikit-Learn and used in other projects as well as the data sets used for training and testing.

## 5.2 API

The prototype uses a REST API for communication between Frontend and Backend components that is available for external usage. The design of the API is expressed in section 3.3.3, although the



final result contains some extra methods used to fetch information that was not initially foreseen as necessary. The resulting API documentation is available on the web application and in Appendix-C section 6.2 there are some examples of API methods and their respective documentation.

### 5.3 BENCHMARKING AND PERFORMANCE

In software engineering it is very important that the developed algorithms and system architectures execute properly and in adequate time periods. When dealing with applications that make use of intense computational efforts over considerable amounts of data, it is decisive that the algorithms and components involved in the execution of these tasks live up to the need of doing it fast. To this extent, the developed application was submitted to a series of benchmarking tests. It was chosen to submit to benchmarking the storage component of the system, the Neo4j database.

The need to benchmark Neo4j resulted from the intensive use that is made of Cypher. Most of the developed API methods involve some sort of querying to the stored data model and it is relevant to understand which of these queries execute smoothly and in a consistent time period. Neo4j has an integrated profiling mechanism accessible through its web interface that allows to analyse Cypher queries in terms of time of execution and database hits necessary to produce the requested results. An example of the Neo4j profiling interface can be found in the following figure:

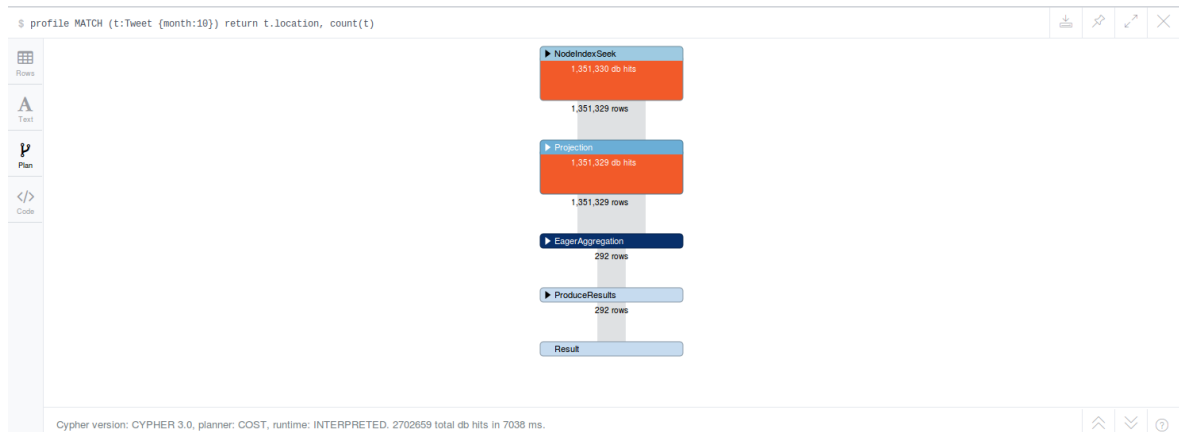


Figure 5.21: Neo4j Profile Interface

The database queries used in the developed API were submitted to a profile run as they are presented in Chapter 4. The machine that was running Neo4j was a virtual machine deployed in an OpenStack instance with two virtual CPUs and eight gigabytes of memory. When profiling queries that regarding a single user, the user picked had 5725 tweets posted and was one of the most influential according to the obtained influence rank. The obtained results are presented as follows.

Table 5.1: Benchmarking of Neo4j queries

Query	Parameters	Database Hits	Time to execute (ms)
Number of tweets in a month	month: 10	1351330	3056
Number of tweets for each day in a month ordered by day	month: 10	2702659	6273
Number of tweets in a day	month: 10 day: 5	722557	1273
Number of tweets by hour in a day ordered by hour	month: 10 day: 5	772416	1797
All tweet locations counts	-	22891545	44877
Tweet locations counts in a month	month: 10	2702659	7014
All tweet source counts	-	31416790	90721
Calculate and write influence score to users	-	113691425	374492
List top 100 users by influence score	-	569403	8471
User activity over months	user_id: 19247625	17178	118
User activity in a month	user_id: 19247625 month: 10	17178	116
User activity for all days in a month	user_id: 19247625 month: 10	14490	157
User activity in a day	user_id: 19247625 month: 10 day: 5	19633	100
User activity for all hours in a day	user_id: 19247625 month: 10 day: 5	14490	97
User activity sources	user_id:19247625	17178	243
User top 5 mentions	user_id:19247625	6816	173
User top 5 repliers	user_id:19247625	20998	545
User distance (4 hops difference)	user_id:1924762 target_id: 12429652	5	800

Looking at these results it is possible to observe that Neo4j does a good job on database querying, considering that the computational resources it had available were inferior to the ones recommended when dealing with this amount of data. This effect is particularly noticeable in operations that involve most tweets entities, which are incredibly superior in number to user entities. The overall performance when handling user entities is within what is expected in a standard DBMS.

## 5.4 USER INTERFACE EVALUATION

To validate the developed prototype in terms of usability and experience, as well as the adequacy of the visual representations to the functionality offered, it was chosen to submit the prototype to a series of usability tests. This was made to have a objective measure of how the developed user interface fulfilled the needs and difficulties found by a common user.

The first evaluation made was a subjective one, at the early stage of development. During Students@Deti, an event organized at University of Aveiro where master’s students present their dissertation work, this prototype was exposed and some feedback was collected regarding user interface and functionality. Although this consisted of an informal evaluation, it was useful to validate some of the already picked representations, like the activity heat map, the activity by locations map and the word cloud for user content. This representations received positive critics when shown during this event.

At the last stage of development, an enquiry was developed to proceed to a more formal and practical evaluation of user experience. This enquiry consisted of a series of tasks to be executed in the web application. The enquired users would then answer to some questions related to the tasks

they had to perform. The questions focused on getting the time to execute a certain task, rating of the adequacy of graphical representations and the compliance of the prototype with Jakob Nielsen's Usability Heuristics for User Interface Design[95], a method commonly used to identify design problems in an application's human interface. Unfortunately, while the enquiry was created and it is available in Appendix-D, due to lack of time and deployment restrictions of the prototype it was not possible to assemble an organized test session with a considerable amount of users to try the application.



# CONCLUSION

---

*This chapter concludes the presentation of developed work for this dissertation, while discussing the obtained results and what work remains to be done to improve this project.*

## 6.1 FINAL CONSIDERATIONS

Data mining research in social media environments is evolving at a sustained pace since the astonishing popularity growth of social media platforms. The amount, structure (or lack of) and properties of social media data pose new challenges everyday, which requires new approaches and solutions to effectively handle these issues.

This work had the ambition to create a robust visualization platform for the characterization and profiling of Portuguese Twitters users, using reliable data mining and machine learning techniques over social media data and a storage system with an innovative data model as support. A data set consisting of tweet objects from the Twitter API capture in an almost continuous time interval of four months served as ground data for the whole development life-cycle, since the projection of system goals, to the modeling and creation of the final data model. The final result consists of a web-based platform that met the desired information requirements and presented them in a visually appealing and interactive fashion.

To develop this prototype it was necessary a lot of background investigation. It was imperative to understand the process of transforming brute data into palpable results, the current state of the art in trending topics in data mining over social media like topic modeling, influence measurement and latent attribute classification, the techniques used to produce meaningful information in each of these areas and how they would commute with this work's objectives and resources.

A solid understanding of the data that was available was crucial to the success of this work. A careful analysis of the existing properties in tweet objects led to the realisation that there were inherent links between entities that could be leveraged for a better understanding of the Twitter network. A property graph data model was implemented resorting to the graph database Neo4j, making it possible to embrace the links in data and use them for a deeper analysis of Twitter users. This database was also used as the storage component in the web application.

While Neo4j was useful for exploring data connections and as database system, some limitations performance-wise were uncovered when the whole data set was loaded into it. The community version, available for personal and academic research, is only possible to use as single instance which results in consuming a lot of memory to handle querying on large amounts of nodes and relationships in a legitimate execution time. It also proved ineffective to do perform text analysis although this is understandable as it is not its focus. Overall, balancing its positives and negatives Neo4j proved to be a good choice to handle the system's storage requirements and made a strong case for graph database systems as a viable choice to tackle issues related with data connectivity. To my best knowledge, there is not any similar system available in scientific community that is built on a property graph modeled after the Twitter network.

The proposed goals for user characterization were all met. The developed prototype allows the end user to explore a part of the Twitter network by itself, with or without focus on a single user. The produced visualization schemes are engaging, allow interactivity and provide faithful representations of the information they use. The system makes use of a REST API that is publicly available and carefully documented for future research on this data set. The same applies to the model used for gender classification.

Overall, this work produced valuable research in a social media platform well known to the public. The focus on the Portuguese community makes it quite singular, as most of the studies made on it are usually focused on a limited set of users and on content rather than the network and its entities.

## 6.2 FUTURE WORK

The vastness of possibilities that social media data provides makes this dissertation an unfinished work, even though the specified requirements were met. There is room for improvement of the work that is already done and for new features.

In terms of non-functional details, the first issue to solve is improving the performance of some of its operations. Some queries used in Neo4j are very time-consuming due to the fact the server where it is allocated does not possess the recommended amount memory that the number of nodes and relationships currently stored requires. The developed user interface also has room for improvement. The realization of the user experience enquiry should be a priority as it will identify eventual design imperfections and even provide crucial feedback about how users feel regarding the application's usability.

The existing features in the prototype also have margin to be improved. The features dealing with content (similarity and topic modeling) should have an exploratory highlighting interface of the user's tweets, to help visualizing what led to similarity scores and topics extracted.

When it comes to new functionality there are also some interesting features that could be added. The biggest priority would be to incorporate real-time collection of tweets to keep the data set updated. This system relies on a static data set and proves that it is enough to gather interesting information from the Twitter network. Therefore adding real-time captures would an incredible upgrade to the system, although it will also require improvements in terms of infrastructure to support the huge amounts of data that will be captured and processed.

# REFERENCES

---

- [1] A. M. Kaplan and M. Haenlein, “Users of the world, unite! the challenges and opportunities of social media”, *Business Horizons*, vol. 53, no. 1, pp. 59–68, Jan. 2010, ISSN: 00076813. DOI: 10.1016/j.bushor.2009.09.003.
- [2] T. Aichner and F. Jacob, “Measuring the degree of corporate social media use”, *International Journal of Market Research*, vol. 57, no. 2, pp. 257–275, 2015, ISSN: 1470-7853. DOI: 10.2501/IJMR-2015-018.
- [3] M. D. Conover, B. Gonçalves, J. Ratkiewicz, A. Flammini, and F. Menczer, “Predicting the political alignment of twitter users”, *Proceedings - 2011 IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011*, pp. 192–199, 2011, ISSN: 1457719312. DOI: 10.1109/PASSAT/SocialCom.2011.34.
- [4] L. Dey, S. M. Haque, A. Khurdiya, and G. Shroff, “Acquiring competitive intelligence from social media”, in *Proceedings of the 2011 Joint Workshop on Multilingual OCR and Analytics for Noisy Unstructured Text Data*, 2011, pp. 1–9, ISBN: 9781450306850. DOI: 10.1145/2034617.2034621. [Online]. Available: <http://dl.acm.org/citation.cfm?doid=2034617.2034621>.
- [5] R. Buettner, “Getting a job via career-oriented social networking sites : the weakness of ties”, *HICSS-49 Proceedings: 49th Hawaii International Conference on System Sciences*, Jan. 2016. DOI: 10.13140/RG.2.1.3249.2241.
- [6] Statista. (2016). Number of daily active facebook users worldwide as of 2nd quarter 2016 (in millions), [Online]. Available: <http://www.statista.com/statistics/346167/facebook-global-dau/> (visited on 08/04/2016).
- [7] —, (2016). Number of monthly active instagram users from january 2013 to june 2016 (in millions), [Online]. Available: <http://www.statista.com/statistics/253577/number-of-monthly-active-instagram-users/> (visited on 08/04/2016).
- [8] —, (2016). Number of monthly active twitter users worldwide from 1st quarter 2010 to 2nd quarter 2016 (in millions), [Online]. Available: <http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/> (visited on 08/04/2016).
- [9] OECD, “Participative web and user-created content: web 2.0, wikis, and social networking. paris.” *Organisation for Economic Co-operation and Development.*, no. 2006, p. 74, 2007, ISSN: 9789264037465. DOI: 10.1787/9789264037472-en.
- [10] peterme. (2002). Thoughts, links, and essays from peter merholz, [Online]. Available: <http://www.peterme.com/archives/00000205.html> (visited on 08/04/2016).
- [11] A. M. Kaplan and M. Haenlein, “The early bird catches the news: nine things you should know about micro-blogging”, *Business Horizons*, vol. 54, no. 2, pp. 105–113, 2011, ISSN: 00076813. DOI: 10.1016/j.bushor.2010.09.004.

- [12] H. Kwak, C. Lee, H. Park, and S. Moon, “What is twitter , a social network or a news media?”, *The International World Wide Web Conference Committee (IW3C2)*, pp. 1–10, 2010, ISSN: 1932-8036. DOI: 10.1145/1772690.1772751. arXiv: 0809.1869v1.
- [13] M. Thelwall, K. Buckley, and G. Paltoglou, “Sentiment in twitter events”, *Journal of the American Society for Information Science and Technology*, vol. 62, no. 2, pp. 406–418, 2011, ISSN: 15322882. DOI: 10.1002/asi.21462. arXiv: 0803.1716.
- [14] D. Easley and J. Kleinberg, “Networks , crowds , and markets : reasoning about a highly connected world”, *Science*, vol. 81, p. 744, 2010, ISSN: 19467567. DOI: 10.1017/CB09780511761942. arXiv: 9809069v1 [arXiv:gr-qc].
- [15] A. Bifet and E. Frank, “Sentiment knowledge discovery in twitter streaming data”, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6332 LNAI, 2010, pp. 1–15, ISBN: 3642161839. DOI: 10.1007/978-3-642-16184-1{\\_}1.
- [16] R. Zafarani, M. A. Abbasi, and H. Liu, “Social media mining an introduction”, *Cambridge university Press*, p. 382, 2014, ISSN: 1882-0875. DOI: 10.1017/CB09781139088510. arXiv: arXiv:1011.1669v3.
- [17] P. Gundecha and H. Liu, “Mining social media: a brief introduction”, *Tutorials in Operations Research*, no. Dmml, pp. 1–17, 2012. DOI: <http://dx.doi.org/10.1287/educ.1120.0105>.
- [18] J. Han, J. Pei, and M. Kamber, *Data mining: Concepts and techniques*. Elsevier, 2011.
- [19] S. Chakrabarti, M. Ester, U. Fayyad, and J. Gehrke, “Data mining curriculum: a proposal, version 1.0 (2006)”, *Www.Kdd.Org/Curriculum/*, pp. 1–10, 2012.
- [20] I. H. Witten, E. Frank, and M. a. Hall, *Data Mining: Practical Machine Learning Tools and Techniques, Third Edition*, 2. 2011, vol. 54, p. 664, ISBN: 9780123748560. DOI: 10.1002/1521-3773(20010316)40:6<9823::AID-ANIE9823>3.3.CO;2-C.
- [21] U. Fayyad and R. Uthurusamy, “Data mining and knowledge discovery in databases”, *Communications of the ACM*, vol. 39, no. 11, pp. 24–26, Nov. 1996, ISSN: 00010782. DOI: 10.1145/240455.240463. arXiv: aimag.v17i3.1230.
- [22] D. Pyle, S. Editor, and D. D. Cerra, *Data Preparation for Data Mining*. 1999, vol. 17, pp. 375–381, ISBN: 4159822665. DOI: 10.1080/713827180.
- [23] J. Tang, Y. Chang, and H. Liu, “Mining social media with social theories”, *ACM SIGKDD Explorations Newsletter*, vol. 15, no. 2, pp. 20–29, 2014, ISSN: 19310145. DOI: 10.1145/2641190.2641195.
- [24] H. Müller and J.-c. Freytag, “Problems, methods, and challenges in comprehensive data cleansing”, *Challenges*, no. HUB-IB-164, pp. 1–23, 2003.
- [25] C.-M. Teng, “Correcting noisy data”, in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML ’99, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 239–248, ISBN: 1-55860-612-2.
- [26] J. E. Jackson, *A User’s Guide To Principal Components*, ISBN: 0471622672.
- [27] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers”, *Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory*, pp. 144–152, 1992, ISSN: 0-89791-497-X. DOI: 10.1.1.21.3818. arXiv: arXiv:1011.1669v3.
- [28] *Understanding Machine Learning: From Theory to Algorithms*. 2014, p. 409, ISBN: 1107057132. DOI: 10.1017/CB09781107298019.
- [29] C.-w. Hsu, C.-c. Chang, and C.-j. Lin, “A practical guide to support vector classification”, vol. 1, no. 1, pp. 1–16, 2016.



- [30] R. Pitre, U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, M. V. Chavan, and P. R. N. Phursule, “A survey paper on data mining with big data”, *AI magazine*, vol. 5, no. 3, pp. 37–53, 2014, ISSN: 0738-4602. DOI: 10.1609/aimag.v17i3.1230.
- [31] Z. Ghahramani, “Unsupervised learning”, in *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*, O. Bousquet, U. von Luxburg, and G. Rätsch, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 72–112, ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9\_5.
- [32] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey”, *ACM Comput. Surv.*, vol. 41, no. 3, 15:1–15:58, Jul. 2009, ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. [Online]. Available: <http://doi.acm.org/10.1145/1541880.1541882>.
- [33] I. Bruha and A. Famili, “Postprocessing in machine learning and data mining”, *ACM SIGKDD Explorations Newsletter*, vol. 2, no. 2, pp. 110–114, 2000, ISSN: 19310145. DOI: 10.1145/380995.381059.
- [34] L. Rokach, “Ensemble-based classifiers”, *Artificial Intelligence Review*, vol. 33, no. 1, pp. 1–39, 2010, ISSN: 1573-7462. DOI: 10.1007/s10462-009-9124-7.
- [35] S. Card, J. Mackinlay, and B. Shneiderman, “Readings in information visualization: using vision to think”, p. 712, 1999, ISSN: 19395108. DOI: 10.1002/wics.89.
- [36] D. A. Norman, *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993, ISBN: 0-201-62695-0.
- [37] E. E. Services, *Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data*. John Wiley & Sons, 2015.
- [38] J. Tang and H. Liu, “Feature selection with linked data in social media.”, *Sdm*, pp. 118–128, 2012. DOI: 10.1137/1.9781611972825.11.
- [39] “Linkage and autocorrelation cause feature selection bias in relational learning”, *Proceedings of the Nineteenth International Conference on Machine Learning (ICML2002)*, pp. 259–266, 2002.
- [40] M. Adedoyin-olowe, M. M. Gaber, and F. Stahl, “A survey of data mining techniques for social network analysis”, *International Journal of Research in Computer Engineering and Electronics*, vol. 3, no. 6, pp. 1–8, 2014. arXiv: 1312.4617.
- [41] G. Stringhini, C. Kruegel, and G. Vigna, “Detecting spammers on social networks”, pp. 1–9, 2010. DOI: 10.1145/1920261.1920263.
- [42] “More than words: social networks’ text mining for consumer brand sentiments”, *Expert Systems with Applications*, vol. 40, no. 10, pp. 4241–4251, 2013, ISSN: 09574174. DOI: 10.1016/j.eswa.2013.01.019.
- [43] M. Kosinski, D. Stillwell, and T. Graepel, “Private traits and attributes are predictable from digital records of human behavior.”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 110, no. 15, pp. 5802–5, 2013, ISSN: 1091-6490. DOI: 10.1073/pnas.1218772110.
- [44] D. Blei, L. Carin, and D. Dunson, “Probabilistic topic models”, *IEEE Signal Processing Magazine*, vol. 27, no. 6, pp. 55–65, 2010, ISSN: 10535888. DOI: 10.1109/MSP.2010.938079. arXiv: 1003.4916.
- [45] D. M. Blei, “Introduction to probabilistic topic modeling”, *Communications of the ACM*, vol. 55, pp. 77–84, 2012, ISSN: 00010782. DOI: 10.1145/2133806.2133826. arXiv: 1003.4916.

- [46] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation”, *Journal of Machine Learning Research*, vol. 3, no. 4-5, pp. 993–1022, 2003, ISSN: 15324435. DOI: 10.1162/jmlr.2003.3.4-5.993. arXiv: 1111.6189v1.
- [47] “Randomization tests for distinguishing social influence and homophily effects”, *The 19Th International Conference*, pp. 601–610, 2010, ISSN: 9781605587998. DOI: 10.1145/1772690.1772752.
- [48] M. McPherson, L. Smith-lovin, and J. M. Cook, “Birds of a feather : homophily in social networks”, *Annual Review of Sociology*, vol. 27, pp. 415–444, 2001, ISSN: 0360-0572. DOI: 10.1146/annurev.soc.27.1.415.
- [49] S. Aral, L. Muchnik, and A. Sundararajan, “Distinguishing influence-based contagion from homophily-driven diffusion in dynamic networks.”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 106, no. 51, pp. 21 544–21 549, 2009, ISSN: 0027-8424. DOI: 10.1073/pnas.0908800106. arXiv: arXiv:1408.1149.
- [50] D. F. Nettleton, *Data mining of social networks represented as graphs*, 2013.
- [51] M. Cha, H. Haddai, F. Benevenuto, and K. P. Gummadi, “Measuring user influence in twitter : the million follower fallacy”, *International AAAI Conference on Weblogs and Social Media*, pp. 10–17, 2010, ISSN: 1556-4029. DOI: 10.1.1.167.192.
- [52] E. Bakshy, J. Hofman, W. Mason, and D. Watts, “Everyone’s an influencer: quantifying influence on twitter”, *Proceedings of the fourth ACM international conference on Web search and data mining SE - WSDM ’11*, pp. 65–74, 2011, ISSN: 09312048. DOI: doi:10.1145/1935826.1935845. [Online]. Available: citeulike-article-id:8754779\$%5Cbackslash\$http://dx.doi.org/10.1145/1935826.1935845.
- [53] S. Brin and L. Page, “The anatomy of a large-scale hypertextual web search engine”, *Comput. Netw. ISDN Syst.*, vol. 30, no. 1-7, pp. 107–117, Apr. 1998, ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X. [Online]. Available: http://dx.doi.org/10.1016/S0169-7552(98)00110-X.
- [54] J. Weng, E. P. Lim, J. Jiang, and Q. He, “Twiterrank: finding topic-sensitive influential twitterers”, *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM 2010)*, pp. 261–270, 2010. DOI: 10.1145/1718487.1718520.
- [55] E. Caladoa and H. Sofia-Pintoa, “User profiling on twitter”, *Semantic-Web-Journal.Net*, pp. 1–15, 2011.
- [56] M. Gupta, R. Li, and K. C.-C. Chang, “Towards a social media analytics platform: event detection and user profiling for twitter”, in *WWW 2014 Companion: Proceedings of the 23rd International Conference on World Wide Web*, 2014, pp. 193–194.
- [57] R. Li, S. Wang, H. Deng, R. Wang, and K. C.-C. Chang, “Towards social user profiling: unified and discriminative influence model for inferring home locations”, *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1023–1031, 2012. DOI: 10.1145/2339530.2339692.
- [58] D. Rao, D. Yarowsky, A. Shreevats, and M. Gupta, “Classifying latent user attributes in twitter”, *Proceedings of the 2nd international workshop on Search and mining user-generated contents - SMUC ’10*, p. 37, 2010. DOI: 10.1145/1871985.1871993. arXiv: 1690219.1690245ããŒ.
- [59] J. D. Burger, J. Henderson, G. Kim, and G. Zarrella, “Discriminating gender on twitter”, *Association for Computational Linguistics*, vol. 146, pp. 1301–1309, 2011. DOI: 10.1007/s00256-005-0933-8.
- [60] A. Vilaça, M. Antunes, and D. Gomes, “Tvpulse: detecting tv highlights in social networks”, *Proc. 10th ConfTele 2015 - Conference on Telecommunications, Aveiro, Portugal*, no. August 2016, Sep. 2015.

- [61] P. Saleiro, S. Amir, M. Silva, and C. Soares, “Popmine: tracking political opinion on the web”, *Proceedings - 15th IEEE International Conference on Computer and Information Technology, CIT 2015, 14th IEEE International Conference on Ubiquitous Computing and Communications, IUCC 2015, 13th IEEE International Conference on Dependable, Autonomic and Secure Computing, DASC 2015 and 13th IEEE International Conference on Pervasive Intelligence and Computing, PICom 2015*, pp. 1521–1526, 2015. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.228. arXiv: 1511.09101.
- [62] J. Celko, *Graph Databases*. 2014, pp. 27–46, ISBN: 9780124071926. DOI: 10.1016/B978-0-12-407192-6.00003-0.
- [63] R. Angles and C. Gutierrez, “Survey of graph database models”, *ACM Computing Surveys*, vol. 40, no. 1, pp. 1–39, 2008, ISSN: 03600300. DOI: 10.1145/1322432.1322433.
- [64] A. Seaborne and G. Carothers, “RDF 1.1 n-triples”, W3C, W3C Recommendation, Feb. 2014, <http://www.w3.org/TR/2014/REC-n-triples-20140225/>.
- [65] Neo4j. (2016). Neo4j: The world’s leading graph database, [Online]. Available: <https://neo4j.com/> (visited on 08/16/2016).
- [66] —, (2016). Neo4j - customer success stories and case studies, [Online]. Available: <https://neo4j.com/customers> (visited on 08/16/2016).
- [67] DB-Engines. (2016). Db-engines ranking of graph dbms, [Online]. Available: <http://db-engines.com/en/ranking/graph+dbms> (visited on 08/16/2016).
- [68] Neo4j. (2016). Neo4j - documentation, [Online]. Available: <http://neo4j.com/docs/developer-manual/current/introduction/#graphdb-neo4j-schema> (visited on 08/16/2016).
- [69] R. V. Bruggen, *Learning Neo4j*. 2014, p. 220, ISBN: 978-1849517164.
- [70] OrientDB. (2016). Orientdb: Distributed multi-model graph/document database, [Online]. Available: <http://orientdb.com/orientdb/> (visited on 10/24/2016).
- [71] —, (2016). Orientdb clients -find out who’s using orientdb, [Online]. Available: <https://orientdb.com/customers> (visited on 10/24/2016).
- [72] —, (2016). Supported types - orientdb manual, [Online]. Available: <http://orientdb.com/docs/last/Types.html> (visited on 10/24/2016).
- [73] —, (2016). Schema - orientdb manual, [Online]. Available: <http://orientdb.com/docs/last/Schema.html> (visited on 10/24/2016).
- [74] —, (2016). Scaling - orientdb manual, [Online]. Available: <http://orientdb.com/docs/last/Distributed-Architecture.html> (visited on 10/24/2016).
- [75] <http://tinkerpop.apache.org/>. (2016). Apache tinkerpop, [Online]. Available: <http://apache.tinkerpop.org> (visited on 10/24/2016).
- [76] Titan. (2015). Titan: Distributed graph database, [Online]. Available: <http://titan.thinkaurelius.com> (visited on 10/24/2016).
- [77] —, (2015). Chapter 5. schema and data modeling, [Online]. Available: [http://s3.thinkaurelius.com/docs/titan/1.0.0/schema.html#\\_defining\\_property\\_keys](http://s3.thinkaurelius.com/docs/titan/1.0.0/schema.html#_defining_property_keys) (visited on 10/24/2016).
- [78] —, (2015). Chapter 2. architectural overview, [Online]. Available: <http://s3.thinkaurelius.com/docs/titan/1.0.0/arch-overview.html> (visited on 10/24/2016).
- [79] S. Beis, S. Papadopoulos, and Y. Kompatsiaris, “Benchmarking graph databases on the problem of community detection”, *Advances in Intelligent Systems and Computing*, vol. 312, pp. 3–14, 2015, ISSN: 21945357. DOI: 10.1007/978-3-319-10518-5\_1.

- [80] S. Jouili and V. Vansteenbergh, “An empirical comparison of graph databases”, *Proceedings - SocialCom/PASSAT/BigData/EconCom/BioMedCom 2013*, pp. 708–715, 2013. DOI: 10.1109/SocialCom.2013.106.
- [81] T. R. Foundation. (2016). R: The r project for statistical computing, [Online]. Available: <https://www.r-project.org/> (visited on 10/24/2016).
- [82] W. N. Venables, D. M. Smith, *et al.*, *An introduction to r*.
- [83] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: An update”, *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.
- [84] (2016). Numpy, [Online]. Available: <http://www.numpy.org/> (visited on 08/16/2016).
- [85] (2016). Scipy library, [Online]. Available: <http://www.scipy.org/scipylib/index.html> (visited on 08/16/2016).
- [86] (2016). Python data analysis library - pandas, [Online]. Available: <http://pandas.pydata.org/> (visited on 08/16/2016).
- [87] 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*, vol. 12, pp. 2825–2830, 2011.
- [88] S. Bird, “Nltk: The natural language toolkit”, in *Proceedings of the COLING/ACL on Interactive presentation sessions*, Association for Computational Linguistics, 2006, pp. 69–72.
- [89] (2016). Twitter api, [Online]. Available: <https://dev.twitter.com/overview/documentation> (visited on 08/16/2016).
- [90] (2016). Twitter api - rest api, [Online]. Available: <https://dev.twitter.com/rest/public> (visited on 08/16/2016).
- [91] (2016). Twitter api - streaming api, [Online]. Available: <https://dev.twitter.com/streaming/overview> (visited on 08/16/2016).
- [92] (2016). Apache cassandra, [Online]. Available: [cassandra.apache.org](http://cassandra.apache.org) (visited on 08/16/2016).
- [93] A. Rajaraman and J. D. Ullman, *Mining of Massive Datasets*: Cambridge: Cambridge University Press, Oct. 2011, ISBN: 9781139058452. DOI: 10.1017/CB09781139058452. [Online]. Available: <https://www.cambridge.org/core/books/mining-of-massive-datasets/A06D57FC616AE3FD10007D89E73F8B92>.
- [94] (2016). Django faq, [Online]. Available: <https://docs.djangoproject.com/en/1.10/faq/general/> (visited on 08/16/2016).
- [95] J. Nielsen, “Heuristic evaluation”, *Usability inspection methods*, vol. 17, no. 1, pp. 25–62, 1994.

# APPENDIX A: TWITTER API OBJECTS

---

---

```
{
  "contributors_enabled": false,
  "created_at": "Sat Dec 14 04:35:55 +0000 2013",
  "default_profile": false,
  "default_profile_image": false,
  "description": "Developer and Platform Relations @Twitter. We are developer
    advocates. We can't answer all your questions, but we listen to all of
    them!",
  "entities": {
    "description": {
      "urls": []
    },
    "url": {
      "urls": [
        {
          "display_url": "dev.twitter.com",
          "expanded_url": "https://dev.twitter.com/",
          "indices": [
            0,
            23
          ],
          "url": "https://t.co/66w26cua10"
        }
      ]
    }
  },
  "favourites_count": 757,
  "follow_request_sent": false,
  "followers_count": 143916,
  "following": false,
  "friends_count": 1484,
  "geo_enabled": true,
  "id": 2244994945,
  "id_str": "2244994945",
  "is_translation_enabled": false,
```

```

"is_translator": false,
"lang": "en",
"listed_count": 516,
"location": "Internet",
"name": "TwitterDev",
"notifications": false,
"profile_background_color": "FFFFFF",
"profile_background_image_url":
  "http://abs.twimg.com/images/themes/theme1/bg.png",
"profile_background_image_url_https":
  "https://abs.twimg.com/images/themes/theme1/bg.png",
"profile_background_tile": false,
"profile_banner_url":
  "https://pbs.twimg.com/profile_banners/2244994945/1396995246",
"profile_image_url":
  "http://pbs.twimg.com/profile_images/530814764687949824/npQQVvkq8_normal.png",
"profile_image_url_https":
  "https://pbs.twimg.com/profile_images/530814764687949824/npQQVvkq8_normal.png",
"profile_link_color": "0084B4",
"profile_location": null,
"profile_sidebar_border_color": "FFFFFF",
"profile_sidebar_fill_color": "DDEEF6",
"profile_text_color": "333333",
"profile_use_background_image": false,
"protected": false,
"screen_name": "TwitterDev",
"status": {
  "contributors": null,
  "coordinates": null,
  "created_at": "Fri Jun 12 19:50:18 +0000 2015",
  "entities": {
    "hashtags": [],
    "symbols": [],
    "urls": [
      {
        "display_url": "github.com/twitterdev/twi\u2026",
        "expanded_url": "https://github.com/twitterdev/twitter-for-bigquery",
        "indices": [
          36,
          59
        ],
        "url": "https://t.co/K5orgXzhOM"
      }
    ]
  },
  "user_mentions": [
    {
      "id": 18518601,
      "id_str": "18518601",
      "indices": [
        3,
        13
      ],
      "name": "William Vambenepe",
      "screen_name": "vambenepe"
    }
  ]
}

```

```

    }
  ]
},
"favorite_count": 0,
"favorited": false,
"geo": null,
"id": 609447655429787648,
"id_str": "609447655429787648",
"in_reply_to_screen_name": null,
"in_reply_to_status_id": null,
"in_reply_to_status_id_str": null,
"in_reply_to_user_id": null,
"in_reply_to_user_id_str": null,
"lang": "en",
"place": null,
"possibly_sensitive": false,
"retweet_count": 19,
"retweeted": false,
"retweeted_status": {
  "contributors": null,
  "coordinates": null,
  "created_at": "Fri Jun 12 05:19:11 +0000 2015",
  "entities": {
    "hashtags": [],
    "symbols": [],
    "urls": [
      {
        "display_url": "github.com/twitterdev/twi\u2026",
        "expanded_url":
          "https://github.com/twitterdev/twitter-for-bigquery",
        "indices": [
          21,
          44
        ],
        "url": "https://t.co/K5orgXzh0M"
      }
    ]
  },
  "user_mentions": []
},
"favorite_count": 23,
"favorited": false,
"geo": null,
"id": 609228428915552257,
"id_str": "609228428915552257",
"in_reply_to_screen_name": null,
"in_reply_to_status_id": null,
"in_reply_to_status_id_str": null,
"in_reply_to_user_id": null,
"in_reply_to_user_id_str": null,
"lang": "en",
"place": null,
"possibly_sensitive": false,
"retweet_count": 19,
"retweeted": false,

```

```

    "source": "<a href="//twitter.com/%5C%22" rel="\>Twitter Web
      Client</a>",
    "text": "Twitter for BigQuery https://t.co/K5orgXzh0M See how easy it is
      to stream Twitter data into BigQuery.",
    "truncated": false
  },
  "source": "<a href="//twitter.com/download/iphone%5C%22"
    rel="\>Twitter for iPhone</a>",
  "text": "RT @vambenepe: Twitter for BigQuery https://t.co/K5orgXzh0M See
    how easy it is to stream Twitter data into BigQuery.",
  "truncated": false
},
"statuses_count": 1279,
"time_zone": "Pacific Time (US & Canada)",
"url": "https://t.co/66w26cua10",
"utc_offset": -25200,
"verified": true
}

```

---

Listing 45: Example of a Twitter API User Object

---

```

{
  "contributors": null,
  "truncated": false,
  "text": "Many people have said I\u2019m the world\u2019s greatest writer of
    140 character sentences.",
  "is_quote_status": false,
  "in_reply_to_status_id": null,
  "id": 491324429184823296,
  "favorite_count": 1571,
  "source": "<a href=\"http://twitter.com\" rel=\">Twitter Web
    Client</a>",
  "retweeted": false,
  "coordinates": null,
  "entities": {
    "symbols": [],
    "user_mentions": [],
    "hashtags": [],
    "urls": []
  },
  "in_reply_to_screen_name": null,
  "in_reply_to_user_id": null,
  "retweet_count": 1724,
  "id_str": "491324429184823296",
  "favorited": false,
  "user": {
    "follow_request_sent": false,
    "has_extended_profile": false,
    "profile_use_background_image": true,
    "default_profile_image": false,
    "id": 25073877,

```



```

"profile_background_image_url_https":
  "https://pbs.twimg.com/profile_background_images/530021613/trump_scotland__43_of_70_c
"verified": true,
"profile_text_color": "333333",
"profile_image_url_https":
  "https://pbs.twimg.com/profile_images/1980294624/DJT_Headshot_V2_normal.jpg",
"profile_sidebar_fill_color": "C5CEC0",
"entities": {
  "url": {
    "urls": [
      {
        "url": "https://t.co/mZB2hymxC9",
        "indices": [
          0,
          23
        ],
        "expanded_url": "http://www.DonaldJTrump.com",
        "display_url": "DonaldJTrump.com"
      }
    ]
  },
  "description": {
    "urls": []
  }
},
"followers_count": 10957946,
"profile_sidebar_border_color": "BDDCAD",
"id_str": "25073877",
"profile_background_color": "6D5C18",
"listed_count": 37621,
"is_translation_enabled": true,
"utc_offset": -14400,
"statuses_count": 32928,
"description": "#TrumpPence16",
"friends_count": 45,
"location": "New York, NY",
"profile_link_color": "0D5B73",
"profile_image_url":
  "http://pbs.twimg.com/profile_images/1980294624/DJT_Headshot_V2_normal.jpg",
"following": false,
"geo_enabled": true,
"profile_banner_url":
  "https://pbs.twimg.com/profile_banners/25073877/1468988952",
"profile_background_image_url":
  "http://pbs.twimg.com/profile_background_images/530021613/trump_scotland__43_of_70_c
"screen_name": "realDonaldTrump",
"lang": "en",
"profile_background_tile": true,
"favourites_count": 36,
"name": "Donald J. Trump",
"notifications": false,
"url": "https://t.co/mZB2hymxC9",
"created_at": "Wed Mar 18 13:46:38 +0000 2009",
"contributors_enabled": false,

```

```
    "time_zone": "Eastern Time (US & Canada)",
    "protected": false,
    "default_profile": false,
    "is_translator": false
  },
  "geo": null,
  "in_reply_to_user_id_str": null,
  "lang": "en",
  "created_at": "Mon Jul 21 20:50:46 +0000 2014",
  "in_reply_to_status_id_str": null,
  "place": null
}
```

---

Listing 46: Example of a Twitter API Status Object

# APPENDIX B: GENDER CLASSIFIER

---

Listing 47: Classifier training script

---

```
from optparse import OptionParser
from sklearn.svm import LinearSVC
from sklearn import preprocessing
from sklearn.feature_extraction.text import CountVectorizer
from Utils.pre_processing import normalize, remove_links,
    remove_hashtags_and_mentions, prepare_char_ngram
from Utils.socling_features import extract_username_socling_features,
    extract_description_socling_features, extract_tweet_socling_features
from sklearn.externals import joblib
import numpy as np
import json

op = OptionParser()

op.add_option('--train_set',
              dest='train_set', type=str, default='Datasets/train_set.json',
              help='Path to training dataset file')

op.add_option('--test_set',
              dest='test_set', type=str, default='Datasets/test_set.json',
              help='Path to testing dataset file')

op.add_option('--name_socling',
              action='store_true', dest='name_socling', default=False,
              help='Use username socio-linguistic features')

op.add_option('--desc_socling',
              action='store_true', dest='desc_socling', default=False,
              help='Use description socio-linguistic features')

op.add_option('--tweet_socling',
              action='store_true', dest='tweet_socling', default=False,
              help='Use tweets socio-linguistic features')

op.add_option('--name_chars',
              dest='name_chars', type=str, default='',
              help='Load username char ngrams file')
```

```

op.add_option('--name_words',
              dest='name_words', type=str, default='',
              help='Load username word ngrams file')
op.add_option('--sc_chars',
              dest='sc_ngrams', type=str, default='',
              help='Load screen_name ngrams file')
op.add_option('--desc_chars',
              dest='desc_chars', type=str, default='',
              help='Load description char ngrams file')
op.add_option('--desc_words',
              dest='desc_words', type=str, default='',
              help='Load description word ngrams file')
op.add_option('--tweet_chars',
              dest='tweet_chars', type=str, default='',
              help='Load tweet char ngrams file')
op.add_option('--tweet_words',
              dest='tweet_words', type=str, default='',
              help='Load tweet word ngrams file')
op.add_option('--char_range',
              dest='char_range', type=int, default=5,
              help='Set char maximum ngram range')
op.add_option('--word_range',
              dest='word_range', type=int, default=2,
              help='Set word maximum ngram range')

#op.print_help()
#print

(opts, args) = op.parse_args()
if len(args) > 0:
    op.error('this script takes no arguments')
    op.print_help()
    exit()

train_set = []
test_set = []
try:
    train_set = json.loads(open(opts.train_set, 'rb').read())
    test_set = json.loads(open(opts.test_set, 'rb').read())
except IOError:
    print 'Dataset file not found'
    exit()

try:
    if opts.name_chars is not '':
        name_chars = json.loads(open(opts.name_chars, 'rb').read())
    if opts.name_words is not '':
        name_words = json.loads(open(opts.name_words, 'rb').read())
    if opts.sc_ngrams is not '':
        sc_ngrams = json.loads(open(opts.sc_ngrams, 'rb').read())
    if opts.desc_chars is not '':
        desc_chars = json.loads(open(opts.desc_chars, 'rb').read())
    if opts.desc_words is not '':
        desc_words = json.loads(open(opts.desc_words, 'rb').read())

```

```

    if opts.tweet_chars is not '':
        tweet_chars = json.loads(open(opts.tweet_chars, 'rb').read())
    if opts.tweet_words is not '':
        tweet_words = json.loads(open(opts.tweet_words, 'rb').read())
except IOError:
    print 'One of the ngrams file was not found'

X = []
targets = []
for user in train_set + test_set:

    result_vector = []
    if opts.name_socling:
        result_vector.append(extract_username_socling_features(user['name']))
    if opts.desc_socling:
        result_vector.append(extract_description_socling_features(user['description']))
    if opts.tweet_socling:
        user_tweets = user['tweets']

        tweet_socling = [0] * 9
        for t in user_tweets:
            temp_socling = extract_tweet_socling_features(t)

            for i in range(0, 9):
                if temp_socling[i] is 1:
                    tweet_socling[i] = 1

        result_vector.append(tweet_socling)

    if opts.name_chars is not '':
        count = CountVectorizer(analyzer='char', binary=False,
                                vocabulary=name_chars, ngram_range=(1, opts.char_range))
        result =
            count.fit_transform([normalize(user['name']).lower().strip()]).toarray().tolist()
        result_vector.append(result[0])

    if opts.name_words is not '':
        count = CountVectorizer(analyzer='word', binary=False,
                                vocabulary=name_words)
        result =
            count.fit_transform([normalize(user['name']).lower().strip()]).toarray().tolist()
        result_vector.append(result[0])

    if opts.sc_ngrams is not '':
        count = CountVectorizer(analyzer='char', binary=False,
                                vocabulary=sc_ngrams, ngram_range=(1, opts.char_range))
        result =
            count.fit_transform([normalize(user['screen_name']).lower().strip()]).toarray().t

    if opts.desc_chars is not '':
        count = CountVectorizer(analyzer='char', binary=False,
                                vocabulary=desc_chars, ngram_range=(1, opts.char_range))

```

```

    result =
        count.fit_transform([normalize(remove_links(remove_hashtags_and_mentions(user['desc
result_vector.append(result[0])

if opts.desc_words is not '':
    count = CountVectorizer(analyzer='word', binary=False,
        vocabulary=desc_words, ngram_range=(1, opts.word_range))
    result =
        count.fit_transform([normalize(remove_links(remove_hashtags_and_mentions(user['desc
result_vector.append(result[0])

if opts.tweet_chars is not '':
    count = CountVectorizer(analyzer='char', binary=False,
        vocabulary=tweet_chars, ngram_range=(1, opts.char_range))
    processed_tweets = map(lambda t:
        normalize(remove_links(remove_hashtags_and_mentions(prepare_char_ngram(t))))
        .lower(user['tweets']))
    result = count.fit_transform(processed_tweets).toarray().tolist()
    result_vector.append(np.sum(result, axis=0).tolist())

if opts.tweet_words is not '':
    count = CountVectorizer(analyzer='word', binary=False,
        vocabulary=tweet_words, ngram_range=(1, opts.word_range))
    processed_tweets = map(lambda t:
        normalize(remove_links(remove_hashtags_and_mentions(t)).lower().strip(),
        user['tweets']))
    result = count.fit_transform(processed_tweets).toarray().tolist()
    result_vector.append(np.sum(result, axis=0).tolist())

X.append([f for vector in result_vector for f in vector])
targets.append(user['gender'])

n_train_samples = len(train_set)/2
n_test_samples = len(test_set)/2

train_samples = X[:n_train_samples] + X[n_train_samples:len(train_set)]
train_targets = targets[:n_train_samples] +
    targets[n_train_samples:len(train_set)]

test_samples = X[len(train_set):len(train_set)+n_test_samples] +
    X[len(train_set)+n_test_samples:]
test_targets = targets[len(train_set):len(train_set)+n_test_samples] +
    targets[len(train_set)+n_test_samples:]

lsvm = LinearSVC(class_weight='balanced')
lsvm.fit_transform(preprocessing.normalize(train_samples), train_targets)

print 'LinearSVC score: %0.3f\n' %
    lsvm.score(preprocessing.normalize(test_samples), test_targets)

joblib.dump(lsvm, 'classifier.pkl')

```

---

Listing 48: List of gendered Portuguese nicknames and abbreviations

---

B ,F,  
Bia ,F,  
Bea ,F,  
Bel , ,  
Bernas ,M,  
Cat ,F,  
Cata ,F,  
Cate ,F,  
Caty ,F,  
Catt ,F,  
Chica ,F,  
Chico ,M,  
Cris , ,  
Di , ,  
Babi ,F,  
Biba ,F,  
Bibs ,F,  
Dani , ,  
Ed ,M,  
Edu ,M,  
Fi ,F,  
Jo , ,  
Joao ,M,  
Joca ,M,  
Jomi ,M,  
Johny ,M,  
Jonny ,M,  
Jony ,M,  
Jose ,M,  
Ju ,F,  
Juu ,F,  
Kata ,F,  
Kate ,F,  
Kika ,F,  
Kiko ,M,  
Lau ,F,  
Lena ,F,  
Leti ,F,  
Lola ,F,  
Maggie ,F,  
Mane ,M,  
Man ,M,  
Manu ,M,  
Marc ,M,  
Mary ,F,  
Megui ,F,  
Mike ,M,  
Migs ,M,  
Mimi ,F,  
Paul ,M,  
Pipa ,F,  
Pipo ,M,  
Rafa , ,

Ricky ,M,  
Sofi , F,  
Sofs , F,  
Suzy , F,  
Tati , F,  
Taty , F,  
Tete , F,  
Tigas ,M,  
Titi , ,  
Tixa , F,  
Toino ,M,  
Toze ,M  
T ze ,M,  
Toz ,M,  
Tuxa , F,  
Xana , F,  
Xano ,M,  
Xavi ,M,  
Zabel , F,  
Ze ,M,  
Zita , F,

---

---

sr ,M,  
mr ,M,  
boy ,M,  
man ,M,  
rapaz ,M,  
mano ,M,  
puto ,M,  
miudo ,M,  
menino ,M,  
gajo ,M,  
rei ,M,  
king ,M,  
sra , F,  
mrs , F,  
ms , F,  
miss , F,  
girl , F,  
woman , F,  
rapariga , F,  
mana , F,  
miuda , F,  
menina , F,  
gaja , F,  
rainha , F,  
queen , F,

---

Listing 49: List of gender-meaningful terms



Listing 50: Regular expressions used to remove URLs hashtags and mentions from tweets

```
# Regex to match an hashtag or a mention in a tweet
\S*(#|@)(?:\[[^\]]+\]|\\S+)'

# Regex to match an url in a tweet
[A-Za-z]+:\//\/[A-Za-z0-9-_\]+\. [A-Za-z0-9-_\%&~\?\/.=]+
```



# APPENDIX C: API DOCUMENTATION

---

## Network - Activity in each month

0.0.0 ▾

GET

```
/api/network/activity/
```

Example usage:

```
http://10.5.40.30/api/network/activity/
```

### Success 200

Field	Type	Description
Response	Object	Tweet counts as a JSON dictionary with month as key

Example Success Response:

```
{
  "1": 1278312,
  "2": 1380942,
  "3": 1503314,
  "9": 1683049
  ...
}
```

Figure 1: API - Network activity in each month

## Network - Activity in locations

0.0.0

GET

```
/api/network/locations/:month/
```

Example usage:

```
http://10.5.40.30/api/network/locations/10/
```

### Parameter

Field	Type	Description
month	Number	Requested date month (September 2015 to March 2016)

### Success 200

Field	Type	Description
Response	Object	Tweet counts as a JSON dictionary with keys as district codes

Example Success Response:

```
{
  "PT.SA": 51863,
  "PT.FA": 50392,
  "PT.CO": 72608,
  ...
  "PT.SE": 153375
}
```

Figure 2: API - Network activity in locations

## Network - Influence Rank

0.0.0

GET

```
/api/network/influence/:minscore/
```

Example usage:

```
http://10.5.40.30/api/network/influence/:minscore/
```

### Parameter

Field	Type	Description
minscore	Number	Minimum score of the users to be present in the rank

### Success 200

Field	Type	Description
Response	Object[]	List of users ordered by score

Example Success Response:

```
[
  {
    "lang": "pt",
    "profile_pic": "http://pbs.twimg.com/profile_images/739939590870605824/iD9nff_a_bigger.jpg",
    "degree_score": 988,
    "favourites_count": 8653,
    "description": "Twitter Oficial do Sporting Clube de Portugal. English Twitter @SportingCP_en",
    "friends_count": 155,
    "url": "http://t.co/PFPT8p0BZc",
    "name": "Sporting CP",
    "followers_count": 449624,
    "location": "Portugal",
    "user_id": 21389998,
    "predicted_gender": "M",
    "screen_name": "Sporting_CP"
  },
  ...
]
```

Figure 3: API - Influence rank

## Network - Network cluster nodes

0.0.0

GET

```
/api/network/clusters/:users/
```

Example usage:

```
http://10.5.40.30/api/network/clusters/1000/
```

### Parameter

Field	Type	Description
users	Number	Number of users to retrieve

### Success 200

Field	Type	Description
Response	Object	JSON object with the clustered nodes

Example Success Response:

```
{
  "nodes": [
    {
      "id": "187675609",
      "name": "A Marques",
      "screen_name": "carlapat28",
      "predicted_gender": "F",
      "label": "2",
      "y": "0.438018681763",
      "x": "-28.7520076365",
    },
    ...
  ]
}
```

Figure 4: API - Cluster nodes

## Network - Network cluster terms

0.0.0

GET

```
/api/network/clusters/:cluster_number/:n_terms/
```

Example usage:

```
http://10.5.40.30/api/network/clusters/1/10/
```

### Parameter

Field	Type	Description
cluster_number	Number	Cluster to get the terms
n_terms	Number	Number of terms to retrieve

### Success 200

Field	Type	Description
Response	Object	List with cluster terms

Example Success Response:

```
[
  "amanha",
  "tous",
  "casa",
  "dormir",
  ...
]
```

Figure 5: API - Cluster terms

### User - User similarity 0.0.0

**GET**

`/api/user/similarity/:userid/:range`

Example usage:

`http://10.5.40.30/api/user/similarity/19247625/10/`

**Parameter**

Field	Type	Description
userid	Number	Twitter user id
range	Number	Number of most similar users

**Success 200**

Field	Type	Description
Response	Object	JOSN of users nodes and links with similarity score

Example Success Response:

```

{
  "nodes": [
    {
      "group": "M",
      "id": "romestafelto",
      "name": "Joao Quadros"
    },
    {
      "group": "M",
      "id": "tyagoclemente",
      "name": "Tiago Clemente"
    },
    ...
  ],
  "links": [
    {
      "source": 0,
      "target": 1,
      "value": 0.31412965643704338
    },
    {
      "source": 0,
      "target": 2,
      "value": 0.32547183970987537
    },
    ...
  ]
}

```

Figure 6: API - User similarity

### User - Terms by topic 0.0.0

**GET**

`/api/user/topics/:userid/`

Example usage:

`http://10.5.40.30/api/user/topics/19247625/`

**Parameter**

Field	Type	Description
userid	Number	Twitter user id

**Success 200**

Field	Type	Description
Response	Object[]	List of terms

Example Success Response:

```

[
  {
    "term": "governo",
    "topic": "0"
  },
  {
    "term": "bola",
    "topic": "1"
  },
  ...
]

```

Figure 7: API - User topics

# APPENDIX D: USABILITY ENQUIRY

---

# User Characterization in Social Prototype Usability

This form aims to evaluate the User Interface of the prototype produced in the course of the dissertation User Characterization in Social Media (Caraterização de Utilizadores em Redes Sociais), in the University of Aveiro

## Age

A sua resposta

---

## Gender

- Male
- Female
- Prefer not to say

## Profession

A sua resposta

---

## Do you regularly use Twitter?

- Yes
- No

**SEGUINTE**

Nunca envie palavras-passe através dos Formulários do Google.

Figure 8: Usability Enquiry - Section 1



## Network Explorer - Activity

A number of tasks will be required on the Activity tab of the Network Explorer. When times are asked please provide the time took to complete the task in seconds.

**Time to find the number of tweets on the first of November**

A sua resposta \_\_\_\_\_

**Rate the visual representation used to represent tweets per month in terms of adequacy to the task**

1            2            3            4            5

**Time to find the tweet distribution on the 31st of January**

A sua resposta \_\_\_\_\_

**Rate the visual representation used to represent tweets per day in terms of adequacy to the task**

1            2            3            4            5

**Time to find how tweets are distributed in the Portuguese territory in the month of February**

A sua resposta \_\_\_\_\_

**Rate the adequacy of the visual representation used to represent tweets distribution over locations**

1            2            3            4            5

Figure 9: Usability Enquiry - Section 2

## Network Explorer - Influence

A number of tasks will be required on the Influence tab of the Network Explorer. When times are asked please provide the time took to complete the task in seconds.

**Time to open the details of fourth more influence user on the network**

A sua resposta

---

**Time to limit the list only to Portuguese users**

A sua resposta

---

**Time to limit the list only to Portuguese female users**

A sua resposta

---

ANTERIOR

SEGUINTE

Nunca envie palavras-passe através dos Formulários do Google.

Figure 10: Usability Enquiry - Section 3

## Network Explorer - Content

Rate the adequacy of the visual representation used to represent similarity between network nodes

1 2 3 4 5

Rate the adequacy of the visual representation used to show the most relevant terms in each cluster

1 2 3 4 5

ANTERIOR

SEGUINTE

Nunca envie palavras-passe através dos Formulários do Google.

Figure 11: Usability Enquiry - Section 4

## User Explorer - Profiling

Time to find the profile information of user "omalestafeito"

A sua resposta \_\_\_\_\_

Time to find tweet distribution of user "omalestafeito"

A sua resposta \_\_\_\_\_

Time to find the similar users of user "omalestafeito"

A sua resposta \_\_\_\_\_

Rate the adequacy of the visual representation used to show the most similar users

1      2      3      4      5

Time to find the network distance between user "omalestafeito" and user "Sporting\_CP"

A sua resposta \_\_\_\_\_

Rate the adequacy of the visual representation used to show network distance

1      2      3      4      5

Rate the adequacy of the visual representation used to show network distance

1      2      3      4      5

Figure 12: Usability Enquiry - Section 5

## Usability Heuristics

Evaluating the compliance with Jakob Nielsen's Usability Heuristics for Interface Design  
<https://www.nngroup.com/articles/ten-usability-heuristics/>

How do you rate the overall system feedback?

	1	2	3	4	5	
Not useful at all	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very intuitive and helpful

Do you feel the communication patterns used in this system (text and graphic) are easily understandable?

- Yes
- No
- Outra: \_\_\_\_\_

Rate the control mechanisms used to interact with the system?

	1	2	3	4	5	
Hard to use	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very easy to use

Do you think the language used is consistent with functionality?

	1	2	3	4	5	
Inconsistent	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very consistent

Did you encounter any error message when handling this system?

- Yes
- No

If you answered yes, did you find it helpful?

- Yes
- No

Figure 13: Usability Enquiry - Section 6