POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Identification and Clustering of Anomalies in Online Social Networks

Supervisor

Prof. Martino TREVISAN

Co-Supervisor

Prof. Luca VASSIO

Candidate

Paola CASO

April 2022

Abstract

Online Social Networks (OSNs) have become an integral part of modern-day life. People use Social Media platforms to share their thoughts, stories, and news, and major events and occurrences throughout the world can be discovered, frequently before they are reported by journals or other forms of classical media outlets.

This work aims at identifying real-world events that have affected the post-stream of OSNs. The proposed novel approach involves detecting anomalous posts, i.e. those that have received more reactions than expected, and then clustering them for each week, on the basis of textual-content similarity, to identify the weekly offline trending or peculiar events that probably induced such atypical engagement, such as the participation of an influencer to a TV show or a contest. The choice for this research fell on the social image sharing platform Instagram, one of the most famous and used OSNs in the world, to exploit multimodal media contents.

The dataset was gathered between 2015 and 2021, including about 1611 Instagram Italian influencer accounts and 2 036 966 posts, from CrowdTangle, a public insights tool owned and operated by Meta ©.

The dataset was pre-processed, in order to extract the main features, remove useless attributes or erroneous data, compute a performance score (in terms of reactions received with respect to expected reactions) for each post (the main feature for the Anomaly Detection phase), and to perform Natural Language Processing (NLP) analyses on the text, contained in the post-caption or within the post-media, to check the textual similarity (the main feature for the Clustering phase) between the posts.

To perform Anomaly Detection on the posts, four classic state-of-the-art methods have been implemented: an ARIMA Model, the Boxplot Rule method, the Isolation Forest and the Z-Score technique, all applied on the time series of the performance scores assigned to the posts of each influencer.

To the anomalies found by the various methods, further textual characteristics machine-intelligible were added, such as the TF-IDF and the Sublinear TF-IDF, calculated for each week of the dataset.

The final phase involves clustering weekly anomalies by textual similarity. Four algorithms have been implemented: the DBSCAN, applied on a pre-computed pairwise distance-matrix involving the captions of anomalous posts, the K-Means, applied on a weighted hashed vector of the caption words, an LDA model, and Community Detection algorithms, applied on a graph of anomalous posts.

The proposed system works well, as it was actually possible to find clusters that

make sense as belonging to the same theme, and they manage well to group posts related to particular events (important football matches, music festivals, TV shows, famous weddings, Oscar awards, etc.). The results obtained suggest that some posts actually received more/less reactions than expected as their content was related to external events in the real world, while others, classified as "noise" are not related to such occurrences, so probably they obtained an anomalous engagement for causes endogenous to the OSN.

Based exclusively on numerical data, taking into consideration the Silhouette Score (and modularity in the case of graphs), the best results would seem to derive from the pair ARIMA model - DBSCAN on hashtags, while the other two pairs are equivalent, with the exception that the graph method is the only one that gives acceptable results when considering all types of words (hashtags, simple words and words within the image) rather than just hashtags. According to these statistics, it would appear that the graph method is the worst of those mentioned, but, in reality, by manual inspection it is the one that gives the best results.

Table of Contents

List of Tables

List of Figures

Acronyms

ACF
Auto-Correlation Function

ADF
Augmented Dickey-Fuller

AIC
Akaike Information Criterion

API
Application Programming Interface

ARIMA
Auto Regressive Integrated Moving Average

ASA
Advertising Standards Authority

BoW
Bag of Words

BST
Binary Search Tree(s)

COVID
COrona Virus Disease

DBSCAN
Density-Based Spatial Clustering of Applications with Noise

ECDF

Empirical Cumulative Distribution Function

FSA

Fire Spread community detection Algorithm

GA

Genetic Algorithm

HCCs

Highly Coordinating Communities

HDFS

Hadoop Distributed File System

HEE

Hot Event Evolution

IG

Instagram

IQR

Interquartile Range

IRA

Internet Research Agency

LCN

Latent Connection Network

LDA

Latent Dirichlet Allocation

LIWC

Linguistic Inquiry and Word Count

LPA

Label Propagation Algorithm

LSTM

Long Short-Term Memory

ML

Machine Learning

NBC

Naïve Bayesian Classifier

NLP

Natural Language Processing

NER

Named Entity Recognition

NN

Neural Network

NoA

Node of Attraction

OSNs

Online Social Networks

PoS

Part-Of-Speech

RDD

Resilient Distributed Dataset

SGD

Stochastic Gradient Descent

SNA

Social Network Analysis

SQL

Structured Query Language

SVI

Search Volume Index

SVM

Support Vector Machine

TF-IDF

Term Frequency — Inverse Document Frequency

XGB

Extreme Gradient Boosting

Chapter 1

Introduction

1.1 Motivation and Objectives

OSNs (Online Social Networks) have become an essential aspect of everyday life. People utilize social media platforms to share their ideas, experiences, and news, and big events and happenings throughout the world can be discovered, often before they are published in journals or other traditional media venues. The goal of this research is to discover real-world events that have had an impact on OSNs' post-stream. The proposed novel approach entails identifying anomalous posts, i.e. those that have gotten more reactions than expected, by exploiting classical Anomaly Detection techniques, and then clustering them for each week based on textual-content similarity in order to identify the weekly offline trending or peculiar events that most likely induced such atypical engagement. In particular, we wish to aggregate the anomalous posts (after recognizing them) using clustering algorithms, and distinguish those that have got a number of anomalous replies owing to external events that occurred in the offline world and are thus exogenous to the OSNs, such as the participation to a TV show or a contest. Because this is an unsupervised task (the themes to which the posts "belong" cannot be determined a priori), a manual review of the results is required to validate the content. The social picture sharing platform Instagram, one of the most well-known and widely used OSNs in the world, was chosen for this study to leverage multimodal media contents.

1.2 Online Social Networks

According to the definition of Encyclopedia Britannica, an online social network, in computers, is:

> [...] an online community of individuals who exchange messages, share information, and, in some cases, cooperate on joint activities. [1]

Figure 1.1: Leading Online Social Networks logos [2][3][4]

In other words, the term Social Network identifies an online computer service[1] that enables the creation of virtual social networks. More in detail, after registration by creating a password-protected personal profile (generally provided by the service), users can interact with each other and share textual content, images, video and audio (including personal data, religious sensitivities, political opinions, sexual relations, etc.): users are not only content consumers, but also content creators.

1.2.1 Difference between Online Social Network and Online Social Media

While the terms social media and social network seem to be similar and interchangeable, they have distinct differences.
According to the definition of Encyclopedia Britannica again, the term social media refers to:

> [...] technologies, platforms, and services that enable individuals to engage in communication from one-to-one, one-to-many, and many-to-many. [5]

In other words, it is an online communication service, a platform for information broadcasting. More specifically, social media platforms are web-based applications, which allow the creation and sharing of content also by the users themselves, make

[1] websites or technologies

interaction with a wide audience possible. This instrument is bidirectional, since it adds the social component to the transmission of information: radio or television, for example, are unidirectional tools, since they transmit information without any interaction with the viewer.
A social network, on the other hand, is a more theoretical concept, used to describe relationships between individuals, groups and organizations: it is a real social structure. Online Social Networks (OSNs) can be considered as a subset of social media, being a smaller group that shares the same interests or causes.
To be clear, social media is the tool, while the social network is the people who use social media to create their own online community network.

1.2.2 Historic Background of Online Social Networks

The origin of today's internet and most of contemporary social media networks dates back to the emergence in the late '60s of the Advanced Research Projects Agency Network — the **ARPANET**: a military internal communication system. It was created by the U.S. Department of Defense, to allow four interconnected universities to share software, hardware, and other data, guaranteeing continuity of communication between different locations in case of a nuclear bombing (USSR).
In the following decades, the debut of E-mails and chat programs led to the creation of **USENET**, a messaging system between two American universities, that enabled users to post and receive messages within thematic areas called *newsgroups*.
Moreover, the advent of the **World Wide Web**, developed by the CERN researcher Sir Tim Berners-Lee, allowed anyone with a mobile connection and a web browser to easily navigate from one site to another (the *resources*) through *hyperlinks*, and faster connections made it possible to share more multimedia content.

In 1994 **Yahoo! GeoCities** was born, a web hosting service that allowed users to create and publish websites for free and to browse them by their theme or interest. The *cities*, containing a list of hyperlinks to web pages, were named as real cities or regions according to their content.
In 1997, the first recognizable social media site, **SixDegrees**[1], launched, which enabled users to upload profile pictures, make friends with each other, send messages and publish bulletin board items to others in first, second, and third-degree connections. It was based on the Theory of the Six Degrees of Separation - hence the name - according to which all people are connected to each other through a chain of at most five intermediaries. This service was followed in 2002 by the Asian **Friendster**, a rudimentary platform with email paid registration and basic online networking features, which was the precursor of modern Online Social Networks

[1] `www.sixdegrees.com`

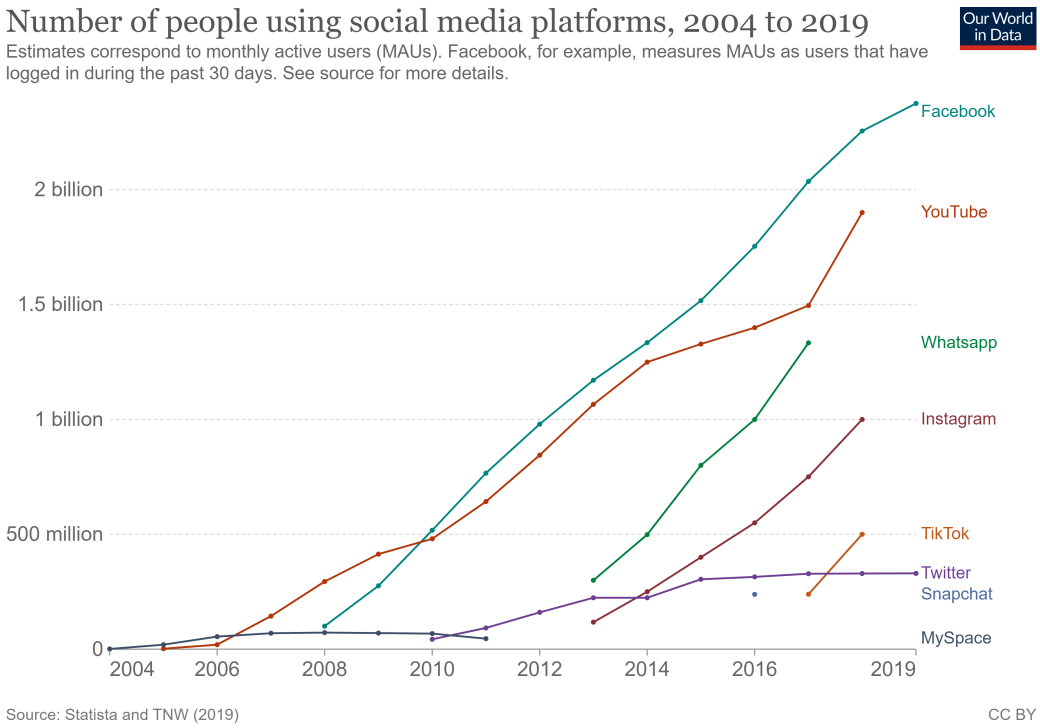


Figure 1.2: Monthly Active Users by Social Platform from 2004 to 2019, taken from [6]

(OSNs). Friendster attracted millions of users, but it soon decayed. One reason was the *popularity index*, which depended on the number of friends that each user had in his friend network (the more friends the higher the popularity) and which led to the creation of *Fakesters*, fake accounts to widen the circle of friends and increase the index.

After the appearance of the first blog sites and the birth of *Wikipedia*[1], **LinkedIn**[2], one of the most popular social media sites for business in the world still today, was founded in 2003 as a networking site for career-minded professionals. The same year also witnessed the launch of **MySpace**[3], which allowed users to completely customize their profile, but also to could embed and share music and videos. To date, MySpace is still present on the web, but the number of users has dropped significantly. **YouTube**[4] first appeared in 2005 and launched an entirely new method of communication with its ability to create and share media (in particular online videos) over very long distances. Time has led YouTube to be the second

[1] `www.wikipedia.com` [2] `www.linkedin.com` [3] `www.myspace.com`
[4] `www.youtube.com`

most visited website in the world after Google.

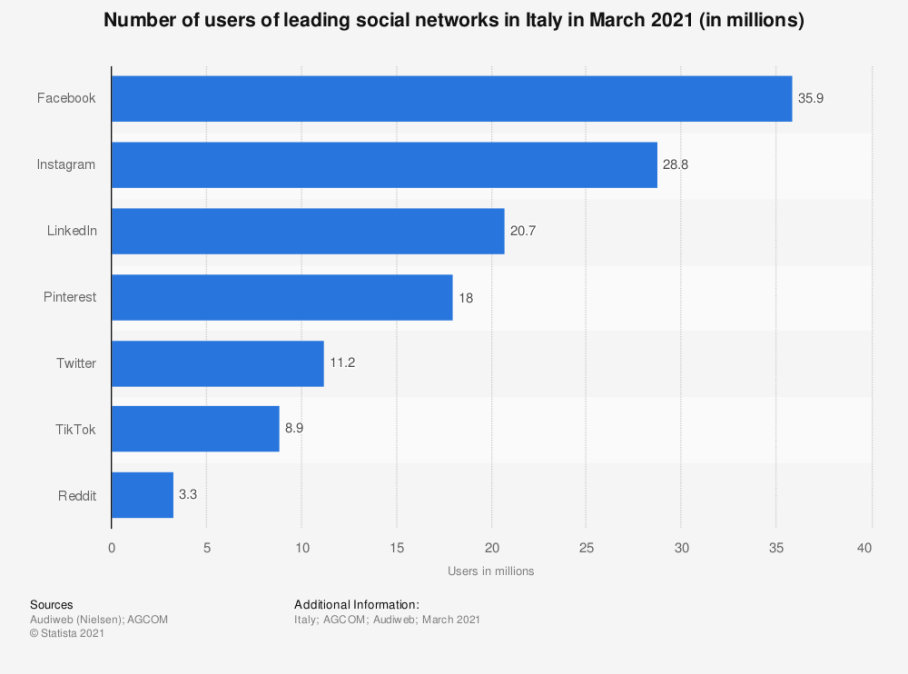


Figure 1.3: Number of users of leading social networks in Italy (March 2021) [7]

2006 saw the advent of **Facebook**[5], which remains one of the most popular social media platforms in the world: it is one of the most visited sites on the web. **Twitter**[6] also, launched in 2006, with the intention of allowing users to share messages in only 140 characters (today 280), the same as SMS text messages, and its ability to enable users to interact directly with celebrities.
Tumblr[7] arrived in 2007, with its micro-blogging and social networking features. Yet another important step in the history of the evolution of social media platforms, was the launch of **Instagram**[8] in 2010. A US-based photo and video sharing platform, Instagram is today one of the biggest social media sites in the world.
In 2016, the now incredibly popular social media platform, the Chinese **TikTok**[9] was born. It is a smartphone app designed to create short video clips lasting up to 60 seconds, with the ability to edit them, use special effects, add parts of songs to create ballets.

[5] `www.facebook.com` [6] `www.twitter.com` [7] `www.tumblr.com`
[8] `www.instagram.com` [9] `www.tiktok.com`

1.2.3 Functionality Framework of Online Social Networks

Today's Online Social Networks (OSNs) offer different functionalities, depending on the chosen platform. While they all have distinctive features, there are almost always certain commonalities between them. The seven key essential functional blocks [8] are the following:

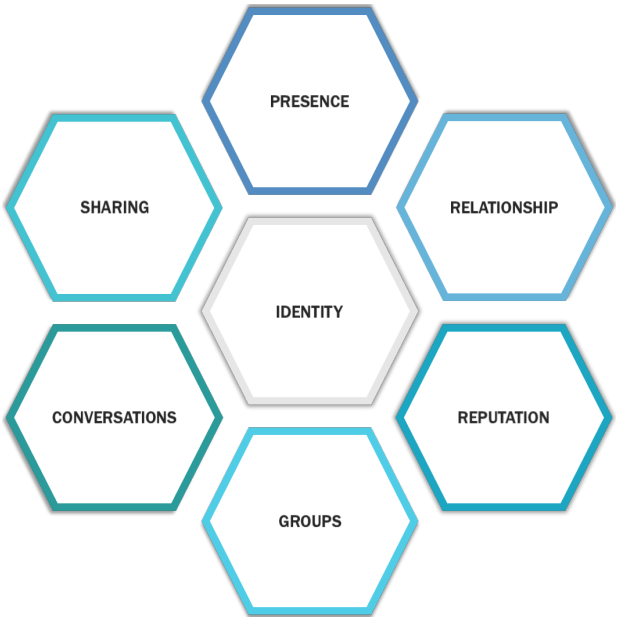


Figure 1.4: Online Social Networks Functionality (*Honeycomb Model*)

- **Identity**: users can customize their own profile and publicly (or partially) share their information (name, surname, gender, age, profession, profile-photo, etc.) and created contents, according to their own privacy settings;
- **Conversations**: users can start conversations among individuals and groups in chat systems;
- **Sharing**: users can exchange, distribute, and receive content, which can be textual, media, or both, in the form of a post. A post is a sort of container of information that can be also re-shared, liked and/or commented by other users;
- **Presence**: users can know if other users are available through *status* lines like "available/online" or "hidden/offline", and they can choose in their settings *selective presences*: one can be visible to some people while staying hidden to others, or everyone. Some platforms can give information also about where the user is in the real world;

- **Relationships**: users can be related to other users through some form of association that leads them to converse, share objects, meet up, build real relationships, or simply just list each other as a friend or fan. In particular, each user can have a contact-list containing other users and the corresponding relations, which can be uni-directional or bi-directional (*followers*, *followee*) or also necessarily mutual (*friends*), depending on the platform;

- **Reputation**: users can identify the standing of others, including themselves, in a social network environment. In most cases, reputation is an indication to determine trustworthiness. In other cases, reputation refers not only to users but also their content, which is often evaluated using content voting systems: *likes*, *comments*, *ratings*, *view counts*, etc. The number of followers is also a metric for reputation but with a limitation: it only indicates the popularity of a person, not how many people actually read the posts;

- **Groups**: users can form communities and sub-communities, which can be open to anyone, closed (approval required), or secret (by invitation only). In the last two cases, generally, some *administrators* manage the group, approve new members, and invite others to join;

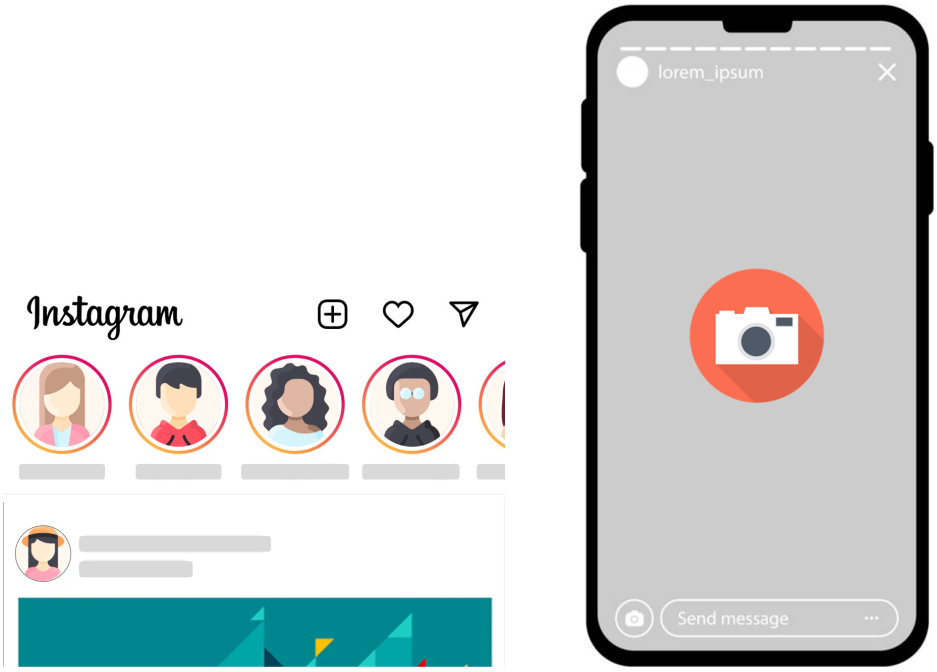
1.2.4 Our research choice: Instagram Social Network

The Online Social Network chosen for our research is Instagram (often colloquially abbreviated as *IG*), a photo and video sharing social networking service founded in 2010 in California. The application allows users to upload media (images and videos) in the form of so-called **posts** that can be edited with photographic *filters*, organized by ***hashtags*** and geographical location tagging (*geotags*) and shared publicly or with previous approval of followers[1] (private profile), according to their customized privacy settings. Users can browse other users' content by tags and locations, visiting their profiles or *swiping* through their personal **feed**. The Instagram Feed is a constantly updating dashboard displaying a list of photos and videos from advertisers and/or accounts that people follow, that appears when opening the Instagram application. The posts can represent a photo, a up to 60 seconds video, an album of up to 10 photos/videos, an **Instagram Reel** (a up to 15 seconds editable multi-clip video with audio, effects, and other tools-additions) or an **Instagram TV** (*IGTV*, a up to one hour video). In particular, "IGTV" is a stand alone application, where each user page is named "channel".

Users can "like", "comment", "save" into a private area of the application, send

[1]Users who subscribe to other users' accounts to "follow" their updates.

in **Direct**[1] and re-share in the **Instagram Stories** (Fig.1.5), the posts. Instagram



(a) Instagram Stories bar: user view example [9]

(b) Instagram Stories slideshow: actual content displayed example [10][11]

Figure 1.5: Instagram Stories template

Stories are vertical photos or (up to 15 seconds videos), that expire after 24 hours from posting, but they can also be collected and displayed (until removed) on the owner profile in the so-called Instagram Stories **Highlights** (Fig.1.6b). They are displayed as coloured circle (Fig.1.5a) around the profile pictures of the owners, at the top of a user's application view, and they need to be clicked through to present the actual content in a slideshow format (Fig.1.5b).

Unfortunately, even if very commonly used by several ordinary users and ***influencers*** (1.2.5), it was not possible to study Instagram Stories in this research due to the unavailability of data.

[1]Instagram chat system to send messages including text, posts, Instagram Stories, (eventually disappearing) photos or videos taken or uploaded from the photo library, to one or more people.

1.2.5 Instagram influencers

In the context of communication and marketing strategies, not only in the Online Social Networks (OSNs) environment, the term *influencer* indicates a popular individual on the Internet, who has the ability to influence the behaviors, opinions and choices of a certain group of users (in particular, potential consumers). In

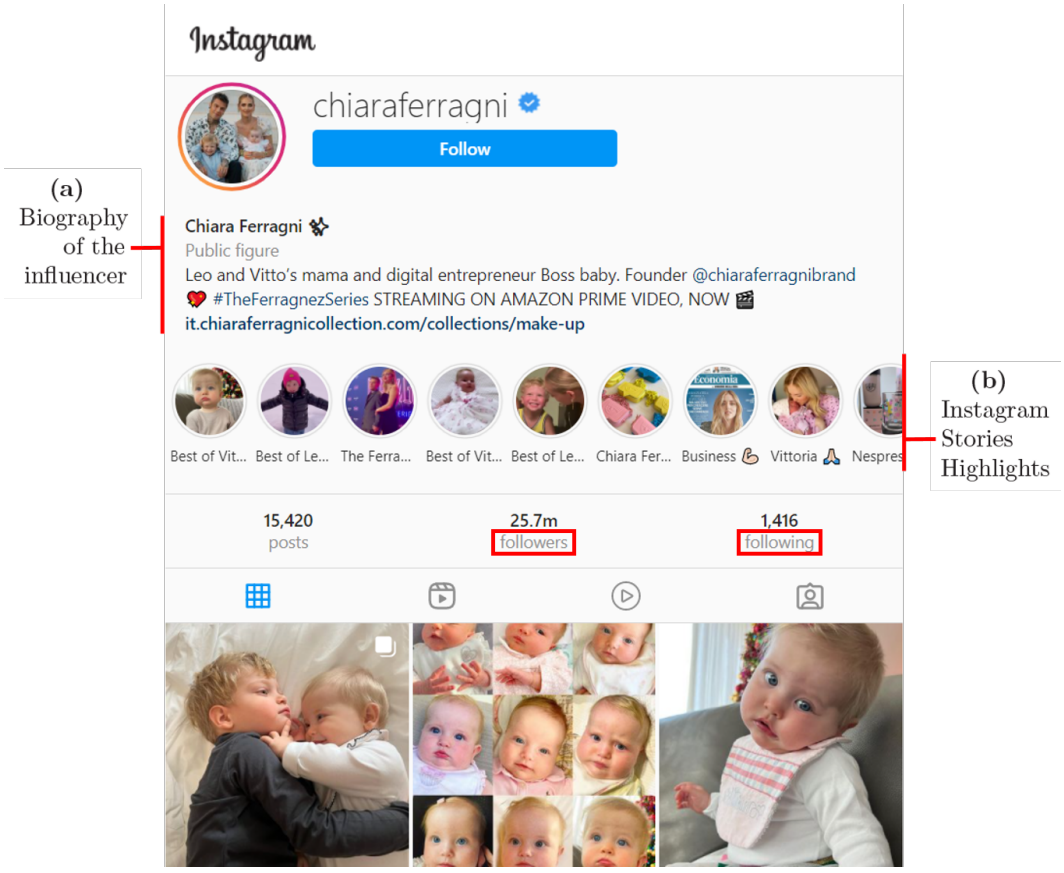


Figure 1.6: Instagram profile of a famous Italian Mega influencer: Chiara Ferragni[1]

other words, influencers are people able to create trends on the Internet and to affect the purchasing decisions of others because of their authority, knowledge and expertise on a specific topic, position, or relationship with their followers. So brands can collaborate with them to achieve their marketing objectives, preferring to advertise through influencers' endorsement rather than through advertising officers. Companies can pay influencers to encourage and persuade their followers to buy products they promote, and, to this extent, some laws have been introduced

[1] `www.instagram.com/chiaraferragni`

that require the usage of some specific hashtags such as *#ad* or *#adv* [12]. Instagram influencers in general publish posts regularly about their preferred topic (a particular area, e.g. fashion, food, travel or technology) and generate large followings of engaged people who are interested in their updates.

There are multiple ways to classify influencers: by followers number, by types of content (e.g. bloggers, YouTubers, podcasters, etc.), by the level of influence or by the niche in which they operate. Generally the first method is the most common one [12] because of its simplicity of application, and it defines four tiers of influencers (Fig. 1.7):

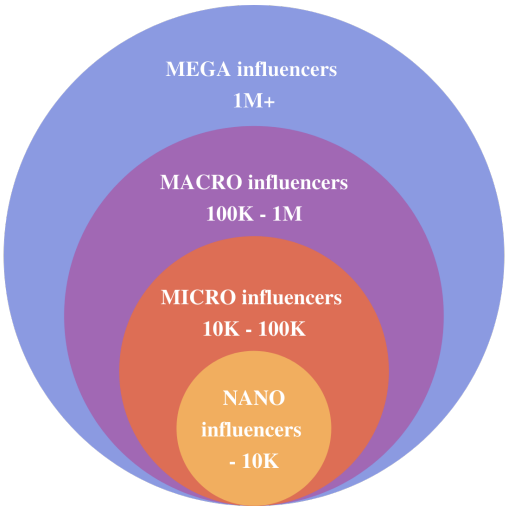


Figure 1.7: Tiers of influencers, graphical representation

- **Mega influencers**: over 1 000 000 followers (Fig.1.6).
- **Macro influencers**: between 100 000 and 1 000 000 followers.
- **Micro influencers**: between 10 000 and 100 000 followers.
- **Nano influencers**: less than 10 000 followers.

According to the paper by Zarei et al. [12], most influencers publish unsponsored posts, otherwise they tend to advertise products belonging to their area of expertise. In general, influencers in the mega, macro and micro categories prefer the usage of Instagram Stories to promote sponsored content rather than the posts, and, in particular, mega influencers are the dominant: they prefer to post "advertise-stories" more regularly than other categories of influencers. Anyway, according to this study, even if mega influencers obtain the greatest attention, measuring likes and comments, small nano influencers sustain attention more effectively, in their niche (Fig.2.1).

Chapter 2

Related Literature

2.1 Online Social Networks General analyses

Online Social Networks (OSNs) have been extensively studied during these years. The work by Zarei et al. [12] focused its analysis on the types of products being promoted by Instagram influencers, their potential reach, and the engagement received from their subscribers (followers). The researchers collected both Instagram posts and stories from users who attached advert-related hashtags to their contents, using the official Instagram APIs to crawl them, between Sep 2018 and April 2019. Once extracted the accounts identified during this phase, all the posts, stories and reactions generated by them were monitored from July 2019 to August 2019. Then the data were categorized in *sponsored/non-sponsored* (simply checking if sponsored metadata, i.e. hashtags, were included), validated (filtering any incorrectly identified influencer) and sampled (using a Random Under-Sampling to reduce the size of the non-sponsored class and to obtain an equal number of sponsored and non sponsored labeled posts). About the types of products being advertised by influencers, they observed that 50% Mega, 58% Macro and 70% Micro influencers promote just a single product, or focus on a particular product type, possibly in their own specialist area (Fig.2.1). To better classify the data, in the paper it is explained that the contents were subdivided into sponsored, non-sponsored and hidden sponsored[1] with the aid of a Random Forest Classifier, compared to the results of a Contextual LSTM Neural Network architecture. For the prediction (manually validated), the most important features were the post captions, the profile biographies (Fig.1.6a) and the post hashtags[2].
The results showed a noticeable set of hidden sponsored posts, particularly for Micro

[1] Advertise-related hashtags are removed, so the posts are not officially declared as sponsored.

[2] Generally the name of the product or producer (or both) is listed as a hashtag.

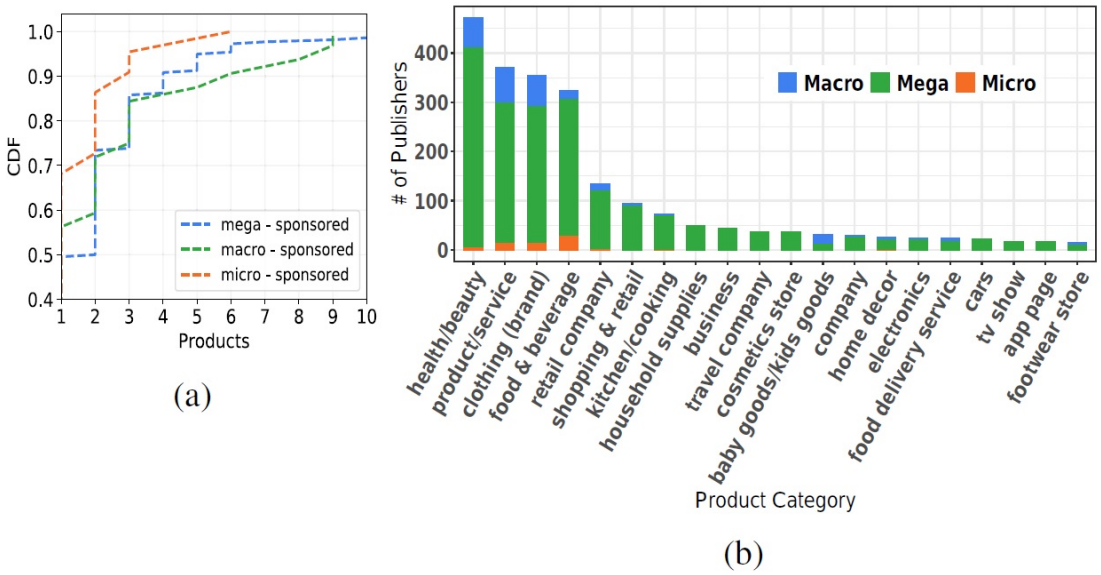


Figure 2.1: (a) CDF of number of promoted products. (b) Number of products promoted in stories, taken from [12]

and Nano influencer, despite of the fact that the Advertising Standards Authority (ASA) advises the usage of commercial hashtags, also because influencers' income is taxable. Further results were already discussed in the last part of section 1.2.5.

Another work, by Javed et al. [13], focused its analysis on WhatsApp[1] messages and posts on Twitter (e.g. *tweets*) in Pakistan. This study was aimed at inspecting what kind of messages about the COVID-19 pandemic were being exchanged, what was the general behavior of users, if there was misinformation about COVID-19 and possibly a relationship with that on Twitter. 227 public WhatsApp groups (reachable by anyone via a URL) including text, images and videos, containing some keywords such as "Corona" or "2019-cov", were analyzed (everything previously anonymized), and, in the end, 14% of pandemic information resulted to be erroneous. Images were more complex to categorize automatically, so a manual tagging was made by two commentators, selecting those containing text related to COVID-19, to restrictions, to precautionary measures and social distancing, and grouping them by similarity (via Hamming distance) with an image hashing tool. Messages were then characterized and divided into 5 categories: information, disinformation (everything that could not be verified or that was certainly false), joke/satire, religious, ambiguous (containing not sufficient data to be assigned to one of the

[1]Instant messaging platform, which allows users to send each other text and voice messages, make voice and video calls (VoIP), share images, videos, documents, locations, and other contents.

previous categories), considering that a single message could belong to multiple categories at the same time (Fig.2.2). Most of the messages resulted to be classified as information, followed by religion. Analyzing the "life" of messages, to evaluate

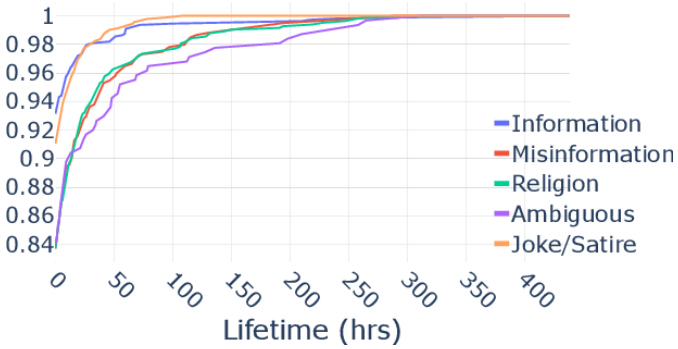


Figure 2.2: CDFs of Text of COVID-19 categories, taken from [13]

the impact of the various types of information, the results showed that the messages that lasted the least were those of satire, while disinformation had a quite high duration and circulated for longer periods than valid information. Misinformation were further categorized, according to their contents: fake news, which included some conspiracy theories or false information about COVID-19; origins, concerning a false origin of the virus, such as having been created in the laboratory; remedies, concerning bogus treatment methods; vaccine: about theories according to which the vaccine had already been developed and used for economic reasons; weather, concerning the fact that the vaccine would disappear with seasonal climate changes; comparison with seasonal flu, information that lowered the severity of COVID-19 symptoms by comparing them to those of the flu. Among these, the category with a shorter duration was the one of fake news, while the most lasting was the one concerning the false remedies. On Twitter, information resulted to have a longer "*die-time*" than incorrect information (3 times more), also because more people could comment and deny the truthfulness of a post.

The study by Weber et al. [14] observed coordination strategies such as boosting, bullying, pollution, and metadata shuffling attempting to reveal networks of cooperating accounts and groups whose behavior was anomalous in degree. The researchers proceeded with an approach based on temporal windows of varying size, user interactions and metadata to to detect accounts engaging in potentially coordinated goal-based strategies. The analysis was performed comparing a randomised dataset (to provide validation) with two relevant datasets, crawled from Twitter: one provided by the IRA and based on October 2018 general activities, while the other containing tweets published during the 2018 Regional Australian Elections. The workflow pipeline (Fig.2.3), from extracting relevant information

from the raw data to the results, consisted of five steps: convert social media posts to common interactions, filter these interactions selecting the ones which were relevant to the research criteria, infer (typed by criterion) links between accounts creating a collection of inferred pairings, construct an LCN from the obtained pairings, where the vertices were the accounts connected by weighted edges of inferred links, identify the highest coordinating communities using a community detection algorithm (FSA_V, a variation of the FSA algorithm).
On one hand, the paper confirmed the usage of the mentioned strategies in organized operations and opened space to explore real-time applications to recognize, and eventually intervene, those schemes. On the other hand, the assumption that political accounts would retweet and mention themselves frequently was not demonstrated by the outcomes.

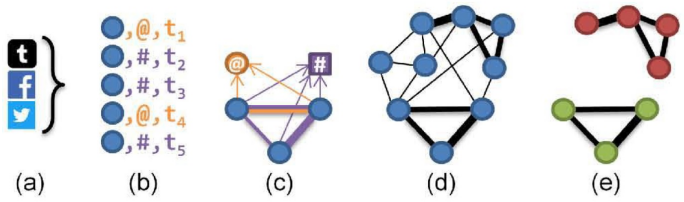


Figure 2.3: Conceptual LCN/HCC pipeline (circles are accounts), taken from [14]

The usage of Online Social Networks has effects on real-world offline phenomena, and this raises the problem of the reliability of data collection for OSNs analyses: the data must be correct, but also complete enough to construct meaningful networks and produce a good mapping between online and offline events. In this direction goes the work from Weber et al. [15], which tries to apply a systematic comparison technique: the researchers collected simultaneously two parallel datasets from Twitter, using different methods and tools, to understand how variations in data acquisition influence the results of social network analyses. To generate the two datasets, the analysts used respectively Twarc[1], as baseline, and RAPID[2]. Afterwards, they built three weighted directed social networks, using the accounts as nodes and starting from direct interactions categories: "mention networks", "reply networks", and "retweet networks". The analyses performed on the datasets were some absolute statistics, such as the count of tweets and hashtags, network statistics like Louvain cluster count or transitivity, centrality values and cluster comparison. According to the results, most prevalent content values remained similar, but some differences were witnessed: different numbers of captured accounts

[1] `https://github.com/DocNow/twarc`
An open source library which wraps Twitter's API.

[2] A social media collection and data analysis platform for Twitter and Reddit.

created a different number of nodes, extra tweets caused extra edges and, more in general, network architecture was different.

In the recent years, the possibility to exploit the potential of new Machine Learning techniques and OSNs data to perform predictions about users' political orientation received great attention and interest from scholars. The work from Cardaioli et al. [16] is related to this possibility: using Twitter APIs, a dataset of more than 6000 users and almost 10 million tweets were downloaded, validated (to detect legitimate accounts and to eliminate Twitter bot accounts) and pre-processed (removing URLs, punctuation, etc.). Further, the accounts were manually labeled (by a pool of four to five independent human judges), using two different labeling techniques, as supporters of six distinct categories of Italian political parties, ranging from extreme right to extreme left. After that, the profiles identified as supporters of ideologically well-defined parties were used to train five different classification algorithms (SVM, Linear Regression, SGD, Random Forest, XGB), to assess the political tendency of people labeled as "Movimento 5 Stelle"[1] electorate. In order to use these Machine Learning methodologies, the authors used the TF-IDF (chap.4.5.1) to represent tweets numerically, analyzing characteristics and relations between the words. When predicting left-right association, the researchers were able to reach an accuracy of up 93%. The outcome of the "Movimento 5 Stelle" voters labeling showed a subdivision in political inclination as in Figure 2.4. This analysis showed that the most frequent words used by the left-wing supporters

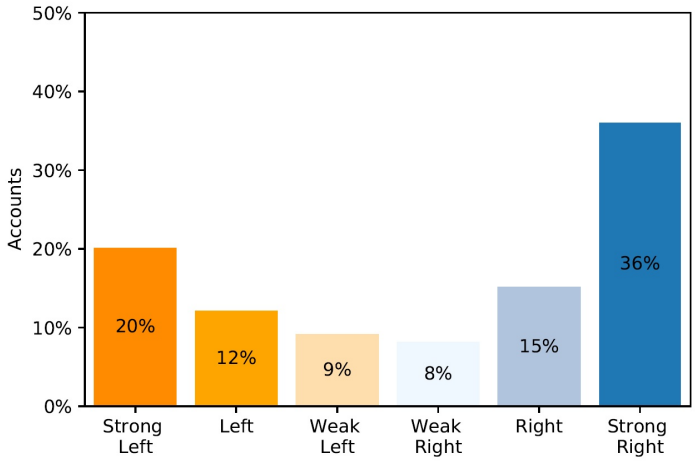


Figure 2.4: Agreement of the tested models on M5S supporters, taken from [16]

in their Tweets are referred to the opposite parties and vice versa. This result

[1] An Italian political party whose inclination towards the left-right line is ambiguous.

reflect the idea that people generally tend to blame the ideas of the out-parties rather than discuss the ideas on the big political topics of the party they support.

2.2 Anomaly Detection in Online Social Networks

In the last years, many researchers have exploited anomaly detection techniques in several research areas, especially for data mining, including Online Social Networks (OSNs). The scholars have adopted different methodologies of classical literature to catch anomalies in a wide range of applications, generally to protect users' security and privacy. Moreover, OSNs make easy the spread of irregular activities, dissemination of fake news, disinformation (sec. 2.1 [13]), rumours, spam, and malicious links (*phishing*). Therefore, detection of anomalies is an important data analysis field.

The authors of the work by Rahman et al. [17], developed a hybrid anomaly detection system (*DT-SVMNB*), to classify Facebook legitimate and anomalous users, that combined three Machine Learning algorithms in cascade: a Decision Tree, a Support Vector Machine (SVM) and a Naïve Bayesian Classifier (NBC). Abnormal messages have negative effects on the society, such as cyberbullying, and, therefore, the final aim of the researchers was to detect abnormal users identifying the potential suicidal ones. To train the proposed model, the researchers extracted some unique features from two types of dataset, synthetic and real: users' profile-based features, including the behavioral changes and the basic information of a user, like the total number of friends and the number of followers, and content-based features, concerning unexpected parts of the comments, status and posts, like the number of negative emoticon symbols per day, the frequency of money words (often anomalous users aim at commercial intent), the hashtags, the number of likes, the number of URLs (to detect advertise contents), the number of sharings (to identify spammers), and others. The proposed anomaly detection framework (Fig. 2.5) consisted of three steps, corresponding to the chosen three Machine Learning algorithms: the first phase consisted in data pre-processing and sampling behaviors, to train the C5.0, and afterwards, using the algorithm to classify the users into "normal" and "abnormal": the algorithm made a decision based on the feature value at each level of the constructed tree; in the second phase, the anomalous users previously extracted, are further classified by SVM into "disappointed" and "happy" users; in the third and final phase, the users labeled as "disappointed" are used to train the NBC, to find out the vulnerable ones, in particular classifying them into suicidal and not suicidal (based on a dictionary feature). A performance analysis on the results, based on a calculation of the false positive and true positive rates used to compute accuracy, precision and recall, showed that the authors were

able to reach an accuracy of 98%, proving the effectiveness of the system.

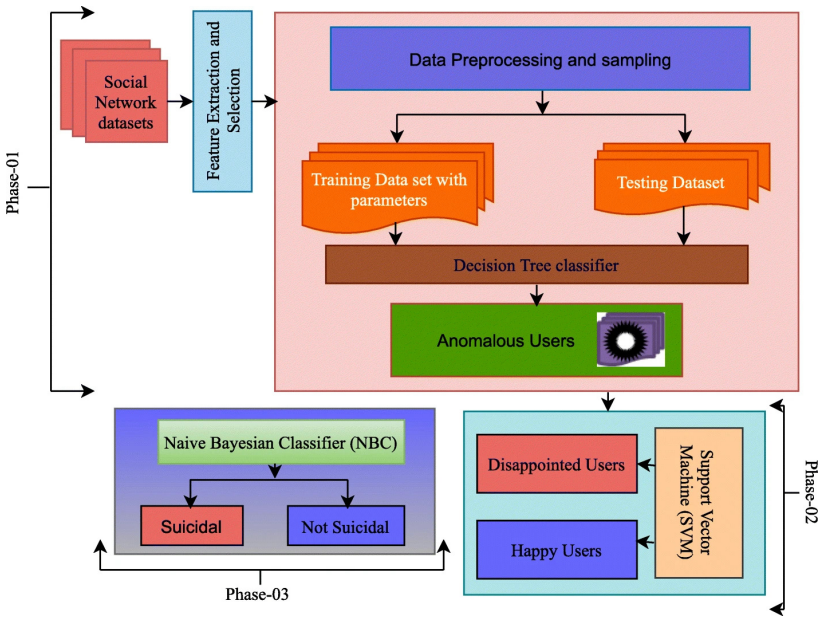


Figure 2.5: Proposed framework for anomaly detection, taken from [17]

The work by Miz et al. [18], instead, is not really focused on OSNs specifically, but it is more related to the web in general. In any case, the results can be applied also to the OSNs' environment, and they are interesting to present. In this research, a scalable unsupervised algorithm, with a distributed implementation, is proposed to detect outliers in dynamic graphs, defining an anomaly as a localized abnormal collective behavior of users in a group (*cluster*) of nodes, during a certain period of time. In other words, an anomaly was an anomalous spike/pattern in the time evolution of the networks, which, as the paper shows, was generally triggered by real-world events. The authors used the Hopfield network model of artificial memory, with the Hebbian learning rule, to combine the network and time information, and they applied this approach to two spatio-temporal dataset: Enron Email dataset January 1999 to July 2002, and Wikipedia page views from September 2014 to April 2015. Through the experiments, the researchers transformed the datasets: for each event or anomaly detected, the proposed model provided a rich and complete spatio-temporal description, which was a group of interconnected nodes (spatial information), where each node had a time series signal (indicating the time when the anomaly occurred) as attribute (temporal information). Therefore, the Enron Email dataset became a temporal network where nodes corresponded to email addresses of employees and they were connected by an edge if employees exchanged an email, and the number of emails sent per day by each employee as the temporal attribute of the nodes. Wikipedia English articles also underwent a transformation,

based on the static Wikipedia hyperlinks network[1] and user visit counts per page per hour, as a node temporal attribute. While highlighting anomalies and extracting the features from the datasets, nodes with similar behavior had a major connection weight, while, to reduce the amount of data, disconnected nodes and nodes with no potential anomaly were pruned in advance. To validate the results, the researchers verified the accuracy of the found anomalous events for the Wikipedia dataset comparing them with the trending topics extracted from Google Trends[2], while the Enron email dataset contained already ground truth and did not need further verifications. The outcomes showed that this approach was effectively able to analyze the sources which caused the spotted anomalies.

2.3 Clustering in Online Social Networks

An Online Social Network (OSN) community can be defined as a set of users (nodes) who interact with each other and participate in some discussions.
The study of the dynamic interactions among these nodes, the period and the way in which such interactions increased or decreased, can provide relevant knowledge about the current events, human behavior and topics of discussion. In the Online Social Network Analysis (SNA), detecting groups of strongly connected nodes and their relations clustering them into communities, may also help to detect, for instance, insider threats, spreading news or rumor in the society, employee attitude, to design target marketing schemes, to identify terrorist cells, or to recognize rivalry interests in social activities. For all these reasons, the problem of clustering of OSNs data has gained a great interest in research, especially in the recent years.
The following subsections and arguments are all related with each other, so the separation of the works is not clear-cut.

2.3.1 Clustering of people

A more generic work is the one by Hajeer et al. [19], where a Genetic Algorithm (GA) is used for Online Social Network Analysis (SNA). Such algorithm exploits biological process as a model, and it was used by the authors in OSNs data to find dense clusters having multi-valued edges, indicating a single user (node) engaged in more than one discussion, and using an edge discharge method based on the context of discussions. They also presented the definition of "Node of Attraction" (NoA), representing the most active node in a network community, and identified as the source of a post/communication which engages other users (nodes) to create a

[1] Each article possibly contains a reference to other articles in the form of hyperlinks.

[2] `https://trends.google.com/trends`

cluster. GA was performed on two distinct dataset, a synthetic one and a real-world one (*Gnutella Peer to Peer Network*, downloaded from the Stanford Large Network Dataset Collection, containing almost 21 thousand edges with a random value associated with each edge), also changing some attribute values during the run, and each obtained "chromosome" represented groups or clusters. From the results, it was observed that the GA was effectively able to form clusters corresponding to each discussion groups, and that some nodes, defined as "linkage nodes", connected two or more independent clusters. These nodes were useful in analyzing the relations between different discussion groups or topics.

Another work related to the clustering of users in OSNs, is the paper by Singh et al. [20], which has some analogies with this thesis project. The final aim of the authors was clustering Twitter users exploiting textual similarity among their tweets. This kind of analysis gives information about the words that are frequently used in a group of people. For the extraction of the data, Tweepy package of Twitter API was used, and afterwards the obtained data were pre-processed: words which contained unnecessary information (*stop-words*) were removed using Lucene[1], then the text was *stemmed*[2] to reduce inflected (or derived) words, using the same library as the previous step, then lexical analysis was done using the *WordNet*[3] lexical database of English to group nouns, verbs, adjectives and adverbs into sets of cognitive synonyms (synsets). At this point, the researchers calculated the *strength* of each couple of users (the result was a symmetric matrix), according to the following rules: strength between two users must be directly proportional to the number of common words between them, inversely proportional to the total number of words used by both the users, and inversely proportional to the difference of the total number of words used by them. Explaining better the last rule, strength should decrease as more frequently the user post a tweet, because the chances of there being textual similarities between the two persons increased, and also with the difference between the occurrences of a word. This approach was applied to two datasets:a "dummy" dataset consisting of 40 nodes and two clusters, and the "real" dataset with 77 nodes. Spectral Clustering, based on the unnormalized graph Laplacian matrix[4], was used to transform the source data into a vector space, and then apply the K-Means algorithm in the same area. From the results it was observed that both simple K-Means and Spectral Clustering algorithm gave almost equal outputs, coherent with the expected outcomes, but the computation cost of

[1] `https://lucene.apache.org/core`

[2] *Stemming*: the process of reducing words to their root form.

[3] `https://wordnet.princeton.edu`

[4] It is a matrix representation of a graph which can be used to find several useful properties of the graph, as the number of its spanning trees.

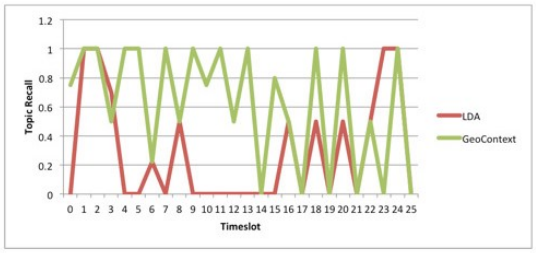
Spectral Clustering for large datasets is very high.

2.3.2 Clustering of posts

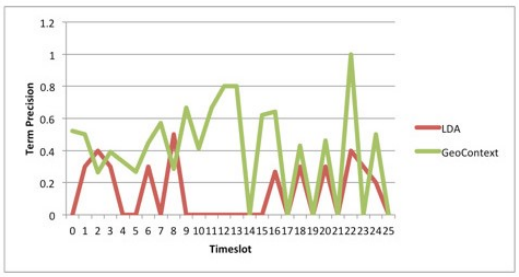
The work by Williams et al. [21] focused the attention on topical clustering of Twitter posts, because many people rely on micro-blogging platforms such as Twitter for knowledge of major events. In this article, two approaches for the topical clustering are compared and applied to two different datasets, to discover the subjects of the users' discussions analyzing the content of tweets. The two analyzed datasets were human-labeled in the contained topics, and they were the following: the Elections dataset, consisting of almost 525 thousand tweets taken from the November 2012, U.S. presidential election, and the Rumors and Truths dataset, consisting of almost 42 thousand tweets concerning various topics. The first approach was GeoContext, a tool to compute the similarity between posts based on semantic text analysis, in contrast with Latent Dirichlet Allocation (LDA), a commonly used topic modeling algorithm that determines topics based on words that appear together frequently. The limitation of the existing methods of topics discovery is the fact they mainly treat all words in the text as having the same value, while GeoContext extracts only terms that are relevant, creating topics ranking words on the basis of the relevance to the search criteria and to the tweet. Moreover, GeoContext works dynamically, does not remove stop-words and does not need training or a prior indication on a fixed number of topics, unlike LDA. To extract significant words from a tweet, GeoContext utilizes two kinds of APIs: AlchemyAPI's Concept Tagging, which abstracts the input text into higher-level concepts, and Keyword Extraction API, which extracts relevant and meaningful words. Afterwords, the algorithm associates each obtained term with a *relevance score* and each couple of tweets with a *similarity score*, clustering together the tweets with a high value of such score (essentially indicating the semantic correlation between tweets). The latter score is calculated first by checking whether the tweets have any hashtags in common and, if not, performing the same test on the extracted concepts and keywords. To evaluate the resulting discovered topics, the authors used three metrics: topic recall, term precision, and term recall (Fig. 2.6). It was finally determined that GeoContext had higher term precision and topic recall metrics, it was able to identify more ground truth topics, to create more topics that had more related terms and that contain the terms from the ground truth topics, than LDA.

2.3.3 Topic Extraction and Event Detection

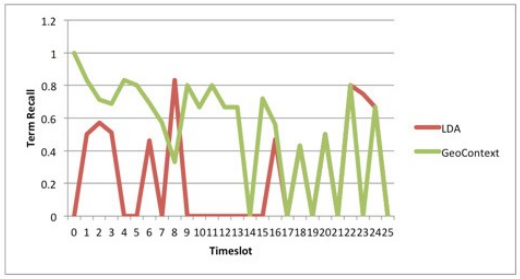
Newer research took aim at applying topic models to posts streams from social media platforms, because discovering topics within Online Social Networks (OSNs)



(a) Topic Recall



(b) Term Precision



(c) Term Recall

Figure 2.6: Metrics across all time-slots, taken from [21]

can help, for instance, in advising content to users based on their interests or in finding major events and occurrences around the world, often before they are reported by journals or other forms of classical media outlets.

The work by Prangnawarat et al. [22] proposes to exploit heterogeneous data concerning the relations among users, posts, and concepts extracted from the post contents, to represent the OSN with some graph based models, with the final aim of clustering the posts by different topics and events. The introduced approach modelled the Twitter network in two different types of networks: the User-Tweet (U-T) Model, a bilateral directed weighted graph between users and tweets, and the User-Tweet-Concept (U-T-C) Model, which extends the former type of network by adding concepts (hashtags, or named entities from Tweet texts, or both) from the tweets' content. Given the graph, after filtering out its small connected components, and the number of clusters as input, the authors used the RankClus algorithm, which provided both clustering and ranking of the tweets (Hyperlink-Induced Topic Search algorithm was used as the ranking function). The model was applied to two different datasets, respectively containing the data descripted in Table 2.1. From the results, in both datsets the same trend were observable, showing that there were still various small connected components that did not interact with the others in the networks, but also discussions among many users. It was also found that taking only top ranked tweets in each clusters, improved the outcomes since much

	Total Users	Total Tweets	Background Tweets
Dataset1	19,127	15,461	9,640
Dataset2	108,770	98,065	92,244

Table 2.1: Datasets from (McMinn, Moshfeghi, and Jose 2013), which proposed a Twitter data corpus, and Wikipedia Current Events Portal to generate a set of Twitter events; statistics taken from [22]

of the noise was removed. The events could be classified into two categories: Local Events, occurred in local areas or in specific user communities, and Global Events, discussed by different communities, even those which did not interact with each other. RankClus algorithm could not capture global events where users were not related, resulting in tweets from the same event assigned to different separated clusters, but in any case using U-T-C models could connect data together more than using just the U-T models (Fig. 2.7).

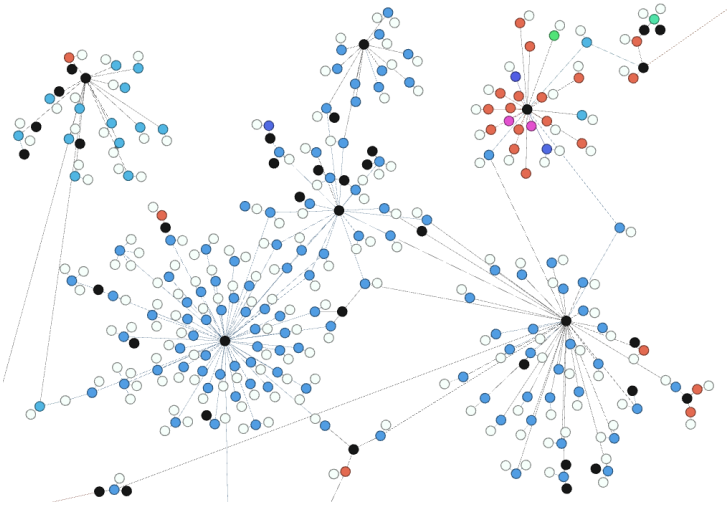


Figure 2.7: U-T-C network: tweets are coloured according to the events in the dataset, users are coloured in white and concepts are coloured in black, taken from [22]

Following this trend, in the same area the work by Shi et al. [23] locates itself, which proposes a user-interest event-evolution model, named the HEE (Hot Event Evolution) model, using the data collected from Twitter through its APIs (almost 127 thousand posts) during eight days period among December 2015 and January 2016, to study the changes in the users' interests during the evolution of salient events. More specifically, the authors used a series of algorithms in cascade: a hot event automatic filtering algorithm to remove the influence of general events, an

automatic topic clustering algorithm to group all short texts with similar topics into clusters, a user-interest model, to combine all the texts in each cluster to create a long unique document, solving the problem of sparse data and improving the determination of the global topic, and, finally, a detection method, based on the cosine similarity measure, to study relatedness between events, detecting the development of such events. The results of the experiments confirmed the efficiency and accuracy of the proposed model for both event detection, providing richer information for the community structure of the detected events, and user interest discovery, during the evolution of significant occurrences. The outcomes were validated and evaluated by comparing them against traditional latent semantic analysis algorithms (as LDA).

Another work, that goes in the direction of Topic Modelling Analysis and needs to be mentioned, is the one by Liu and Jansson [24], which explores an Instagram dataset, by the Digital Georgraphy Lab at University of Helsinki, containing posts and comments of the Helsinki region during three months (June-August 2016), to capture overall English topics and their popularity on the OSN. The data was prior cleaned and pre-processed by tokenization and stop-words, special symbols and less than three characters words removal. Afterwords, the researchers applied LDA modelling methods to the English corpus, comparing the analyses with both posts and comments included, against posts data only, and also understanding the effects of hashtags by including/excluding them. The results showed that it would be safer to include comments for analysis and that eliminating the hash sign (#) from hashtags not only makes the outcome content more coherent but also less redundant in topic terms. It seemed, in fact, that the hashtag sign had the only effect of splitting the words extracted from the analysis into: one topic containing only words, one topic contains only hashtags, and the third topic a mixture of the previous ones. Moreover, setting a large number of topics in the input parameters of LDA, generated many overlapping topics.

2.4 Previous publications by our research group

The study for this thesis project derives from the wider work on Online Social Networks (OSNs) carried out by the research group SmartData@Polito[1]; this center focuses on Big Data technologies, Data Science (from data management, to data modeling, analytics, and engineering), and Machine Learning methodologies applied to several domains of knowledge, finding solutions for both theoretical problems and helping companies toward applications.

An initial work of Data Analytics applied to OSNs is the study by Trevisan

[1] `https://smartdata.polito.it`

et al. [25]: a first research regarding how people behaved and interacted with politicians and personalities on Instagram before the European Elections of May 2019. A custom crawler was used to collect the data used for the analyses: it downloaded and stored data and meta-data about the profiles of top public Italian figures (i.e. influencers), the related activities (i.e. their posts) and the interactions (i.e. users' likes and comments in the first 24 hours after posting time), at the turn of two months. The study focuses on checking if interactions across political figures follow general patterns, and if there are any differences with those ones across profiles of different categories of influencers, such as music, sport and show entertainment. The paper provides also a characterisation of the so-called *mentions*[1], analyzed to quantify interactions among commenters, and the related reactions, subdividing them into four categories: answered/unanswered first mention and solicited/unsolicited *reply*[2]. The results suggested that comments to politicians' posts come from a small group of users that actively participate in discussions. These comments take place for longer time periods, and they are more numerous and lengthier. Moreover, users rarely mention other people when commenting politicians' posts, compared to the posts of other categories of influencers, but these comments attract a large number of replies, most of which not explicitly solicited. This means that users are not dragged into the discussion, but they reply autonomously after reading the previous interactions, possibly with the aim of influencing the online political debate. Differences among profiles of different parties were also witnessed. It is then clear that interactions with political and other categories posts were markedly different at both quantitative and qualitative levels.

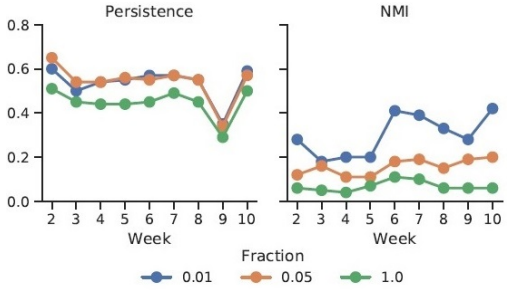
Another paper related to politics and OSNs is the one provided by Ferreira et al. [26], in which the researchers' goal was to study communities of co-commenters[3] to reveal characteristics and dynamics of interactions on the Instagram environment, to highlight common trends as well as particularities, the level of engagement and coordination. To this extent, a null model was designed to extract the backbones of the interaction networks, to obtain salient interactions between commenters while removing random occasional interactions, and, using that, identify communities through the *Louvain* algorithm. The analysis was performed on a dataset crawled from Instagram, containing the activity of several public political (politicians and political parties) and general influencers' profiles (the latter used as a control group), during the electoral periods of Italy and Brazil, subdivided into week intervals (ten weeks in total). Considering all the posts from homogeneous groups

[1]Comments containing explicit references to other users.

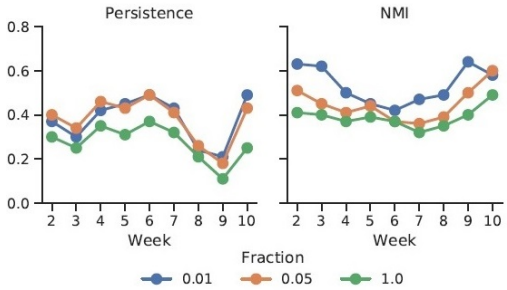
[2]A comment to previous comments.

[3]Commenters that comment on the same post.

of influencers that have been posted in the same week, each network was composed in the following way: the nodes represented commenters, the edges represented comments between any two commenters if both had commented on the same post (co-commenters), weighted as the number of posts in which they both commented. Then the researcher tried to study the similarity of profiles, clustering politicians on the basis of their community structure, and the evolution over time of the backbone of the graphs. Such methodology uncovered some interesting observations:
the backbones of commenter networks were split into fewer and better defined communities, but these communities were weaker in politics than in general topics, even if their participants were more numerous and more active; as expected, political communities had a peak in in the online debate during the elections, while in general topics they were persistent and consistent over time (Fig. 2.8); most commenters with the highest activity levels remained coherently active over time even if communities changed.



(a) Italy - Politics



(b) Italy - General

Figure 2.8: Temporal evolution in Italy of commenters in communities. Blue: top 1%, Orange: top 5%, Green: all commenters, taken from [26]

A work more related to our contribution but not to OSNs, is the one by Soro et al. [27], focused on comparing existing well-known algorithms and Machine Learning (ML) techniques of anomaly detection, applied to banking transactions data, in particular to time series of the transaction history of customers. The

analyzed events, over a period of two years, were: cyclic events, e.g. repeated peaks, single isolated anomalies and salient changes in increasing or decreasing trends, e.g. periodical spikes whose height does not show particular changes over time. The dataset included 50 million transactions and about 500 thousand customers, all from the customers' point of view, and it was filtered by setting a monthly threshold of minimum payments. Then the researchers set different time granularity and studied the periodic phenomena, distinguishing the payments of salaries to employees from those addressed to suppliers, by means of an internally developed algorithm which labeled the transaction using the Random Forest algorithm and a set of input features. After revealing periodicity, trends and cyclic behaviors in per-customer time series with some threshold-based methods, in order to spot isolated anomalous points, the researchers used various techniques. One of them was the Auto Regressive Integrated Moving Average (ARIMA) model (sec. 4.2.1), a technique to forecast time series. This approach provides a prediction of the future values of a time series on the basis of its past values, so, after defining an acceptable confidence interval for the predicted values, the points that fell outside such confidence interval boundaries could be labeled as outliers. Among the supervised techniques, other procedures were some driven heuristics driven by the domain knowledge, such as the definition of threshold-based control criteria, the Support Vector Regressor (with three different kernel functions), the Stochastic Gradient Descent Regressor, the AdaBoost Regressor and the Random Forest Regressor. Among the unsupervised techniques, instead, the researcher considered four clustering algorithms (after evaluating some cluster quality measures to find the best configuration parameters): K-Means (sec. 4.4.3), DBSCAN (sec. 4.4.2), Hierarchical clustering and Isolation Forest (sec. 4.2.3). After that, they detected the anomalies by choosing the points belonging to clusters containing less than a threshold-number of minimum elements, or those recognized as noise points. According to the performance metrics for the algorithms (Tables 2.2 and 2.3), the

	ARIMA	**ADA**	**DT**	**RF**	**SGDR**	**Lin**	**Poly**	**RBF**
Accuracy	0.75	0.88	0.89	0.73	0.51	0.61	0.28	0.54
Precision	0.06	0.19	0.18	0.14	0.08	0.11	0.01	0.03
Recall	1	0.89	0.89	0.89	0.4	0.58	0.79	0.88

Table 2.2: ARIMA and Supervised algorithms scores, taken from [27]

	DBScan	**Agglomerative**	**Isolation**	**K-Means**
Accuracy	0.99	0.98	0.97	0.99
Precision	0.52	0.51	0.19	0.77
Recall	0.63	0.62	0.48	0.8

Table 2.3: Unsupervised algorithms scores, taken from [27]

found results showed that the Support Vector-based algorithms and the Stochastic Gradient Descent Regressor were unreliable, as they had a small recall (they were not good at detecting true anomalies) and presented a very low precision with a large number of false positives. The unsupervised models showed instead a better average behavior, as a consequence of the fact that generally they flag anomalies only if their classification is very sure, leading them to give a better precision.

The COVID-19 pandemic profoundly changed economy, culture, politics, but, above all, the society and, as a consequence, it was important and interesting to study its impact on OSNs. The study by Trevisan et al. [28] was focused on understanding the effects on social life of the total lockdown imposed during the first six months of the year 2020, offline but also online, because OSNs represented an alternative solution to physical meetings. In particular, the work analyzes the changes in Italian influencers' trends of activity patterns, interactions, and engagement in discussions about specific topics on Instagram and Facebook during this historical event. The dataset was built through custom web crawlers and consisted of Instagram and Facebook posts of 639 popular influencers, previously irreversibly anonymized. Then there was a step of *data augmentation* deriving from each comment its psycho linguistic properties (LIWC) and extracting its topic (Fig.2.9). Some interesting variations between the periods before, during

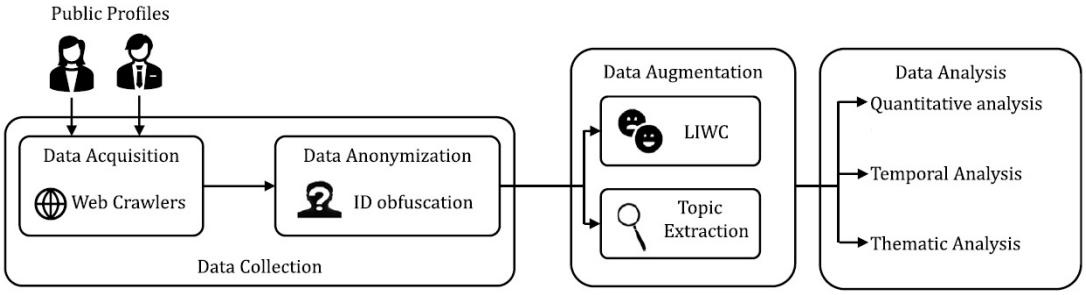


Figure 2.9: Data collection and analysis methods, taken from [28]

and after the lockdown and distinguishing the two social networks were observed: Facebook showed an increment in the number of posts, comments and likes during the lockdown while, on the other hand, Instagram was characterized by a flat trend in terms of posts and comments, while the number of likes decreased significantly. Moreover, the prohibition on social activities during first weeks of lockdown had effects on the hourly patterns: in both social applications an increase in the morning and in the weekend activities and a decrease in the early afternoon and in the evening were witnessed. In the same period, as expected, the researchers observed also an increase in popularity of comments about negative sensations such as anxiety and inhibition, people began to discuss about the health emergency and

personal life, such as unemployment. Analysing the online debate, it was observed that people were engaged less in discussion, especially for politicians, even if they had obtained a great number of new subscribers during those weeks.

Vassio et al. [29] conducted research on how influencer postings attract interactions (number of likes or reactions) and how content popularity increases over time, as well as defining the behavior of influencers and followers over time and the progression of interactions across time, from their peak to the conclusion of a post-life. The researchers looked into the activities of Italian influencers and their followers on Facebook and Instagram for more than five years (from 2016 to 2021). The following are some of the key findings: both influencer and user activity follow a consistent daily pattern, although with a different form; the posts' inter-arrival time has a long-tailed distribution that is reasonably adapted by a log-normal distribution; on average, 50% of user interactions occur within the first 4 hours after content creation on Facebook and after 2 hours on Instagram, and the number of user interactions grows faster on Instagram than on Facebook (however after about 30 hours, the two curves converge); influencers' posts have a short life-span, with exponential temporal decay of the arrival-rate of interactions, lasting between 20 and 50 hours (after which they stop attracting interactions), however the decay rate varies greatly from post to post and depending on the OSN in question; the number of interactions obtained during the first hour or even 15 minutes might give a good indication of the content's popularity (the freshness of the post has a great impact on its attractiveness); because the generation of new posts by the same influencer gradually fades the attractiveness of the original post, it seems that followers tend to focus their attention at the top of the timeline, and therefore the fraction of total interactions obtained in a given time interval is related to the number of posts just published in the same interval. Furthermore, the findings revealed that, independent of the online platform, follower and influencer actions follow similar patterns throughout the two OSNs studied: the activities of followers and influencers diminish overnight, with two peaks during the day, while followers are more engaged in the late evening than influencers. Moreover, Facebook posts have a longer engagement time than Instagram posts: the predicted interaction time for Facebook is 15 hours against 11 hours for Instagram.

One more work to be cited is the one by Bertone et al. [30], demonstrating that the OSNs ecosystem, in which prominent influencers struggle to recruit new followers in order to grow their visibility (value), has parallels to the stock exchange market, where investors choose to purchase a stock, and companies with a large number of investors rise in value. Influencers may be viewed in this light as stocks having a market value that can be grown, as measured by the number of followers. Regular users operate as private investors, following (buying) influencers depending on their personal tastes and data gathered from other sources. This study will use statistical methods from the financial industry to examine the dynamics of

influencers on OSNs, assisting in the development of decision support systems to assist influencers and advertisers in accurately estimating short-term trends.
The data was downloaded using the CrowdTangle tool (chap. 3) and covered a period of more than three years, from November 2017 to March 2021, and included 60 Italian public figures who were active on Instagram (obtained through the analytics platform `www.hypeauditor.com`) and fit into three categories: singers/musicians, athletes, and VIPs. The Google Trends API was used to download historical trends for the set of influencers and collect the Google Trends Search Volume Index (SVI), a monthly granularity time series representing search queries (the name/stage of each influencer was used as a search keyword) normalized over the highest value observed in the considered period. Bollinger bands, which are normally employed in decision support system applications to provide traders suggestions about stock transactions, might assist influencers, advertisers, and social platforms in the OSNs area shape their tactics and compare entirely distinct time measurements dynamically. This approach demonstrated how the short-term trends derived by Bollinger bands for Instagram influencers' followers were similar to those found in external sources, such as Google Trends, similar to what academics have discovered for the stock market. The researchers also discovered that influencers with less followers have the most well-coordinated short-term trends, since they are more likely to get new followers than those with a large following.

2.5 Contribution of This Work

This work differs from those proposed in this chapter (*Related Literature*) as it combines Anomaly Detection with Clustering of posts (not influencers). The application of Anomaly Detection algorithms is aimed at identifying anomalous posts, i.e. posts that differ greatly in the number of likes/comments received compared to the expected engagement, while clustering is not properly aimed at identification of topics within the OSN, but rather to group the posts concerning the same topic, then manually identifying and separating the groups of posts that have received a number of anomalous reactions to causes of external events, exogenous to the OSN. For this purpose, 4 classic Anomaly Detection methods and 4 state-of-the-art algorithms for clustering are proposed, identifying the best pair of methods through cohesion and coherence measures.
To this end, in chapter 3 a description of the raw dataset and its characteristics provides an overview of the data used to perform the analyses. In chapter 4 we define the Anomaly Detection and Clustering problems, providing an overview of their main characteristics. Furthermore, we illustrate both the transformation pipeline to modify the data according to our needs, and the theory of all the techniques used to produce, from the raw datapoints, the solution to our problem.

In chapter 5, on the other hand, the data are transformed and pre-processed further, in order to extract the necessary features for the analyzes and remove all the noise possible. Finally, the performances of the various methods are presented and compared in chapter 6. Being an unsupervised problem (it is not possible to identify a priori the topics to which the posts "belong"), a manual inspection of the results is necessary to verify the content.

Chapter 3

The Instagram Post Dataset

In order to soddisfy the purposes of this research, a big archive of social media data was needed. While Twitter's public data is accessible through its APIs, it can be much more difficult to access platforms such as Facebook and Instagram for scholars. For this reason, our choice fell on the most easily attainable platform for managing three of the most important social networks - Facebook, Instagram and Reddit - and has free access for journalists, academics and researchers. More specifically, the data was downloaded from **CrowdTangle**[1], a Meta-owned tool that tracks interactions on public content from Facebook pages and groups, verified profiles, Instagram accounts, and subreddits. It does not include paid ads unless those ads began as organic, non-paid posts that were subsequently "boosted" using Facebook's advertising tools. It also does not include activity on private accounts, or posts made visible only to specific groups of followers.
In particular, CrowdTangle tracks influential public accounts and groups across Facebook, Instagram, and Reddit, including accounts like politicians, celebrities, sports teams, journalists, media and publishers, public figures, and others. The purpose of the platform is giving greater accuracy and transparency into what's happening on Online Social Networks (OSNs) to help researchers in their activities, such as monitoring a given topic looking for combinations of words and phrases to discover trends and patterns, tracking the activity of public accounts and communities, studying which accounts post the most and who gets the most interactions, i.e. reactions, comments and shares, on specific issues, monitoring performance and identify emerging stories.

[1] `www.crowdtangle.com`

3.1 Selection of the Social Media Platform

Online Social Networks' (OSNs) data, collected by platforms such as Twitter, has enabled a new field of "social sensing", where participatory user-generated content has been harnessed to identify trending events occurring in the real offline world. In contrast to mainly textual channels, the choice for this research fell on the social image sharing platform Instagram (*Introduction*, sec. 1.2.4), to study how this social sensing can exploit multimodal multimedia contents.
The choice to carry out our research on Instagram was made precisely for the characteristics of the aforementioned OSN: it is, in fact, a social network strongly aimed at creating public content accessible to anyone, and consequently easily accessible by real common people, and the features to "measure" the popularity of the posts, and thus detect the anomalous ones, are few and easily identifiable, such as the number of reactions (i.e. likes and comments).
Facebook, for example, provides on the contrary different types of reactions with different corresponding meanings, like the "love" or "sad" or "angry" reactions, which could make the analysis more difficult.

3.2 Characterization of the Dataset

The dataset was retrieved employing the CrowdTangle public APIs and reporting the activities of the top ranked **1611 Italian influencers**, over the course of **six years, from 1 January 2015 to 31 December 2020** (313 weeks): the result of this collection was generating a dataset of **2 036 966 posts** in total. The influencers were selected through the portal "`www.influenceritalia.it`", a network platform which aims at simplifying the choice of the right influencer, for all brands and communication agencies that want to invest in digital campaigns and integrated communication activities, or at helping influencers who want to increase their popularity and visibility. The platform contains politicians, VIPs, institutions and football teams.
The dataset is composed of data objects that represent the posts of the influencers, published in the period mentioned above, and are characterized by numerous and various attributes contained in each record (for every post), that can be subdivided into the following categories:

- **Account attributes**: the information describing the influencer account that published the post, at the time of the sampling by CrowdTangle. They include the account handle, the name, the IDs used by the platform, the subscribers count, the URL of the profile, an information about the verification of the account and some other metrics;

- **Sponsor attributes**: if the post contains some branded content, these characteristics report the descriptive aspects (similar to the previous ones) of the sponsor account;

- **User-regulated post attributes**: they include features depending on the user's intention at the creation time of the post, such as the media type (photo, album, video, IGTV), the publication date, the text in the description.

- **Platform-regulated post attributes**: these features are generated automatically by the platform when creating the post, such as the history of the features of the previous posts published by the influencer, the URL to reach the post, and the expected amount of reactions (likes and comments) that the post will obtain;

- **Popularity attributes**: the statistics defining the post's success in the sense of popularity (reactions actually obtained), which are not under the control of the user, including also a *score*, computed by CrowdTangle by means of some post's analytics, to measure the effectiveness of the post's outcome in terms of interactions received.

3.3 Statistical Analysis and Data Cleaning

The dataset, subdivided as explained in sec. 3.2, can be therefore characterized from different points of view, by analyzing the statistics of the various features present in each record (post).
Before proceeding with the analyses, it is important to specify some preliminary considerations.
After a first check on the correctness of the data, it turned out that a noticeable part of the dataset was composed of posts created by influencers whose followers count was equal to zero. Sampling and verifying their actual number of followers led to the deduction that this event was due to an error, probably caused by the fact that these users started being tracked after the "corrupt" activity and, consequently, some data was not registered correctly. Due to the size of the dataset and difficulties in restoring the correct metrics, the decision was made to remove all those posts whose follower count was zero.
The **purged dataset**, containing **1 396 influencers** and **1 400 697 posts** in total (215 influencers and 636 269 posts deleted from the original data), was used for all the analyses presented and illustrated in the following paragraphs.

3.3.1 Followers Characterization

One of the first analyses was to verify the distribution of influencers' followers at the time of CrowdTangle's sampling. In the following table (Table 3.1), the metrics "Posts Count" and "Influencers Count" are added to the general statistics for completeness.

	Complete dataset	Purged dataset
Posts Count	2 036 966	1 400 697
Influencers Count	1 611	1 396
Mean	682 862	993 053.2
Standard Deviation	1 917 473.7	2 244 731.5
Minimum	**0**	143
Maximum	45 926 689	45 926 689

Table 3.1: Statistical description of the followers distribution

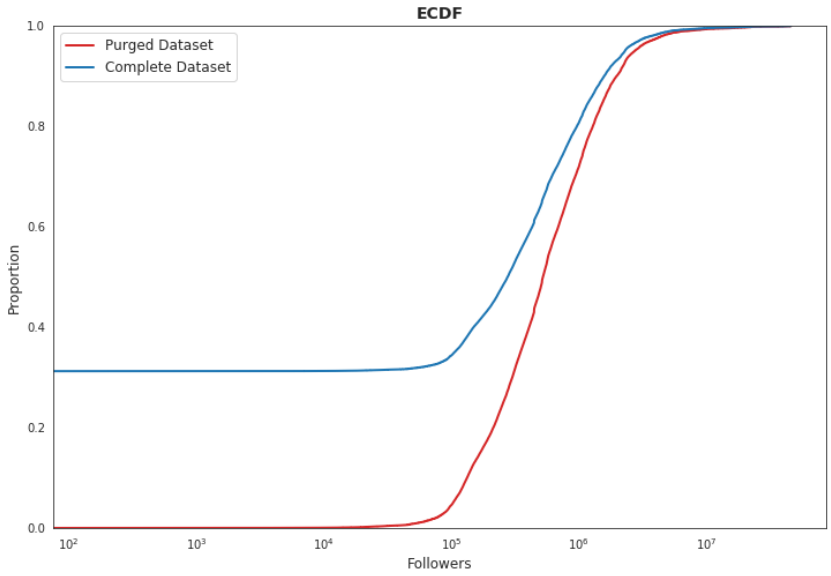


Figure 3.1: ECDF of influencers' followers

Plotting the Empirical Cumulative Distribution Function (ECDF) of followers, which appears Gaussian for the logarithmic scale of the abscissa, as shown in Figure 3.1, it is noticeable that it can be approximated to a log-normal distribution with a peak of half a million and the vast majority less than a million: a predictable event since it's easy to imagine that only a handful of influencer followers can reach a large audience. Looking at the curve regarding the complete dataset, consistent

with the numerical results, it can be seen that about 30 percent of the posts were published by influencers who at that time appeared to have 0 followers.
The figure below (Fig. 3.2) shows instead the ECDF distribution of the number of followers for each class of influencers (for the purged dataset), which have been subdivided by following the schema illustrated in paragraph 1.2.5. In particular,

	Mega inf.	Macro inf.	Micro inf.	Nano inf.
Posts Count	404 284	974 004	22 358	51
Influencers Count	174	1 160	61	1
Mean	2 343 187.7	453 720.2	77 282	5 219.3
Standard Deviation	3 826 194.4	320 629.4	44 058.4	2 679.6
Minimum	44 651	143	2 596	1 110
Maximum	45 926 689	2 238 894	320 012	11 265

Table 3.2: Statistical description of the followers distribution for each class of influencers

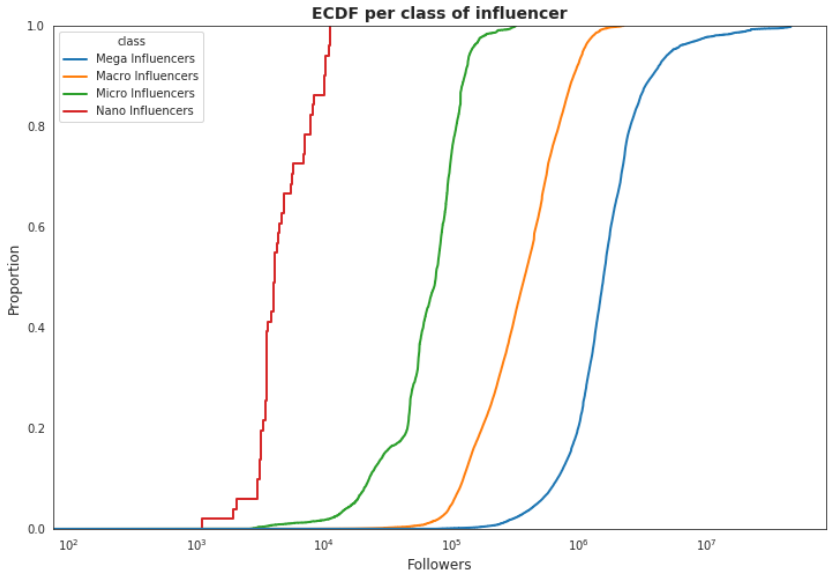


Figure 3.2: ECDF of influencers' followers for each class of influencers

initially we calculated the **average** of the number of followers for each influencer and then, on the basis of this average, we carried out the division into classes, otherwise some influencers could have been part of several classes at the same time if, during the 6 years analyzed, they had passed from one class to another for having their number of followers increased / decreased.
From the plot it is possible to identify that all the curves are approximate, also

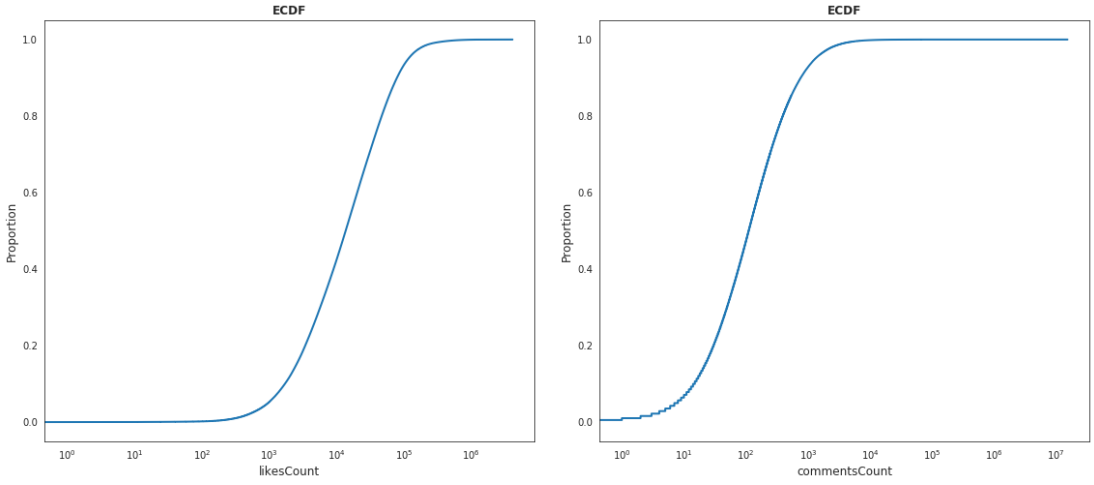
in this case, to a log-normal distribution, respectively, with a peak of 1 million followers and the majority less than 5 million for mega influencers, a peak of 500 thousand followers and a majority of less than one million for macro influencers, a peak of 60 thousand followers and a majority of less than 100 thousand for micro influencers, and a peak of 5 thousand followers and a majority of less than 10 thousand for nano influencers. Naturally, the distributions reflect the split with nano influencers having the fewest number of followers, as was foreseeable by the way the division into classes was performed.

3.3.2 Reactions Characterization

The following essential parameters to analyze were the distribution of the number of reactions, i.e. likes (*passive* attention) and comments (*active* attention), to the published posts, to measure "influence" by inspecting engagement levels on a users' posts [12].

	Likes	**Comments**
Mean	32 568.1	362.1
Median	13 650	110
Standard Deviation	63 900.7	12 894.7
Minimum	**0**	**0**
Maximum	4 083 006	15 169 309

Table 3.3: Statistical description of the reactions (likes and comments) distribution



(a) ECDF of likes **(b)** ECDF of comments

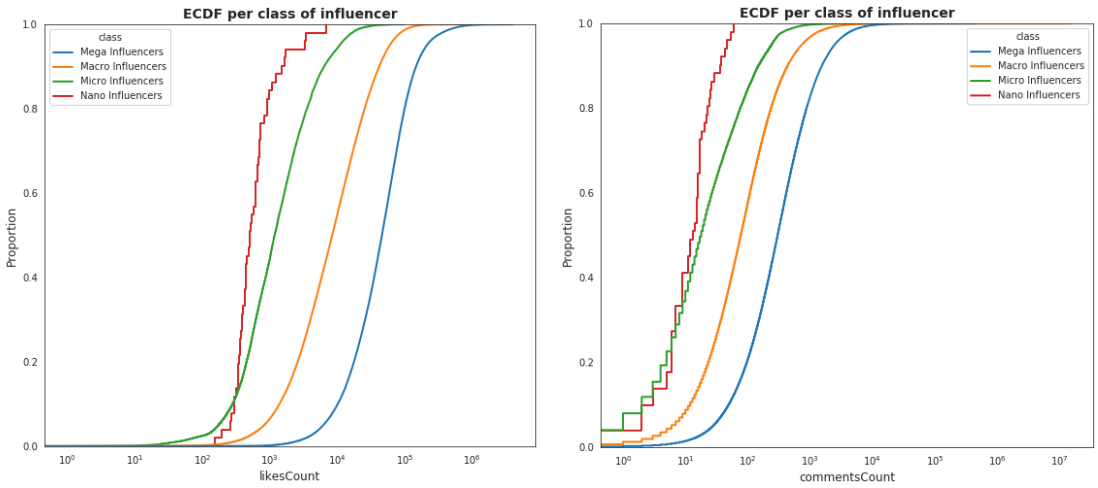
Figure 3.3: ECDFs of reactions to posts

By looking the Table 3.3, it is noticeable that also in the case of reactions, some posts seem to have 0 likes and comments, but they have not been removed from the dataset. Fig. 3.3, presents the ECDFs of reactions to influencers' posts: almost all posts have a smaller amount of comments (the majority is in the range between 50 and 1 000 comments) compared to the number of likes (the majority is in the range between 5 000 and 100 000 likes).

By performing again the subdivision of influencers in classes, the results in Table 3.4 and in the ECDFs plots in Fig. 3.4, seem to show that the various categories exhibit broadly similar patterns, with mega influencers gaining the most and nano influencers gaining the fewest engagement in absolute terms. Also the same trend seen in the general ECDF of reactions (Fig. 3.3), for the difference between the number of likes and the amount of comments, is reflected in this plot.

	Likes - Comments			
	Mega inf.	**Macro inf.**	**Micro inf.**	**Nano inf.**
Mean	72 500.5 - 707.1	16 680.5 - 225.9	2 700.3 - 58.1	837.8 - 15.8
Std. Dev.	103 028.7 - 2 057.2	23 040 - 15 404.2	4 543.1 - 145.8	1 089.7 - 13.4
Min	**0** - **0**	**0** - **0**	**0** - **0**	156 - **0**
Max	4 083 006 - 437 145	859 414 - 15 169 309	99 548 - 8 897	6 980 - 59

Table 3.4: Statistical description of the reactions (likes - comments) distribution for each class of influencers



(a) ECDF of likes per class of influencers **(b)** ECDF of comments per class of influencers

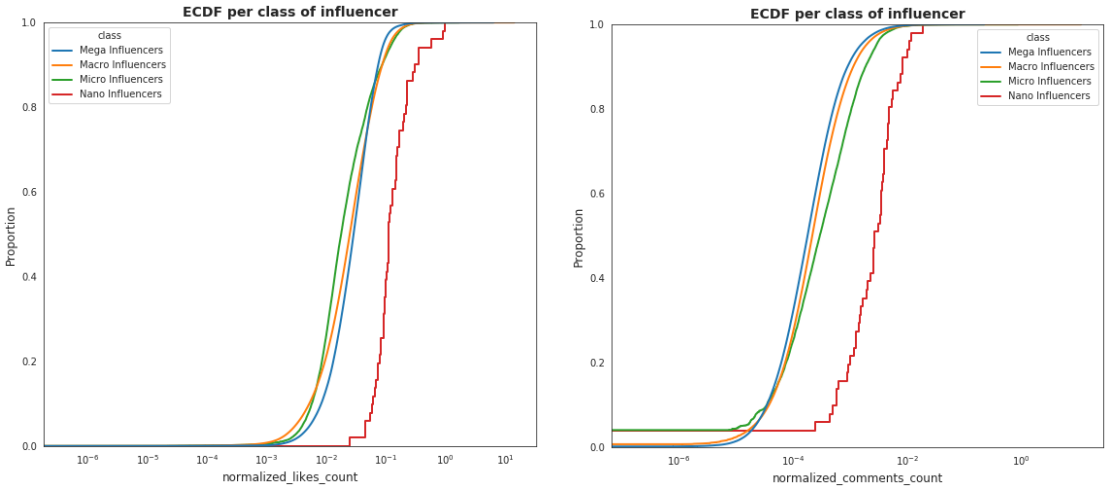
Figure 3.4: ECDFs of reactions to posts for each class of influencers

The above analysis of absolute counts can provide a misleading perspective as influencers with a large number of followers will obviously get higher reaction counts. So, we performed a **normalization** of the comments count as a fraction of the followers count, and plotted the results in Fig. 3.5.
In the statistical description in Table 3.5, some of the metrics for comments are expressed in scientific notation for the sake of brevity and space, because the numbers are very small due to the normalization.

	Normalized Likes - Normalized Comments			
	Mega inf.	**Macro inf.**	**Micro inf.**	**Nano inf.**
Mean	0.04 - $4.3e-4$	0.04 - $5.7e-4$	0.03 - $7.8e-4$	0.17 - 0.004
Std. Dev.	0.05 - 0.001	0.05 - 0.01	0.05 - 0.006	0.18 - 0.003
Minimum	**0** - **0**	**0** - **0**	**0** - **0**	0.02 - **0**
Maximum	6.4 - 0.2	14.2 - 11	1.7 - 0.9	0.96 - 0.02

Table 3.5: Statistical description of the **normalized** number of reactions (likes - comments) distribution for each class of influencers

In the ECDF plot, quite different trends are noticeable, with the nano influencers getting the most engagement, but there are no further significant divergences between the classes.



(a) ECDF of normalized number of likes per class of influencers

(b) ECDF of normalized number of comments per class of influencers

Figure 3.5: ECDFs of **normalized** number of reactions to posts for each class of influencers

3.3.3 Textual Characterization

As previously specified, each data point has features that are measurable and regulated directly by the influencer creating the post. Instagram users can fill the post description, a short text of 2 200 characters, to depict and characterize the content of the post, mention another user or a page, or attach some hashtags.
It is important to notice that, in the below representations, the *Total Words* field includes also hashtags and mentions, but not the text contained in the posts' media (the image) because it is not properly part of the description. The *Simple Words*, instead, exclude both hashtags and mentions, to obtain a more precise analysis.

	Tot. Words	Simple Words	Hashtags	Image Words	Mentions
Mean	26.5	22.7	3.2	1.9	0.7
Std. Dev.	44.2	43.3	5.2	7.6	1.5
Minimum	0	0	0	0	0
Maximum	638	638	60	100	117

Table 3.6: Statistical description of the words distribution, divided by category, for each post

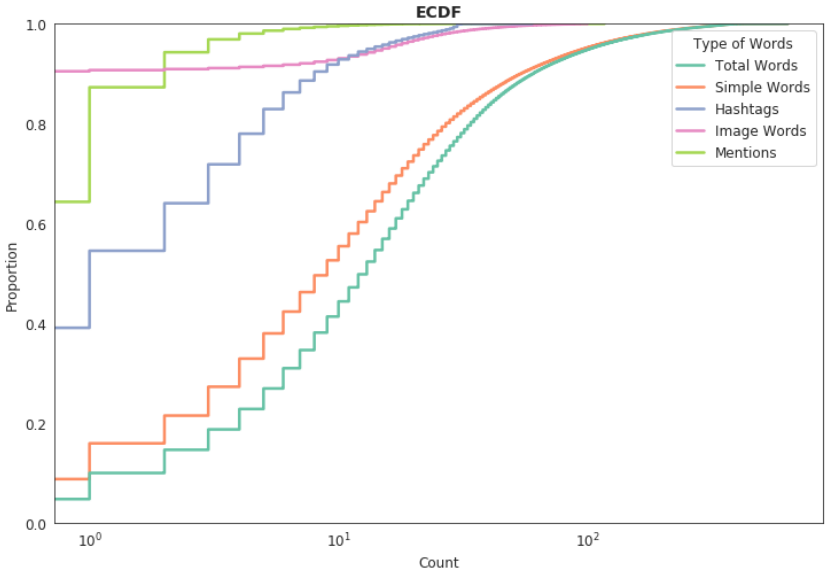


Figure 3.6: ECDF of the post description metrics (caption-words, hashtags, image-text and mentions)

After a quantitative analysis on the posts description, shown in Table 3.6 and in the ECDF plotted in Figure 3.6, it is clear that the vast majority of posts contain very few mentions, hashtags (which lately have been limited by Instagram to a

maximum of 30) and words in the image (90% of the posts contains media without integrated text). This is a foreseeable result because of the nature of this type of words, while the text length (number of *Total Words*), in general, shows a more logarithmic trend.

Operating the usual division of influencers into classes, the results in Tables 3.7, 3.8 and 3.9, and in the ECDFs in Figure 3.7, show that there are minor behavioral differences between the categories of influencers.

Tot. Words - Simple Words				
	Mega inf.	**Macro inf.**	**Micro inf.**	**Nano inf.**
Mean	18.9 - 15.9	29.6 - 25.5	27.2 - 21.9	76.6 - 52.6
Std. Dev.	33.7 - 32.5	47.7 - 46.8	39.3 - 37.3	34 - 31
Minimum	0 - 0	0 - 0	0 - 0	6 - 3
Maximum	453 - 453	638 - 638	401 - 398	149 - 133

Table 3.7: Statistical description of the words distribution for each class of influencers

Hashtags - Mentions				
	Mega inf.	**Macro inf.**	**Micro inf.**	**Nano inf.**
Mean	2.5 - 0.5	3.4 - 0.7	4.4 - 0.9	23.7 - 0.3
Std. Dev.	3.5 - 0.1	5.7 - 1.6	6.2 - 1.9	9.1 - 0.6
Minimum	0 - 0	0 - 0	0 - 0	3 - 0
Maximum	50 - 115	60 - 117	33 - 64	30 - 2

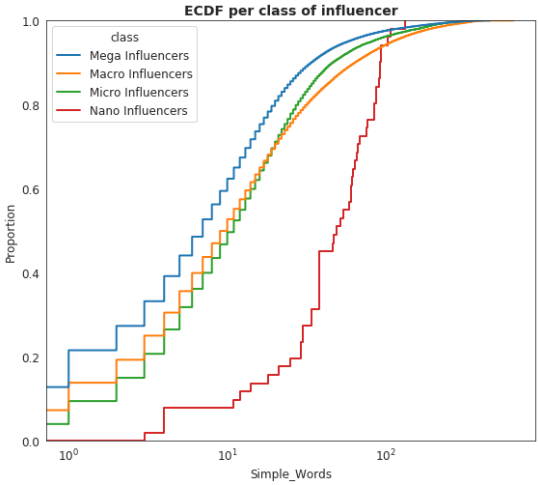
Table 3.8: Statistical description of the hashtags and mentions distributions for each class of influencers

Image Words				
	Mega inf.	**Macro inf.**	**Micro inf.**	**Nano inf.**
Mean	2.2	1.8	0.7	0.7
Std. Dev.	7.7	7.6	4.2	3.7
Minimum	0	0	0	0
Maximum	100	100	97	26

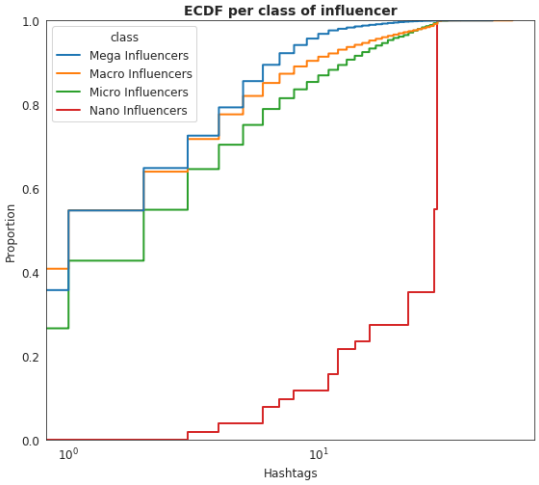
Table 3.9: Statistical description of the post image-words distribution for each class of influencers

In particular, the most important difference regards the nano influencers, which stand out from the others for the number of simple words and, above all, hashtags. Second, after them, are the micro influencers, who also seem to be the ones using the most mentions. These results suggest that "smaller" influencers tend to write

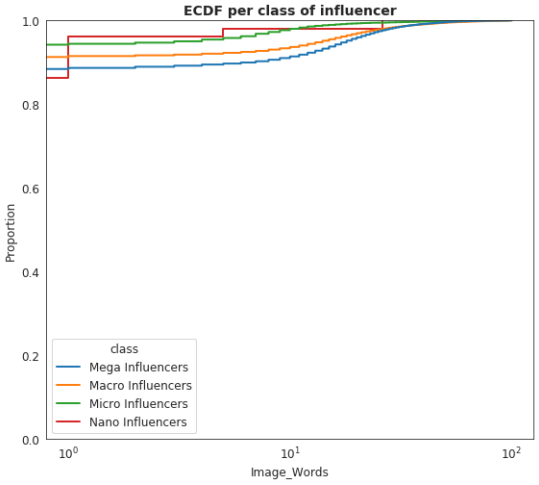
more words than larger ones, possibly to create network effects with other similar influencers (mentions) or to be more easily reachable by new followers (hashtags) than already famous people, while there is no reason to publish media containing more words.



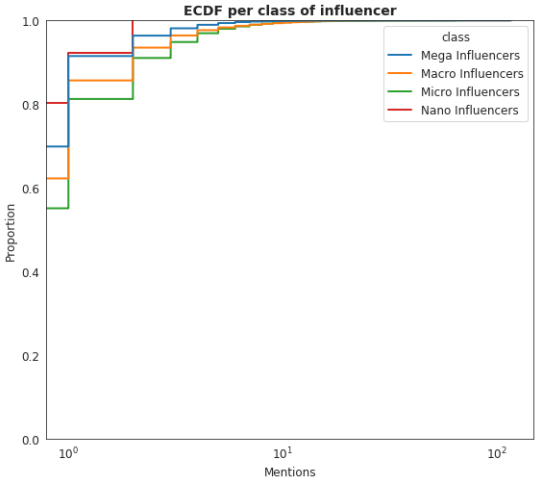
(a) ECDF of number of simple words per class of influencers



(b) ECDF of number of hashtags per class of influencers



(c) ECDF of number of words in the post image per class of influencers



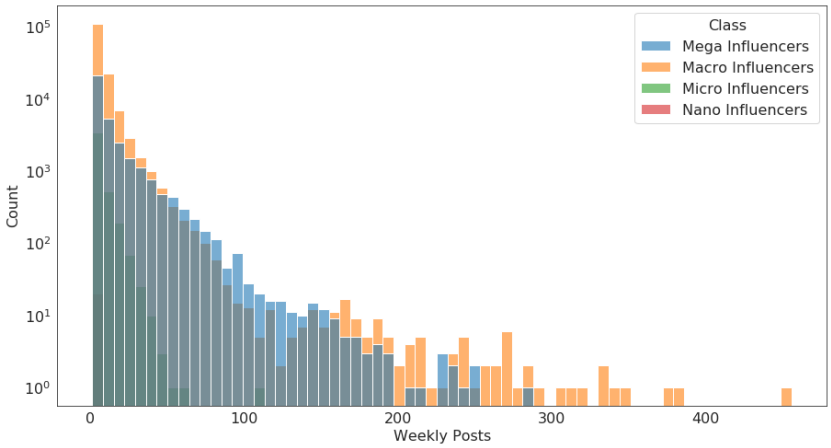
(d) ECDF of number of mentions per class of influencers

Figure 3.7: ECDFs of the post description metrics (caption-words, hashtags, image-text and mentions) for each class of influencers

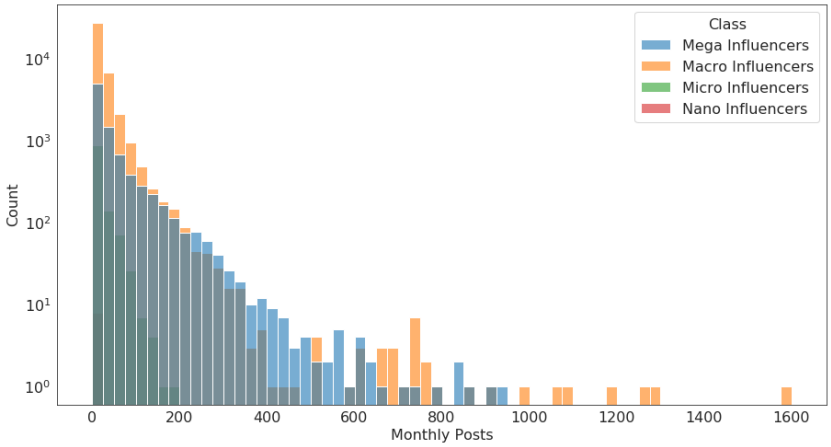
3.3.4 Time Features Characterization

Posting Frequency

Finally, we performed an analysis about the distribution of the number of posts published over time (posting frequency), which provided some interesting insights. From the histograms of frequencies illustrated in Figure 3.8, for the weekly and monthly intervals, it is noticeable that the vast majority of influencers produce a lower amount of posts, in particular much less than one per day.



(a) Weekly Posts



(b) Monthly Posts

Figure 3.8: Histograms of the posting frequencies for each class of influencers

Posting Trends

From the time series plotted in Figure 3.9, it is noticeable that the number of published posts per week increases and decreases over years, with some very low peaks in the last week of year 2016 and year 2020. Macro influencers are the most numerous, so it is natural that they are also those with the highest number of posts published per week. Another relevant detail concerns the nano influencers, which apparently "appeared" (started to be tracked) only around the twentieth week of the year 2020.

Figure 3.9: Posting trend over years for each class of influencers (time series)

3.3.5 Other Features

As already explained in sec. 1.2.4, when publishing a post, an influencer can decide what kind of post to create, choosing from four possible types: photos, videos, albums or IGTV.
Not surprisingly, as shown in Figure 3.10, the most published posts are those containing a single photo, followed by videos, albums and finally IGTV. There are no particular differences between the various classes of influencers, with the sole exception of the nano influencers who have published more than 50% of the posts in the form of videos, followed by albums, photos and then IGTV. In any case, it should be remembered that their posts are just 51, so the numbers involved are still very small.
For what concerns the media information, since there was no interest in extracting the qualitative content of the images or their meaning, the extracted data was not particularly complex or diverse. Beyond the type of the post, CrowdTangle saves

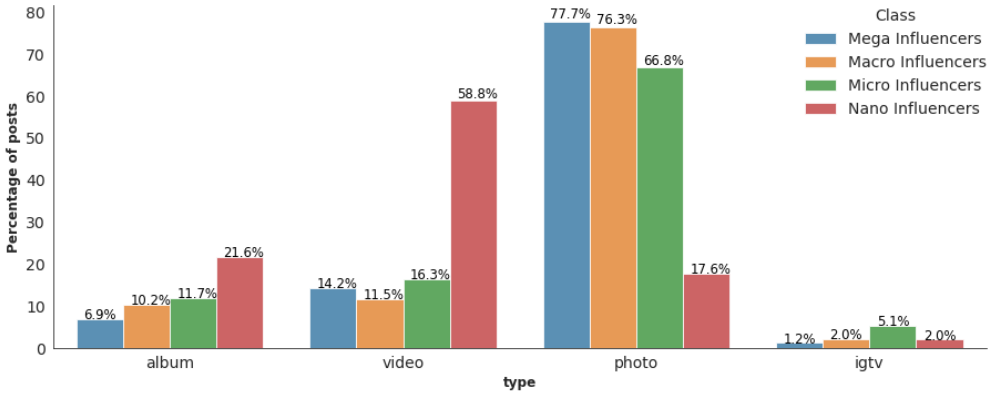


Figure 3.10: Histogram of the post types for each class of influencers: the heights of the bars represent the percentage of posts, relative to the total number of posts for each category of influencers, of the selected type

the text contained in the image, as previously said, the represented images/videos and their dimensions (height and width).

As for sponsorship, it turned out that only 0.66% of posts (in total) are sponsored, but more than 40.97% of influencer accounts have published at least one sponsored post. The class of influencers with the highest (relative) percentage of sponsored posts is the nano one, followed by the macro, mega and then the micro (Fig. 3.11).

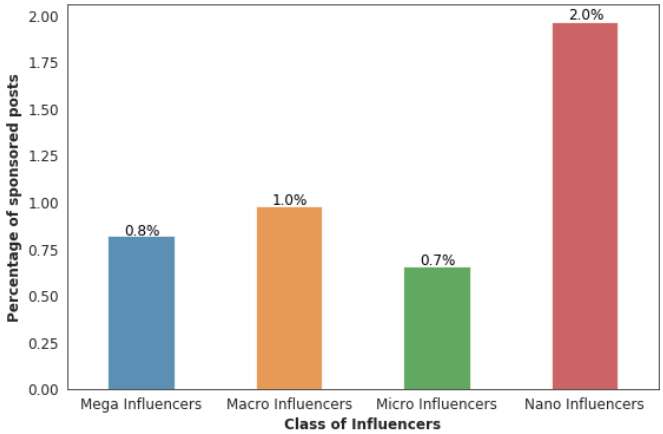


Figure 3.11: Histogram of the relative percentage of sponsored posts for each class of influencers

Chapter 4

Relevant Theory

In the following chapter, the theory relevant to this thesis project will be presented. First, the Anomaly Detection problem will be presented in general terms, as a methodology background. Afterwards, the various Anomaly Detection Methods will be explored: an overview of the most commonly used techniques is provided, describing all their characteristics. Subsequently, after a general presentation of the clustering problem, the algorithms used (and compared) for grouping the obtained results (the found anomalies) in clusters, will be defined. Finally, the approaches used to pre-process data or to extract text-features from the data objects, useful for the clustering methods, will be explained.

4.1 The Anomaly Detection Problem

4.1.1 Definitions

Anomalies are commonly defined as a significant divergence from some expected (*normal*) behaviour. Some scholars defined anomalies, in computer science (data mining), as "patterns in data that do not conform to a well defined notion of normal behaviour" (Chandola et al. [31], 2009), or as "an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data" (Barnett and Lewis [32], 1984). From these definitions, it is then clear that the determination of a "normal" pattern is a key aspect in order to find any abnormal or unexpected behavior in a data instance, and, typically, it depends on the specific problem, the target application domain and the analysis requirements. Therefore, ***Anomaly Detection*** is the research field which aims at finding unexpected patterns in data [33], i.e. uncommon data points, events, and/or observations which do not fit into any established model and differ significantly from the majority of the data. Machine Learning techniques are widely being used to exploit and automate anomaly detection in various application domains to solve classical problems in computer networks, e.g. detection of port scans and DDoS attacks (Intrusion Detection), time series monitoring, fraud detection, fault detection, image processing, event detection in sensor networks, system health monitoring, or pre-processing to remove abnormal data (*noise*) from a dataset improving the accuracy of the analyses results.

Figure 4.1: Different meanings of outliers in data

More specifically, in the literature, Novelty Detection, Outlier Detection and Rare Event Detection are some of the related research fields that are commonly unified together as Anomaly Detection. They all have similar definitions with some differences.
Outliers are values excluded from the residual samples, e.g. they are out of the tolerable variable interval, or far from the norm (expected values). **Rare Events** are commonly defined in the same way as outliers, with the only difference that they embody events that usually occur rarely. Finally, **Novelties** represent a (perpetual) behavioural variation of a data instance. Novelty Detection's aim is catching if a new sample can be defined as anomalous or not, comparing it to the past history: the previously unobserved novel behavior-pattern could become the

new "normal" after the abnormal alteration stabilizes, and could be integrated into the model of regular behavior.

4.1.2 Characterization of the Problem

Each anomaly detection problem can be characterized on the basis of various aspects, on which the applicability of an algorithm depends: the kind of the **input** data, the **type** of anomaly, the accessible **labels** for the learning stage and the **output** format (Fig.4.2).

As explained above, the key challenge of Anomaly Detection is defining an

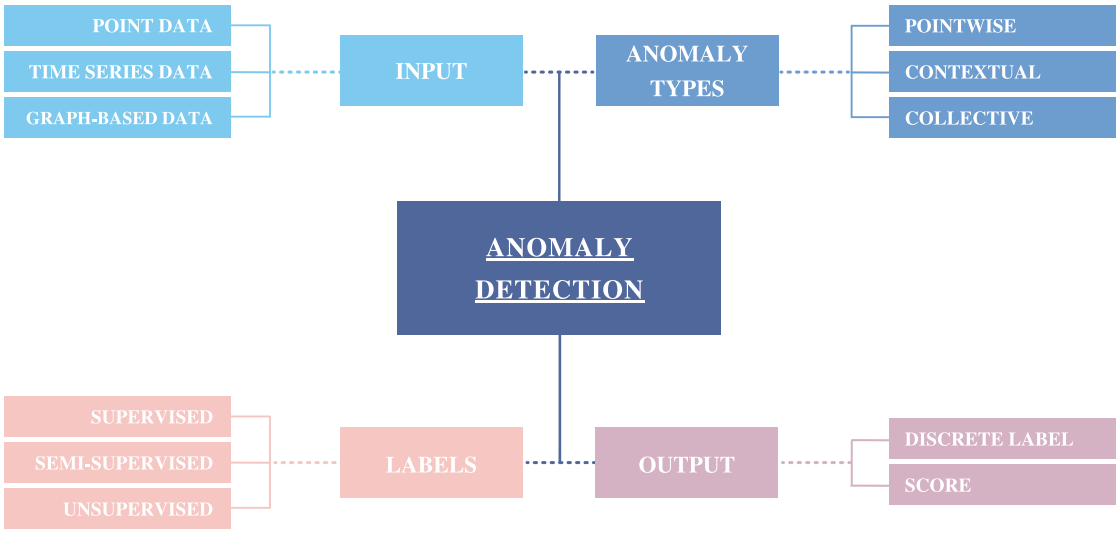


Figure 4.2: Characteristics and properties of Anomaly Detection

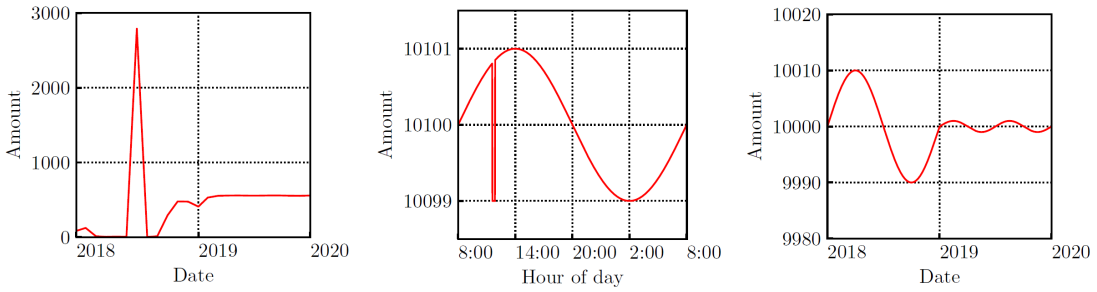
appropriate set of features, to be observed in order to differentiate regular and anomalous behavior, but, to understand whether an algorithm is applicable, also the nature of the input data, the relationship among instances and their attributes must be taken into consideration. *Point data* refers to instances with no relationship, *time series* are instances recorded over time and possibly linearly ordered as *sequential data*, *graph-based* data refers to instances correlated by any other generic relatedness criteria.

The anomalies themselves can be classified into three macro-categories (Fig.4.3), independently on the type of data:

- ***Pointwise anomalies***: isolated data objects (i.e. data points), also referred to as *global anomalies*, that deviate from the norm in a dataset (Fig.4.3a). Similarly are defined also *local anomalies*, which are not considered with respect to the entire set of data, but only to their local close neighbors (e.g.

in a graph). In time series analysis pointwise outliers can be *uni-variate* or *multi-variate*, depending on whether they affect one or more time-dependent variables, respectively;

- ***Contextual anomalies***: data instances that are considered anomalous in a certain specific context but, possibly, would be labeled as normal if taken in isolation (Fig.4.3b). An example may be temperature, which is considered as abnormal or not depending on the time and location (*contextual attributes*, that define the context of a data object) and on its characteristics, such as temperature measure, humidity, etc. (*behaviour attributes* of a data instance);
- ***Collective anomalies***: anomalies whose joint behavior is unusual, formed by the collection of various data instances as a whole, which depict a different pattern with respect to the entire dataset, while the individual data points may not be anomalous (Fig.4.3c).



(a) Pointwise anomaly **(b)** Contextual anomaly **(c)** Collective anomaly

Figure 4.3: Examples of anomaly macro-categories, considering the time series of a numeric attribute, taken from [33]

It is noticeable that point anomalies can occur in any type of dataset, collective anomalies can occur only if data instances are related with each other with some criteria, contextual anomalies can be detected if context information are available in the data.
Another method of classification distincts anomalies into *static*, occurring with respect to the rest of data ignoring the time factor, and *dynamic*, which arise compared to the past behaviour. Anomalies can be further classified into: *white crow* anomalies, which occur when a data point differs very significantly from other observations, and *in-disguise* anomalies, considered as a hidden small deviation from the normal measures [34] (Fig.4.4).

In addition, another important aspect to consider is related to the presence of labels, which indicate the availability or not of some ground truth for recognizing

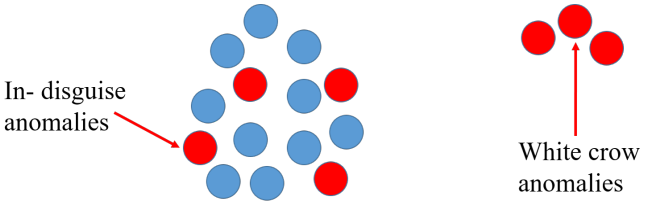


Figure 4.4: In-disguise and white crow anomalies

the data behavior, for allowing the usage of (*supervised*) algorithms that learn anomalous templates from labeled datasets, and for validating the results. Therefore, depending on the presence of this ground truth, there are three deriving manners of defining an Anomaly Detection problem:

- ***Supervised***: both normal and anomalous classes of data instances are known;
- ***Semi-supervised***: only the regular behavior is known;
- ***Unsupervised***: neither normal nor abnormal classes of instances are known.

After the application of the appropriate Anomaly Detection algorithm, the forecasting results (*output*) could be reported in two different formats: a *discrete label* indicating whether the test data instance is normal or anomalous, or an anomaly *score*, defined on the basis of the nature of the problem (e.g. an arbitrary measure of distance from the expected value for ordinary instances) and depending on the magnitude to which that instance is considered an anomaly.

4.1.3 Anomalies in Online Social Networks

In Online Social Networks (OSNs) research, the study is centered on the interactions between pairs of individuals, on the relationship between these interactions and the attributes of the involved users, and on the effects of such interactions in the system. Therefore, it is natural that OSNs can be viewed as a graphical structure with vertices (nodes) and edges (links) representing the people and their interactions respectively. This characteristic separates Anomaly Detection in OSNs from classic, non-network based analyses.
In this context, other types of anomalies can be identified. One of these is named *horizontal anomaly*, and it is based on the different data sources available: for example, a same user may be present in different communities and may have similar friends on different OSNs, but totally distinct kinds of friends for another social network [34]. In addition, putting together the previous anomalies characterizations (sec. 4.1.2) in such network environment, it is possible to distinguish the following

types of anomalies in OSNs, based on static/dynamic nature of network structure and on information available in such structure [34][35]:

- ***Static unlabeled anomalies***: the behavior of an individual or group of people remains static, leading to anomalous network structures, and the labels on vertices and edges (attributes) are ignored. Only the occurring of an interaction is relevant, and, to this extent, it could be useful to make hypothesis about the likelihood of interaction between pairs of users;

- ***Static labeled anomalies***: the changes in the network structure together with the labels, in combination, are taken into consideration for the definition of an anomalous pattern. They are often used for spam detection;

- ***Dynamic unlabeled anomalies***: interaction patterns that evolve over time, so that the (dynamic) network structure at a certain time differs considerably from that in previous time intervals. Anomaly Detection approaches for general time series can be used in this case;

- ***Dynamic labeled anomalies***: observed by considering labels of the vertices and edges, the graph structure at fixed time passages and treating each of them as for a static network;

It is noticeable that all OSNs are dynamic, but, in any case, usually they are analyzed if they were static.
When it comes to OSNs, anomalies can also be classified based on [34]:

- **Graphical properties and structural operations**:
  - *Insertion*: the presence of unexpected nodes or edges in the graph;
  - *Modification*: the existence of unforeseen labels on nodes or edges;
  - *Deletion*: the absence of expected nodes or edges;

- **Interaction patterns in the structure**:
  - *Near Stars/Cliques*: the presence of totally disconnected (stars) or all linked (cliques) peers;
  - *Heavy locality*: an heavy weight around a particular zone or a community;
  - *Particular dominant edges*: unusual heavy load at a certain node or edge, with respect to the rest of the network;

4.1.4 Classical Approaches

A dataset/network can contain more than one type of anomaly. The above classifications are important because the most significant methods of Anomaly Detection, applied in Data Mining and Social Network Analysis (SNA), are based on the kind of anomaly to be detected. The most prominent method of subdivision of the main approaches for Anomaly Detection discussed in the literature [34][33][31] is the following.

Supervised Methods

Supervised methods model both normal and abnormal patterns. It involves Anomaly Detection as a classification problem with data that has been pre-tagged, perhaps manually by some experts, as normal, so that the remaining data is considered abnormal, or, conversely, pre-labeled as anomalous, in such a way that the remaining data is considered to be regular. These methods are based on the assumption that anomalies are usually very less with respect to the number of normal data instances (this assumption is common to many methodologies), and on considering more important to detect precisely as many anomalies as possible than avoiding false positives. The major problems are due to unbalanced class distributions, because the training data contain far fewer anomalous items, compared to the number of instances belonging to the normal class, and, in addition, obtaining detailed and descriptive labels, especially for the anomaly class, is usually difficult.

Semi-supervised Methods

Semi-supervised methods are used when there are labels on the normal / abnormal behavior of only a few data items, rather than for the whole dataset. More specifically, generally they assume that the training data has labeled instances only for the normal class and, not requiring labels for the anomaly class, they result to be more widely applicable than supervised techniques. From the small amount of labeled data, a classifier can be built, that constructs a model for normal data instances and then attempts to label the unlabeled data, identifying as anomalies the objects that do not fit the model. This approach is called self-training, but there is another one that can be employed, called co-training, in which two or more classifiers train each other. The problem with semi-supervised techniques is associated to lack of labels for anomalous data, from which derives the difficulty of building a model in charge of predicting every kind of possible anomaly.
The most important approaches for this category are summarized as follows [34]:

- ***Proximity*-based**: each data instance is analyzed relative to its neighborhood, assuming that normal data objects follow a similarity pattern, while anomalous objects are far away from their closest neighbors. Various measures can be used

to calculate closeness between vertices, which lead to the further classification of proximity-based methods into two categories:

  - Distance-based: the anomaly score is computed using the distance of a data point from its k neighbors;
  - Density-based: the anomaly score using the relative density of each data point.

  These methods are simple and can be applied to a large number of domains, the only key-challenge is the choice of an appropriate distance/density measure, but when it comes to datasets characterized by areas with markedly differing densities, or by group of anomalies close to each other, Anomaly Detection becomes challenging. An example is the DBSCAN algorithm with its numerous variants;

- ***Clustering*-based**: anomalies are assumed to belong to small sparse clusters, or to be far away from their closest cluster, or to be not included in any cluster at all, while normal instances are assumed to be part of large and dense clusters.
  The major merits of these techniques concern the fact that they do not require a set of pre-labeled data to be applied, and, once the clusters are built, it is easy and fast to compare them. The issues, instead, are related to the computational complexity associated to the construction of clusters for large datasets, when they are built before the detection of anomalies, and to the possible false alarms (data instances not belonging to any cluster could be just noise rather than anomalies) or missed anomalies (anomalous objects having a similar behavior could form big clusters and hence be considered as regular data). An example is the K-Means algorithm, used for Anomaly Detection;

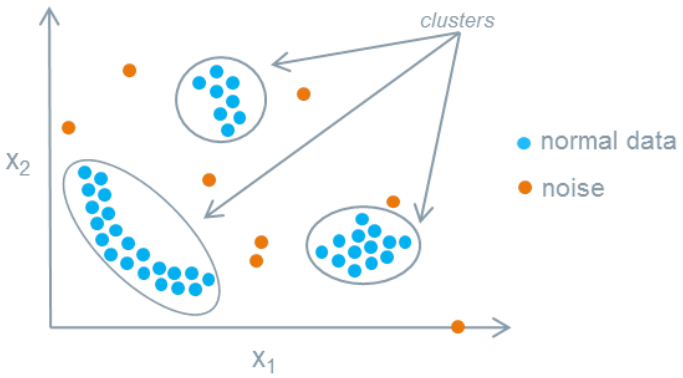


Figure 4.5: Example of clusters and some noise data, taken from [36]

- ***Classification*-based**: similarly to supervised methods, they include a learning phase (*training*), in which labeled data instances are used to build a model, and a classification phase (*test*) in which that model is used to predict the data class-labels (normal/abnormal). Classification-based techniques can use either a one-class model, where the classifier is built to define just the normal class (all those data objects that belong to that class are defined as regular, while those that do not fall into that class are identified as anomalies) or a multi-class model (the available data instances belong to several classes rather than to a single one).
  On one hand, these approaches usually embody a fast process, and they result to be efficient especially when integrating various classifiers. On the other hand, the disadvantages of these methods relate to the dependence and reliability of the training data, which, if not adequately available, can lead to loss of performance, and with the possibility that only a few items may belong to the principal class. Some examples are the Bayesian classifier (NBC), SVM and neural-network-based classification methods.

Unsupervised Methods

Unsupervised methods are generally exploited when no predefined labels are available in the data, and, not requiring any training data, they are the most widely applicable. They are often studied as a clustering problem, assuming that normal instances can be clustered into distinct groups having some similarity patterns. The problem is that this hypothesis is not always true, since sometimes it is the exact opposite, i.e. anomalies are those that can be grouped according to similar patterns, or maybe they can be noise rather than anomalies. Another assumption can be made in this case, stating that normal instances are far more frequent than anomalies in the test dataset, but if this is not true, the techniques will product false alarms. For these reasons, these methods are not very efficient, and they tend to generate a large number of false positives. Basically, the clustering-methodology usually consists in finding clusters based on a similarity between objects, and elements not belonging to any cluster are classified as anomalous (similarly to semi-supervised clustering-based techniques). Since the number of anomalies present in a dataset is rather less than with normal data objects, this usually results in a high computational cost.
Many semi-supervised methods can be adapted to work in an unsupervised mode by employing a sample of the unlabeled dataset for the training stage, assuming that the test data contain very few anomalies and relying on the robustness to these few anomalies of the learnt model. In any case, techniques in semi-supervised mode perform better than those in unsupervised mode in terms of false negatives (anomalies considered as normal instances), since the probability of an anomaly

forming a close neighborhood in the training dataset is very low.

4.2 Anomaly Detection Methods

In this section, the Anomaly Detection methods used in this thesis project, which are those most frequently described in the classical literature for this field of study, are proposed and explained in general terms. The choice of the algorithms fell on both supervised and unsupervised learning methods. In the next chapters (chap.6) more details will be provided on how they have been adapted to the dataset (chap.3) and the research requirements.

4.2.1 Auto Regressive Integrated Moving Average (ARIMA) for Anomaly Detection

In time series analysis, an Auto Regressive Integrated Moving Average (ARIMA) model is one of the simplest and effective supervised Machine Learning algorithm for time series forecasting, and it is an abstraction of an Auto Regressive Moving Average (ARMA) model. Both of these models are fitted to time series data to predict future points in the series (*forecasting*) or to better analyze the data. The main assumption is that the information in the past values of the time series can alone be used to predict the future values.
On one hand, if enough input is available, ARIMA models could suggest hidden patterns, therefore **outliers** can be targeted. On the other hand, with limited data, the quality of prediction would be lower, and so would the accuracy of Anomaly Detection. More specifically, the time series models use the data in the training step, finding the general behaviour and capturing different standard temporal structures of the data, and then attempt to provide a forecasting. If an observation is normal, the prediction would be as close as possible to the actual value, otherwise the forecast of an anomaly would be as far as possible to the true value (Fig.4.6). Therefore, for Anomaly Detection purposes, an examination of the prediction errors is necessary.

ARIMA models need stationary time series as input. According to the Wold's Decomposition Theorem, they are theoretically satisfactory to describe steady broad-sense stationary time series and profile the data, but the applicability only to stationary time series is one of the most serious drawbacks of the algorithm. In some cases, ARIMA models can be applied even when the data is non-stationary in the mean function (but not in variance/autocovariance), by applying an initial differencing phase one or more times, before fitting the ARMA model to the data, to eliminate the non-stationarity. The AR part of ARIMA describes the analytical part of of the signal, denoting that the developing variable of interest is *regressed* on its own (*auto*regression) past values (*lag*). The MA part describes the noise content

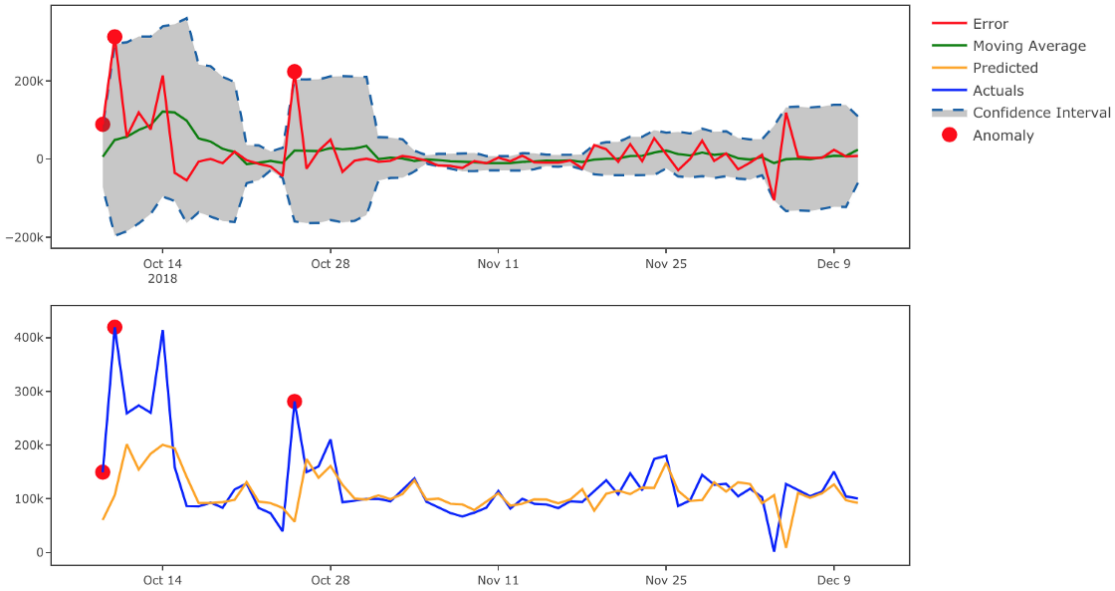


Figure 4.6: An example of Anomaly Detection using Time Series Forecasting, for the sequence of a generic metric, taken from [37]

of the signal, specifying that the regression error is actually a linear combination of the past forecast error values that took place simultaneously and at various times. The I (for "integrated") indicates that the data values have been replaced by a differencing process, with the difference between their values and the past values (possibly more than once).
Non-seasonal ARIMA models are generally connoted as ARIMA(p, d, q), where the terms p, d, and q are non-negative integer parameters: p is the order (number of time lags) included in the autoregressive model, d is the degree of differencing (the number of times that the raw observations are differenced), and q is the order of the moving-average model (the size of the moving-average window).

Stationarity and Differencing

A time series, which is a sequence where a metric is recorded over regular time intervals (depending on the frequency, monthly-wise, weekly-wise, hourly-wise, etc.) and associated to a timestamp, is defined "stationary" when its properties do not depend on the time at which the series is observed: the mean and the variance/autocovariance remain constant over time. In general, a stationary time series will have no predictable long-term patterns: time series with trends, or with seasonality, are not stationary because these properties alter the value of the series at different times (negatively affecting a regression model). Time graphs will show the series to be approximately horizontal (although some cyclic behaviour is

possible), with constant variance (Fig.4.7).

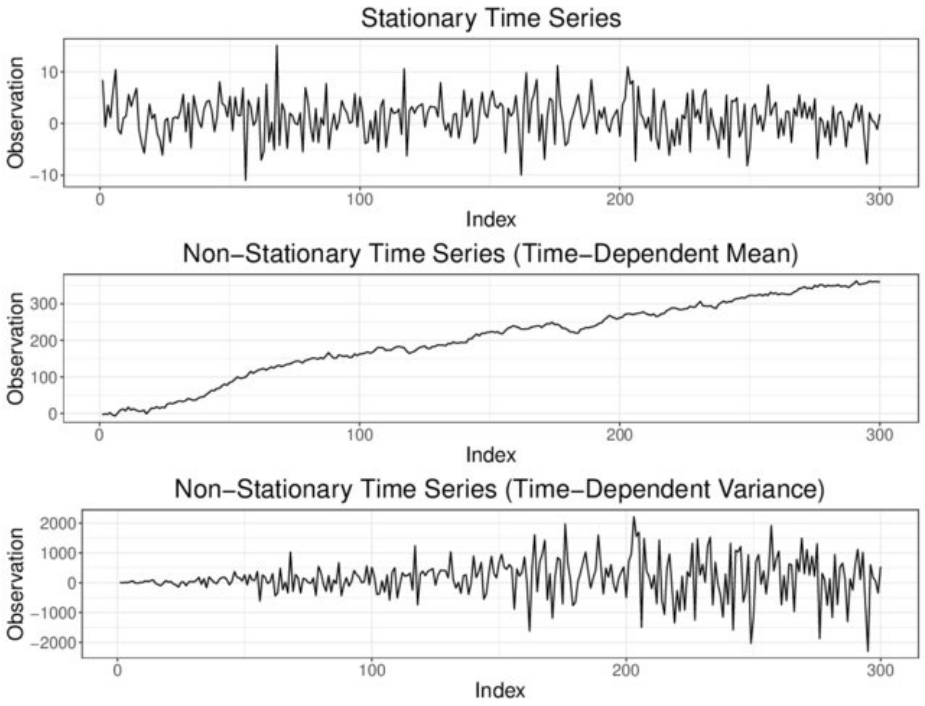


Figure 4.7: Examples for stationary and non-stationary time series, taken from [38]

The observations in a time series data are auto-correlated: observations are highly related to their previous observations. For this reason, as well as looking at the time plot of the data, the Auto-Correlation Function (ACF) graph is also useful to recognize non-stationary time series. For a stationary time series, the ACF will drop to zero relatively quickly, while, for non-stationary data, it slowly decreases. To difference the data and make a time series stationary, the subtraction between consecutive observations is computed. Mathematically, this is formally denoted as:

$$y'_t = y_t - y_{t-1}.$$

Differencing removes the variations in the level of a time series, removing (or reducing) trend and seasonality and consequently stabilizing the mean of the time series. Occasionally the differenced data will not appear to be stationary and it may be necessary to difference the data more than once to obtain a stationary series:

$$\begin{aligned} y''_t &= y'_t - y'_{t-1} \\ &= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\ &= y_t - 2y_{t-1} + y_{t-2} \end{aligned}$$

It is almost never necessary to go beyond second-order differences and apply the differencing process more than a second time. The right order of differencing is the minimum difference required to obtain a nearly stationary series that hovers around a definite mean and the ACF graph reaches zero fairly quickly. Likewise, the *seasonal* differencing, i.e. the difference between a measurement and the previous observation from the same season, is applied to a seasonal time series to remove the seasonal component.

The Akaike Information Criterion (AIC)

To determine the order of a non-seasonal ARIMA model, a useful method is the Akaike Information Criterion (AIC), which is an estimator of prediction error and thereby relative quality of statistical models for a given dataset. When showing the process that generated the data, the representation with a statistical model will almost never be exact, because some information will be lost. AIC estimates the relative amount of **loss of information** due to the use of a given model: the less information a model loses, the higher the quality of that model. AIC provides a trade-off between the quality and rightness of fit of the model and its simplicity, dealing with both the risk of overfitting and the risk of underfitting. In a collection of models for the data, AIC estimates the goodness of each model, relative to each of the other models. Therefore, AIC offers a means for model selection: the aim is to **minimize the AIC** value in order to choose the model that minimizes the information loss. It is computed as:

$$AIC = \frac{-2}{log(L)} + 2(p + q + k),$$

where L is the likelihood of the data, p is the order of the auto-regressive part, q represents the order of the moving average part, and k is the intercept of the ARIMA model. For AIC, if $k = 1$, then there is an intercept in the ARIMA model ($c \neq 0$) and if $k = 0$, then there is no intercept ($c = 0$).

4.2.2 Boxplot Rule

In descriptive statistics, a boxplot is a standardized way to visualize the distribution of data based on the five-number summary: the minimum, the maximum, the sample median, and the first and third percentiles (quartiles). Boxplots are a type of chart often used in informative data analysis.
The body of the boxplot consists of a "box" - hence, the name - which goes from the first quartile (Q_1) to the third quartile (Q_3), and a vertical line is drawn at the median of the dataset (Q_2) within such box. The longer the box the more dispersed the data, the shorter the less dispersed the data. Two horizontal lines, named

whiskers, extend from the front and back of the box: the front one goes from Q_1 to the smallest non-outlier point in the dataset, and the back one goes from Q_3 to the largest non-outlier point. If the dataset contains one or more anomalies, they are plotted separately as points on the chart (Fig.4.8).

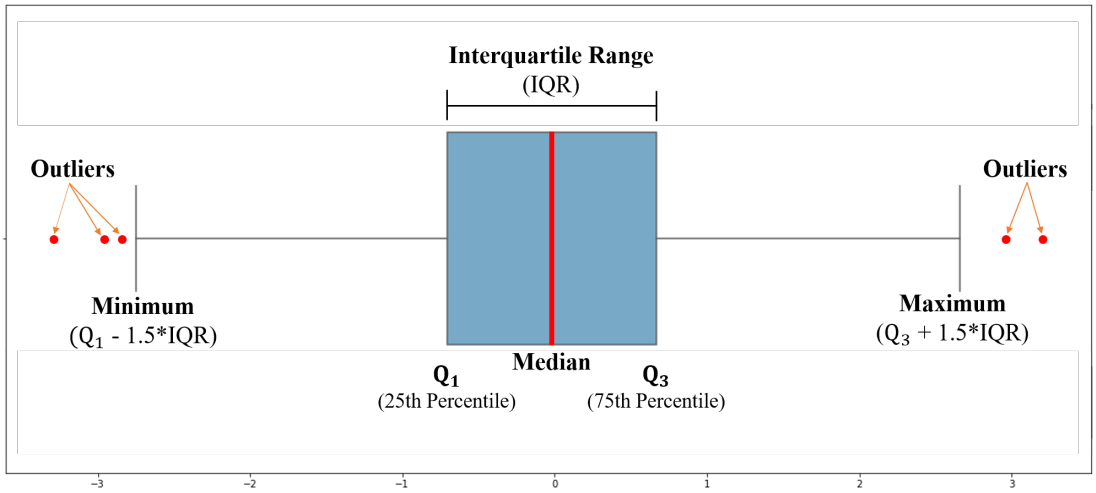


Figure 4.8: Different parts of a boxplot

- **Minimum** (Q_0 or 0th percentile): the lowest data point in the dataset excluding any outliers, $Q_0 = Q_1 - 1.5 * IQR$;
- **Maximum** (Q_4 or 100th percentile): the highest data point in the dataset excluding any outliers, $Q_4 = Q_3 + 1.5 * IQR$;
- **Median** (Q_2 or 50th percentile): The average value in the dataset
- **First quartile** (Q_1 or 25th percentile): the median of the lower half of the dataset;
- **Third quartile** (Q_3 or 75th percentile): the median of the upper half of the dataset;
- **Interquartile Range** ($IQR = Q_3 - Q_1$): the distance between the upper and lower quartiles;
- **Outliers**: the numerical data that are less than $Q_1 - 1.5 * IQR$ or greater than $Q_3 + 1.5 * IQR$.

The spacing in each subsection of the boxplot graphically indicate the degree of dispersion (*spread*) and asymmetry groups of numerical data, i.e. showing whether

the data are symmetrical and how tightly the data are grouped, and allow to visually measure various L-estimators, in particular the interquartile range, the mid-hinge, the range, the mid-range and the trimean. If the data is normally distributed, the various parts of the boxplot will be equidistant.
When the median is in the center of the box and the whiskers are roughly long the same on both sides of the box, then the distribution is symmetrical.
When the median is closer to the bottom of the box, and if the back whisker is shorter, then the distribution is positively skewed.
When the median is closer to the top of the box and if the front whisker is shorter, the distribution is negatively skewed (Fig.4.9 and 4.10).

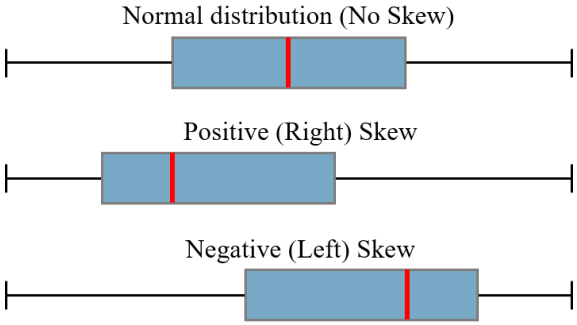


Figure 4.9: Possible different symmetries in the data, represented by the boxplots

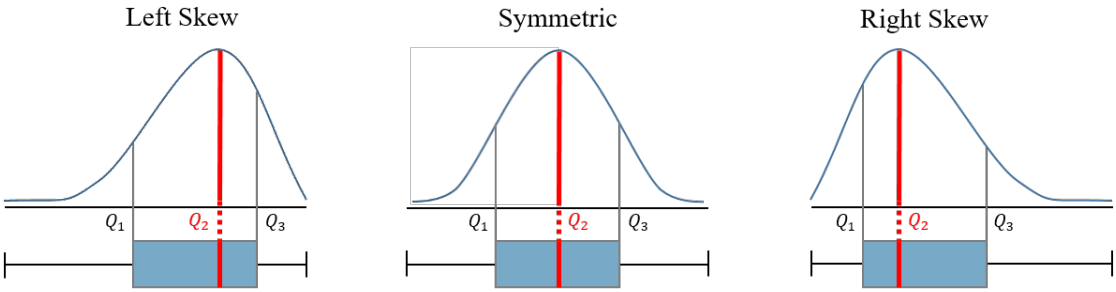


Figure 4.10: Different skews with corresponding distribution plots and boxplots

4.2.3 Isolation Forest

The Isolation Forest is an unsupervised (since there are no pre-defined labels) Anomaly Detection algorithm, which directly targets **anomalies** explicitly isolating them, without the need of previously profiling all the normal instances. Similarly to the Random Forest algorithm, it is based on decision trees.

To isolate a data point, the algorithm performs partitions on the sample by selecting a random feature (from the set of all N features) and then generating a random value divided by such attribute (a *threshold*), in the interval between the minimum and maximum values allowed for the selected feature. If the value of a data point is less than the selected threshold, it goes to the left branch, else to the right. This random branching is generated recursively, until each data point is completely isolated or until the maximum depth (if defined) is reached. In mathematical formalism, recursive partitioning can be represented by a tree structure - the Isolation Tree (iTree) - and the number of divisions needed to isolate an observation can be viewed as the length of the path, within the tree, starting from the root node until reaching the terminal node. After an ensemble of iTrees (Isolation Forest) is built, the training of the model is complete.
Given an anomalous point, random partitioning produces significantly shorter branches, as it was easier for the tree to separate it from other observations. Therefore, when a forest of random trees collectively produces shorter paths for particular samples, it is very likely that they are anomalies, while, similarly, the samples that travel deeper into the tree are less likely to be outliers as they required more cuts to isolate them. It is observable that the architecture of iTree is comparable to the structure of Binary Search Trees (BST): a termination to an outer iTree node is the same as a failed search in the BST (Fig.4.11).

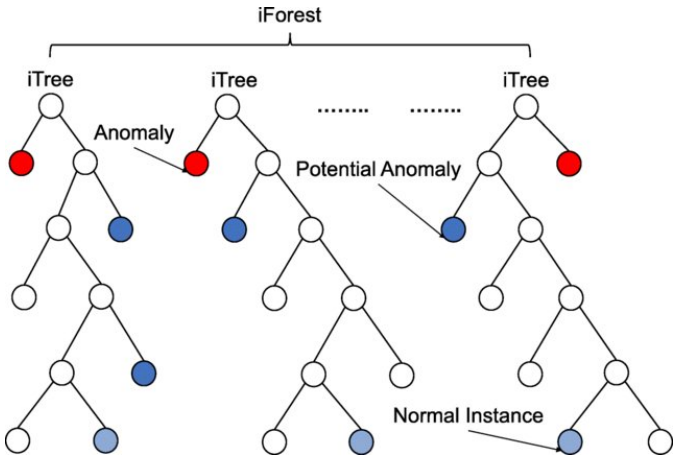


Figure 4.11: Overview of the Isolation Forest algorithm. Light blue circles represent common normal samples, dark blue circles indicate uncommon normal samples (potential anomalies), and red circles stand for outliers. Taken from [39]

More specifically, given a sample, the algorithm calculates an anomaly score for each data point, which is traversed through all the trees that were trained previously, based on the provided *contamination* parameter (percentage of outliers existent in the dataset). This score is an aggregation of the depth, required to

arrive at that point, obtained from each of the iTrees: if the calculated score is close to 1, the point is most likely an anomaly, if it is less than 0.5, the sample is likely is a regular value. If all instances in a dataset are assigned an anomaly score of approximately 0.5, it is safe to assume that the dataset does not contain any anomalies.

The algorithm has the following properties:

- It has a low linear time complexity, so it is computationally efficient, and requires small memory;
- It is able to process high-dimensional data with irrelevant features;
- It can be learned with or without anomalies in the training set;
- It can provide detection results with varying degrees of granularity without re-training.

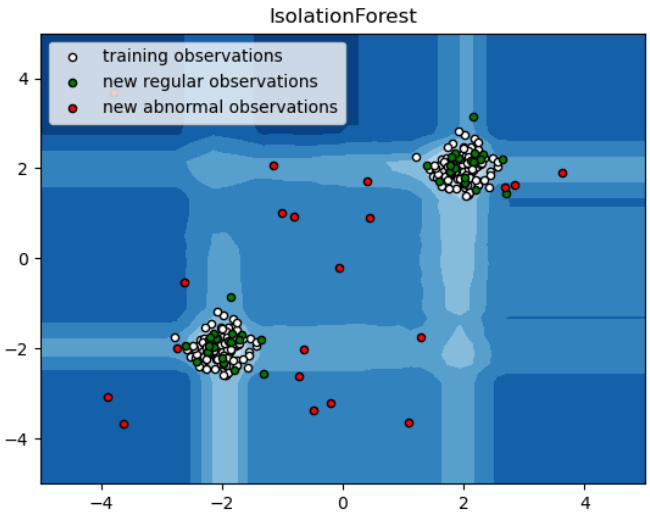


Figure 4.12: An example of usage of the Isolation Forest algorithm by SciKit Learn for Anomaly Detection. Taken from [40]

In addition to the advantageous properties described above, there are also a few limitations of the algorithm to be mentioned. Since the final anomaly score depends on the contamination parameter, provided while training the model, it is clear that a knowing about the percentage of anomalous data is required to obtain a better prediction. Moreover, the model suffers from a bias due to the manner in which the partitioning takes place. In any case, despite these restrictions, Isolation Forest

has been proven to be very effective in various fields for Anomaly Detection, so it is widely exploited for this kind of research.

4.2.4 Z-Score for Anomaly Detection

Z-Score, also called standard-score, is a statistical measure used to determine the distance of a data value with respect to the mean, showing and how far away it is from the rest of the dataset. In a more technical term, Z-score, which can fall in the interval $[0, +\infty)$, tells how many standard deviations away a given observation is distant from the mean of the distribution. To compute such score, the method involves subtracting the mean from the value of the data point, and then divide by the standard deviation:

$$Z = \frac{x - \mu}{\sigma},$$

where μ represents the mean and σ is the standard deviation.
In a normal distribution (Fig.4.13), it is estimated that:

- 68% of the data points lie between ± 1 standard deviation;
- 95% of the data points lie between ± 2 standard deviation;
- 99.7% of the data points lie between ± 3 standard deviation.

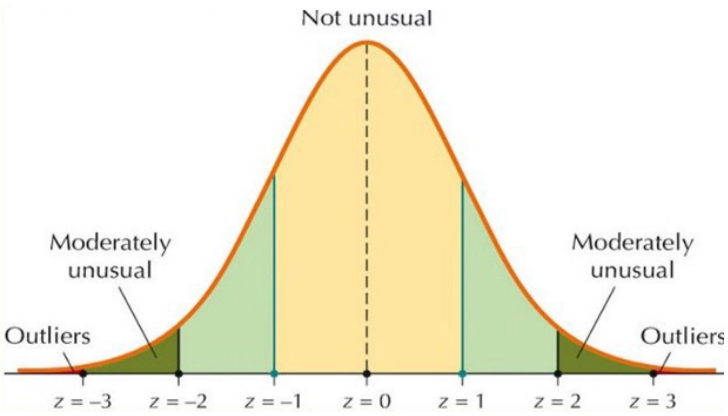


Figure 4.13: Outlier Detection with Z-Scores on a Normal Distribution, taken from [41]

For this reason, the first three integer values of the Z-score, 1, 2 and 3, are commonly used.
In other words, Z-scores can quantify the anomaly magnitude of an observation when the data follow the normal distribution: each data point has its own Z-score,

indicating the number of standard deviations above and below the mean where its value falls, and the further away an observation's Z-score is from zero, the more unusual it is. For example, a Z-score of 2 denotes that an observation is two standard deviations above the mean, while a Z-score of -2 indicates it is two standard deviations below the mean. A Z-score equal to zero represents a value corresponding to the average.
A standard cut-off value to look for outliers is Z-scores of ± 3 or further from zero, so outliers are detected by setting a threshold on the score, using **three standard deviations**: every data object that lies beyond the upper limit $(3 * \sigma)$ and lower limit $(-3 * \sigma)$ is identified as an outlier.

4.3 The Clustering Problem

4.3.1 Definitions

Clustering is the unsupervised, semi-supervised, and supervised undirected[1] data mining technique which provides a classification of patterns of *similar* kinds into respective categories (groups/clusters), without prior knowledge of the group definitions. More specifically, it is the task of grouping a set of items (e.g. data elements) in such a way that elements in the same group or class (a ***cluster***) are more "similar" (this concept depends on the specific problem) to each other than to those in other groups or classes (Fig.4.14).

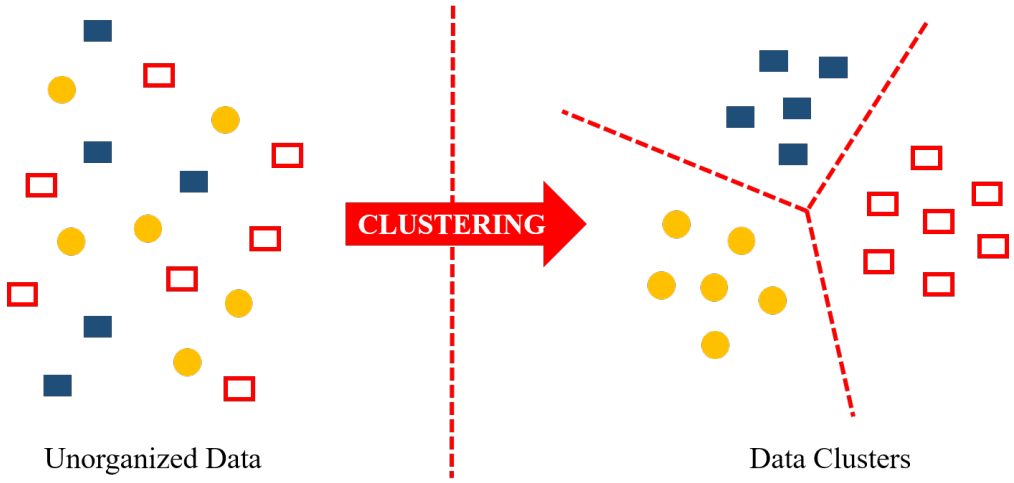


Figure 4.14: Basic example of clustering

[1]The goal of non-direct data mining is to discover the structure of the data as a whole.

A mapping is provided between the set of input vectors to a fixed set of discrete labels indicating the total number of class types. This task has been addressed in many contexts and research-fields, including pattern recognition, information retrieval, image analysis, Machine Learning and more others. The phases of a typical cluster analysis process include sequentially pattern representation, the selection of an appropriate similarity/dissimilarity (generalized as *proximity*) measure, the choice of the most conform clustering algorithm, the evaluation of the output, and the illustration of the final clusters [42].

4.3.2 Characterization of Cluster Analysis

Cluster analyses can be approximately categorized as:

- ***Hard* clustering**: each data observation belongs to a cluster or not;
- ***Soft*** (or *fuzzy*) **clustering**: each data item belongs to each cluster to a certain magnitude, for example according to a probability measure;

There are also finer classifications possible, such as, for example:

- ***Strict partitioning* clustering**: each data object belongs to exactly one cluster, the ones that do not fall into any cluster are considered outliers;
- ***Overlapping*** (*alternative* or *multi-view*) **clustering**: data objects may be members of more than one cluster at the same time;
- ***Subspace* clustering**: on the contrary, clusters are not expected to be overlapped in any case;
- ***Hierarchical* clustering**: data items belonging to a child cluster are also contained in the parent cluster (Fig.4.15);

4.3.3 Classification of Classical Methods and Models

Cluster analysis encompasses different methods, models and algorithms, the most typical of which include:

- ***Connectivity models***: models based on **distance** connectivity, e.g. **hierarchical clustering**. They are based on the key assumption of data items being more similar or connected to nearby objects than to the ones farther away, thus they link instances to form clusters based on their distance. According to this hypothesis, clusters can be described completely by the maximum distance needed to connect elements inside of them. Different distances lead

to the creation of different clusters, which can be represented using a dendrogram, which shows the merges or partitions made in each subsequent stage of the analysis: the y-axis indicates the distance at which the clusters merge, while the data instances are placed along the x-axis thus the clusters do not mix (Fig.4.15).

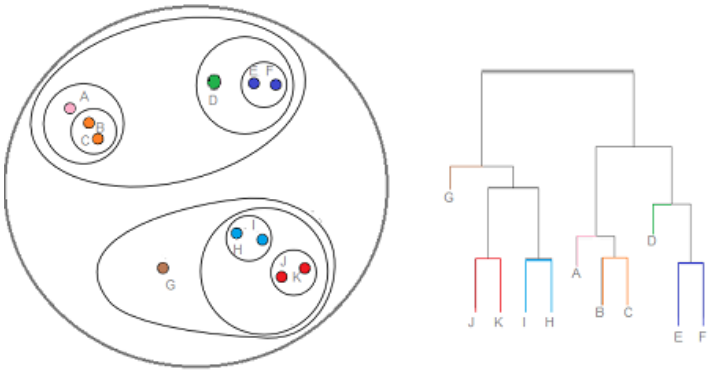


Figure 4.15: Example of hierarchical clustering: a dendrogram (right) representing nested clusters (left), taken from [43]

These algorithms, rather than a single partitioning of the dataset, seek at building and providing an extensive hierarchy of clusters that join with each other at specific distances. Data is not partitioned into a particular cluster in a single step, but a series of partitions takes place. They are not very safe and effective towards outliers, which will either be grouped in additional clusters or even lead other clusters to combine. Finally, Hierarchical Clustering can be subdivided into *agglomerative* methods, which proceed by series of mergers of the observations into groups, and *divisive* methods, which subsequently separate the items into finer groupings [44];

- ***Centroid models***: each cluster is showed as a central vector, which is not necessarily an instance belonging to the dataset, such that an observation in a cluster is closer (more similar) to the "center" of a cluster, than to the center of any other cluster (e.g. the K-Means algorithm, sec. 4.4.3). Most algorithms, associated with this category, require a fixed number of clusters – k – to be determined beforehand, which is considered to be one of the biggest inconveniences of these algorithms. The center of each cluster is usually a ***centroid***, a weighted average of all the data points in the cluster, or a ***medoid***, the most "representative" data point of a cluster. Moreover, they prefer clusters of roughly similar size, as they will always attach an instance to the nearest centroid, often causing erroneous cut borders of clusters;

- ***Distribution models***: clusters are profiled as instances belonging most likely to the same statistical distribution, such as multivariate normal distributions for the expectation-maximization algorithm. They usually suffer from the problem of overfitting, unless the model complexity is bound, but, if a mathematical model can be defined, they are able to capture correlation and dependence between attributes;
- ***Density models***: clusters are defined as connected regions of higher density than the rest of the dataset, while instances in sparse areas are usually considered as noise and border objects. Some examples are the algorithms of DBSCAN (sec. 4.4.2), which is the most popular one, and OPTICS;
- ***Graph-based models*** (sec. 4.4.1): the clusters are identified by the so-called *cliques*, subsets of nodes in a graph, such that each set of nodes is densely connected internally and can be grouped separately from the others (Fig.4.16).

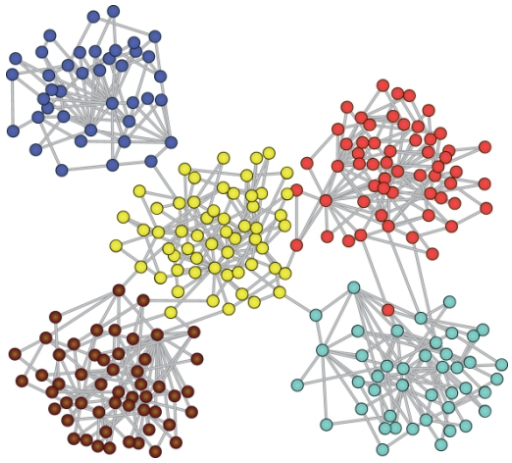


Figure 4.16: Example of graph-based clustering, taken from [45]

4.4 Clustering Methods

As discussed in sec. 4.3.3, clustering algorithms can be classified based on their cluster model. The following section provides an overview of the clustering algorithms used for this thesis, which are the most important and well-known ones. There is no objectively "correct" clustering algorithm, but as noted, "clustering is in the eye of the beholder" [46]: the most appropriate clustering algorithm for a particular problem often has to be determined experimentally, unless there is a mathematical proof to prefer the choice of one cluster model to another.

4.4.1 Graphs and Community Detection Algorithms

A ***graph*** is a set of elements named ***nodes*** (or *vertices*) that can be connected together by lines named ***edges*** (or *arcs* or *links*). More formally, a graph is an ordered pair $G = (V, E)$ of sets, with V set of nodes and E set of edges, such that the components of E are pairs of elements in V. If E is a symmetric relation, then the graph is said to be *undirected* (or indirect), otherwise it is said to be *directed* (or oriented). ***Community Detection***, also known as *graph clustering*/partition, is a common problem in graph data analytics and an important task in many scientific domains, such as sociology, biology and computer science (disciplines where systems are often represented as graphs), that has been extensively studied in the literature. It is also a growing branch of interest in the area of Online Social Networks (OSNs), that can be naturally viewed as a graphical structure representing the users and their interactions, helping to reveal the hidden relations among the nodes (users or posts) in the network. According to M. Girvan and M. E. J. Newman, two popular researchers in the domain of community detection, the aim of such discipline is to partition vertices in a complex graph into densely-connected nodes, the so-called communities (organization of vertices in clusters), with few edges connecting components outside of the group (joining vertices of different clusters), as shown in Figure 4.17.

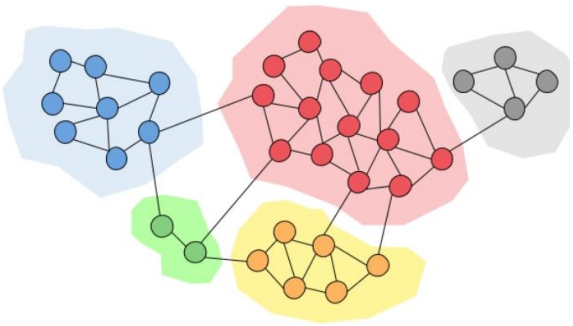


Figure 4.17: Example of community detection in a graph, taken from [47]

Such clusters can be considered as quite independent subsections of a graph, which play a similar role. Overlapping communities, where vertices are in more than one cluster, are also allowed.

Modularity

The ***modularity*** is a benefit function that quantifies the quality of division of a network into modules (the communities), by evaluating the density of the connections between nodes within a community, compared against the connections

that would be if edges were distributed randomly in the network. Therefore, graphs with high modularity have thick links between the nodes inside communities, but sparse connections between nodes in different communities.
However, it has been shown that modularity suffers a resolution limit and, therefore, it is unable to detect small modules. Despite of its known drawbacks, one of the most widely used methods for optimizing community detection is modularity maximization.

Modularity is mathematically defined as the difference between the fraction of the edges that fall within the given communities and the expected fraction if edges were in a random network, and its value lies in the range [−0.5,1]. Some examples are provided in Figure 4.18.

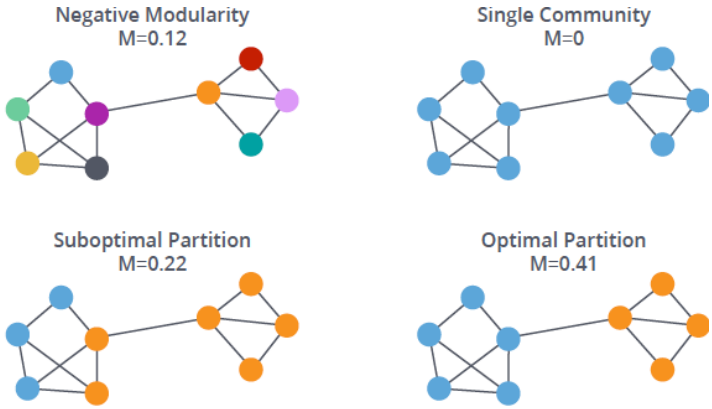


Figure 4.18: Example of different modularities, taken from [48]

For a weighted graph, the formula to compute modularity is:

$$Q = \frac{1}{2m} \sum_{ij} \left(A_{ij} - \gamma \frac{k_i k_j}{2m} \right) \delta(c_i, c_j),$$

where m is the number of edges, A is the adjacency matrix of the graph, $k_{i/j}$ is the degree of the node i or j respectively, γ is the resolution parameter, and $\delta(c_i, c_j)$ is 1 if i and j are in the same community, else 0 [49].
With some algebra, this can be reduced to:

$$Q = \frac{1}{2m} \sum_{c=1}^{n} \left[\frac{L_c}{m} - \gamma \left(\frac{k_c}{2m} \right)^2 \right],$$

where the sum iterates over all communities c, m is the number of edges, L_c is the number of intra-community edges for community c, k_c is the sum of degrees of the nodes in community c, and γ is the resolution parameter, which sets an arbitrary trade-off between intra-group edges and inter-group edges [49].

Many algorithms have been developed for detecting overlapping and/or disjoint communities, some of which are listed below.

Louvain Algorithm

The Louvain algorithm is a simple and fast method to detect communities in large networks. This method has two elementary phases that are repeated iteratively, as shown in Figure 4.19: initially individual starting nodes are moved locally to the neighbour's community that yields the largest increase of modularity, then, based on the partition obtained from the previous phase, each community is incorporated into one node, creating a new coarse-grained aggregate network. The two phases are repeated iteratively until the algorithm, given perturbations to the current community state, is unable to increase further the chosen quality function (the modularity). At this point, it is guaranteed that each individual node is optimally assigned and that communities are well separated (the modularity is maximized).

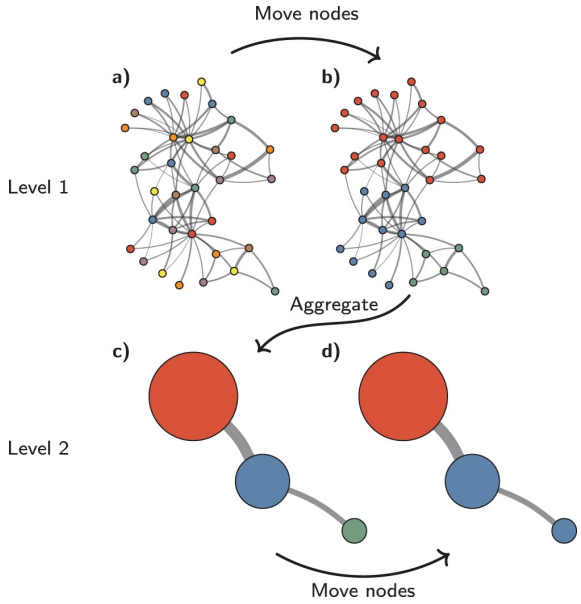


Figure 4.19: Schema of the process followed by the Louvain algorithm: (**a**) it starts from a singleton partition in which each node is in its own community; (**b**) the algorithm moves single nodes from one community to another to find a partition; (**c**) on the basis of this partition, it creates an aggregate network; (**d**) individual nodes are moved in the aggregate network. These steps are iterated until the quality cannot be improved further. Taken from [50]

Usually, the Louvain algorithm starts from a singleton partition, in which each node belongs to its own community, but is also possible to start the algorithm

from a different partition. Multiple successive iterations of the algorithm can be performed, using the partition identified in one iteration as input for the next iteration.

Louvain's algorithm can find arbitrarily badly connected communities, which are internally disconnected: a node can be moved to a different community while it may have acted as a bridge between the different components of its old community and, by removing it, the old community becomes disconnected. However, the other nodes may still be sufficiently strongly connected to their community, despite the fact that the community has been disconnected.

Label Propagation Algorithm

Label Propagation (LPA) is a semi-supervised Machine Learning algorithm that identifies both disjoint and overlapping communities in a network, by assigning labels, representing community membership between nodes, to previously unlabeled data points.
At the start of the algorithm, the memory of each vertex is initialized with a unique label, indicating the independent community each vertex belongs to. Afterwards, a node is selected as a *listener*, whose label is propagated to each of its neighbors (*speakers*), as shown in Figure 4.20.

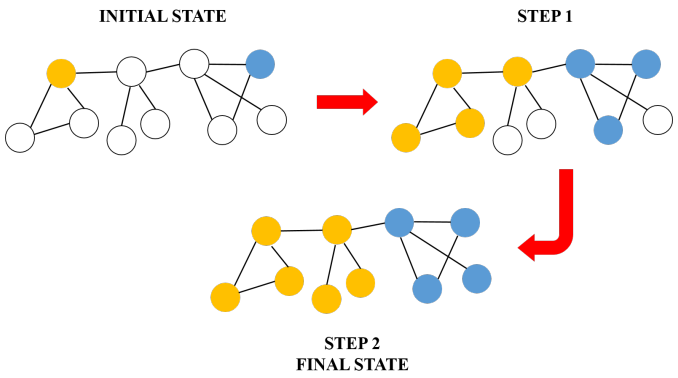


Figure 4.20: Schema of the process followed by the Label Propagation algorithm: some nodes have labels (*Initial State*), more labels are added (*Step 1*). This iteration is repeated until there is convergence on a solution, a set solution range, or a set number of iterations (*Final State*)

Each diffused label is randomly selected with a probability proportional to its frequency in the memory of the speaker that sends it. The listener then integrates the most common of the labels received into its memory. This process is repeated for a maximum number of iterations defined by the user. Finally, a probability distribution of labels is built for each vertex: if the probability for a particular label

of a node is below a given threshold, then the label is removed. Nodes with common labels are then grouped into a community, and it is natural to argue that densely connected groups reach a common label quickly. If a node has multiple labels it will be part of multiple overlapping communities, identified by the algorithm: the smaller the value of the threshold, the greater the number of communities that overlap.
Compared to other algorithms, LPA has convenience in its running time and quantity of information needed beforehand about the network structure (no parameters needed). The disadvantage concerns that it produces no unique solution, but an aggregate of many solutions (possible communities structures) starting from the same initial condition.

Girvan-Newman Algorithm

Another commonly used algorithm for detecting communities in complex systems (with a network structure) is the Girvan–Newman algorithm. This hierarchical algorithm identifies edges in a network that lie between communities by employing the graph-theoretic measure ***betweenness centrality*** (Fig.4.21), which assigns a number to each edge which is large if the edge lies "between" many pairs of nodes and, therefore, that are most likely "between" communities. Afterwards, these edges are progressively removed from the original network, such that the connected components of the remaining graph are the communities, revealing the underlying internal structure of the network.
More specifically, betweenness centrality indicates highly central nodes in networks, and, extending this definition to the arcs, the "edge betweenness" of an edge is defined by the number of shortest paths between pairs of nodes that cross it, divided by the total number of shortest paths.

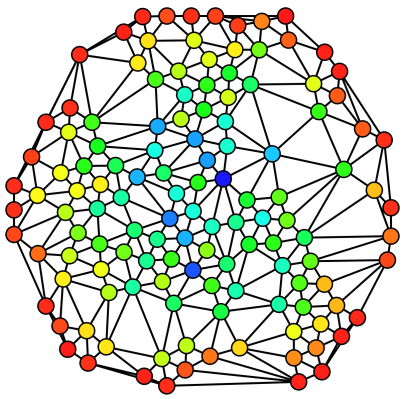
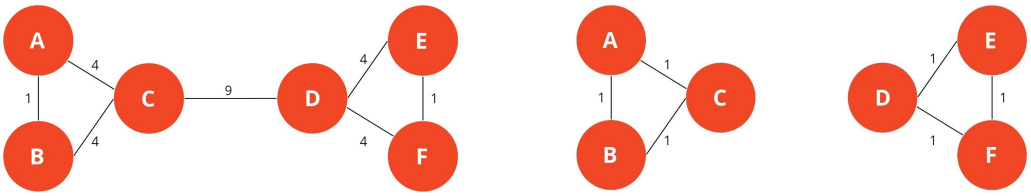


Figure 4.21: An undirected graph colored according to the betweenness centrality of each node from the minimum (red) to the maximum (blue), taken from [51]

The algorithm's steps can be summarized in the following way (Fig.4.22):

1. The betweenness centrality of all the edges present in the graph is computed;
2. The edge(s) with the highest betweenness is (are) eliminated;
3. The betweenness of all remaining edges (affected by the deletion) is recomputed;
4. Steps 2 and 3 are iterated until there are no more edges left.



(a) Initial state: weights are assigned to edges on the basis of the number of shorter paths that pass through them

(b) The edge between vertices C and D is removed, because it is the one with the highest weight (it is located between communities). The betweenness centrality has to be recomputed for every remaining edge until every edge has the same betweenness centrality

Figure 4.22: Simple example of the process followed by the Girvan-Newman algorithm, taken from [51]

The end result of the Girvan–Newman algorithm is a dendrogram, whose leaves are individual nodes: the network is divided into different communities with the consecutive removal of edges.

4.4.2 DBSCAN Algorithm

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is one of the most common clustering non-parametric algorithms and also most cited in scientific literature. The algorithm requires two input parameters: the radius of a cluster (ϵ, epsilon), which is, in other words, the maximum distance between two points for one to be considered as in the neighborhood of the other but not a maximum bound on the distances of elements within a cluster, and *minimum points* required to form a dense cluster. Given a set of points in a certain space, it clusters together points that are tightly packed together (these points' ϵ-neighborhood contain sufficiently many points such that they can be considered as core points),

marking as outliers (or noise) points that lie alone in low-density regions (whose nearest neighbors are too far away).

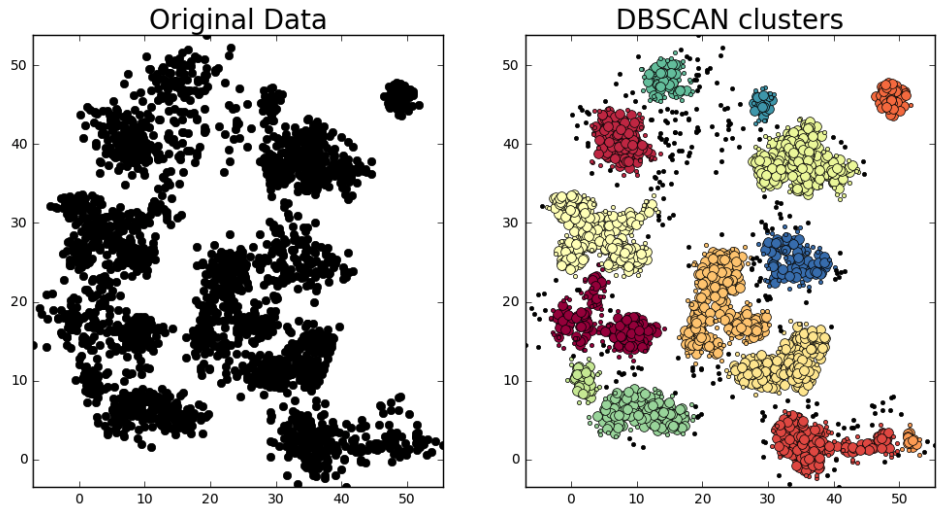


Figure 4.23: Original data on the left and clusters identified by the DBSCAN algorithm on the right: large colored circles represent core cluster members, small colored circles indicate cluster edge members, and small black points represent noise/outliers, taken from [52]

The DBSCAN algorithm can be abstracted into the following five steps:

1. Select an arbitrary point p;
2. Retrieve all points density-reachable from p, i.e. in the ϵ neighborhood of p;
3. If p is a core point (with more than minimum points neighbors), a cluster is created;
4. If p is a border point, it is assigned to a nearby (ϵ) cluster, otherwise it is assigned to noise;
5. The process is repeated until all the points have been processed.

DBSCAN can be also used with any distance function as well as similarity functions, the most common of which is Euclidean distance. In our case, for this thesis project, the Jaccard Index has been used to compute the distance matrix given to the DBSCAN algorithm as input.

Jaccard Index

The Jaccard Index, or Jaccard similarity coefficient, is a statistic used to measure the similarity/dissimilarity of finite sample sets, and is defined in the range [0,1], as the fraction between the size of the *intersection* and the size of the *union* of the sample sets (A and B):

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

4.4.3 K-Means Algorithm

K-Means clustering is a vector quantization technique, which aims to partition observations in a dataset into k clusters (this parameter must be predefined in advance), where each observation belongs to the cluster with the closest mean (centroid or cluster center), acting as a prototype of the cluster.
Given an initial set of k "means" as input, the naive algorithm (there are some existing variants) proceeds alternating the following two phases:

- **Assignment step**: each data observation is assigned to the cluster with the closest mean (centroid), i.e. the Euclidean distance squared;
- **Update step**: the means (centroids) are recomputed for the data points assigned to each cluster.

The algorithm achieves convergence when there are no further changes in the assignments, and there is no guarantee that it will find the optimum. Using a distance function other than Euclidean (squared) distance can prevent the algorithm from converging.

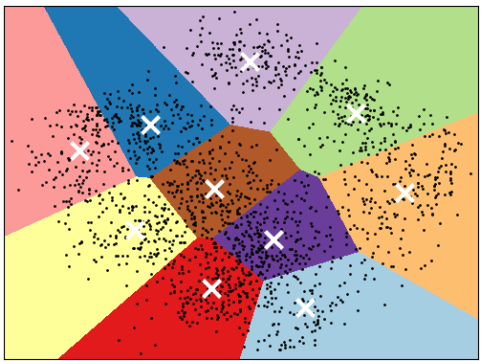


Figure 4.24: An example of usage of the K-Means algorithm by SciKit Learn for clustering on a handwritten digits data (PCA-reduced). Centroids are marked with white cross, taken from [53]

The algorithm has some limitations, for example it cannot be used with arbitrary distance functions or on non-numerical data, and the parameter k is known to be hard to choose when there are no external constraints.
To resolve the latter problem, there are some cluster quality measure that can be used as an interpretation and validation of consistency inside clusters, in order to identify the best k-parameter for clustering a given set of samples and evaluate the goodness of the resulting split. Some examples are:

- the ***Silhouette Score***: a measure of cohesion, how similar an object is to its own cluster, and separation, a similarity comparison to other different clusters. The higher it is, the more appropriate is the clustering configuration;
- the ***Davies-Bouldin Index***: the average similarity of each cluster with the most similar one, measured by comparing the distance between clusters with the size of the clusters themselves. The lower it is, the better is the result of the clustering performed;
- the ***Calinski-Harabasz Index***: it is defined as the fraction between the sum of intra-cluster and of inter-cluster dispersions, computed for all clusters. The higher the score, the better the performances.

Cosine Similarity

In data analysis, Cosine Similarity is a measure of how similar two sequences of numbers are. To define it, sequences are represented as vectors in an inner product space, and the cosine of the angle between them is defined as the scalar product of the vectors divided by the product of their lengths. The Cosine Similarity is always in the interval $[-1,1]$: two proportional vectors have a similarity of 1, the similarity of two orthogonal vectors is 0, and the similarity of two opposing vectors is -1.
In information retrieval and text extraction, each word is assigned a separate coordinate and a document is represented by the vector of the number of occurrences of each word in the document. Cosine similarity is a helpful metric for determining how similar two documents are in terms of topic content, independent of their length. In the realm of data mining, the approach is also used to quantify **cluster cohesiveness**.
Given two vectors of attributes, A and B, the Cosine Similarity is defined as:

$$S_C(A,B) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

The Term Frequency Vectors (sec. 4.5.1) of the documents are commonly used as attribute vectors A and B for text matching.

4.5 Textual Analysis Methods: Natural Language Processing (NLP)

Natural Language Processing (NLP) refers to the branch of computer science (Artificial Intelligence) concerned with the ability of software to process human language words, understanding their full meaning as human beings do, by combining computational linguistics with statistical, Machine Learning, and Deep Learning models. For this thesis project, the ML part, such as *Part of Speech Tagging* (or *grammatical tagging*), has been processed through some libraries (*Data Mining and Pre-Processing*, sec. 5.2.3), but there are also some classic methods that have been implemented to filter the data or extract text features (or topics), that need to be mentioned and explained.

4.5.1 Term Frequency — Inverse Document Frequency (TF-IDF)

In information retrieval (and other research fields such as text mining and user modeling), TF-IDF, acronym for Term Frequency–Inverse Document Frequency, one of the most popular term-weighting schemes. It is a numerical statistic that aims at reflecting the importance of a word within a document in a collection or *corpus*. Its value increases proportionally to the frequency of a word appearing in a document, but inversely to the number of documents in the collection containing that word. The underlying idea behind this behavior is to give more relevance to the terms that appear in the document, but which in general are infrequent.
The TF-IDF is calculated as the product of two statistics, the term-frequency and the inverse-document-frequency, hence the name:

$$TF_IDF(t, d, D) = TF(t, d) \cdot IDF(t, D),$$

where t is a term, d is the document in which t appears, and D is the set of documents.
Term-frequency: it is the frequency (generally the raw count) of term t, i.e. the number of times that term t occurs in document d.
Inverse-document-frequency:

$$IDF(t, D) = \log \frac{|D|}{1 + |d \in D : t \in d|},$$

where $|D|$ is the total number of documents in the corpus and the denominator indicates the number of documents where the term t appears, adjusted (it is increased by one) to avoid division by zero if t is in none of the documents. Since logarithm is used, if a term appears in all documents, its IDF value becomes 0.

Sublinear TF-IDF

Sublinear TF-IDF is a variant which modifies the term-frequency component, giving prevalence to TF over IDF, by replacing TF with $1 + \log(TF)$.

Hashing TF-IDF

In this variant, each raw feature is mapped into an index (term) by applying a hash function and then calculating term frequencies based on the mapped indices. With this approach, computing a global term-to-index map, which can be expensive for a large corpus, can be avoided, but there is a limitation: potential hash collisions can occur, where different raw features may become the same term after the application of the hash function. To escape this problem and reduce the chance of collision, the target feature dimension (the number of buckets of the hash table) can be increased.

4.5.2 Latent Dirichlet Allocation (LDA)

In Natural Language Processing (NLP), the latent Dirichlet allocation (LDA) is a generative probabilistic **topic modeling algorithm** which takes as input a *corpus* (a collection of documents containing a set of terms, i.e. single words or phrases, assumed to be not ordered) to discover the contained hidden topics (shared themes).The underlying idea is that the semantic content of each document is viewed as a combination of a small number of various latent topics, where each topic is characterized by a Dirichlet prior distribution over words. Some terms' meaning is uncertain, since such terms belong to more than one topic, with different probability. However, in a document, the joint presence of specific neighboring terms (belonging to only one topic) will disambiguate their usage.
All terms are initially treated uniformly, there is no indication of prevalence of some words over others, and then LDA automatically classify any individual document of he corpus in terms of how "relevant" it is to each of the found topics. If the document collection is large enough, LDA will reveal topics based upon the mutual occurrence of individual terms, though the task of assigning a meaningful label to an individual topic has to be done manually by a user.
LDA uses a sampling method to produce two types of output:

1. **Topic distribution over documents**: the mixture of topics within each document, and the percentage of each topic contained in the document. This output derives from the LDA assumption that, in the corpus, individual topics will occur with differing frequencies: they have a probability distribution, so that a given document is more likely to contain some topics than others.

2. **Topic distribution over words**: the mixture of words found in each topic, and the percentage each word contributes semantically to each topic. This output derives from the LDA assumption that, within a topic, certain terms appear much more frequently than others, such that they have a probability distribution.

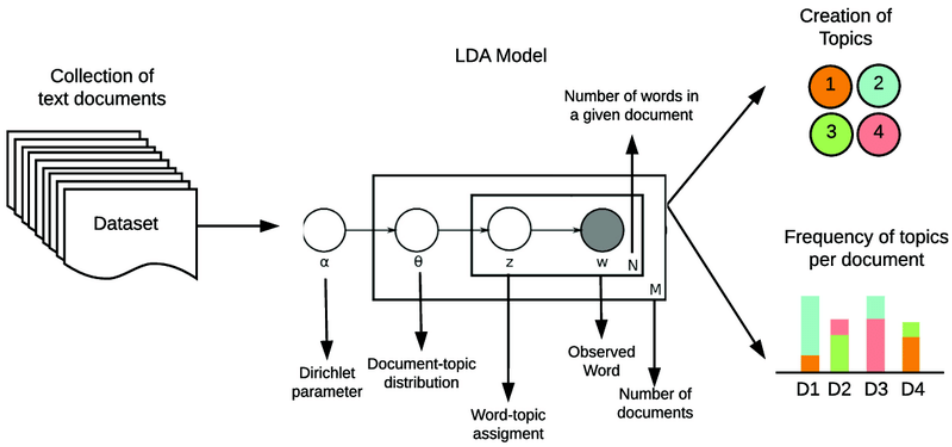


Figure 4.25: Schematic of LDA algorithm, taken from [54]

Chapter 5

Data Mining and Pre-Processing

5.1 Data Sources

As already explained in chap. 3, we mined two data sources to generate the Instagram Post Dataset, used in this research: CrowdTangle Data and a list of influencers taken from "`www.influenceritalia.it`".

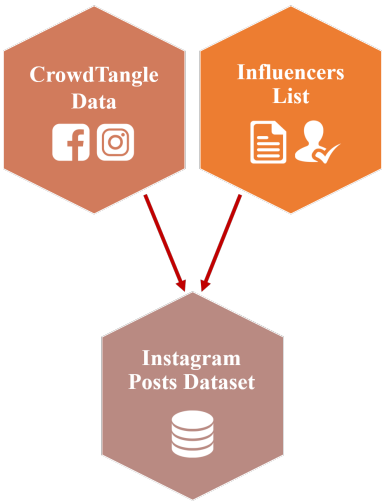


Figure 5.1: Data Sources

Downloading the data took four days of a script's runtime, while maintaining a cautionary query limit of one every ten seconds to stay below the suggested limits and avoid overloading both the CrowdTangle servers and our hardware.

5.2 Instruments

5.2.1 Apache Spark

Apache Spark, developed in 2009 at Berkeley, is an open-source unified analysis engine written in the Scala programming language and based on an advanced large-scale distributed data SQL engine, used for Big Data and Machine Learning processing. It provides high-level APIs in Java, Scala, Python, and R, and supports general execution graphs thanks to an optimized engine.
Spark essentially provides an interface for **cluster programming**, with implicit **data parallelism** and **fault tolerance** integrated in its architectural foundation, which derives from the concept of Resilient Distributed Dataset (*RDD*, a read-only multi-set of data items distributed on a cluster of machines).
In fact, Spark has the ability to perform processing tasks on huge datasets by distributing them among multiple *worker nodes* (processes that perform computations and store data for the application), and, for this reason, it has become a solution widely exploited by companies and research institutes. In other words, in the most common usage, Spark applications consist of independent sets of processes on a cluster (coordinated by the *Driver* program thanks to the *Spark Context* object). In this case, Apache Spark is said to be deployed in "**Cluster Mode**", i.e. across multiple computers or servers.
Its architecture, on a conceptual level, consists of three components:

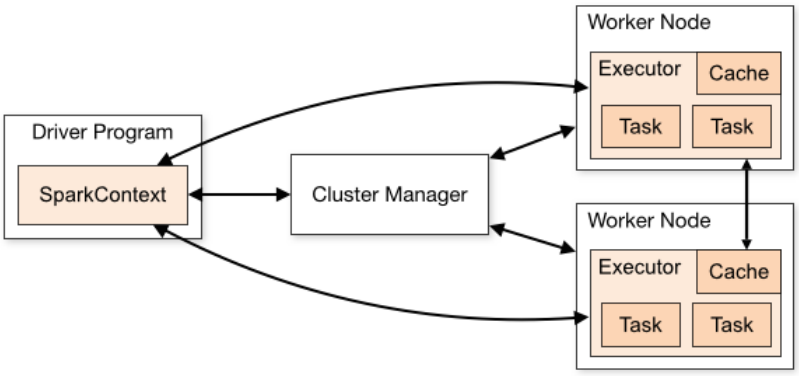


Figure 5.2: Organization of a Spark Cluster, taken from [55]

- The **Driver**: the process running the main function of the application and creating the Spark Context, which organizes the user code into multiple that are then distributed across workers, and executes various *parallel* operations on the cluster;

- The **Cluster Manager**: an external service for acquiring resources on the cluster, distributing the tasks, managing the workers and performing control routines;
- The **Executor**: a process running the assigned tasks on a worker node, also keeping data in memory or disk storage.

More specifically, for cluster management, Spark supports connections to several different types of cluster managers, which allocate resources between applications: *standalone* (native Spark cluster), ***Hadoop YARN***, Apache Mesos (deprecated) or Kubernetes. Once connected, it captures the executors on the cluster nodes (workers) and sends them the application code (defined by the Python files passed to the Spark Context). Finally, Spark Context sends the tasks to the executor processes for computation.
It is important to mention that each application has its own executors for the whole running time, which run tasks in multiple threads. For this reason, distinct Spark applications are separated from each other, from both scheduling and executor points of view: each driver schedules its own tasks and each task from a different application runs in a different Java Virtual Machine. On the other hand, the resulting disadvantage is that data cannot be shared across distinct applications, unless it is written to an external storage system.

Apache Spark also supports many higher-level tools, including Spark SQL, MLlib for Machine Learning, GraphX for graphs creation, and Streaming for incremental computation and flow processing.

Figure 5.3: Spark ecosystem and its components

- **Spark Core**: the underlying kernel of the Apache Spark framework on which all other functionality is based, providing the overall execution engine to the platform, defining and managing the native data structure (RDD), and offering in-memory computations;
- **Spark SQL**: a component for structured data processing, acting as a distributed SQL query engine. It offers the so-called DataFrame, one of the most relevant programming abstractions used in Spark;

- **Spark Streaming**: library used to perform analytics on streaming and historical data, which actually works by dividing the data stream into blocks to be processed;
- **MLlib**: a low-level Machine Learning library that supports some popular statistical, analytical, and machine learning methods, providing a set of high-level APIs to build and optimize large-scale ML pipeline practices;
- **GraphX**: a framework used to process graph data in a distributed way, based on RDDs. It should be remembered that RDDs are immutable and, for this reason, graphs are also immutable and, therefore, GraphX is not suitable for graphs that need to be updated;
- **Spark R**: package of Apache Spark able to process large datasets with R, supporting operations like selection, filtering, aggregation etc.

Finally, Apache Spark basically provides three types of data structures:

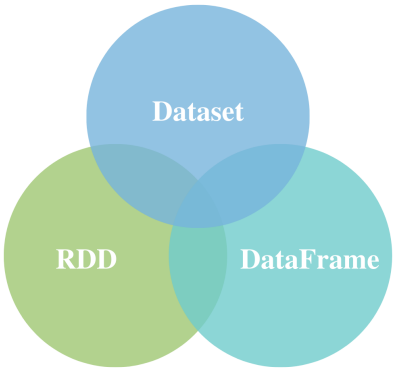


Figure 5.4: Spark data structures

- **RDD**: Resilient Distributed Dataset, Spark's fundamental data structure that implements an immutable collection of data objects, partitioned between cluster nodes, on which it is possible to operate in parallel. They are fast and fault tolerant, but cannot be used together with Spark SQL;
- **DataFrame**: a distributed collection of data organized into optimized structures with named columns (*c*ollections of Rows), similar to tables in a relational database, defined by Spark SQL;
- **Dataset**: a strongly typed collection of data elements, which offers the advantages of both RDDs and the SQL optimized execution engine, allowing parallel *transformations* (those that produce new datasets) or *actions* (those

that trigger the calculation[1] and return results) on data, by using functional or relational operations. DataFrames are Datasets of *Rows*.

5.2.2 The Cluster and The Hadoop Distributed File System (HDFS)

Figure 5.5: Jupyter and PySpark logos ©

The data, processed with PySpark, was saved on the computing cluster using the Hadoop Distributed File System (HDFS), which is the main data storage used by Hadoop (and, consequently, PySpark) applications. Essentially, HDFS is a scalable fault-tolerant distributed system for Big Data, in which the archived files are split into chunks, as shown in Figure 5.6, each of which containing a part of the content of one single file, saved on Data Nodes, and spread across the servers. Such nodes have their mapping managed by one (or more) Master Nodes which also perform file system operations.

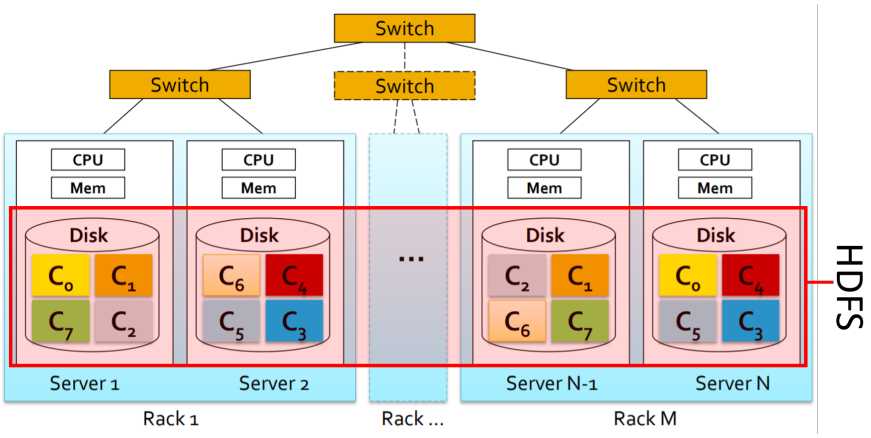


Figure 5.6: Hadoop main components, taken from [56]

[1] ***Lazy evaluation***: in Apache Spark data is not loaded until it is necessary, i.e. an action is triggered, so transformations are not executed immediately.

Our research group, SmartData@PoliTO, to satisfy the center's research aims, operates on Politecnico's **BigData@Polito Cluster**, which is built on the above described complex software infrastructure. The cluster, after 2020, has reached the capabilities of:

- 33 storage workers equipped with:
  - 216 TB of disk storage space;
  - 384 GB of RAM;
  - Two CPUs with 18 cores/36 threads each.
- Two nodes equipped with 4 GPUs for experimentation;
- 50 GB/s of data reading and processing speed.

In particular, for the present thesis project, we exploited a server with 8 reserved CPU Threads/40 GB of memory, maximum 48 CPU Threads/120 GB of memory. **Jupyter Lab** (Fig. 5.5), which is a web-based interactive development environment for **Jupyter Notebooks**, interactive computational environments with a modular design accessible from a web browser, provided us access to the cluster and its software by means of the spawn of a virtual server with the described features.

5.2.3 Tools and Libraries

The data was elaborated with the help of some tools and libraries: Pandas, NumPy, SciKit-Learn, SciPy, NetworkX, SpaCy, Gensim, and, for the visualization of the results Matplotlib and Seaborn.

Figure 5.7: Logos of the libraries used ©

- **Pandas**: it is a fast and flexible open source data analysis and manipulation tool based on the Python programming language, offering data structures with

integrated indexing and operations for manipulating numerical tables and time series. It has been extensively employed for data science and machine learning tasks;

- **Numpy**: it is an open source package for scientific computing with Python, offering vectorization, indexing and broadcasting for array computing, comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more;

- **SciKit-Learn**: it is an open source software Machine Learning library for Python, which features various classification, regression and clustering algorithms, dimensionality reduction, model selection and pre-processing, offering efficient tools for predictive data analysis;

- **SciPy**: it is an open source Python library used for scientific computing and technical computing, providing algorithms for optimization, integration, interpolation, special functions, statistics and many other classes of problems, and extending NumPy offering additional tools for array computing and specialized data structures, such as sparse matrices and k-dimensional trees;

- **NetworkX**: it is a Python package for the creation, manipulation, and study of the topology, dynamics, and functions of complex networks, providing data structures to represent graphs, digraphs, multigraphs, random graphs, and synthetic networks, many standard algorithms to manipulate graphs, network structure and analysis measures;

- **SpaCy**: it is a Python open source library for advanced Natural Language Processing (NLP), which helps to process large amounts of text, build information extraction or NLP systems, and pre-process text for deep learning or other kinds of analysis. It features convolutional neural network models for Part-Of-Speech (POS) Tagging, dependency parsing, text categorization and Named Entity Recognition (NER);

- **Gensim**: it is an open source library for unsupervised topic modeling and NLP, using modern statistical machine learning, and handling large text collections for semantic analysis by means of data streaming and incremental online algorithms;

- **Matplotlib**: it is a comprehensive library for creating static, animated, and interactive visualizations in Python, i.e. quality plots and interactive figures that can be customized with different styles and layouts and exported to many file formats;

- **Seaborn**: it is a Python data visualization library based on Matplotlib, providing a high-level interface for drawing attractive and informative statistical graphics.

5.3 The Original Data

The data was first downloaded locally, and afterwards it was stored in the HDFS of the cluster in order to be processed by the Spark instruments presented in sec. 5.2. Each row-entry of the original dataset, scanned by the CrowdTangle APIs, was characterized by various attributes, described in the following Table 5.1:

Property	**Type**	**Description**
`account`	struct	Information about the user who created the post: • unique identifier of the user (long) • handle and name of the user (strings) • platform where the user published the post (string) • Instagram user unique identifier (string) • profile image (string) • current number of user's followers (long) • account URL (string) • information about the account verification (boolean)
`brandedContentSponsor`	struct	Information about the sponsoring page who promoted the post, if available (same structure of `account`)
`date`	string	Posting date
`description`	string	Text description (caption) of the post
`expandedLinks`	array	Links contained in the post
`history`	array	List of elements, consisting of: • *actual* number of received likes and comments (long) • date (string) • *expected* number of reactions to receive (long)
`id`	string	CrowdTangle content unique identifier
`imageText`	string	Text, if existing, within the post media (image).
`legacyId`	long	Unique ID of the post, now deprecated by the system
`media`	array	Array of media for the post (URLs and dimensions)
`platform`	string	Platform where the post was published (here Instagram)
`platformId`	string	Instagram content unique identifier
`postUrl`	string	URL to access the post on its platform
`score`	double	Performance-score given to the post by CrowdTangle
`statistics`	struct	Performance metrics associated with the post, as `history`
`subscriberCount`	long	Number of influencer's followers at the posting time
`type`	string	Media type
`updated`	string	Last updating time of the post by CrowdTangle

Table 5.1: Attributes of the data objects (Rows) contained in the raw dataset, and description of their characteristics

From this raw form, the dataset can be organized neatly by distributing the attributes of the posts into the following categories:

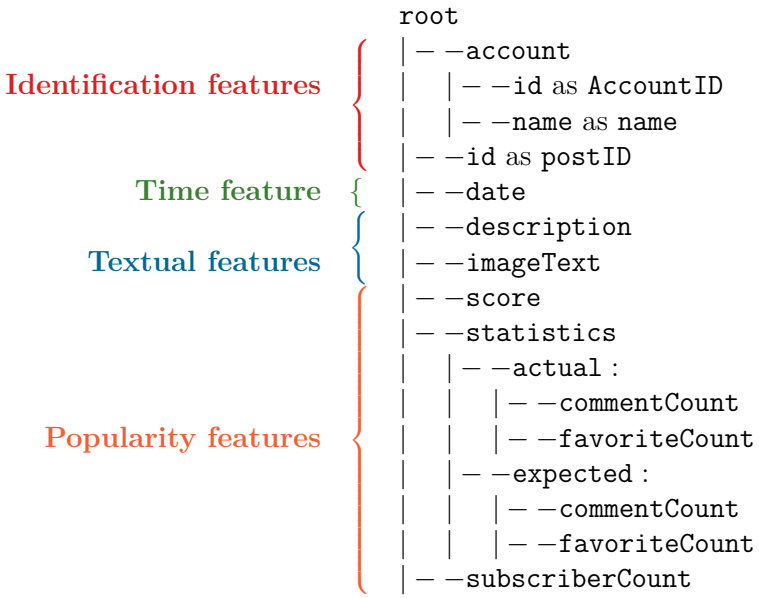
- Identification features: the various unique identifiers and URLs to identify properly the creator's account and the post (sec. 5.4.2);
- Temporal features: the posting and the updating dates (sec. 5.4.3);
- Popularity features: the post's received reactions (actual and expected), the subscribers count at the posting time and the performance-score given to the post by CrowdTangle (sec. 5.4.4);
- Media and Textual features: the post's media, its type and its textual content, and the caption of the post (sec. 5.5).

These categories have been identified empirically, based on the changes needed for the data, and, for this reason, they are fully described in the following sections, while presenting the transformations performed on the dataset.

5.4 Initial Data Transformation

5.4.1 Data Fields Filtering

As seen in Table 5.1, each row of the dataset contains numerous fields, most of which are useless for our purposes. For this reason we have decided to filter these fields by removing them and keeping only the following scheme to manipulate and on which to perform the various analyses:



Such decision was taken in order to reduce the total amount of data and also to avoid some relative redundancy of the post characteristics.

5.4.2 Identification Features

Each data object needed to be identified, and uniquely distinguished from the others, during the execution of the various analyses, and, to do this, we employed the attributes already provided for each post.

Post Identification

Instagram attributes three unique identifiers to each content: an alphanumeric string, called "`id`", used as the unique identifier of the post in the CrowdTangle system, which is composed of the numeric unique identifier of the account that published the post, pre-poned in charge of a long integer (`LongInt`), as shown in Figure 5.8; the `legacyId`, which is the legacy version of the unique identifier of the post (on CrowdTangle), and is owned by just under half of the posts (probably it has been deprecated by the system); an external URL that the post links to, pointing to the web page containing the post itself.

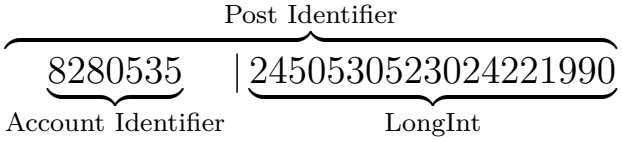


Figure 5.8: Post unique identifier example

id	legacyId	postUrl
741175|2282986242916233648	89654293484	https://www.instagram.com/p/B-uy-RkIDGw/
191654|2282816185976522862	0	https://www.instagram.com/p/B-uMTnspIhu/
2071620|2282752311256407927	89580219350	https://www.instagram.com/p/B-t9yHuo993/
2138208|2282656409468489139	89554816232	https://www.instagram.com/p/B-tn-kOo02z/
532839|2401242724362577280	0	https://www.instagram.com/p/CFS7YkVH2WA/

Figure 5.9: Post entries example

Account Identification

The characterization of the users' accounts is slightly richer (Fig. 5.10): in addition to the id and the URL also provided for the posts, the user profiles are also designated through an attribute called `name` that can be chosen and widely modified by the user (it can in fact contain a variety of characters) and a handle (commonly known as a "username") that can only contain Latin letters, numbers, underscores, and dots. The name does not have to be exclusive on the platform (Instagram), while the username must necessarily be unique for each user.

handle	id	name	platform	platformId	profileImage	subscriberCount	url	verified
mahmood	4801862	Mahmood	Instagram	211275393	https://scontent-sea1-1.cd...	1175208	https://www.instagram.com/...	True
surry	530855	Surry	Instagram	35250093	https://scontent-sea1-1.cd...	1754379	https://www.instagram.com/...	True
cookist	1594290	Cookist	Instagram	4360839247	https://scontent-sea1-1.cd...	1148128	https://www.instagram.com/...	True
cleotoms	545026	CLEO	Instagram	188180558	https://scontent-sea1-1.cd...	564721	https://www.instagram.com/...	True
linus_dj	730247	Linus	Instagram	1666704811	https://scontent-sea1-1.cd...	761258	https://www.instagram.com/...	True

Figure 5.10: Example of account entries (URLs truncated for pagination constraints)

5.4.3 Temporal Features

As stated earlier in chap. 3, the dataset was extracted over a six-year time frame. CrowdTangle essentially provided three types of temporal data, as represented in Table 5.1:

- `date`: the date, represented as a string, of the post's publication time;
- `date`: the date, saved in the same format, of the last update of the post by CrowdTangle;
- `date` saved in `history`: the data of each sampling, saved within an array in parallel with the reference metrics (`actual` number of likes and comments obtained from the post and `expected` number of likes and comments to be received by the post).

This information needed some transformations to be used more easily and to locate posts in certain periods of time.
Due to the fact that a timestamp would have been too accurate and would have identified the data objects too accurately, and because there was no interest in having precision information on the actual days of each week (from Monday to Sunday) and each month (for example from January 1st as of January 31), the weeks and months have been uniquely identified **in conjunction with the** information

regarding the corresponding **year**.
In particular, the information on the date has not been removed for completeness, but transformed into the `DateType` in the format "`yyyy-MM-dd`" (removing the part concerning the time, hours, minutes and seconds in the format "`HH: mm: ss`"), and then we added the fields **`week`**, **`month`**, and **`year`**, calculated using the `pyspark.sql.functions` library methods, respectively `weekofyear`, which extracts the week number (from 1 to 52) of a given date as integer, `month`, and `year`, which extract respectively the month and the year of a given date as integer, applied on the new transformed Column `date` of the PySpark DataFrame.

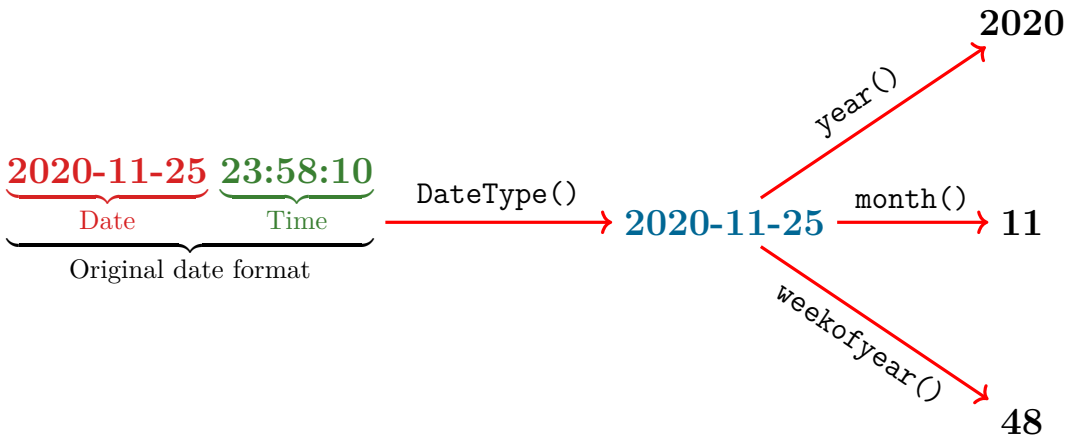


Figure 5.11: Example of date transformation

5.4.4 Popularity Features

As mentioned above in chap. 3, CrowdTangle tracks interactions, also known as engagement, which is defined as the number of reactions (likes and comments) and, for Facebook, also of shares, during a given set period of a time frame, over 10 minute intervals. As for comments, CrowdTangle does not provide the comment text, only the total number of comments, and does not differentiate between whether the comment was in response to the post or in reply to a mention or another comment to the post.
Consequently, from each post some information on popularity was extracted, in particular four types of metrics, already provided by the platform (CrowdTangle):

- Number of **likes** of the post at a specific time;
- Number of **comments** to the post after a certain time;
- Number of **followers** of the influencer (the creator of the content) at the posting time;

- The **performance-score** of the post calculated by CrowdTangle.

This choice was made because these metrics are present for all influencers, easily accessible and commonly considered the most concrete effect of reputation and popularity.

CrowdTangle Performance-Score

CrowdTangle defines **Overperforming** or **Underperforming scores** for each post, based on some analysis of the post interactions.
To calculate these scores, they first generate **benchmarks** to identify the number of likes and comments the post is *expected* to receive, by taking the last 100 posts from a given account/page and from a given type of posts (link-post, image-post, etc.) for reference, dropping the top and bottom 25% of those 100 posts, and computing the average number of interactions to the middle 50% of the posts at each age (15 minutes, 60 minutes old, 5 hours, etc.).
Afterwards, when a new post is published from that account, it is compared to that average, and the difference is multiplied by the corresponding *weights* in each dashboard[1]. A score greater than 1 indicates that the post is performing better than usual for a post of that sort, from that account/page, and at that time (10 minutes after posting, 1 hour after posting, etc.). A negative score means it is faring worse than the mean.

The equation performed to compare the post to the benchmark is:

$$\texttt{score} = \frac{actual}{expected} \tag{5.1}$$

In other words, if a post has 100 interactions and its benchmark was 50, the score is 100/50 = 2.0x.

According to this equation, any under-performance should be between 0 and 1, but it is rendered negative. If a post is expected to have 200 interactions and get 100 effective, it is 50% of its expected value, which can be viewed as -2 times underperforming.
Then the equation is reversed and its sign is changed to negative:

$$\texttt{score} = -1 * \frac{expected}{actual} \tag{5.2}$$

In the event that a post does not receive any interaction, it would be impossible to divide by 0, so in this case CrowdTangle simply does the following calculation:

$$\texttt{score} = -2 * expected$$

[1]Actual and expected values have some multipliers (weights): a like, for example, is worth three times the value of a video view.

If a post is overperforming but is below a minimum threshold (variable depending on the platform), its score is entered in the range between 0x and 1x.

$$\texttt{score} = \frac{actual}{minimum}$$

This means that a post in this case presents itself as overperforming, but is simply ranked below all other overperforming posts that have more interactions than the minimum.

In the event that a post is underperforming in the range between 0x and 1x, the corollary of the over-performance in this range is applied:

$$\texttt{score} = -1 * \frac{expected - actual}{expected}$$

The end result is that it is correctly noted as underperforming, but does not jump to the very bottom of the list.

New Performance-Score Computation and (Post) Filtering

To avoid all the complications of CrowdTangle's calculations, we have recalculated the performance score for each post.
The formulas for calculating the overperforming and underperforming scores are the same as those used by the platform (5.1 and 5.2), while the special cases have been simplified and incorporated into the first two. In particular, the posts that had a number of expected reactions equal to 0 were in total 4 in the dataset and therefore were simply removed, while to avoid the division by 0 in the formula to calculate the underperforming score, the posts with an actual number equal to 0 reactions (which were 561 in total) were simply given a score of 0.
In addition, with the same formulas, two other scores have been calculated for completeness, one for the likes and the other for the comments, but considering individually the actual and expected numbers of likes and comments instead of adding them together, as for the general equations.

Characterization of the Types of Score

As shown by the values in Table 5.2 and by the ECDFs in Figure 5.12, which represent the distribution of posts for each type of score, the score calculated by CrowdTangle and ours have the same distribution. As for the score calculated individually for the number of likes and comments, it turns out that the score of the likes tends to be slightly higher than the other for underperforming scores and exactly the opposite for the overperforming one.

From the ECDFs it can also be seen that most of the posts (in particular 60%) were underperforming and obtained a negative score.

	Score			
	CrowdTangle	Reactions (new)	Likes	Comments
Mean	-37.1	-2.7	-2.6	-2.3
Std. Dev.	2 717.8	244.1	241.3	26.4
Minimum	-1 489 218	-129 665	-129 344	-1 519.5
Maximum	872.9	872.9	1 278.3	25 974.8

Table 5.2: Statistical description of the post scores, for each type

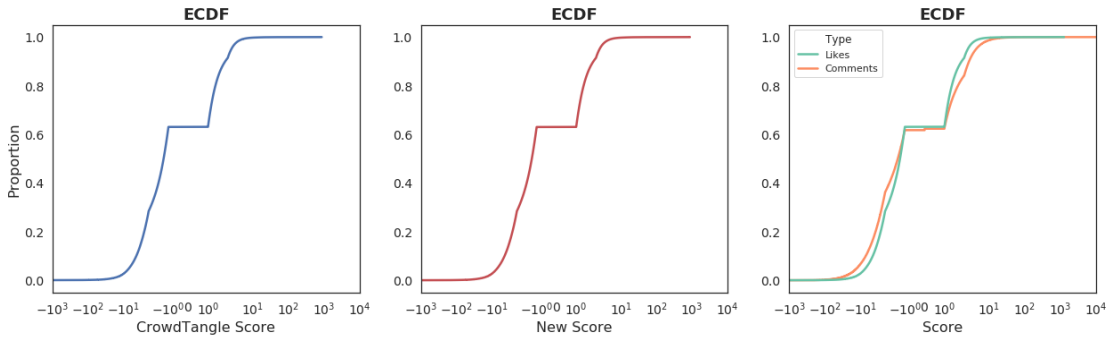


Figure 5.12: ECDFs of the post scores

Since no particularly significant differences were found between the score calculated by CrowdTangle and the new one, only the latter will be taken into consideration for subsequent analyses, concerning the usual subdivision of influencers (and their posts) into classes, already performed for the characterization of the dataset described in chap. 3.

	Mega inf.	Macro inf.	Micro inf.	Nano inf.
Mean	-3.8	-2.2	-1.7	3.9
Standard Deviation	403.5	134.5	12.9	5.5
Minimum	-129 665	-70 351	-1 095	-1.7
Maximum	700	872.9	123.3	33.5

Table 5.3: Statistical description of the (new) post score distribution for each class of influencers

From the Table 5.3 and the ECDF in Figure 5.13, which represent the same previous data but reflecting the division into classes, it can be seen that the posts of mega,

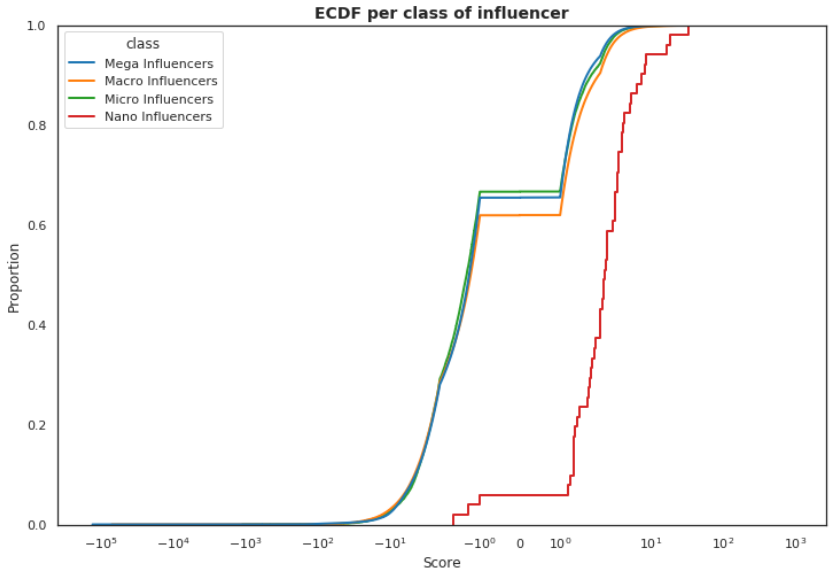


Figure 5.13: ECDF of the (new) post score for each class of influencers

macro and micro influencers do not present substantial differences, while the nano influencers' ones would seem to have received the highest performance scores. But it should be remembered that there are only 51 posts in this category, so the stats can never be as accurate as those for the other three classes.

5.5 Textual Features Pre-Processing

Text pre-processing is the process of cleaning up text data before using it for analysis or prediction by feeding the data into a Machine Learning model. When it comes to dealing with Human Language, text data is one of the most unstructured types of accessible data, which could include some noise in the form of emoticons, punctuation and text in several cases.
Natural Language Processing (NLP)-based challenges are addressed using a variety of libraries and techniques, which mainly use regular expressions to reorder the text. The NLTK and SpaCy next-level libraries (sec. 5.2.3) are used to perform natural language tasks, including stopword removal, named entity recognition, Part of Speech (PoS) Tagging, sentence matching, and so on. NLTK is an outdated library that novices can use to learn NLP methods, while SpaCy is the newest library to be released, which features the latest methods, and is widely used in the production environment.

5.5.1 Textual Information Extraction

Although it is not the main focus of the Social Network, it is common for Instagram posts to include a text description (a caption): this text can contain up to 2200 characters of any type and frequently includes emojis, hashtags, and tags. In addition to a message to describe the context of the media, add important information, or converse with followers, on Instagram it is common to add some hashtags or mentions in this text space (immediately below the average of the post, or possibly in the first comment, but such occurrence was not traceable due to lack of information in the dataset). Although it used to be greater, the maximum number of mentions is now set at 20, at 30 for hashtags.
The features of the text extracted for each post, if available, were the following:

- the list of the **total words** used;
- the list of **hashtags**;
- the list of **mentions**;
- the list of **simple words**, hashtags excluded;
- the list of **words** contained **in the image**.

These lists of words were generated by **splitting the string** of the `description` field, using the the space (“ ”) and the new line (“`\n`”) characters as separators, and by means of the regular expressions:

```
#(\w+)          \B@\w+
```

Figure 5.14: Regular Expressions used to extract respectively hashtags and mentions

5.5.2 Special Characters Cleaning

Lower Case

Since lowercase and uppercase letters are processed differently by a computer (words like “ball” and “Ball”, for example, are handled differently), it is simple for a machine to read the words if the text is in the same case.
As a result, we transformed each word in the content in the same case, to prevent such issues. To do this, we exploited the Python’s String library, which provides the `lower()` method, converting all uppercase characters in a string into lowercase characters.

Punctuation and Empty Strings Removal

Many times the text data contains extra spaces or while performing the pre-processing techniques listed in this section, more than one space is left between the text, so it is necessary to check and fix such problem: blank strings or additional space characters have been deleted from the various word lists.
The removal of punctuation is another text pre-processing classic approach, and there are a total of 32 major punctuation marks that must be considered. We used again the String module to replace any punctuation in the text with an empty string. Python's String library has a pre-defined set of punctuation, such as:

```
!"#\$\% \&'()*,-./:;?\@[]'|+
```

Therefore, we exploited the following line of code to remove them:

```
1 text.translate (str.maketrans("", "", string.punctuation))
```

Unicode Words

A post caption might also contain human-readable Unicode strings that should still be somehow machine-intelligible. For this reason, we used the `Unidecode`[1] library to decode the text in case special characters were present. In particular, the `unidecode()` function takes the Unicode data and tries to represent it in ASCII characters (i.e. the universally visible characters between "`0x00`" and "`0x7F`"), where the compromises made when mapping between two character sets are chosen for be close to what a human with an American keyboard would choose.

Emojis Removal

It is common in the caption of Instagram posts to find some special characters aimed at enriching the content, called "Emojis", which are pictograms, logograms, ideograms or emoticons, of various kinds, including facial expressions, common objects, places and types of time and animals. The main function of emojis is to fill in the otherwise missing emotional cues in the typed conversation. They are very similar to emoticons, a facial expression representations created with simple punctuation marks, but emojis are more like real small images rather than typographical approximations, and each of them is associated with a Unicode code. Many of them are meaningless and for this it is necessary to remove them from the

[1] `www.pypi.org/project/Unidecode`

text as they do not provide useful information. To do this, after uni-decoding the text, we exploited again a regular expression to find and eliminate Emojis:

```
[\u263a-\U0001f645]
```

Figure 5.15: Regular Expression to find Emoji characters

5.5.3 Part-of-Speech (PoS) Tagging Analysis

Part-of-Speech Tagging, also known as grammatical tagging in corpus linguistics, is the technique of identifying a word in a text (corpus) as belonging to a certain part of speech based on its meaning and context. Simply put, PoS Tagging is the process of classifying words as nouns, pronouns, verbs, adjectives, and so on, and it is important in sentence analysis, information retrieval, sentiment analysis, and other applications.
PoS Tagging, which was once done manually, is now done in the framework of computer linguistics, with algorithms that link discrete terms, as well as concealed parts of speech, with a set of descriptive tags.For example, in English, a word that comes after "the" is almost always a noun. Because some words can represent more than one part of speech at different times, i.e. they can work in multiple ways when used in different circumstances, and because some parts of speech are complex or unspoken, this task is more difficult than simply having a list of words and their parts of speech.

To categorize which tag or label a token belongs to, the SpaCy library uses a trained pipeline and statistical models, used to provide a series of annotations for each term, given a certain text. In particular, each token (word) is given a PoS Tag from the list in Table 5.4.

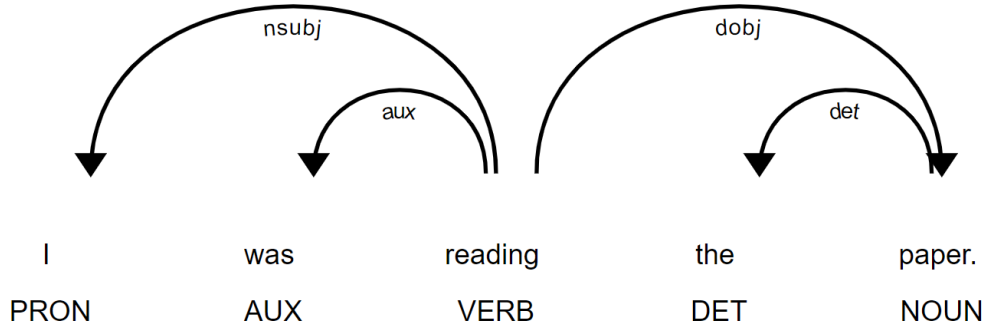


Figure 5.16: Example of PoS Tagging performed by the SpaCy library

PoS Tag	Description
ADJ	Adjective
ADP	Adposition
ADV	Adverb
AUX	Auxiliary
CONJ	Conjunction
CCONJ	Coordinating Conjunction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numeral
PART	Particle
PRON	Pronoun
PROPN	Proper Noun
PUNCT	Punctuation
SCONJ	Subordinating Conjunction
SYM	Symbol
VERB	Verb
X	Other
SPACE	Space

Table 5.4: SpaCy Part-of-Speech Tags List

For our purposes, the PoS Tagging analysis was performed for all the extracted word lists presented in sec. 5.5.1, but **excluding** that of **hashtags**, due to the nature of this kind of words, which are inserted in the posts as a sort of list, potentially of unrelated words, without creating sentences with full meaning.

Language Detection

Each language is unique, full of exceptions and special circumstances, even among the most popular terms. Some of these exceptions are shared between languages, while others are completely specific, often necessitating hard-coding to handle. All language-specific data is structured in Python files under SpaCy's `lang` module. In order to execute a good PoS Tagging analysis, it is necessary to first determine the language of each post caption, and use the corresponding SpaCy package. The library supports 63 languages in total, but is constantly developing for additional functionality.

By running the SpaCy Language Detection algorithm on each post of our dataset, a total of **32 different languages** were detected, in which Italian would seem

to prevail, followed by English, as shown in Figure 5.17. The "`Unknown`" column refers to posts that did not have a caption or that was completely filtered out by the analyses above, and it includes 151 960 posts in total.

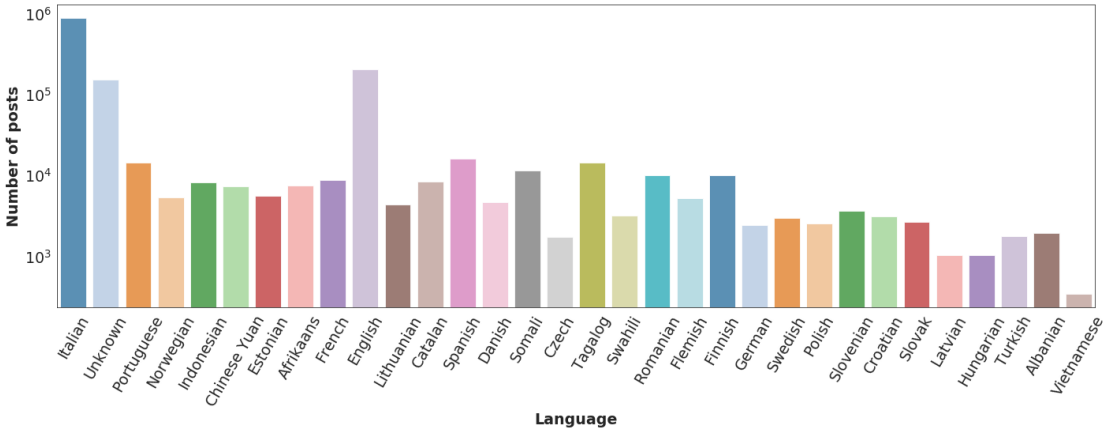


Figure 5.17: Language Detection performed by the SpaCy library on the Instagram Post Dataset

By manually testing the content of the captions of posts identified as belonging to uncommon languages, such as Tagalog, it turned out that this detection is probably due to an error, on the part of the user who created the post or the library, as they were consisting of lists containing only one or two words, or just mentions, acronyms, misspelled words, abbreviations, words not divided by spaces, contractions, colloquial and dialectal expressions.

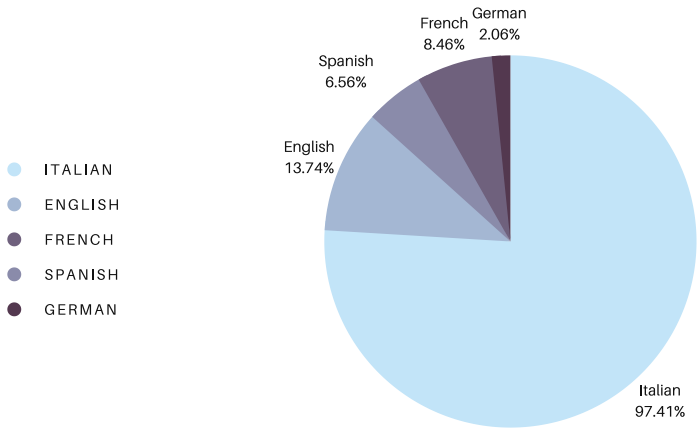


Figure 5.18: Percentage of people in Italy who speak the illustrated languages as a mother tongue or foreign language

By taking the data from the statistics reported on LanguageKnowledge[1] website and in Figure 5.18, the most used languages in Italy are, in order: Italian, English, French and Spanish. For this reason, only these four languages were used to perform PoS Tagging on posts.

PoS-Tagging (Words) Filtering

This SpaCy PoS Tagging Analysis was performed on all the posts of the dataset to keep for each list of words (remember, hashtags excluded) only **nouns** (NOUN tags) and **proper nouns** (PROPN tags), which are the terms that most contribute to the identification of the general topic of each post, in accordance with our purposes, pruning useless ones instead.

5.5.4 Stop-Words Removal

Stop words are the most used words in any human language (such as articles, prepositions, pronouns, conjunctions, etc.), which do not provide useful information for textual analysis. “They”, “there”, “this”, “where” and others are examples of stop-words. Stop words are available in abundance and therefore, by removing those words, we reduce the dataset size and remove low-level information from texts, to focus more on the important information.
The NLTK library is a widely used stop-words removal library that provides approximately 180 stop-words for the English language, 280 for Italian, 160 for French and 310 for Spanish. SpaCy, on the other hand, provides 326 stop-words only for the English language, 607 for Italian, 507 for French and 551 for Spanish.

In order to get the best possible results, we combined both stop-words collections into a single set of terms (removing duplicates). Following that, we utilized the set to filter these terms out of the list of words collected from each post (in this case hashtags included).
Furthermore, in order to properly filter the hashtags, we manually included the **non-significant hashtags** most commonly used by Instagram influencers, as well as other terms (manually checked) that were not appropriately excluded by the aforementioned libraries. These additional words are listed in Table 5.5, including 266 hashtags and terms.
We finally obtained a set of **2 390 stop-words** in total.

Other terms that were removed were those containing **less than 4 characters** or **just numbers**.

[1] `www.languageknowledge.eu/countries/italy`

Dictionary of additional stop-words
abbi, abbia, advertising, agli, alcune, alcuni, alla, alle, allo, alwaysfollowback, amazing, amore, anche, anzi, beautiful, beautifulday, beauty, best, bestoftheday, boysofinstagram, ciao, come, cool, cosa, cose, cute, daily, dargli, darglielo, darle, darmela, darmelo, darmi, darvela, darvelo, darvi, davvero, degli, dimmi, diretta, dirgli, dirglielo, dirvelo, dirvi, diventare, domani, dormo, dovete, durano, entrambe, entrambi, facebook, faresti, fatto, follow, follow4follow, follow4followback, followall, followalways, followback, followbackalways, followbackteam, follower, followerforfollower, followerforfollowers, followers, followersforfollower, followersforfollowers, followforfollow, followforfollowback, followher, followme, followtrick, followxfollow, forever, fosse, foto, giele, goodlookingguys, goodmorning, grazie, happiness, happy, happyness, hello, ieri, ierisera, ifollow, igers, ilove, incontrarti, insta, instabeauty, instablog, instaboy, instacolor, instacolors, instacool, instadaily, instadailypic, instafamous, instafashion, instagirl, instaglam, instago, instagood, instagram, instagramer, instagramers, instagrammer, instagrammers, instahome, instalife, instalike, instalive, instalove, instaminchia, instamoment, instamood, instangram, instanlike, instaphoto, instaphotography, instaphotos, instapic, instapics, instasaturday, instastories, instatag, instavideo, instavideos, ioete, iphoneonly, lets, life, lifestyle, like, like4like, like4likeback, like4likes, likeback, likeforfollow, likeforfollower, likeforfollowers, likeforfollows, likeforlike, likeforlikealways, likeforlikeback, likeforlikes, likeforliketeam, likes, likes4like, likes4likes, likesforfollow, likesforfollower, likesforfollowers, likesforlikes, link, linkinbio, love, lovelife, loveu, loveyou, meandyou, meme, memeita, memes, meno, messo, mood, moodoftheday, mylove, oggi, online, onthisday, ootd, parole, perche, percio, photo, photodaily, photography, photooftheday, picby, picoftheday, pics, pleasefollow, pleasefollowme, pochino, porti, posso, post, puoi, pure, quando, quant, quanta, quante, quanti, quanto, questo, repost, rispondi, saresti, scrivera, seguitemi, sempre, senza, shop, shot, shots, shout, shoutout, shoutout4shoutout, shoutout4shoutouts, shoutouter, shoutouters, shoutoutforshoutout, shoutoutforshoutouts, shoutouts, shoutouts4shoutout, shoutouts4shoutouts, shoutoutsforshoutout, shoutoutsforshoutouts, siate, simply, smile, sono, stasera, storia, storie, story, tagforlike, tagga, taggaituoiamici, tagsforlike, tagsforlikes, tagsta, tagstagramers, tanta, tante, tanti, tanto, teamfollowback, thanks, thankyou, tipo, today, tomorrow, toptag, toptags, torno, trovate, tumblr, unforgettableinstagrammer, unlimlikes, vabbe, vabbè, vederti, video, voglio, vogliono, vuoi, webstagram, youandme, youtube, youtuber.

Table 5.5: Dictionary of manually added stop-words, in alphabetical order

Chapter 6

Methodologies and Results

In this chapter the software architecture implemented in order to achieve our objectives will be defined, the parameters of the various methods and the results achieved will be presented and the latter will also be suitably compared to try to determine which methodology offers the best outcomes.

6.1 Workflow

The workflow of this project can be divided into 4 phases, each of which is further composed of sub-phases, as shown in Figure 6.1:

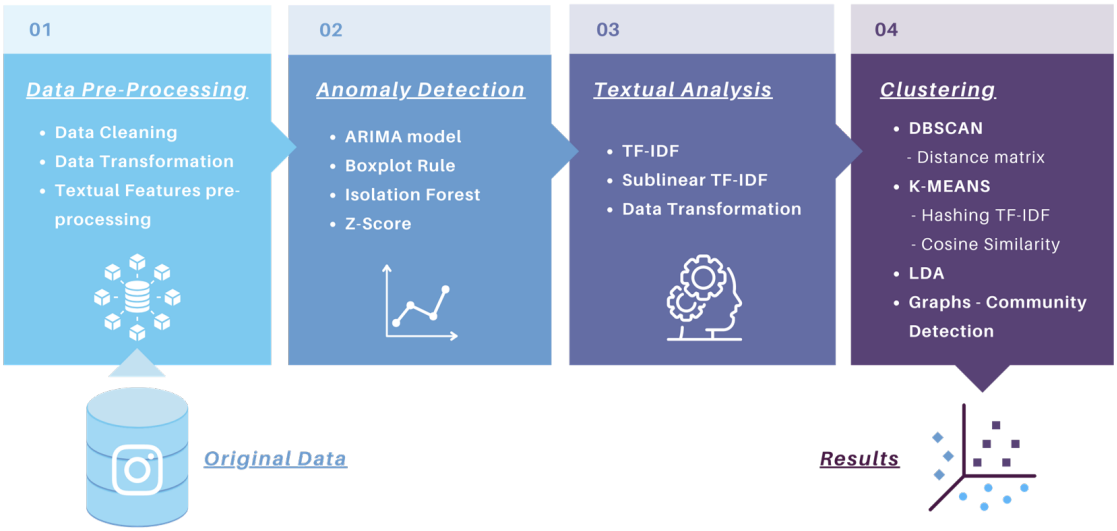


Figure 6.1: Project workflow

1. **Data Pre-Processing**: already fully described in chap. 5, consisting of data cleaning, data fields transformation and textual features extraction and pre-processing (cleaning and filtering);

2. **Anomaly Detection**: the time series, during the 6 years of our dataset, of the posts published by the individual influencers are analyzed in order to find anomalous posts (with respect to the number of reactions received, more details in sec. 6.2). Four classical techniques/algorithms, described theoretically in chap. 4, have been implemented: an **ARIMA Model** that forecasts the time series pattern and finds the posts that deviate too much from this prediction, the **Boxplot Rule** method, the **Isolation Forest** and the **Z-Score** method;

3. **Textual Analysis**: once the anomalies have been detected using the various approaches mentioned above, a more in-depth textual analysis is performed on the resulting anomalous posts **for each week** of the dataset. This analysis includes three components: the computation of the Term Frequency - Inverse Document Frequency (**TF-IDF**) and the **Sublinear TF-IDF** for each word contained in the captions of anomalous posts published during a certain week, and, subsequently, a new and final transformation of the dataset scheme into a more complex one that includes these last analyses (more details in sec. 6.3);

4. **Clustering**: the final phase involves the grouping of the anomalous posts obtained in phase 2, using the textual features obtained during phase 1 and 3, also in this case for each week of the dataset, trying to identify the posts that have received an anomalous number of reactions due to external events to the social network, that is, dealing with topics deriving from external factors. Four **textual-clustering** techniques have been implemented: the **DBSCAN** (applied on a pre-computed pairwise textual distance-matrix involving the weekly captions of anomalous posts) and **K-Means** (applied on a weighted hashed vector of the caption words) algorithms, an **LDA** model, and **Community Detection** algorithms applied on a **graph** of anomalous posts built for each week, more details are illustrated in sec. 6.4).

6.2 Anomaly Detection

After the pre-processing of the dataset, it was possible to proceed with the detection of anomalies. To do this, as mentioned above, four state-of-the-art classical methods have been implemented: an ARIMA model, the Boxplot Rule method, the Isolation Forest and the Z-Score method.
The first definition to be given was the meaning of "anomalies" in our research, and we concluded that a post is identified as an outlier if it is assigned a score

(calculated as described in the sec. 5.4.4) that is too overperforming or, conversely, too much underperforming, i.e. a score which deviates significantly from the scores usually received by the influencer who created the post.
In detail, to detect such anomalies, we **individually** analyzed the various influencers of the dataset and the **time series of the score** assigned to the posts they published during the six years mentioned above. The outliers were then identified for each individual time series, therefore with respect to the **context** of the **post-score history of each influencer**.
In total, with the various techniques proposed, **106 866 anomalous posts** were extracted (removing duplicates, i.e. posts in common found by multiple methods), corresponding to approximately **7.6% of the purged dataset**, that contained 1 400 697 posts. Among these anomalous posts, 43 674 received an overperforming score and 63 192 an underperforming one.
It is specified that the following methods and algorithms presented do not necessarily have to be performed in sequence, but separately from each other, with the aim of choosing the best (s). Furthermore, the subsequent operations (Textual Analysis and Clustering) are performed individually for each group of outliers detected by each method, so that, in the end, it is possible to identify the sequence of methodologies which, as a whole, provides the best possible results.

6.2.1 ARIMA Model

Time series data, as stated earlier (sec. 4.2.1), is strictly sequential and very susceptible to auto-correlation. Time series Machine Learning models, whichever model is chosen, would use the data to train and find the general behavior of the data in order to try to predict the data: if an observation is normal, the forecast would be as close to the actual value as possible; otherwise, if an observation is an outlier, the forecast would be as far away from the real value as possible. As a result, outliers in the data can be discovered by examining the prediction errors, in particular by performing the following steps:

1. Determine whether the data are stationary and, if not, transform them to stationary (with an ARIMA model, such conversion can be performed by using differencing);
2. Fit the pre-processed data to a time series model (find the best order for the ARIMA model);
3. Calculate the squared error for each data observation;
4. Establish a data error threshold;
5. If an observation's errors exceed such threshold, we can classify such observation as an anomaly.

In order to use ARIMA time series forecasting, the first step was to check whether the time series of the score assigned to the posts of each influencer was stationary or not, and this could be tested through the **Augmented Dickey-Fuller (ADF) test**. This test returns the statistics, which contain a p-value, used to determine whether the data is stationary or not: if this value is less than 0.05 (called the "significance level", which corresponds to 95%), the data is stationary, otherwise, if the p-value is greater, the data are not stationary, i.e. they have no constant mean, variance and auto-correlation.
The Python `statsmodel` package provides a reliable implementation of the ADF test via the `adfuller()` function (from `statsmodels.tsa.stattools`), and also of the ARIMA model (from `statsmodels.tsa.arima.model`), which are the ones we exploited for this research.
The second step was to determine the number of **differencing** (up to a maximum of second order differencing, it makes no sense to differentiate further) required to convert the time series data of each influencer into stationary, if they were not already, and, after that, we could use the resulting data to train the ARIMA model.

It is remembered that the ARIMA model has **three parameters** namely $\boldsymbol{p}$, $\boldsymbol{q}$ and $\boldsymbol{d}$: p stands for AR (Auto Regression), which uses the previous lags to model the data, and q for MA (Moving Average), which uses the previous forecast errors for modelling the data, and d, which is the order of differencing. In particular, Automatic Regression utilizes the data lags as characteristics and the given data as a target to model and predict the relationship between the data, using the Least-Squares (or Linear Regression) approach.
In order to find the best order (p, q, d) for the model, we selected the combination of values (the order) that reduced the **Akaike Information Criterion (AIC)**, keeping in mind, as a general best practice, that the values of p, q and d are generally kept below 3 [27]. The Python code we implemented, fitted the model for different p, q and d values (each of which in the range [0,2]), recording the AIC for each combination (multi-threading was used to execute the code faster), obtained through the `aic` attribute (of each fitted model). After completing, we found the best order by using the **minimum AIC** value for the model.

The following step was to find the **Squared Error** for each and every post-score in the data: the `resid` attribute of the fitted results had the residuals of each observation and forecast.

Afterwards, we found the threshold for the squared errors above which the data was considered to be anomalous, by using the following formula:

$$\texttt{threshold} = \mu(\texttt{squared\_errors}) + (z * \sigma(\texttt{squared\_errors})),$$

where μ is the mean function, σ is the standard deviation function and z, in our case, is an integer equal to 1.

The predictions contained the integers 0 and 1, indicating normal and anomalous observations respectively.

Results

With the proposed method, namely the ARIMA model, we extracted from the dataset **48 656 anomalous posts** in total, of which 19 728 with an overperforming score and 28 928 with an underperforming score. The total statistics, concerning also the usual subdivision of influencers in classes (chap. 3), are illustrated in Table 6.1. The column "Influencers" refers to the number of influencers, for the corresponding category, who published at least an anomalous post.

	Influencers	**Tot. Anom.**	**Overperf.**	**Underperf.**
All influencers	1 373	48 656	19 728	28 928
Mega infl.	173	9 515	4 227	5 288
Macro infl.	1 147	38 538	15 232	23 306
Micro infl.	52	600	266	334
Nano infl.	1	3	3	0

Table 6.1: Statistics of anomalies found by the ARIMA Model in the post-score history of each influencer, subdivided per class

An example of output of the results is shown in Figure 6.2:

Elisabetta Franchi[1]

Fitting ARIMA with order p, d, q = (0, 0, 0) ...
Fitting ARIMA with order p, d, q = (0, 0, 1) ...
Fitting ARIMA with order p, d, q = (0, 0, 2) ...
Fitting ARIMA with order p, d, q = (1, 0, 0) ...
Fitting ARIMA with order p, d, q = (1, 0, 1) ...
Fitting ARIMA with order p, d, q = (1, 0, 2) ...
Fitting ARIMA with order p, d, q = (2, 0, 0) ...
Fitting ARIMA with order p, d, q = (2, 0, 1) ...
Fitting ARIMA with order p, d, q = (2, 0, 2) ...

Minimum AIC : 88033.05
Minimum order : (0, 0, 1)
Threshold : 34256.82

[1] `www.instagram.com/elisabettafranchi`

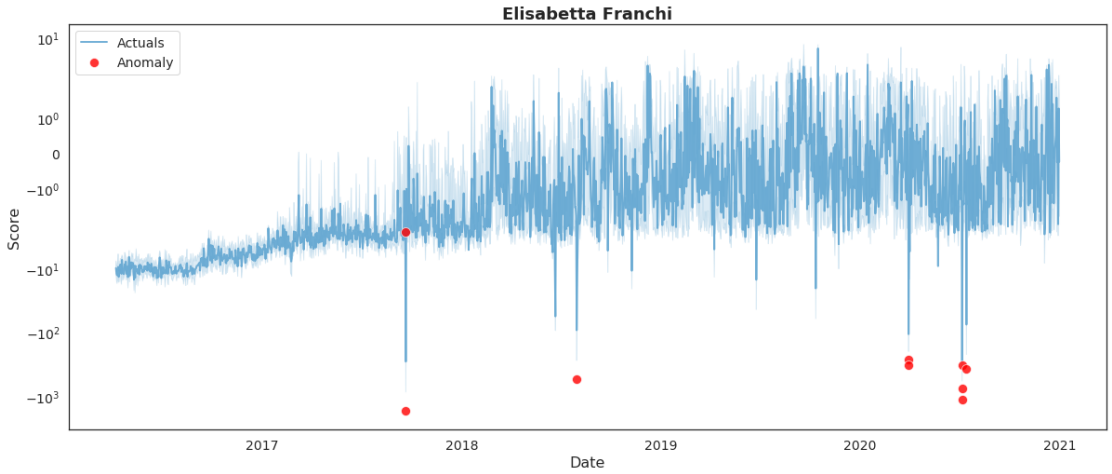


Figure 6.2: Example of anomalies (9 in total) found by the ARIMA Model in the post-score history of the Italian Mega influencer Elisabetta Franchi

During the tuning of the parameters, shown immediately above the Figure 6.2, it can be seen that the value for parameter d was set directly to 0 and did not vary as p and q: this means that the ADF test had given a "stationary" result for the time series of the post-scores of this influencer, so that it was not necessary to differentiate the data.

The influencer Elisabetta Franchi was chosen and reported as an example as it turned out to have the greatest disparity in the number of anomalies detected by the various methods proposed.

6.2.2 Boxplot Rule Method

The Boxplot Rule method applied to the Anomaly Detection field is based on verifying that the tested values have exceeded a certain threshold.
As explained in the chapter on *Relevant Theory* (sec. 4.2.2), we computed **two thresholds**, a lower limit and an upper limit, using the formulas already described:

$$\begin{aligned} \text{IQR} &= Q_3 - Q_1 \\ \text{lower\_limit} &= Q_1 - 1.5 * \text{IQR} \\ \text{upper\_limit} &= Q_3 + 1.5 * \text{IQR}, \end{aligned}$$

where Q_1 and Q_3 are, respectively, the first and third quartile and IQR is the Interquartile Range.
It should be remembered that the zero quartile (Q_0), the first (Q_1), the second (Q_2, the *median*), the third (Q_3) and the fourth quartile (Q_4) correspond with

the first modes whose cumulative percentage frequency is at least 0, 25, 50, 75 and 100 respectively. Therefore, Q_1 corresponds to the i_{th} mode if the cumulative percentage frequency $P_{i-1} < 25$ and $P_i \geq 25$, while Q3 corresponds to the i_{th} mode if the cumulative percentage frequency $P_{i-1} < 75$ and $P_i \geq 75$.
To calculate the quantiles Q_1 and Q_3, respectively, we applied the `quantile` function of the Pandas Python library to the *series* of the scores assigned to the posts of each influencer. In particular:

```
1    Q1 = series.quantile (0.25)
2    Q3 = series.quantile (0.75)
```

We then used the two thresholds to define as anomalies all the posts that had been assigned a score lower than `lower_limit` or greater than or equal to `upper_limit`.

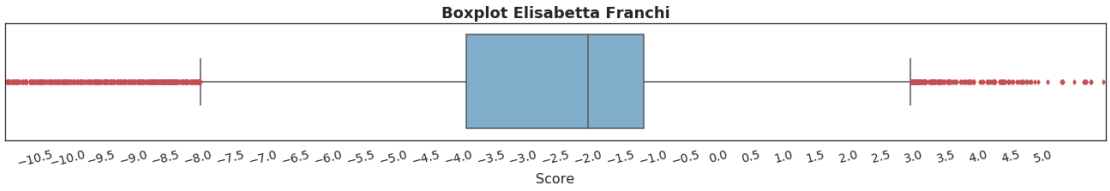
Results

With the proposed method, namely the Boxplot Rule method, we extracted from the dataset **77 812 anomalous posts** in total, of which 30 049 with an overperforming score and 47 763 with an underperforming score. The total statistics, concerning also the usual subdivision of influencers in classes, are illustrated in Table 6.2.

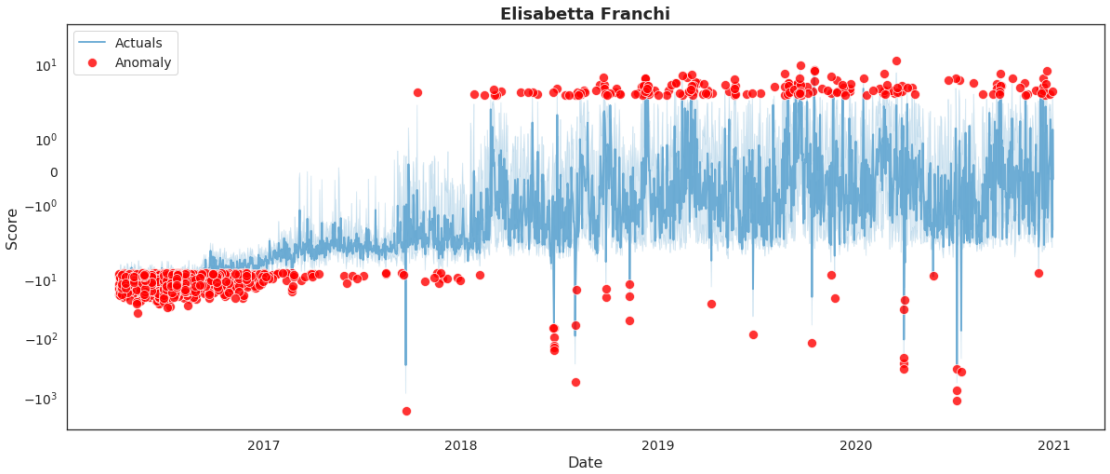
	Influencers	Tot. Anom.	Overperf.	Underperf.
All influencers	1 070	77 812	30 049	47 763
Mega infl.	159	25 572	9 896	15 676
Macro infl.	877	51 300	19 830	31 470
Micro infl.	33	933	317	616
Nano infl.	1	7	6	1

Table 6.2: Statistics of anomalies found through the Boxplot Rule method in the post-score history of each influencer, subdivided per class

An example of output of the results is shown in Figure 6.3:
Considering the high number of anomalies found for the influencer Elisabetta Franchi through this method, the Figure 6.3b clearly shows the range of values (in blue, separated by the red markers) within which the scores assigned to the posts are considered “normal”.



(a) Boxplot



(b) Time series with anomalies

Figure 6.3: Example of anomalies (3 193 in total) found through the Boxplot Rule method in the post-score history of the Italian Mega influencer Elisabetta Franchi

6.2.3 Isolation Forest

To implement the Isolation Forest algorithm we used the model already present in the Python `scikit-learn` library, which returns the anomaly score of each sample to be tested. The parameters of `sklearn.ensemble.IsolationForest` are the following [57]:

- **`n_estimators`**: `int`, default = 100
  The number of base estimators in the ensemble;

- **`max_samples`**: “auto”, `int` or `float`, default = ”auto”
  To train each base estimator, the number of samples to take from the input data. If the number of samples supplied is insufficient, all samples will be utilized for all trees (no sampling);

- **`contamination`**: ”auto” or `float` (in the range $(0, 0.5]$), default = ”auto”
  The quantity of contamination in the dataset, i.e. the percentage of outliers

in the dataset. When fitting the samples, this function is used to define the threshold;

- **`max_features`**: `int` or `float`, default = 1.0
  The number of features to use to train each base estimator from the input samples;

- **`bootstrap`**: `bool`, default = `False`
  Individual trees are fitted to random subsets of the training data selected with replacement if `True`; otherwise, sampling without replacement is executed;

- **`n_jobs`**: `int`, default = `None`
  The number of tasks to run in parallel for both fit and predict, -1 means using all processors;

- **`random_state`**: `int`, `RandomState` or `None`, default = `None`
  Manages the pseudo-randomness of feature and split values selection for each branching step and each forest tree. It is utilized to get consistent and reproducible outcomes over a number of different function calls;

- **`verbose`**: `int`, default = 0
  Manages the tree-building process's verbosity;

- **`warm_start`**: `bool`, default = `False`
  When `True`, repeat the previous call to fit-solution and add extra estimators to the ensemble; otherwise, fit a completely new forest.

All parameters have been left at the default value, except for `contamination` set to 0.01, assuming that the percentage of outlier points in the score-data of each influencer is 1% (no tuning performed), `n_jobs` set to 10 to not overload the server, `random_state` set to 42.
The estimator has been fitted to the score assigned to each influencer's posts and, for each sample, the `predict` method returns an integer as a label, -1 if the sample is an outlier, 1 if not.

Results

With the proposed method, namely the Isolation Forest algorithm, we extracted from the dataset **14 596 anomalous posts** in total, of which 4 271 with an overperforming score and 10 325 with an underperforming score. The total statistics, concerning also the usual subdivision of influencers in classes, are illustrated in Table 6.3.

An example of output of the results is shown in Figure 6.4:

	Influencers	Tot. Anom.	Overperf.	Underperf.
All influencers	1 377	14 596	4 271	10 325
Mega infl.	173	4 091	1 020	3 071
Macro infl.	1 152	10 250	3 152	7 098
Micro infl.	51	254	98	156
Nano infl.	1	1	1	0

Table 6.3: Statistics of anomalies found by the Isolation Forest algorithm in the post-score history of each influencer, subdivided per class

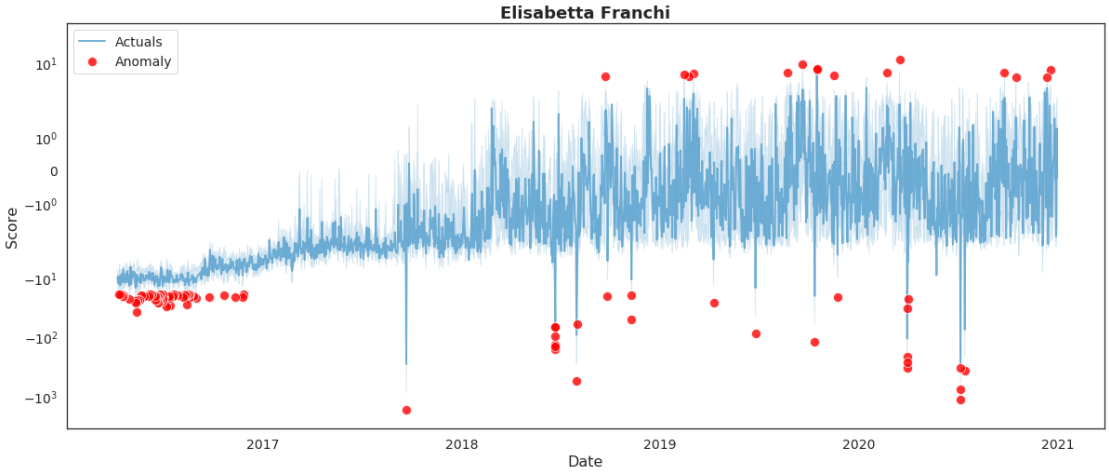


Figure 6.4: Example of anomalies (91 in total) found by the Isolation Forest algorithm in the post-score history of the Italian Mega influencer Elisabetta Franchi

6.2.4 Z-Score Method

Similarly to the Boxplot Rule method, also for the Z-Score method it is necessary to define **two thresholds**, one lower and one upper, beyond which the data objects are defined as outliers
Also in this case, the formulas used are the same as defined in the chapter of the *Relevant Theory* (sec. 4.2.4), where, in particular, the mean and the standard deviation have been calculated on the historical series of the scores assigned to the posts of each individual influencer:

$$\text{lower\_limit} = \mu - 3 * \sigma$$
$$\text{upper\_limit} = \mu + 3 * \sigma,$$

where μ is the mean and σ is the standard deviation, computed through the `pyspark.sql.functions` library methods, respectively `mean` and `stddev`, applied on the PySpark Column of the post-score for each influencer.

Results

With the proposed method, namely the Z-Score method, we extracted from the dataset **9 969 anomalous posts** in total, of which 1 649 with an overperforming score and 8 320 with an underperforming score. The total statistics, concerning also the usual subdivision of influencers in classes, are illustrated in Table 6.4.

	Influencers	**Tot. Anom.**	**Overperf.**	**Underperf.**
All influencers	1 023	9 969	1 649	8 320
Mega infl.	162	2 495	373	2 122
Macro infl.	827	7 325	1 233	6 092
Micro infl.	33	148	42	106
Nano infl.	1	1	1	0

Table 6.4: Statistics of anomalies found through the Z-Score method in the post-score history of each influencer, subdivided per class

An example of output of the results is shown in Figure 6.5:

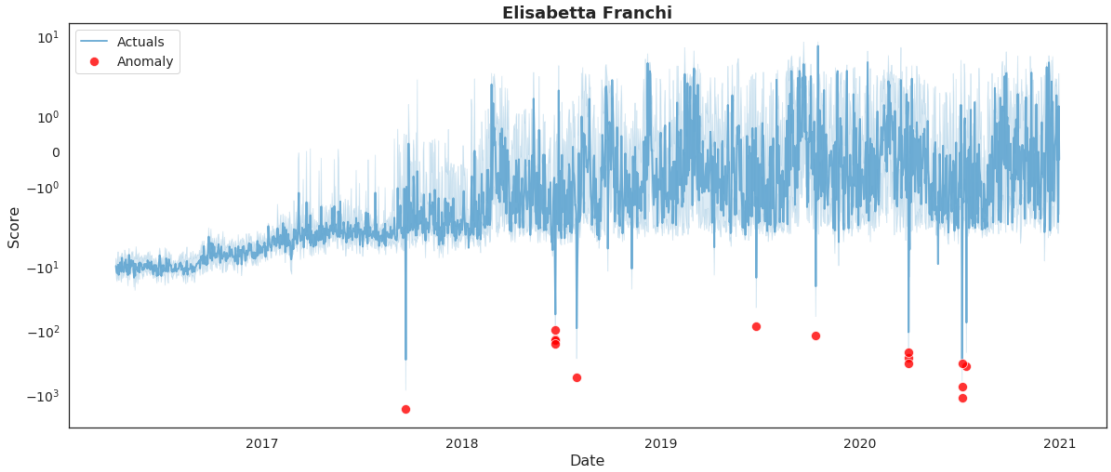


Figure 6.5: Example of anomalies (15 in total) found through the Z-Score method in the post-score history of the Italian Mega influencer Elisabetta Franchi

As also noted in the example reported for the ARIMA model, also with this method only anomalies relating to posts to which an underperforming score has been assigned (for Elisabetta Franchi) were detected. With the Boxplot Rule method and the Isolation Forest algorithm, on the other hand, anomalous posts that had received a score overperforming were also discovered.

6.2.5 Comparison of Anomaly Detection Methods and Results

The graph in Figure 6.6 shows the main characteristics (and their counts) of the various Anomaly Detection methodologies proposed, highlighting the differences from method to method.

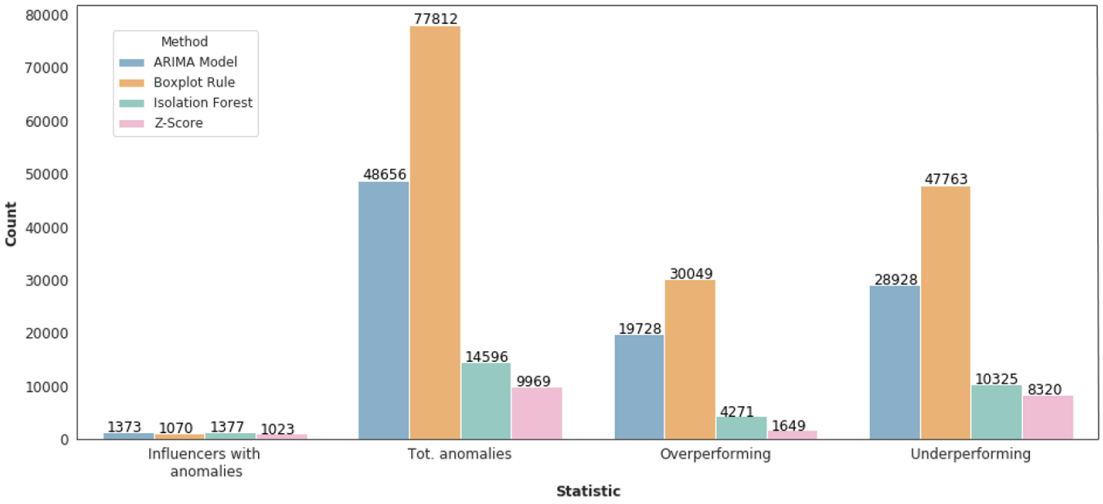


Figure 6.6: Comparison of results from different Anomaly Detection Methods

As for the number of **influencers** for which at least one anomalous post was detected, the counts are very similar to each other, with a difference of about 300 influencers between the results found by the ARIMA and Isolation Forest methods (1 373 and 1 377 influencers respectively) and those found by the methods of the Boxplot Rule and Z-Score (1 070 and 1 023 influencers, respectively).
It should be remembered that the total number of influencers in the filtered dataset (chap. 3) was 1 396, which means that:

- ARIMA Model: about 98.4% of influencers published at least one post identified as anomalous;
- Boxplot Rule method: about 76.7% of influencers published at least one post detected as an outlier;
- Isolation Forest algorithm: about 98.6% of influencers published at least one post identified as anomalous;
- Z-Score method: around 73.3% of influencers published at least one post detected as an outlier.

As regards the number of **total anomalies** detected, the Boxplot Rule method is the one to have found the largest number, followed by the ARIMA Model, the Isolation Forest and finally the Z-Score method, which is, instead, the one to have found the least number of anomalous posts.
Between the ARIMA model and the Boxplot Rule method there is a difference of about 29 thousand outliers, but the order of magnitude of the count is still sufficient to carry out the clustering phase (sec. 6.4). The Isolation Forest and the Z-Score method, on the other hand, found too few anomalies on the 6 years of the dataset (more than 36 thousand posts on average less than the ARIMA Model and even more than 65 thousand posts on average less compared to the Boxplot Rule method), so the results of future clustering step cannot be satisfactory.

Another interesting information attainable from the graph concerns the difference between the number of anomalous posts that have been assigned an **overperforming score** and those that have received an **underperforming score**. In fact, it can be noted that the former are less numerous than the latter, regardless of the Anomaly Detection method used. This could mean that the statistics assigned to the posts by CrowdTangle, on the number of reactions expected to get, in some cases are too optimistic, or endogenous/exogenous causes to the Social Network meant that these posts received a lower number of likes and comments than CrowdTangle expected.

The last interesting analysis concerns the comparison of the number of **anomalies** detected **per week** by each method, as it is the data that we need most for the clustering phase. A statistical description is in Table 6.5, while Figure 6.7 shows the ECDF of the distribution of the number of weekly anomalies by each method.

Overperforming - Underperforming					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Weeks with anomalies		283 - 315	275 - 304	247 - 308	218 - 300
Anomalies per week	Mean	69.7 - 91.8	109.3 - 142.6	17.3 - 33.5	7.6 - 27.7
	Median	77 - 94	113 - 146	17 - 30	6 - 25.5
	Std. Dev.	51.7 - 52.5	88.4 - 86.8	13.7 - 22.8	6.8 - 20.4
	Minimum	1 - 2	1 - 1	1 - 1	1 - 1
	Maximum	205 - 294	346 - 416	79 - 149	41 - 130

Table 6.5: Statistical description of weekly anomalies for each Anomaly Detection method

The results obtained are consistent with the total number of anomalies detected by each method, and therefore reflect the same numerical differences.

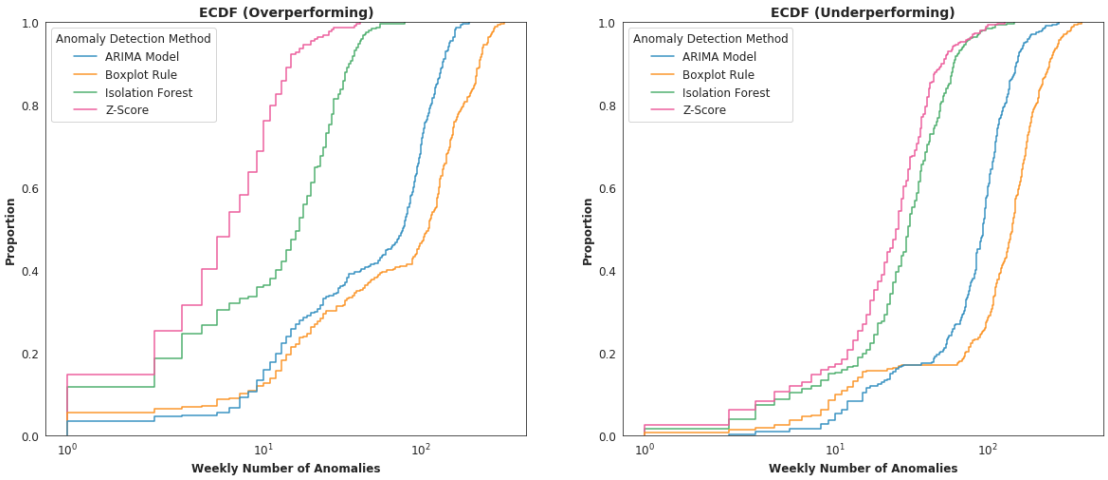


Figure 6.7: ECDFs of the distribution of the weekly number of anomalies for each Anomaly Detection method

6.3 Textual Analysis

In addition to the pre-processing of the data (chap. 5), before moving on to the next step, that of clustering, it is necessary to carry out other analyses on the textual attributes (the hashtags, the content of the caption and the text within the image) of the remaining posts, i.e. anomalous posts found in the previous phase. These analyses add further **machine-intelligible features**, which can be useful for clustering methods. Therefore, it can be said that this step is actually a sort of preparatory bridge, a transition between the previous phase and the next.

6.3.1 TF-IDF and Sublinear TF-IDF

As explained above (sec. 4.5.1), The term Frequency - Inverse Document Frequency (TF-IDF) is a numerical statistic that indicates the importance of a certain word to a document in a collection or corpus. The TF-IDF value is directly proportional to the number of times a word appears in the document and inversely proportional to the number of documents in the corpus that contain the term (to soften the weight of some words that appear more frequently in general).

In our case, the "**documents**" are the **influencers' accounts**: each user is represented by a vector. The words used by each influencer are grouped into a "sentence", a string of words separated by spaces, which is treated like a document. So, in other words, the weight of the aforementioned vector is calculated by a TF-IDF scheme where TF is the frequency of the keywords used by the influencer and IDF is the number of influencers who have used the keywords: the more an

influencer uses a certain word, the higher the TF-IDF of that word, but the more influencers will use that word, the lower the TF-IDF value. We used the same approach for the calculation of the Sublinear TF-IDF, which gives priority to TF over IDF.
To implement these vectors, we again used the Python `scikit-learn` library, in particular the `sklearn.feature_extraction.text.TfidfVectorizer`, which converts a collection of raw documents to a matrix of TF-IDF features, setting the `sublinear` parameter to `True` in case of the Sublinear TF-IDF.
The TF-IDF/Sublinear TF-IDF value was then calculated for each word used in each week and saved as a value in a key-value map stored for each anomalous post, where the key is the word (more details in sec. 6.3.2).

6.3.2 Final Data Transformation

In order to represent and describe both the textual properties, obtained by analyzing the anomalous posts found with each Anomaly Detection algorithm/method, and general weekly information about the Anomaly Detection performed, we implemented the following PySpark DataFrame schema:

- **`year`** (`int`): year of publication of the post;
- **`week`** (`int`): week of publication of of the post;
- **`n_anomalies`** (`int`): total number of anomalous posts of the week;
- **`n_anomalies_accounts`** (`int`): total number of accounts that published anomalous posts in the week;
- **`AccountID`** (`long`): unique identifier of the influencer account that published the post;
- **`name`** (`string`): name of the influencer account that published the post;
- **`postID`** (`string`): unique identifier of the post;
- **`date`** (`date`): exact publication date of the post;
- **`score`** (`double`): score of the post;
- **`hashtags_properties`**: `array` of `maps`: `key` = an hashtag, `value` = `struct`, (structure) with values calculated considering the week (there is a map for each word of the post, so they are grouped in an array). The sub-fields of such structure are the following:
  - **`posts_count`** (`long`): number of posts of the week that used this hashtag;

  - **posts_ids** (array of strings): list of unique identifiers of the posts of the week that used this hashtag;
  - **posts_prevalence** (double): ratio between the number of posts that used this hashtag and the total number of (anomalous) posts of the week;
  - **names_count** (long): number of influencer accounts of the week that used this hashtag;
  - **account_names** (array of strings): list of names of the influencer accounts of the week that used this hashtag;
  - **account_ids** (array of longs): list of unique identifiers of the influencer accounts of the week that used this hashtag;
  - **names_prevalence** (double): ratio between the number of accounts that used this hashtag and the total number of influencer accounts of the week;
  - **TF_IDF** (double): TF-IDF of the hashtag calculated in the week context (each document is an influencer account);
  - **Sublinear_TF_IDF** (double): Sublinear TF-IDF of the hashtag calculated in the week context (each document is an influencer account);

- **words_properties**: array of maps: key = a word, value = struct, with values calculated considering the week (there is a map for each word of the post, so they are in an array). The sub-fields of the structure are the same as hashtags_properties but obviously refer to the simple words of the caption, hashtags excluded;

- **image_words_properties**: array of maps: key = a word of the image, value = struct,
  with values calculated considering the week (there is a map for each word of the post, so they are in an array). The sub-fields of the structure are the same as hashtags_properties but obviously refer to the words within the post-image;

year	week	n_anomalies	n_anomalies_accounts	AccountID	name	postID	date
2020	19	114	78	81777	Chiara Ferragni	81777|2305399858668119570	2020-05-08
neg_my_score		**hashtags_properties**		**words_properties**		**image_words_properties**	
1.256494		[{'theferragnez': (2, ['81777|2303900680049784691', '81777|2305399858668119570'], 0.017543859649122806, 1, ['Chiara Ferragni'], [81777], 0.0128205128205128, 0.0, 0.0)}]		[{'ferragnez': (1, ['81777|2305399858668119570'], 0.008771929824561403, 1, ['Chiara Ferragni'], [81777], 0.0128205128205128, 0.0, 0.0)}]		[{'tiktok': (1, ['81777|2305399858668119570'], 0.008771929824561403, 1, ['Chiara Ferragni'], [81777], 0.0128205128205128, 0.0, 0.0)}]	

Table 6.6: Example of an entry after Anomaly Detection (performed by the ARIMA Model for week 19 of year 2020) and Textual Analysis. The anomalous post, with an overperforming score, is by the Italian Mega influencer Chiara Ferragni

This schema represents and describes the individual rows of the DataFrame, each of which containing an anomalous post found in the previous phase of the workflow. It is natural that in the rows representing the posts published in the same week, the fields `year`, `week`, `n_anomalies` and `n_anomalies_accounts`, are the same for each post (row) of that week, and are therefore repeated.

6.4 Clustering

As stated earlier, the final phase involves clustering **weekly anomalies** by **textual similarity**. Four algorithms have been implemented: the DBSCAN, applied on a pre-computed pairwise distance-matrix involving the captions of anomalous posts, the K-Means, applied on a weighted (by TF-IDF values) hashed vector of the words contained in the caption of the posts, an LDA model, and Community Detection algorithms, applied on the graph of anomalous posts, built for each week.
We have chosen a **weekly time granularity** because it is the most reasonable time interval in order not to have anomalous posts attributable to too many events external events, in the case of a longer time interval, or too few or none, in the occurence that the chosen time granularity was minor.
Since our goal is to find the best pair of Anomaly Detection method - Clustering method, all results are analyzed and compared also **for each Anomaly Detection technique**. Furthermore, the distinction between anomalous posts to which an **overperforming** score has been assigned and those with an **underperforming** score is maintained.
The analyses are carried out both on **hashtags only** and on **all types of words** extracted from the caption and from the image, to understand which of the two is more precise, as analyzing only the hashtags could be too simplistic, but grouping the posts that use all words could lead to overlapping groupings, and therefore to put posts in the same cluster that should actually be separated (because they do not concern the same topic).
When referring to all words, further clarification is required. To reduce the number of terms and try to improve the results, we filtered the words contained in the caption (hashtags and words within the image excluded), taking into consideration only those that had been assigned, from the previous textual analysis, a strictly positive value (greater than 0) of the TF-IDF or, otherwise, a strictly positive value (greater than 0) of the Sublinear TF-IDF.

6.4.1 DBSCAN Algorithm

To implement the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm (sec. 4.4.2), we used the model already present in the Python `scikit-learn` library, which performs DBSCAN clustering from vector array or

distance matrix and returns a label assigned to each sample tested. Essentially, it finds core samples of high density and expands clusters from them.
The parameters of `sklearn.cluster.DBSCAN` are the following [58]:

- **`eps`**: `float`, default = 0.5
  The largest distance between two samples at which one can be called a neighbor of the other. This isn't a maximum distance between points inside a cluster;

- **`min_samples`**: `int`, default = 5
  The number of samples required for a point to be deemed a core point in a surrounding community. This includes the actual point;

- **`metric`**: `str` or callable, default = "euclidean"
  The measure to calculate the distance between instances in a feature matrix. The input sample is supposed to be a distance matrix and must be square if the metric is "precomputed";

- **`metric_params`**: `dict`, default = `None`
  Additional metric function keyword arguments, if necessary;

- **`algorithm`**: {"auto", "ball_tree", "kd_tree", "brute"}, default = "auto"
  The NearestNeighbors module's algorithm for calculating punctual distances and locating the nearest neighbors;

- **`leaf_size`**: `int`, default = 30
  Leaf size passed to BallTree or cKDTree (two of the algorithms of the previous parameter). This can have an impact on the speed with which the tree is built and queried, as well as the amount of memory required to hold the tree;

- **`p`**: `float`, default = `None`
  Minkowski metric's power to determine distance between points. If `None`, then `p` = 2 (equal to the Euclidean distance);

- **`n_jobs`**: `int`, default = `None`
  The number of tasks to run in parallel for both fit and predict, -1 means using all processors;

All parameters have been left at the default value, except for `metric` set to "precomputed", as the input is a weekly pre-calculated square distance matrix, `n_jobs` set to 10 to not overload the server, `min_samples` set to 2, i.e. a core point should have at least two samples (the point itself included, so there should be at least one sample (post)) in its neighborhood (with a caption similar to its), and the value of `eps` depending on the particular week.
According to Rahmah et al. [59], one technique for automatically determining the

optimal ϵ value involves calculating the average distance between each point and its nearest k neighbors, where k = the selected value `min_samples`. The average k-distances are then plotted in ascending order on a k-distance graph and the optimal value for ϵ corresponds to the **point of maximum curvature** (i.e. where the graph has the greatest slope).
In our case, the distances have already been calculated, so we can simply plot the distribution, the **ECDF diagram**, of these **distances**. Empirically, it can be noted that for each week the curves all rise in proximity to the distance value 1, so that, by interpolating, the **ordinate value** corresponding to **0.995 on the abscissa axis** is automatically chosen as the optimal epsilon value.

Distance Matrix

The distance matrix is pre-calculated to be used as a direct input to the DBSCAN algorithm. It is a **symmetrical square matrix** of dimensions $n \times n$, where n is the number of anomalous posts of the week (n varies according to the week and the Anomaly Detection method used). The anomalous posts are placed symmetrically along the first row and the first column of the matrix, in such a way as to calculate the "textual distance" between the captions of all the possible pairs of such posts. Then, taking the posts two by two, the distance between their captions is a number in the range [0,1] and is calculated as follows:

1. The two lists of hashtags/words are converted to **sets**, s_1 and s_2, so as to remove duplicate words and not offset the count;

2. If the obtained sets are identical, the distance is minimum, i.e. 0;

3. The **partial intersection** of each pair of words not already contained in the intersection of the two sets is calculated, using the `get_matching_blocks` function of the SequenceMatcher class belonging to the Python `difflib` library: it returns lists of triples describing matching sequences. Each triple is of the form (i, j, n), and means that $s_1[i : i + n]$ is equal to $s_2[j : j + n]$. Adjacent triples always describe non-adjacent equal blocks;

4. For each pair of words not already contained in the intersection of the two sets, if there is at least one sub-sequence of at least 3 characters in common between the two words, the length of the matching block (or the sum of the lengths of the matching blocks) is divided by the total length of the two words, in order to have a measure of how similar the two words actually are. This value is added to the current value of partial intersection;

5. Once all these pairs of words have been compared, the final value of the partial intersection is divided by the number of comparisons made (recorded and increased by 1 for each combination of pairs of terms);

6. Finally, the Jaccard Index definition (sec. 4.4.2) is used, but adding the partial intersection value to the intersection size. The **Jaccard *distance*** is the complementary, and so it is obtained by subtracting the Jaccard Index from 1. The final formula is:

$$1 - \frac{|s_1 \cap s_2| + partial\_intersection}{|s_1 \cup s_2|}.$$

   Intersection and union are are calculated by means of the homonymous pre-existing functions for Python sets.

For the sake of clarity and completeness, the code for the distance function is the following:

```
def distance (lst1, lst2):

    s1 = set (lst1)
    s2 = set (lst2)

    if(s1 == s2):
        return 0

    partial_inters = 0

    for word1 in s1 - s1.intersection(s2):
        combinations = 0

        for word2 in s2 - s1.intersection(s2):
            match = SequenceMatcher(None, word1, word2).get_matching_blocks()
            match_sizes_sum = sum ([m.size for m in match if m.size > 2])

            if(match_sizes_sum > 0):
                partial_inters = partial_inters + float(
                match_sizes_sum / (len(word1) + len(word2)))
                combinations = combinations + 1

        if(combinations > 0):
            partial_inters = float(partial_inters / combinations)

    return 1 - float((
    len(s1.intersection(s2)) + partial_inters) / len(s1.union(s2)))
```

This function is passed as the `metric` parameter to the **`pdist`** function of the Python library SciPy (`scipy.spatial.distance.pdist`), which computes pairwise distances between observations in n-dimensional space, using the distance `metric`. From the same package, the function **`squareform`** converts the resulting vector-form distance array to a square-form distance matrix.

Results

In the Tables below, 6.7 and 6.8, the "Percentage of noise" indicates the percentage of posts classified as noise with respect to the number of anomalies of the week,

Anomalous posts with an OVERPERFORMING score					
Hashtags - All words					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Epsilon (ϵ)	Mean	0.33 - 0.3	0.35 - 0.31	0.41 - 0.37	0.48 - 0.44
	Median	0.31 - 0.28	0.33 - 0.29	0.34 - 0.3	0.37 - 0.33
	Std. Dev.	0.11 - 0.11	0.11 - 0.12	0.23 - 0.23	0.27 - 0.28
	Minimum	0.13 - 0.07	0.1 - 0.12	0.08 - 0.08	0.13 - 0.08
	Maximum	1 - 1	1 - 1	1 - 1	1 - 1
Noise posts	Mean	32.1 - 0.5	43.6 - 0.1	8.9 - 0.2	3.9 - 0.19
	Median	38 - 0	50.5 - 0	9 - 0	3 - 0
	Std. Dev.	20.9 - 0.8	26.7 - 0.3	5.9 - 0.5	3.1 - 0.6
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	86 - 4	119 - 1	34 - 2	19 - 4
Percentage of noise	Mean	46.8% - 0.7%	46.1% - 0.1%	46% - 1.3%	43.8% - 2.7%
	Median	47% - 0%	44.4% - 0%	46.7% - 0%	50% - 0%
	Std. Dev.	14.2% - 1.3%	17.3% - 0.2%	22.8% - 3.8%	27.8% - 10.8%
	Minimum	0% - 0%	0% - 0%	0% - 0%	0% - 0%
	Maximum	100% - 8.3%	100% - 2.3%	100% - 33.3%	100% - 100%
Number of clusters	Mean	2.6 - 1.1	5.4 - 1	1.2 - 1	1 - 1
	Median	2 - 1	5 - 1	1 - 1	1 - 1
	Std. Dev.	1.6 - 0.3	4.1 - 0.1	0.7 - 0.2	0.5 - 0.13
	Minimum	0 - 1	0 - 1	0 - 1	0 - 0
	Maximum	8 - 3	17 - 2	4 - 3	3 - 2
Silhouette Score	Mean	0.63 - −0.51	0.57 - −0.87	0.46 - −0.78	0.29 - −0.84
	Median	0.65 - −1	0.63 - −1	0.7 - −1	0.62 - −1
	Std. Dev.	0.25 - 0.63	0.32 - 0.4	0.62 - 0.48	0.7 - 0.44
	Minimum	−1 - −1	−1 - −1	−1 - −1	−1 - −1
	Maximum	1 - 0.43	0.89 - 0.4	0.95 - 0.4	1 - 0.79

Table 6.7: Statistical description of weekly DBSCAN clustering on anomalous posts with an overperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

while the Silhouette Score (sec. 4.4.3), which compares how similar an object is to its own cluster (cohesion) to other clusters (separation), in some cases is -1: because all the posts of the week have been classified as noise and therefore no cluster has been found, or a single cluster has been identified.
Given the results, in particular the Silhouette Score, it is superfluous to represent the ECDFs of the distribution of the weekly number of clusters and the Silhouette Score for clustering based on all types of words, and therefore the two distributions are reported only for the hashtags of the anomalous posts with an overperforming (Fig. 6.8) and underperforming score (Fig. 6.9).

Anomalous posts with an UNDERPERFORMING score					
Hashtags - All words					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Epsilon (ϵ)	Mean	0.35 - 0.33	0.36 - 0.36	0.4 - 0.36	0.44 - 0.39
	Median	0.32 - 0.29	0.31 - 0.3	0.34 - 0.3	0.37 - 0.32
	Std. Dev.	0.14 - 0.15	0.16 - 0.18	0.2 - 0.2	0.22 - 0.21
	Minimum	0.1 - 0.1	0.13 - 0.13	0.16 - 0.12	0.14 - 0.1
	Maximum	1 - 1	1 - 1	1 - 1	1 - 1
Noise posts	Mean	40.3 - 0.65	64.8 - 0.17	14.3 - 0.2	11.3 - 0.2
	Median	38 - 0	57 - 0	12 - 0	9 - 0
	Std. Dev.	24.9 - 0.5	42.9 - 0.5	10.1 - 0.6	9.1 - 0.6
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	146 - 4	225 - 4	63 - 4	59 - 4
Percentage of noise	Mean	45.2% - 0.61%	44.7% - 0.5%	44.4% - 0.8%	39.6% - 1.2%
	Median	43.5% - 0%	45.9% - 0%	44.4% - 0%	40.6% - 0%
	Std. Dev.	15.1% - 5.8%	15.6% - 5.8%	22% - 2.4%	21.3% - 6.4%
	Minimum	0% - 0%	0% - 0%	0% - 0%	0% - 0%
	Maximum	100% - 100%	100% - 100%	100% - 25%	100% - 100%
Number of clusters	Mean	3.9 - 1.2	7.3 - 1.1	2.1 - 1.1	1.9 - 1
	Median	3 - 1	6 - 1	2 - 1	2 - 1
	Std. Dev.	2.5 - 0.5	5.1 - 0.4	1.5 - 0.3	1.2 - 0.3
	Minimum	0 - 1	0 - 0	0 - 1	0 - 0
	Maximum	15 - 4	23 - 4	7 - 3	7 - 3
Silhouette Score	Mean	0.58 - -0.45	0.5 - -0.8	0.54 - -0.78	0.57 - -0.8
	Median	0.63 - -1	0.57 - -1	0.68 - -1	0.73 - -1
	Std. Dev.	0.29 - 0.65	0.33 - 0.49	0.52 - -0.49	0.52 - 0.46
	Minimum	-1 - -1	-1 - -1	-1 - -1	-1 - -1
	Maximum	0.91 - 0.46	0.91 - 0.67	0.99 - 0.56	0.98 - 0.67

Table 6.8: Statistical description of weekly DBSCAN clustering on anomalous posts with an underperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

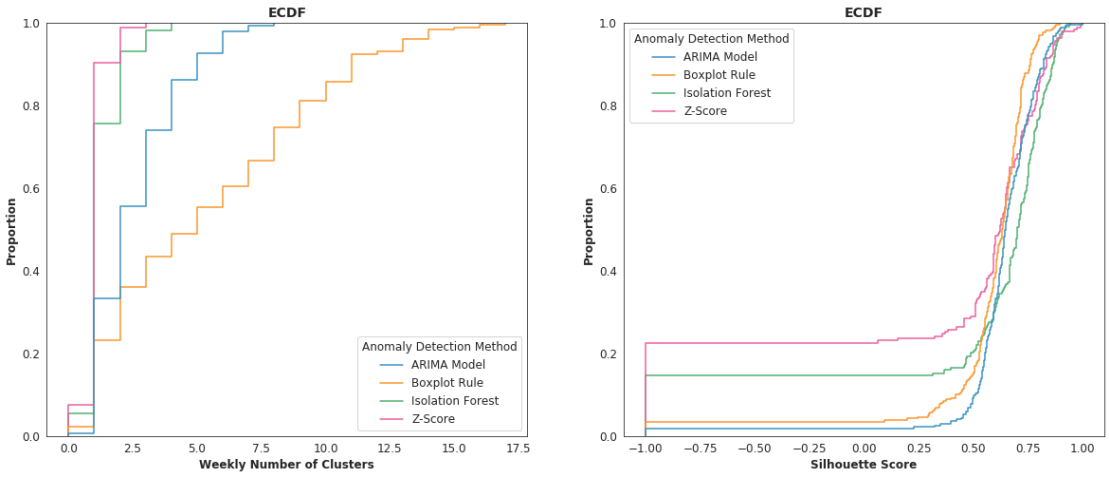


Figure 6.8: ECDFs of the weekly number of clusters (left) and of the Silhouette Score (right) for the DBSCAN applied on **hashtags** in the captions of anomalous posts with an **overperforming** score (for each Anomaly Detection method)

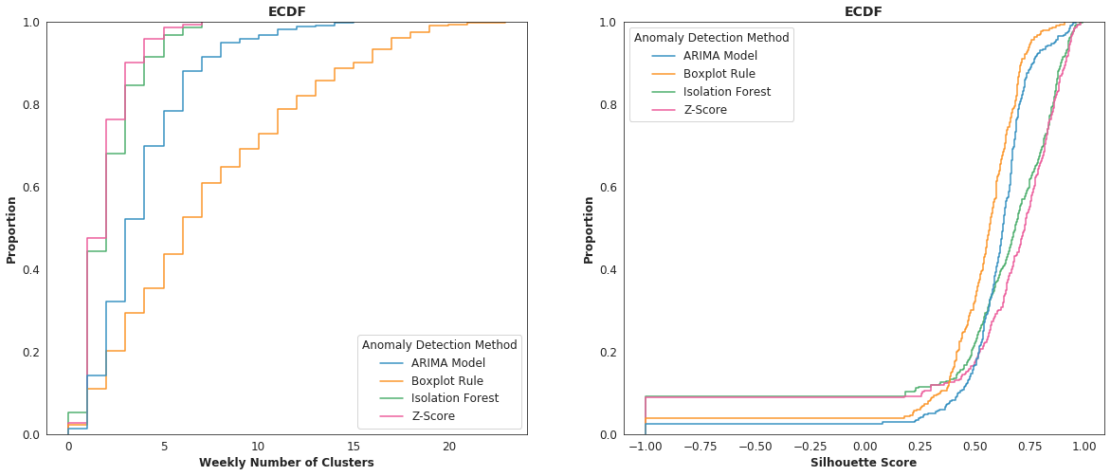


Figure 6.9: ECDFs of the weekly number of clusters (left) and of the Silhouette Score (right) for the DBSCAN applied on **hashtags** in the captions of anomalous posts with an **underperforming** score (for each Anomaly Detection method)

By looking at the results, it can be seen that the **number of clusters** found on average per week increases as the number of weekly anomalies detected increases, and therefore also depends on the Anomaly Detection method chosen. In this regard, it is also noted that the number of clusters found for posts with an underperforming score is greater than for those with an overperforming one, for the same reasons. Then by looking at the statistics regarding the **Silhouette Score**, it turns out that

the best results come from the **pair ARIMA Model - DBSCAN on hashtags** for the posts with an overperforming score and the pair Z-Score - DBSCAN on hashtags for the posts with an underperforming score. However, the number of anomalous posts detected with the Z-Score method is much less, so the best choice falls on the same couple anyway (ARIMA Model - DBSCAN on hashtags). It is interesting, to see that the performances drop to peak if all the words are used instead of just the hashtags: as expected, in fact, we are faced with a few clusters, larger, and therefore containing posts that are actually unrelated , i.e. not linked to the same topic/event, whether internal or external to the OSN.
On the other hand, however, by using all types of words, the number of posts classified as **noise** decreases significantly, precisely because posts that should not actually be taken into consideration are instead placed in clusters.
There are no particular comments to make or particularly significant differences on the value of ϵ, except that the highest values are recorded for the anomalies detected with the Z-Score method.

Example

An example of output of the results is shown in the Figures below. The Figure 6.10

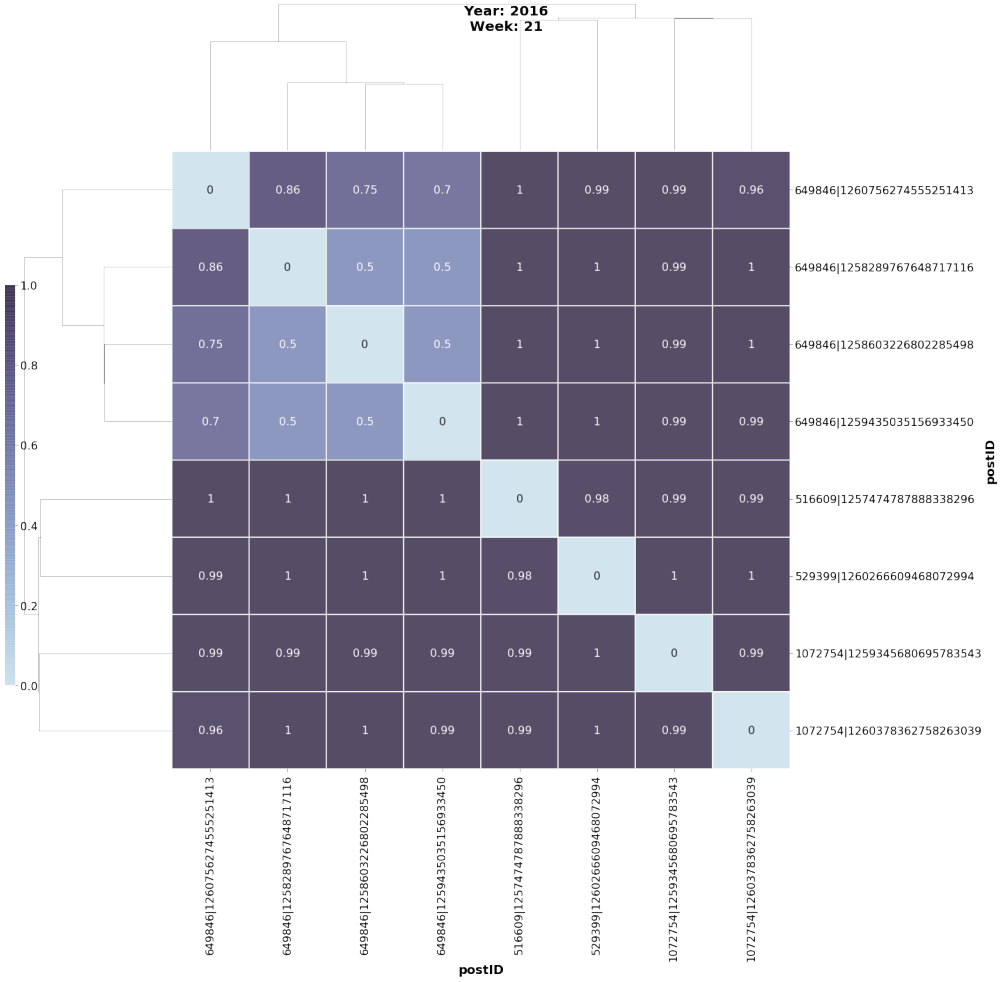


Figure 6.10: Example of distance matrix for hashtags used for the DBSCAN, for week 21 of year 2016 (anomalous posts with an overperforming score detected by the ARIMA Model)

represents the distance matrix computed for hashtags of the captions of anomalous posts (with an overperforming score, detected by the ARIMA Model) for week 21 of year 2016. The choice of the week and the Anomaly Detection method was dictated only by the small number of anomalous posts detected, to facilitate the visualization of the results.
The following step is to determine the best value for ϵ for the same posts, the ECDF is illustrated in Figure 6.11. The ordinate value, corresponding to 0.995 on

the abscissa axis, as well as the best value for ϵ, is 0.63 for the chosen week.

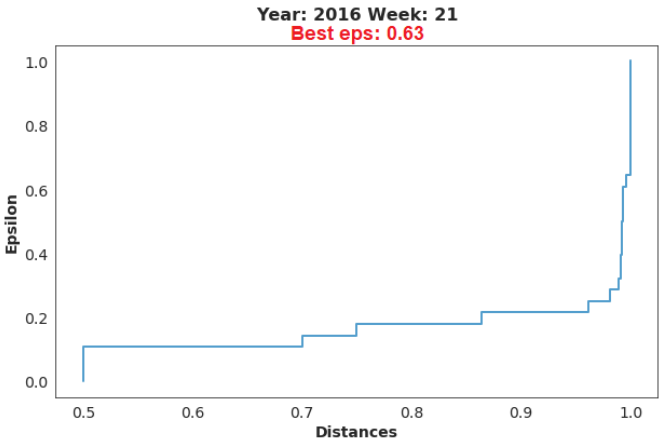


Figure 6.11: Example of determination the best value of ϵ used for the DBSCAN applied on hashtags: ECDF of pairwise textual distances for week 21 of year 2016 (anomalous posts with an overperforming score detected by the ARIMA Model)

The final result, with the label of the clusters on the right of each post, is in Table 6.9, which shows the entries of the final DataFrame.

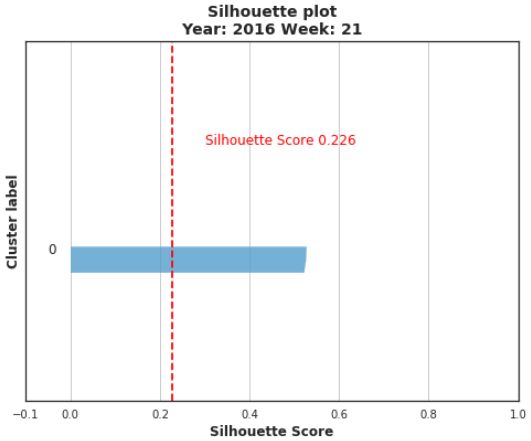
Year	Week	PostID	Name	List Hashtags	Cluster
2016	21	516609|1257474787888338296	Lodovica Comello	[aliceattraversolospecchio, tess]	-1
2016	21	529399|1260266609468072994	Karina Cascella	[senonvoleteuncanenonprendetelopernulla, ecomediceginni]	-1
2016	21	1072754|1260378362758263039	Tommaso Scala	[tourist, traveling, instapassport, vacation, trip, mytravelgram, holiday, travelgram, instatraveling, tourism, travel, travelblog, igtravel, travelling, visiting, traveler, instatravel, travelingram]	-1
2016	21	649846|1260756274555251413	Sergio SylVestre	[instore, bigboy, mondadori, napoli, stefanodemartino, sergiosylvestre, vulcanobuono]	-1
2016	21	1072754|1259345680695783543	Tommaso Scala	[mornings, ocean, naturelovers, seascape, beach, blue, nature, wave, lake, sand, wave, water, morning, waves]	-1
2016	21	649846|1258603226802285498	Sergio SylVestre	[amici15, sergiosylvestre, bigboy]	0
2016	21	649846|1258289767648717116	Sergio SylVestre	[amici15, ilvincitore, sergiosylvestre]	0
2016	21	649846|1259435035156933450	Sergio SylVestre	[amici15, bigboy, instore, tati, sergiosylvestre, ilvincitore]	0

Table 6.9: Example of entries after DBSCAN applied on hashtags for week 21 of year 2016 (anomalous posts with an overperforming score detected by the ARIMA Model). "cluster: -1" indicates a noise point, here not represented for the sake of brevity

The first 4 posts, which are assigned the cluster "-1", are classified as "noise", since

they have no hashtags in common with all the other posts of that week. Instead, it can be noted that the other posts, to which cluster number "0" is assigned, are attributable to the 2015/2016 edition of "Amici", a famous Italian television program, and therefore to an event external to the OSN.

(a) Wordcloud for the output clusters (light-blue represents the noise, red represents the only cluster)



(b) Silhouette Score plot

Figure 6.12: Example of outputs for the DBSCAN applied on hashtags, for week 21 of year 2016 (anomalous posts with an overperforming score detected by the ARIMA Model)

6.4.2 K-Means Algorithm

To implement the K-Means algorithm (sec. 4.4.3), we used the model already present in the Python `scikit-learn` library, which performs K-Means clustering from an array or sparse matrix. It computes cluster centers and predict cluster index for each input sample. In particular, K-Means employs vector quantization techniques to split the data space into Voronoi cells, which are then returned as clusters. The technique typically starts with a set of random locations called "seeds" and then alternates between two phases to pick fresh cluster *centroids*.
The parameters of `sklearn.cluster.KMeans` are the following [60]:

- **`n_clusters`**: `int`, default = 8
  The number of clusters that will created (and the number of centroids that will be produced);

- **`init`**: {"k-means++", "random"}, callable or array-like of shape (**`n_clusters`**, **`n_features`**), default = "k-means++"
  This is the initialization method:

  - "k-means++": intelligently picks initial cluster centers for K-Means clustering to accelerate convergence;
  - "random": choose n clusters observations (rows) at random from the data for the first centroids;

- **`n_init`**: `int`, default = 10
  The K-Means algorithm will be performed a number of times `n_init`, with various centroid seeds. The best output of successive runs in terms of inertia will be the final result;

- **`max_iter`**: `int`, default = 300
  For a single run, the maximum number of iterations of the K-Means;

- **`tol`**: `float`, default = To proclaim convergence, the difference in the cluster centers of two successive iterations must be within a certain range of tolerance in terms of the Frobenius norm;

- **`verbose`**: `int`, default = 0
  Manages the process's verbosity;

- **`random_state`**: `int`, `RandomState` or `None`, default = `None`
  For centroid initialization, this function generates random numbers to make the randomness deterministic. It's used to produce consistent and repeatable results across a variety of function calls;

- `copy_x`: `bool`, default = `True`
  If `copy_x` is set to `True` (the default), the original data is not altered, otherwise the original data is updated and then returned before the function returns; nonetheless, slight numerical discrepancies can be generated by removing and then adding the data mean;

- `algorithm`: {"auto", "full", "elkan"}, default = "auto"
  This is the K-means algorithm to use. By employing the triangle inequality, the "elkan" variation is more efficient than "full" on data with well-defined clusters, while being more memory heavy;

All parameters have been left at the default value, except for `max_iter` set to 100, `random_state` set to 42, and the value of `n_clusters`, or k, which historically is the most difficult parameter to determine for this algorithm, depending on the particular week.
To determine the **best k-value** for each week, we calculated three clustering quality measure coefficients: the Silhouette Score, and the Davies-Bouldin and Calinski-Harabasz indices (sec. 4.4.3), each of which for each k value in the range [2,10], the most reasonable interval if we think about the number of events that can be discussed during the same week. Thereafter, the k-value corresponding to the maximum Silhouette Score and Calinski-Harabasz Index and the minimum Davies-Bouldin Index is automatically chosen. If this is not found, because different values of k correspond to the best coefficients, the one that maximizes the Silhouette Score is chosen. An example is shown in Figure 6.13.

Hashing TF-IDF

The K-Means algorithm needs numeric features as input, so it is not possible to use word-lists extracted from each post or a distance matrix as we did previously for DBSCAN. For this reason, we decided to use as input the TF-IDF calculated for each word within the group of anomalous posts of each week (each post is considered a document), using a `HashingVectorizer`, which applies a **hashing function** to the frequency of terms counts in every document (i.e. converts a collection of text documents to a **sparse matrix of token occurrences**), to reduce the number of features, followed by the already mentioned `TfidfTransformer`. To implement them, we used the modules in the Python `scikit-learn` library, in particular in the `sklearn.feature_extraction.text` package. The hash function employed is the default one, i.e. the signed 32-bit version of *Murmurhash3*.
This strategy has several advantages, for example it requires very little memory and is scalable to large datasets because no vocabulary dictionary is stored in memory, but there are also some drawbacks (as compared to using a CountVectorizer), for example there is no method to perform the inverse transform (from feature indices

Year: 2020 Week: 4

K-value	Silhouette	Davies-Bouldin	Calinski-Harabasz
2	0.345925	0.535553	8.051934
3	0.373714	0.523532	7.371563
4	0.410531	0.505757	8.197083
5	0.438824	0.491774	8.220854
6	0.456870	0.615908	8.136499
7	0.467372	1.341273	8.061275
8	0.485808	1.264390	7.916505
9	0.488918	1.254008	7.515725
10	0.495489	1.208423	7.245301

(a) Example of k found for year 2020 and week 4, $k = 5$ chosen

Year: 2019 Week: 44

K-value	Silhouette	Davies-Bouldin	Calinski-Harabasz
2	0.266398	1.997199	4.554322
3	0.268517	2.148744	3.616452
4	0.292013	1.490590	4.067983
5	0.293206	1.313537	3.550918
6	0.301419	1.617076	3.752124
7	0.308961	1.590304	3.851020
8	0.310309	1.483249	3.617771
9	0.313585	1.512109	3.590518
10	0.314999	1.434121	3.451090

(b) Example of k not found for year 2019 and week 44, $k = 10$ chosen, corresponding to maximum Silhouette Score

Figure 6.13: Example of determination of the parameter k for the K-Means algorithm performed on hashtags of anomalous posts with an overperforming score detected by ARIMA Model

to string feature names), and **collisions** can occur (distinct tokens can be mapped to the same feature index). However this is rarely an issue, since the default number of features in the output matrices for `HashingVectorizer` is 2^{20}.

Cosine Similarity

The output of the `HashingVectorizer-TfidfTransformer` pipeline, a weighted term vector for each anomalous post of the week, is then used to construct the **similarity matrix**, a square symmetric matrix, using the Cosine Similarity method. The method is implemented using the `linear_kernel` function of the `sklearn.metrics.pairwise` module. Each value of the matrix is in the range [0,1], which increases the more similar the posts are, so a value of 0 indicates a pair of posts with nothing in common, and a value of 1 indicates two posts that contain exactly the same words. The resulting matrix is then finally used as K-Means input feature.

Results

In the Tables below, 6.10 and 6.11, the Silhouette Score in some cases is -1: because just one post is anomalous during that particular week and therefore no cluster has been found, or a single cluster has been identified.

Anomalous posts with an OVERPERFORMING score					
Hashtags - All words					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Number of clusters	Mean	7.2 - 7.4	7.1 - 7.2	6.3 - 6.5	3.6 - 4.2
	Median	10 - 10	10 - 10	9 - 10	3 - 4
	Std. Dev.	3.6 - 3.6	3.9 - 3.9	4.4 - 4.5	3.5 - 3.9
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	10 - 10	10 - 10	10 - 10	10 - 10
Silhouette Score	Mean	0.35 - 0.26	0.3 - 0.22	0.17 - 0.05	0.09 - -0.04
	Median	0.39 - 0.31	0.38 - 0.29	0.41 - 0.24	0.4 - 0.098
	Std. Dev.	0.32 - 0.3	0.36 - 0.34	0.6 - 0.53	0.68 - 0.59
	Minimum	-1 - -1	-1 - -1	-1 - -1	-1 - -1
	Maximum	1 - 0.78	0.8 - 0.8	0.86 - 0.75	1 - 0.93

Table 6.10: Statistical description of weekly K-Means clustering on anomalous posts with an overperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

Given the results, in particular the Silhouette Score, as for the DBSCAN algorithm, it is superfluous to represent the ECDFs of the distribution of the weekly number of clusters and the Silhouette Score for clustering based on all types of words, and therefore the two distributions are reported only for the hashtags of the anomalous posts with an overperforming (Fig. 6.14) and underperforming (Fig. 6.15) score.

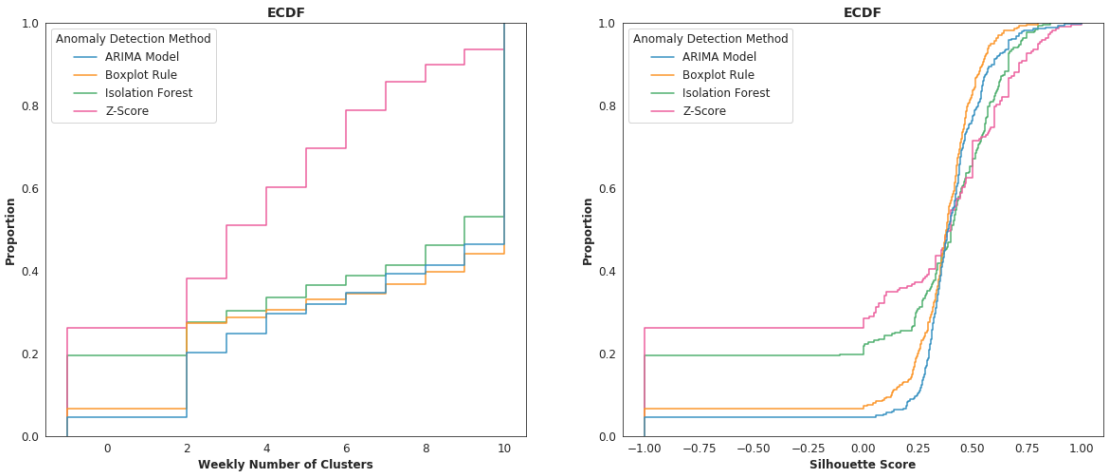


Figure 6.14: ECDFs of the weekly number of clusters (left) and of the Silhouette Score (right) for the K-Means applied on **hashtags** in the captions of anomalous posts with an **overperforming** score (for each Anomaly Detection method)

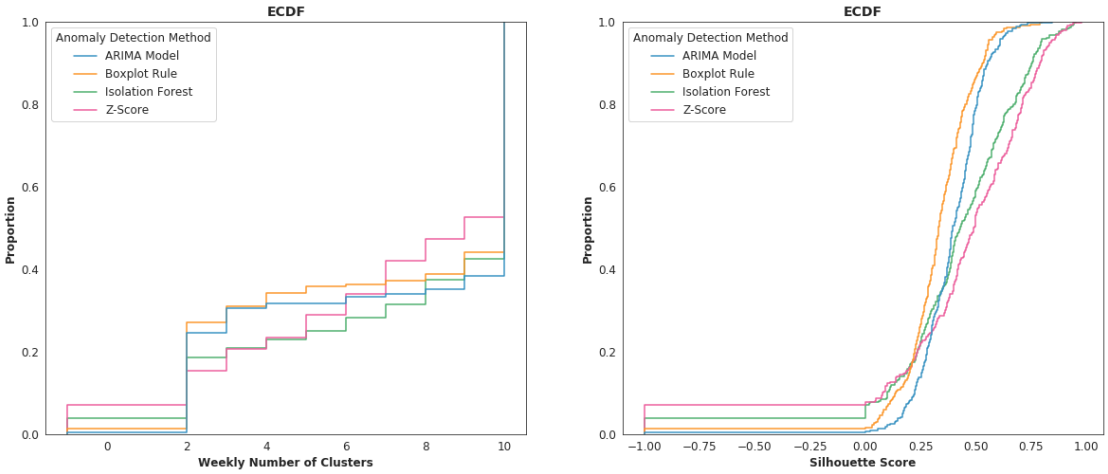


Figure 6.15: ECDFs of the weekly number of clusters (left) and of the Silhouette Score (right) for the K-Means applied on **hashtags** in the captions of anomalous posts with an **underperforming** score (for each Anomaly Detection method)

By looking at the results, it can be seen that the **number of clusters** found on average per week increases as the number of weekly anomalies detected increases, and therefore also depends on the Anomaly Detection method chosen. In this regard, it should also be noted that the number of clusters found for places with an underperforming score is higher than for those with an overperforming score, for the same reasons. In the latter case, however, the number of clusters found

Anomalous posts with an UNDERPERFORMING score					
Hashtags - All words					
		ARIMA M.	Boxplot R.	Isol. For.	Z-Score
Number of clusters	Mean	7.4 - 7.3	7.1 - 7.1	7.6 - 8	7.2 - 7.7
	Median	10 - 10	10 - 10	10 - 10	9 - 10
	Std. Dev.	3.6 - 3.6	3.7 - 3.6	3.5 - 3.4	3.5 - 3.5
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	10 - 10	10 - 10	10 - 10	10 - 10
Silhouette Score	Mean	0.39 - 0.32	0.32 - 0.26	0.4 - 0.29	0.4 - 0.31
	Median	0.39 - 0.33	0.33 - 0.28	0.43 - 0.31	0.49 - 0.37
	Std. Dev.	0.16 - 0.14	0.21 - 0.19	0.36 - 0.33	0.44 - 0.39
	Minimum	-1 - -1	-1 - -1	-1 - -1	-1 - -1
	Maximum	0.85 - 0.65	0.8 - 0.77	0.96 - 0.92	0.98 - 0.93

Table 6.11: Statistical description of weekly K-Means clustering on anomalous posts with an underperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

by K-Means is similar across all Anomaly Detection methods, regardless of the approach chosen.
Then by looking at the statistics regarding the **Silhouette Score**, it turns out that the best results come from the **pair ARIMA Model - K-Means on hashtags** for the posts with an overperforming score and the pair Z-Score - K-Means on hashtags for the posts with an underperforming score, which are the same results as for the DBSCAN algorithm. However, the number of anomalous posts detected with the Z-Score method is much less, so the best choice falls on the same couple anyway (ARIMA Model - K-Means on hashtags). It is interesting to see that, unlike DBSCAN, the performances do not drop to peak if all the words are used instead of just the hashtags: in this case, in fact, the performances are lower in the first case, but not too different.

Example

An example of output of the results is shown in the Figures below.
The first step is the transformation of the lists of hashtags of the captions of the 27 anomalous posts (with an overperforming score, detected by the Isolation Forest algorithm) for week 6 of year 2019, into a weighted (by the TF-IDF values of terms) hashed vector. Afterwards, the Cosine Similarity method is applied, and the Figure 6.16 represents the similarity matrix obtained.
The next step is to determine the best value for k (i.e. the number of clusters), which, as shown in Figure 6.17, has been automatically assigned to 9.

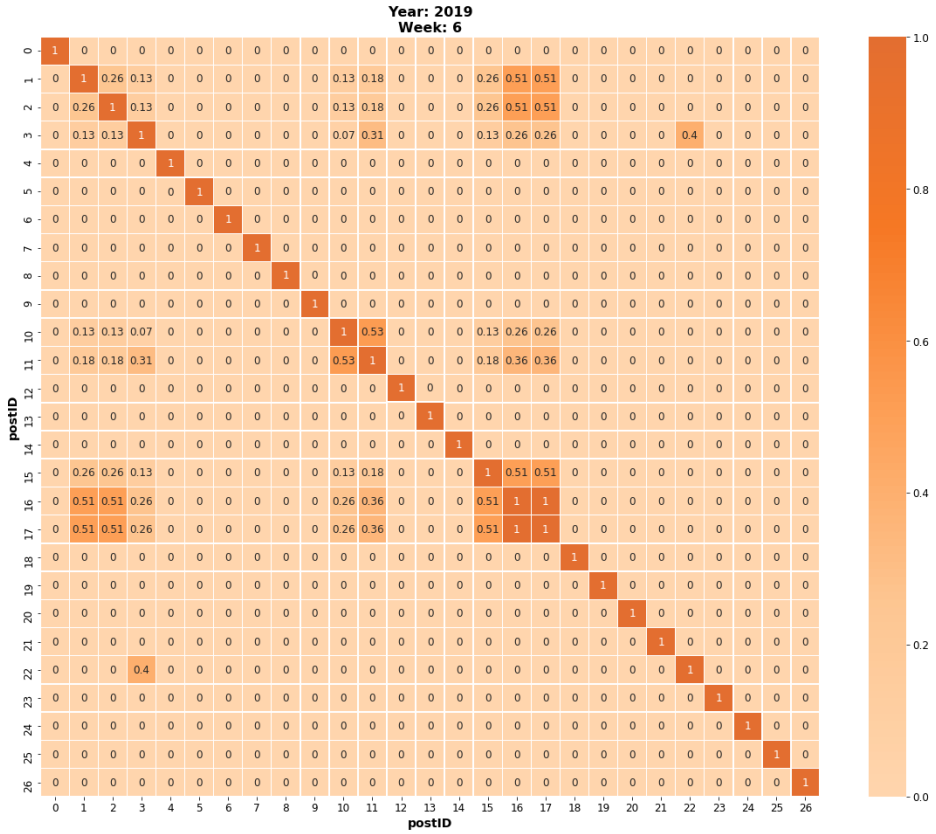


Figure 6.16: Example of similarity matrix for hashtags used for the K-Means, for week 6 of year 2019 (anomalous posts with an overperforming score detected by the Isolation Forest algorithm)

The final result with the entries of the DataFrame is not represented for reasons of brevity and visualization, but includes 9 clusters: the first containing 4 totally unrelated posts, the second containing 5 posts concerning the edition of the 2019 of the Sanremo Festival (of the Italian Song), the third and the eighth containing 2 posts related to the same event as above, and finally 5 clusters of 1 post each, some of which are related, others with nothing in common. A visual representation of the results is in Figures 6.18 and 6.19 , which shows the passage from the initial word bag (the wordcloud) and the subsequent subdivision into clusters carried out by the K-Means algorithm.
This example suggests that K-Means does not actually provide promising results, except in some cases, as often too many clusters are created, splitting content that actually needed to be grouped.

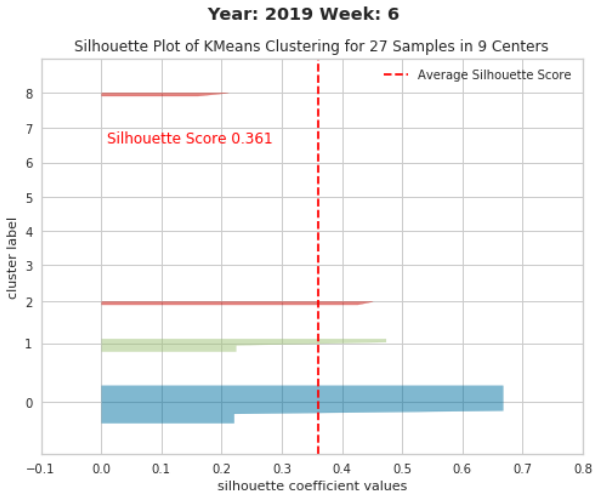


Figure 6.17: Example of Silhouette Score plot (with Silhouette Visualizer) for the K-Means applied on hashtags, for week 6 of year 2019 (anomalous posts with an overperforming score detected by the Isolation Forest algorithm)

Figure 6.18: Example of wordcloud built using hashtags, for week 6 of year 2019 (anomalous posts with an overperforming score detected by the Isolation Forest algorithm)

Figure 6.19: Example of wordclouds for each cluster built using hashtags, for week 6 of year 2019 (anomalous posts with an overperforming score detected by the Isolation Forest algorithm)

6.4.3 LDA Algorithm

Latent Dirichlet Allocation (LDA) (sec. 4.5.2) is not a clustering technique in the strictest sense of the word. This is due to the fact that clustering algorithms create a single grouping per clustered item, whereas LDA produces a distribution of groupings over the clustered items. When we apply LDA to a set of documents, the result is a probability distribution of groupings (or *topics*) for each document (rather than a single cluster). In other words, each document will be allocated to a topic distribution, with each topic having its own probability. LDA is not regarded a true (or *hard*) clustering method because it yields more than one topic per document, although it does yield topic groupings, so it is also referred to as a soft clustering method.
For this reason, it is presented here as an example, but not compared to the other methods. Another reason is that this algorithm is mostly used, precisely, on real text documents, while the words contained in the captions of posts on Online Social Networks are usually few.
To implement the Latent Dirichlet Allocation (LDA) algorithm, we used the model already present in the Python `gensim` library, which performs LDA taking a stream of document vectors or sparse matrix of shape (`num_documents`, `num_terms`) as input. In particular, we used `gensim.models.LdaMulticore` implementation, employing all CPU cores to parallelize and speed up model training by means of multiprocessing, that is equivalent to the `gensim.models.LdaModel` class, which is instead a more straightforward and single-core implementation.

It is not necessary to explain in detail all the parameters as they are very numerous, also because they have all been left at the default value. The only important parameters are the **input** (the *corpus*) and the **number of** requested latent **topics** to be extracted from the training corpus, which will be discussed below.

Lemmatization

Generally before using an LDA algorithm it is necessary to do Natural Language pre-processing, but in this case it is not necessary because we have already done so during the first phase of data elaboration. The only neglected NLP phase was lemmatization, to avoid changing the content of the captions too much. Lemmatization is the process of reducing an inflected form of a word to its canonical form, called *lemma*. In NLP, lemmatization is the algorithmic process that automatically determines the lemma of a given word.
Proceeding in order, each post was treated as a document, and the various word lists were transformed into strings separated by spaces, and analyzed separately. Each of these strings, each corresponding to a post caption, was further pre-processed and transformed into **tokens** using the `gensim.utils.simple_preprocess` function. Each token was finally lemmatized using the `lemmatize` function of the WordNetLemmatizer module of the Python `ntlk` library. We also tried to use the lemmatizer of the SpaCy library, and the relative language detection we have already talked about in the previous chapter (sec. 5.5.3), but the results remained almost the same.
Later, using the `gensim.corpora.Dictionary` function we created a **dictionary** that encapsulates the mapping between normalized words and their integer identifiers for each document in the list used as input. We then applied the `doc2bow` function to each document which converts each document into the bag-of-words format, i.e. a list of (`token_id`, `token_count`) tuples. We used the resulting BoW corpus (for each week) as input for the LDA model.

Coherence Model

To optimize the `num_topics` parameter and determine the best value of the number of topics for each week, we iterated several times the creation of an LDA model over the range [2,50] in steps of 2, for the number of topics parameter. For each model we calculated the **coherence of the discovered topics**, using the `gensim.models.CoherenceModel` function, and then we chose the value that maximized the coherence measure `c_v` for each week's LDA model. The `c_v` measure (reported in the Tables 6.12 and 6.13 of the statistic description of the results as "Coherence Score") is based on a sliding window, a segmentation into a set of the main words, and an indirect confirmatory measure, using the normalized point-like reciprocal information and Cosine Similarity.

Results

In the following Tables, 6.12 and 6.13, "Number of topics" can be compared to the usual "Number of clusters", while the "Coherence Score" can be compared to the usual "Silhouette Score", as an index of the goodness of the method. Since we have not particularly focused on this algorithm, for simplicity the ECDFs (Fig. 6.20 and Fig. 6.21) of the distributions of the number of topics and the coherence score of the algorithm applied only on the hashtags in the captions of anomalous posts are reported (the ECDFs with the statistics about the use of all types of words are missing), for each Anomaly Detection method used, as usual. Also the distinction between posts with an overperforming score and an underperforming score is maintained.

Anomalous posts with an OVERPERFORMING score					
Hashtags - All words					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Number of topics	Mean	14.3 - 14.3	16.5 - 14.7	10.7 - 11.3	13.6 - 12.9
	Median	12 - 12	12 - 10	10 - 10	7 - 10
	Std. Dev.	12.9 - 14.1	15.9 - 15.3	11 - 10.4	14.6 - 13.3
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	48 - 48	48 - 48	48 - 48	48 - 48
Coherence Score	Mean	0.63 - 0.64	0.63 - 0.64	0.61 - 0.66	0.53 - 0.61
	Median	0.63 - 0.63	0.63 - 0.63	0.67 - 0.7	0.5 - 0.6
	Std. Dev.	0.16 - 0.12	0.14 - 0.12	0.27 - 0.22	0.32 - 0.25
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	1 - 1	1 - 1	1 - 1	1 - 1

Table 6.12: Statistical description of LDA algorithm applied on weekly anomalous posts with an overperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

Looking at the results, it can be seen that the **number of topics** discovered is, unlike the other clustering methods, higher on average for anomalous posts with a score overperforming, with the exception of the anomalies detected by the Isolation Forest algorithm for which the 'exact opposite. This indicates that the number of weekly anomalies does not affect the number of topics detected, also because, to confirm this, there are no notable differences between the various Anomaly Detection methods used (the topics are slightly more numerous for the anomalous posts detected by the ARIMA model and the Boxplot Rule method). Furthermore, there are no particular differences taking into consideration only hashtags or all types of words (hashtags, simple words in the caption and words within the image): in general, strangely enough, the number of topics is on average greater or almost

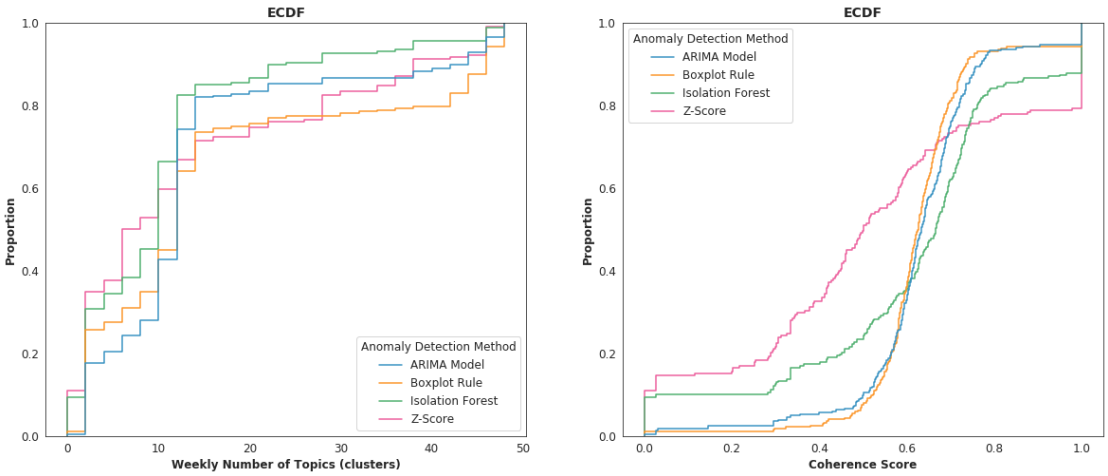


Figure 6.20: ECDFs of the weekly number of topics/clusters (left), and the Coherence Score (right) for the LDA algorithm applied on **hashtags** in the captions of anomalous posts with an **overperforming** score (for each Anomaly Detection method)

Anomalous posts with an UNDERPERFORMING score					
Hashtags - All words					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Number of topics	Mean	13 - 13.5	13.1 - 11.5	12.9 - 11.8	12.1 - 12.4
	Median	12 - 12	10 - 6	10 - 12	10 - 10
	Std. Dev.	11.1 - 11.9	14.4 - 13.5	11.3 - 9	11.3 - 10.8
	Minimum	0 - 2	2 - 2	0 - 2	0 - 0
	Maximum	48 - 48	48 - 48	48 - 48	48 - 48
Coherence Score	Mean	0.65 - 0.65	0.64 - 0.62	0.66 - 0.66	0.63 - 0.66
	Median	0.66 - 0.65	0.64 - 0.61	0.69 - 0.68	0.68 - 0.69
	Std. Dev.	0.11 - 0.08	0.1 - 0.09	0.16 - 0.14	0.18 - 0.15
	Minimum	0 - 0.17	0.05 - 0.2	0 - 0.03	0 - 0
	Maximum	1 - 1	1 - 1	1 - 1	1 - 1

Table 6.13: Statistical description of LDA algorithm applied on weekly anomalous posts with an underperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

equal using only the hashtags, compared to all the words of the posts.
Considering the **coherence score**, also in this case the values are very similar to each other, regardless of the type of performance score of the posts, the type of words used for the analysis and the Anomaly Detection method used. The only

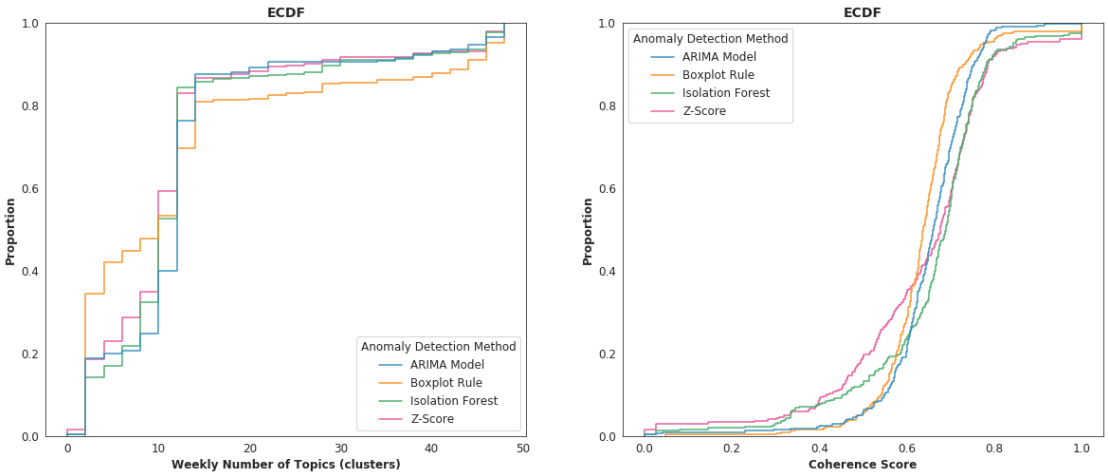


Figure 6.21: ECDFs of the weekly number of topics/clusters (left), and the Coherence Score (right) for the LDA algorithm applied on **hashtags** in the captions of anomalous posts with an **underperforming** score (for each Anomaly Detection method)

difference that could be mentioned concerns the anomalies detected by the Isolation Forest algorithm and the Z-Score method which have a higher Standard Deviation of the coherence score. Although there are slight differences, according to this score, LDA seems to work better on anomalous posts detected by the Isolation Forest algorithm.

Example

An example of output of the results is shown in Figure 6.22, below.
The picture was created using `pyLDAvis`, a Python library for interactive topic model visualization that was created to assist users in understanding the themes in a topic model that has been fitted to a corpus of text data. The software uses data from a fitted LDA topic model to create a web-based interactive representation. The sample corpus is derived from anomalous posts with an overperforming score found using the Z-Score algorithm for week 35 of the year 2018.
One of the major themes is represented by the hashtag "theferragnez", which indicates the marriage of the aforementioned influencer Chiara Ferragni, which in September 2018 was followed on the OSNs by more than 20 million followers. Unfortunately, looking at the other terms inserted in this topic by the algorithm, they do not seem to be very related to the event of this wedding or to the theme of the wedding in general.The other big topic of the week instead revealed style and fashion in general, and therefore does not seem linked to an external event that

happened in the real world.

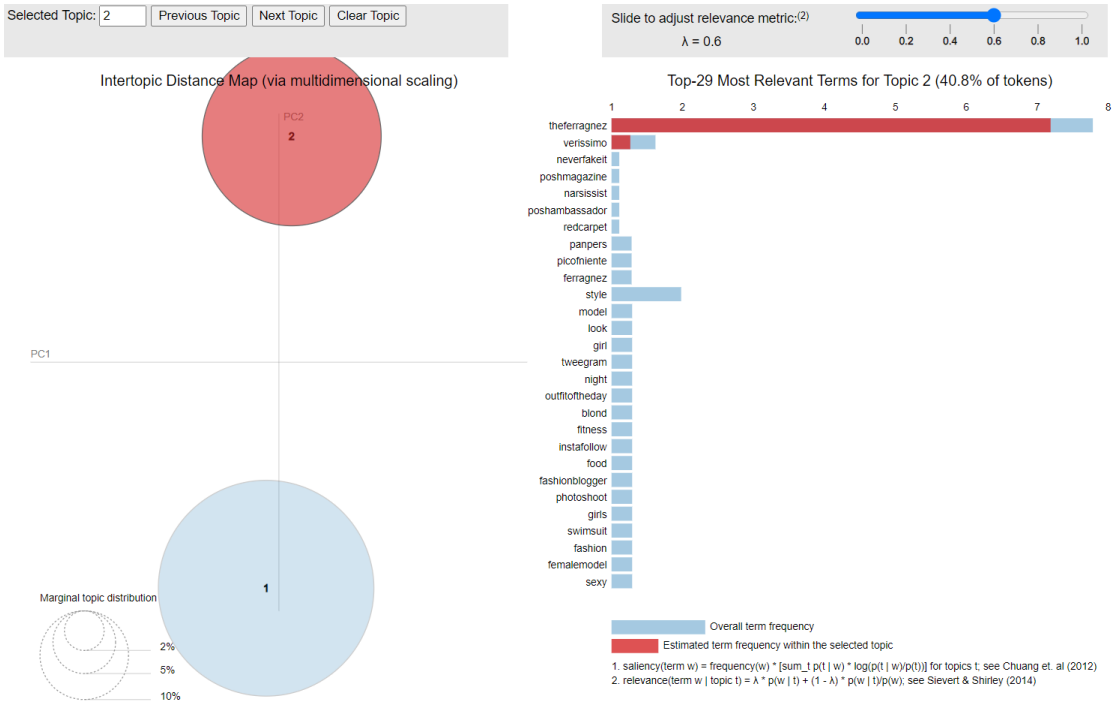


Figure 6.22: Example of output of topics for the LDA algorithm applied on hashtags, for week 35 of year 2018 (anomalous posts with an overperforming score detected through the Z-Score method)

6.4.4 Graphs and Community Detection

To implement this approach, we built **4 different graphs** for each week, using the `NetworkX` Python library: one for hashtags and one for all types of words, respectively for weekly anomalous posts with a score overperforming and a score underperforming. Obviously these 4 graphs have been constructed for each Anomaly Detection method used, as usual.
In particular, each **node** represents a **post**, connected to other posts by a **weighted edge** if it had at least one **hashtag/word in common** with those posts. For the computation of the weight of the arcs, it was decided to use the same distance function already used for the DBSCAN algorithm for each pair of nodes (posts) connected by an edge.

Subsequently, for each graph thus constructed, we applied the **Louvain Community Detection algorithm** to identify the communities, and therefore the post clusters. Initially, we tried to use 3 Community Detection algorithms, Louvain, Label Propagation and Girvan Newman, then automatically choosing the method that maximized modularity for each week. Empirically we noticed that in 90% of cases the designated algorithm was Louvain's, for this reason, for simplicity, we decided to use only this algorithm.

Results

In the following Tables, 6.14 and 6.15, "Number of communities" can be compared to the usual "Number of clusters", while the "Modularity" is an index of the goodness of the method. It is important to specify that the "Number of communities" refers to the communities, found in each weekly graph, containing **at least 2 posts**. The communities containing a single post, i.e. without edges connecting it to other posts, are therefore not considered in the counts.
Looking at the results, it can be seen that the **number of communities** increases as the number of weekly anomalies increases, and therefore depends on the Anomaly Detection method used. The number of communities found for the anomalies detected with the Boxplot Rule method is in fact the highest, followed by that for the anomalies detected by the ARIMA model. A separate case, on the other hand, is represented by the anomalies found with the Isolation Forest algorithm and the Z-Score method, for which the number of weekly communities detected by the Louvain algorithm is between 0 and 1 on average.

Anomalous posts with an OVERPERFORMING score					
Hashtags - All words					
		ARIMA M.	Boxplot R.	Isol. For.	Z-Score
Number of communities	Mean	4 - 4.7	6.6 - 7.8	0.61 - 0.98	0.29 - 0.43
	Median	4 - 5	7 - 7	0 - 1	0 - 0
	Std. Dev.	3.2 - 3.8	4.7 - 6	0.92 - 1.24	0.59 - 0.75
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	12 - 17	19 - 24	5 - 7	3 - 4
Modularity	Mean	0.18 - 0.27	0.34 - 0.38	−0.56 - −0.37	−0.76 - −0.65
	Median	0.48 - 0.52	0.57 - 0.62	−1 - 0	−1 - −1
	Std. Dev.	0.66 - 0.61	0.57 - 0.56	0.58 - 0.63	0.46 - 0.52
	Minimum	−1 - −1	−1 - −1	−1 - −1	−1 - −1
	Maximum	0.87 - 0.88	0.9 - 0.9	0.66 - 0.73	0.5 - 0.65
Silhouette Score	Mean	−0.12 - −0.11	0.02 - −0.03	−0.56 - −0.47	−0.75 - −0.67
	Median	0.07 - 0.04	0.13 - 0.07	−1 - −1	−1 - −1
	Std. Dev.	0.48 - 0.4	0.41 - 0.37	0.57 - 0.54	0.51 - 0.53
	Minimum	−1 - −1	−1 - −1	−1 - −1	−1 - −1
	Maximum	0.63 - 0.5	0.62 - 0.5	0.56 - 0.55	0.8 - 0.8

Table 6.14: Statistical description of Louvain algorithm applied on weekly graphs of anomalous posts with an overperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

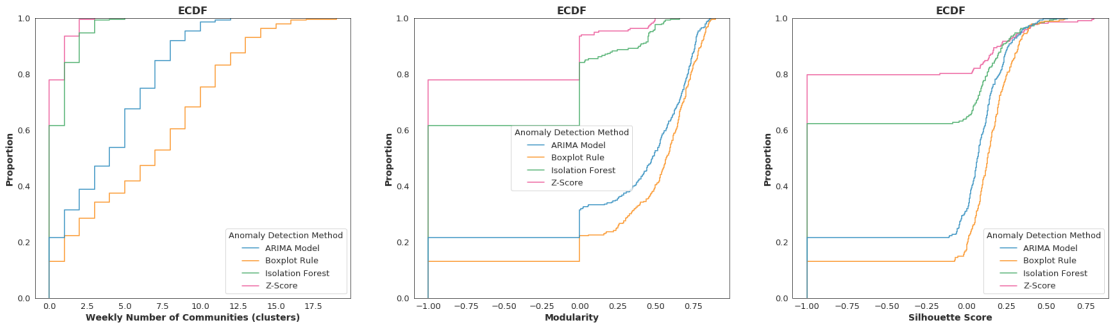


Figure 6.23: ECDFs of the weekly number of communities/clusters (left), the modularity (center) and of the Silhouette Score (right) for the Louvain algorithm applied on graphs of **hashtags** in the captions of anomalous posts with an **overperforming** score (for each Anomaly Detection method)

As far as **modularity** is concerned, remember that its value can move in the interval $[-0.5, 1]$: it is positive if the number of edges present is greater than the number expected and negative otherwise. It can be seen, both from the Tables and from the ECDFs in the Figures, that the modularity increases as the number of communities increases and, for this reason, the execution of Community Detection

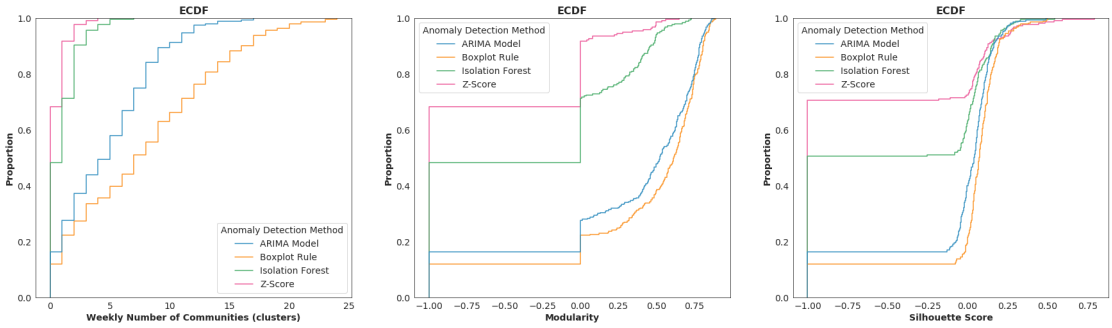


Figure 6.24: ECDFs of the weekly number of communities/clusters (left), the modularity (center) and of the Silhouette Score (right) for the Louvain algorithm applied on graphs of **all types of words** (hashtags, simple words and words within the image) in the captions of anomalous posts with an **overperforming** score (for each Anomaly Detection method)

Anomalous posts with an UNDERPERFORMING score					
Hashtags - All words					
		ARIMA M.	**Boxplot R.**	**Isol. For.**	**Z-Score**
Number of communities	Mean	5.3 - 6	8.7 - 9.7	1.5 - 2	1.2 - 1.5
	Median	5 - 5	8 - 9	1 - 2	1 - 1
	Std. Dev.	3.8 - 3.9	5.8 - 5.9	1.7 - 1.9	1.4 - 1.6
	Minimum	0 - 0	0 - 0	0 - 0	0 - 0
	Maximum	25 - 25	27 - 29	9 - 9	8 - 9
Modularity	Mean	0.52 - 0.57	0.47 - 0.5	−0.18 - −0.02	−0.29 - −0.13
	Median	0.62 - 0.65	0.58 - 0.61	0 - 0.03	0 - 0
	Std. Dev.	0.35 - 0.32	0.4 - 0.38	0.66 - 0.63	0.64 - 0.63
	Minimum	−1 - −1	−1 - −1	−1 - −1	−1 - −1
	Maximum	0.89 - 0.9	0.89 - 0.88	0.86 - 0.85	0.82 - 0.82
Silhouette Score	Mean	0.06 - 0.02	0.05 - 0.02	−0.29 - −0.22	−0.33 - −0.25
	Median	0.08 - 0.03	0.07 - 0.03	−0.01 - 0	−0.04 - 0.01
	Std. Dev.	0.23 - 0.19	0.28 - 0.25	0.56 - 0.5	0.59 - 0.53
	Minimum	−1 - −1	−1 - −1	−1 - −1	−1 - −1
	Maximum	0.47 - 0.41	0.69 - 0.69	0.7 - 0.58	0.96 - 0.64

Table 6.15: Statistical description of Louvain algorithm applied on weekly graphs of anomalous posts with an underperforming score, analyzing only hashtags and all types of word (hashtags, simple words in the caption and terms within the image)

applied to the graphs constructed with the anomalies detected with the Boxplot Rule method it is the best from this point of view. An exception are the anomal posts with an underperforming score, since even in this case the maximum average number of communities was detected for anomalies found with the Boxplot Rule

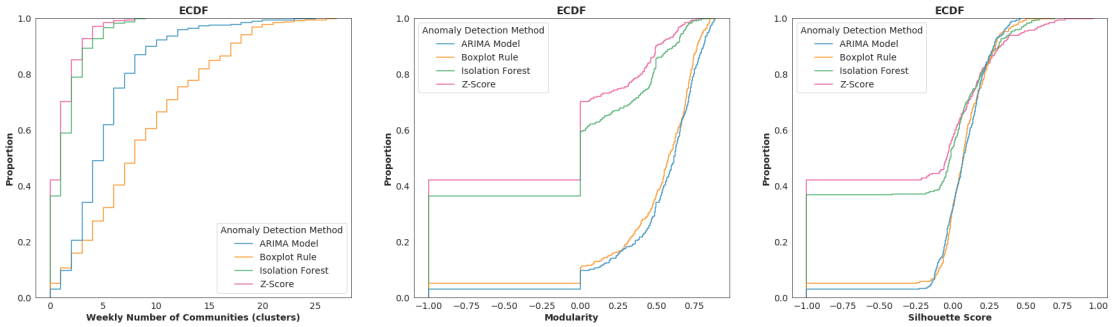


Figure 6.25: ECDFs of the weekly number of communities/clusters (left), the modularity (center) and of the Silhouette Score (right) for the Louvain algorithm applied on graphs of **hashtags** in the captions of anomalous posts with an **underperforming** score (for each Anomaly Detection method)

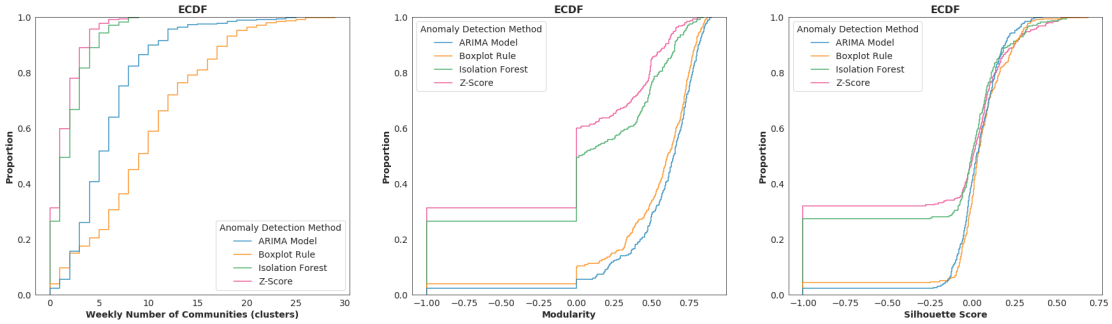


Figure 6.26: ECDFs of the weekly number of communities/clusters (left), the modularity (center) and of the Silhouette Score (right) for the Louvain algorithm applied on graphs of **all types of words** (hashtags, simple words and words within the image) in the captions of anomalous posts with an **underperforming** score (for each Anomaly Detection method)

method, the modularity is greater for the posts detected with the ARIMA model. Furthermore, if we use **all types of words** (hashtags, simple words in the caption and words within the image), the number of communities detected is on average greater than the use of **hashtags** alone, and consequently, also the modularity is greater. The graph method therefore seems to be the only one to provide better results when all types of words are considered instead of just hashtags.
Since this method is not a classic clustering algorithm, a compromise had to be found to calculate the **Silhouette Score**. In particular, given that the weight of the arcs of the graphs is calculated using the same distance function already used to create the weekly distance matrix for the DBSCAN algorithm, also in this case we have calculated this matrix, but using the numbered communities detected by the

Louvain algorithm, for the `scikit-learn` function which calculates the Silhouette Score automatically (which accepts as input the sample and the labels found by the chosen clustering algorithm). As you can see, on average the Silhouette Score thus obtained is on average always negative, with the only exception for the anomalous posts with a score overperforming detected by the Boxplot Rule method, and those with an underperforming score detected by the Boxplot Rule itself and from the ARIMA model. However, these are very low values, close to 0. According to these statistics, it would seem that the graph method is the worst of those mentioned, but, in reality, by manual inspection it is the one that gives the best results. Note that the results described above are valid both for posts that have been assigned an **overperforming** score and for those with an **underperforming** score, except for the special cases mentioned.

Example

An example of output of the results is shown in the Figures below.

Figure 6.27: Example of wordcloud built using all types of words (hashtags, simple words and words within the image), for week 46 of year 2018 (anomalous posts with an overperforming score detected through the Boxplot Rule method)

The Figure 6.27 represents the wordcloud constructed using all types of words (hashtag, simple words and words within the image), for week 46 of the year 2018

(anomalous posts with an overperforming score detected through the Boxplot Rule method), as an indication of the initial bag of words contained in all the chosen anomalous posts of the week. Since the graph method is the only one that gives more promising results when all words are used instead of just hashtags, we have decided to represent this example.
Starting from the anomalous posts, the graph in Figure 6.28 is built, on which the Louvain Community Detection algorithm has already been applied. Different colors represent different communities, while white markers indicate isolated posts, which had no words in common with the other anomalous posts of the week. It can be seen that, in this case, the communities are well defined and separate.

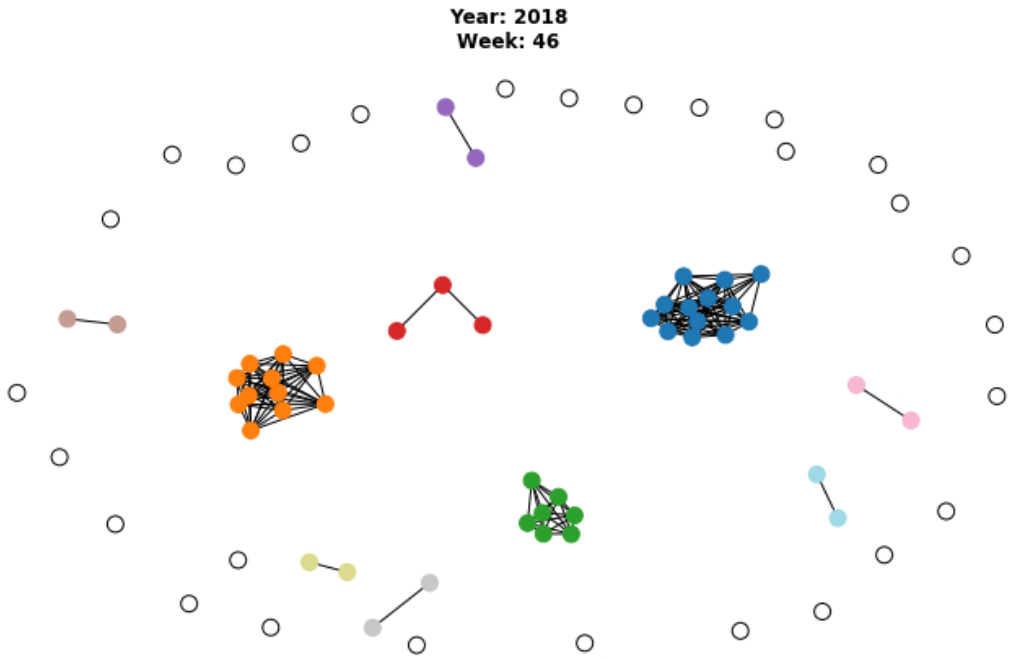


Figure 6.28: Example of the Louvain algorithm applied on a graph of all types of words (hashtags, simple words and words within the image), for week 46 of year 2018 (nodes represent anomalous posts with an overperforming score detected through the Boxplot Rule method, edges represent at least a word in common between connected posts)

Subsequently, the Figure 6.29 shows a Bubble Chart, represented with the same colors as the previous communities, containing, for each community, the words that led to the creation of the edges of the graph. A total of 10 communities are detected, some smaller, which, as can be seen from the Bubble Chart, deal with minor topics. The two largest ones instead (in orange and blue) deal, respectively,

with the theme of fashion for the influencer Elisabetta Franchi (who is a stylist, so it is not excluded that she had launched a new collection of her line at that time) and , perhaps more important, the football theme, present in the anomalous posts of almost every week. In this case, in particular, the most frequent words refer to the Nations League match of Italy-Portugal, played on November 17, 2018, actually an event outside the OSN and which took place in the real world.

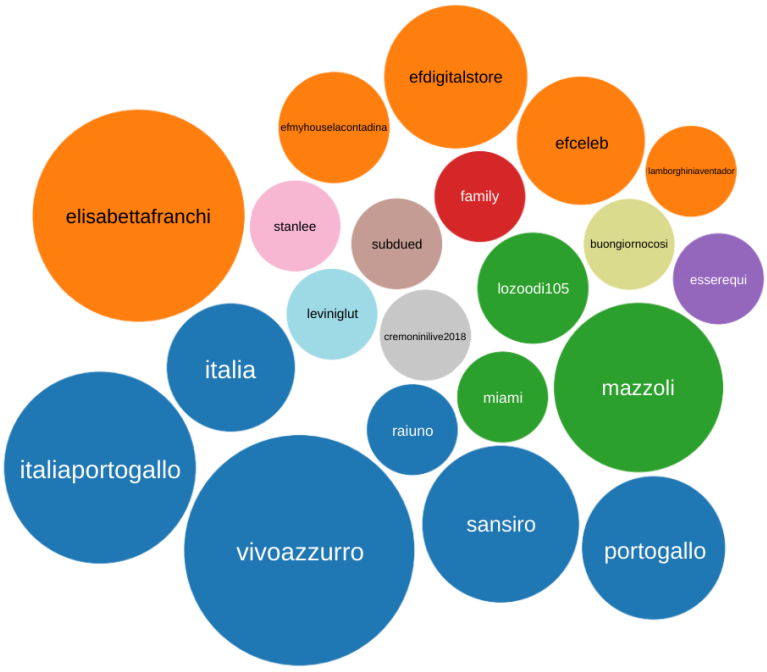


Figure 6.29: Example of bubble chart built using all types of words (hashtags, simple words and words within the image), for week 46 of year 2018 (anomalous posts with an overperforming score detected through the Boxplot Rule method). Different colors correspond to different communities detected by the Louvain algorithm

6.4.5 Comparison of Clustering Methods and Results

Summarizing all the results obtained and presented in this chapter so far, we have found that the best pairs of Anomaly Detection method - Clustering algorithm are respectively:

- **ARIMA model - DBSCAN on hashtags**,
- **ARIMA model - K-Means on hashtags**,
- **Isolation Forest - LDA on hashtags/all types of words** (not covered further),
- **Boxplot Rule method - Graph method and Louvain** Community Detection algorithm **on hashtags/all types of words** (to evaluate the goodness of this method it makes more sense to look at the modularity compared to the Silhouette Score),

regardless of the type of performance score of the anomalous posts analyzed.
In any case, for our purposes, we have focused more on posts that have been scored **overperforming** for manual inspection of results, as it seems more sensible to focus more on anomalous posts that have been scored overperforming, as an external event is more likely to "positively" affect the popularity of an influencer or a certain post.
By excluding the LDA algorithm from comparisons, and taking into account only the three remaining method pairs, some comparisons can be made. Already from this first analysis it emerges that, among the Anomaly Detection methods, the ARIMA model and the Boxplot Rule method provide better results for our research, and therefore the Isolation Forest algorithm and the Z-Score method must be excluded.
Based exclusively on numerical data, taking into consideration the **Silhouette Score** (and modularity in the case of graphs), the best results would seem to derive from the **ARIMA model - DBSCAN on hashtags** pair, while the other two pairs are equivalent, with the exception that the graph method is the only one to provide acceptable results when considering all types of words instead of just hashtags. To make further changes, a more thorough **manual inspection** of the contents of the resulting clusters is required, avoiding giving too much importance to particular cases.

Comparing the DBSCAN and K-Means algorithms, from a first observation it can be seen that DBSCAN tends to put in the same cluster (typically the number 0 or 1) the posts that do not contain hashtags, and in a separate cluster (numbered with -1) those classified as "**noise**", that is, they do not have hashtags in common with each other. K-Means, on the other hand, tends to put in the same cluster

(typically the number 0 or 1) both posts that do not contain hashtags and those that are not related (in some cases even if they are semantically related but do not contain common words), making it more difficult noise distinction at a glance. With the graph method, however, posts that do not contain hashtags / words are automatically ignored and not included in the graph, while those considered as "noise" are visually distinguishable as they are isolated from the others and not connected to any other node (post) with an edge.
Another feature of K-Means is that it tends, unlike the other two clustering methods presented, to form clusters even from a single element (post), which doesn't make much sense for the purposes of our work. Excluding these useless posts, the results are very similar. The K-Means algorithm sometimes manages to insert one or two more posts in the clusters (in an appropriate way) than the DBSCAN, and, at times, even finds small more clusters (not more than one in general), but the whose content actually in some cases had to be inserted in one of the already existing and found clusters.
Generally both DBSCAN and K-Means are able to group posts that contain the same hashtags or with one or two words of difference. With the graph method, on the other hand, it is possible to group posts that are apparently unrelated but which actually have something in common with a post that bridges and connects them: this can be positive in some cases, because the posts that are judged "uncorrelated" by the other two algorithms but which instead share the same semantic meaning, they are grouped in the same cluster with this method, and negative in other cases, when the clusters are actually too large and confused and some posts bridge some which actually dealt with different topics.
For all these reasons, according to our manual inspection of the results, the **best method** is actually that of **graphs**, built only on **hashtags** (and using the anomalous posts detected with the Boxplot Rule method) to avoid as much as possible the last error of we discussed. Using all types of words, the risk is to lose some edge between posts that would have been appropriate, but the loss of data is less frequent and, above all, less relevant than the loss of correctness and validity of the clusters.

Comparison Example

To clarify the considerations made so far, a comparative example is proposed below, showing the differences in the clusters found by method and method, considering the **same week**. For simplicity, the wordclouds corresponding to the clusters found with the **graph method** (the graph was built using **all types of words**), in the ninth week of 2019, are represented in Figure 6.30, as it is the method which has found more clusters and also includes those found with the other algorithms.

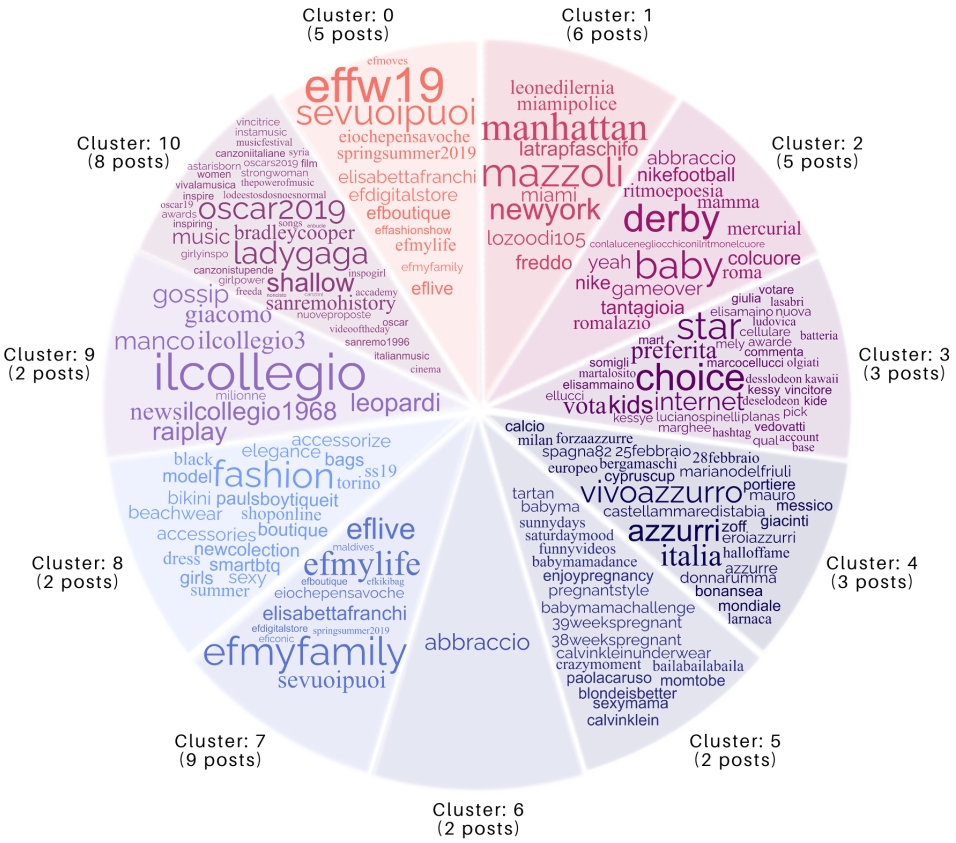


Figure 6.30: Wordclouds for each cluster (detected by the **graphs method**) built using **all types of words** (hashtags, simple words and words within the image), for week 9 of year 2019 (anomalous posts with an overperforming score detected through the Boxplot Rule method). Different colors correspond to different communities detected by the Louvain algorithm

The DBSCAN (Table 6.16) and K-Means (Table 6.17) algorithms are able to find, for this week, the same two clusters: one containing 5 posts with the single word "bikini" in common, and the other containing 6 posts concerning an Italian television program broadcast in that period, called "Uomini E Donne - La Scelta". It is notable that these two clusters, of which only the second is relevant for our research, are not found with the graph method, whose clusters are shown in the image, but the reason is that the posts that form those clusters they have not been classified as anomalous by the Boxplot Rule method. K-Means also manages, unlike DBSCAN, to find the most important cluster of all, that is the one concerning the 2019 Oscars awarding event, also present in the clusters found with the graph method. K-Means, however, also inserts 6 clusters each containing only one post,

useless for our purposes.

DBSCAN outputs					
Year	**Week**	**PostID**	**Name**	**List_hashtags**	**Cluster**
2019	9	2085523|1988013328401628018	Margherita Molinari	[bikini]	0
2019	9	2085523|1987166392199065983	Margherita Molinari	[bikini]	0
2019	9	2085523|1991650968350994141	Margherita Molinari	[bikini]	0
2019	9	2085523|1989455893831348594	Margherita Molinari	[bikini]	0
2019	9	2085523|1987483138420264211	Margherita Molinari	[bikini]	0
2019	9	536943|1989935092214618264	Uomini e Donne	[lascelta, uominiedonne]	2
2019	9	536943|1989427514078583503	Uomini e Donne	[lascelta, uominiedonne]	2
2019	9	536943|1990691285631256653	Uomini e Donne	[uominiedonne, lascelta]	2
2019	9	536943|1990315192256775523	Uomini e Donne	[uominiedonne, lascelta]	2
2019	9	536943|1990654586956223469	Uomini e Donne	[uominiedonne, lascelta]	2
2019	9	536943|1990273768018566388	Uomini e Donne	[uominiedonne, lascelta]	2

Table 6.16: Outputs for the **DBSCAN** applied on **hashtags**, for week 9 of year 2019 (anomalous posts with an overperforming score detected by the ARIMA Model). For reasons of brevity and visualization, only the actual clusters are represented, and not the 47 posts classified as noise (cluster: -1) or belonging to cluster 1, containing 62 posts without hashtags in their caption

Focusing instead only on the graph method, it can be observed that clusters 0 and 7 in the figure concern, once again, the stylist Elisabetta Franchi, and should, in truth, be gathered in a single cluster. The graph method, applied on this same week but only on hashtags instead of all types of words, does not make this mistake and actually creates a single cluster with the sum of these posts.
Both by analyzing all types of words and only hashtags, an error is detected in cluster 2: due to the word "Rome" which acts as a bridge between the posts in the graph, two actually unrelated themes are brought together in the same cluster: football (as you can see from the word "derby") and motherhood. In any case, using only hashtags, the loss of data is minimal: only 2 fewer clusters are found (one of which is incorporated into another, as mentioned before, and the other is cluster 6, useless for our work). Both in this case, but also more generally, it is noted that the best result provided is actually the one obtained with the graph method applied only to hashtags, since the K-Means and DBSCAN algorithms are too dispersive, in the first case, or too much reductive, both in the first and in the second case.

K-MEANS outputs					
Year	**Week**	**PostID**	**Name**	**List_hashtags**	**Cluster**
2019	9	536943|1989935092214618264	Uomini e Donne	[lascelta, uominiedonne]	1
2019	9	536943|1989427514078583503	Uomini e Donne	[lascelta, uominiedonne]	1
2019	9	536943|1990273768018566388	Uomini e Donne	[uominiedonne, lascelta]	1
2019	9	536943|1990315192256775523	Uomini e Donne	[uominiedonne, lascelta]	1
2019	9	536943|1990654586956223469	Uomini e Donne	[uominiedonne, lascelta]	1
2019	9	536943|1990691285631256653	Uomini e Donne	[uominiedonne, lascelta]	1
2019	9	2085523|1991650968350994141	Margherita Molinari	[bikini]	2
2019	9	2085523|1989455893831348594	Margherita Molinari	[bikini]	2
2019	9	2085523|1987166392199065983	Margherita Molinari	[bikini]	2
2019	9	2085523|1987483138420264211	Margherita Molinari	[bikini]	2
2019	9	2085523|1988013328401628018	Margherita Molinari	[bikini]	2
2019	9	2064999|1987201291826516016	Gli Autogol	[gliautogol, abisso, inter, rigore, polemiche, scoop, fiorentina, seriea]	3
2019	9	2541047|1990578788300496209	stefania saettone	[view, madonnadicampiglio, suite, luxury, sensualshot, blonde, natural, mountain, italy, nopost, body]	4
2019	9	2064999|1990905750301766394	Gli Autogol	[terzo, championsleague, gliautogol, seriea, tifosi, gerryscotti, milan, zona]	5
2019	9	3046091|1991362230046091271	Laura freddi Official	[mamma, occhi, roma, sguardi, crescere, figlia, figlia, bimba, compleanno, 14mesi]	6
2019	9	51941|1991714564871133435	Mario Balotelli	[allezlom, thefirstinstagramhistoryafteragol, bravolesgars, supporterambiancemagnifiquemerci]	7
2019	9	531133|1987181715384162706	Verissimo	[ladygaga, oscars, bradleycooper, verissimo]	8
2019	9	1585817|1987062136647492054	Syria	[ladygaga, oscars2019]	8
2019	9	2064999|1991698934862872329	Gli Autogol	[juventus, napoli, gliautogol, finito, allegri, seriea, campionato]	9

Table 6.17: Outputs for the **K-Means** applied on **hashtags**, for week 9 of year 2019 (anomalous posts with an overperforming score detected by the ARIMA Model). For reasons of brevity and visualization, only the actual clusters are represented, and not the 101 posts classified as noise (cluster: 0, including also posts without hashtags in their caption without distinction)

Chapter 7

Conclusions

In this thesis work, we presented the problems of Anomaly Detection and Clustering on Online Social Networks (OSN). The new approach proposed involves the identification of anomalous posts, i.e. those that have received more reactions than expected, and the weekly grouping of the same, on the basis of textual-content similarity, to identify weekly offline trends or peculiar events that probably led to such atypical engagement, such as the participation of an influencer in a TV show or contest.

First we defined the practices and characteristics of the online presence, of the OSNs (in particular Instagram) and, more generally, of online life with a brief historical description of the past and present market players.

Next, we divided the existing literature (on such problems) into categories and identified the shortcomings and values of other proposed solutions, highlighting the differences with the proposed work. We also presented the relevant theory necessary to carry out our work.

To perform this search, Instagram CrowdTangle Database was used to produce a dataset of all posts between 2015 and 2021, including approximately 1 611 Italian Instagram influencer accounts and 2 036 966 posts, with the most descriptive attributes that could be extracted.

The dataset has been pre-processed, in order to extract the main characteristics, remove unnecessary attributes or erroneous data, calculate a performance score (in terms of reactions received versus expected reactions) for each post (the main feature of the survey Anomaly phase) and perform Natural Language Processing (NLP) analyzes on the text, contained in the post-caption or within the post-media, to verify the textual similarity (main feature for the Clustering phase) between the posts. The anomalies found by the various methods were joined by further machine intelligible textual characteristics, such as the TF-IDF and Sublinear TF-IDF, calculated for each week of the dataset.

The final phase involves grouping the weekly anomalies by textual similarity.

Four algorithms have been implemented: the DBSCAN, applied on a pre-calculated paired distance matrix involving the captions of the anomalous posts, the K-Means, applied on a weighted hash vector of the caption words, an LDA model (shown only as an example and not in-depth) and Community Detection algorithms, applied on a graph of anomalous posts.

The proposed system works well, as it was actually possible to find clusters that make sense as belonging to the same theme, and they manage well to group posts related to particular events (important football matches, music festivals, TV shows, famous weddings, Oscar awards, etc.). The results obtained suggest that some posts actually received more/less reactions than expected as their content was related to events outside the real world, while others, classified as "noise" are not related to those events, so they probably have obtained an anomalous involvement due to causes endogenous to the OSN.

Based exclusively on numerical data, taking into consideration the Silhouette Score (and modularity in the case of graphs), the best results would seem to derive from the pair ARIMA model - DBSCAN on hashtags, while the other two pairs are equivalent, with the exception that the graph method is the only one that gives acceptable results when considering all types of words (hashtags, simple words and words within the image) rather than just hashtags. According to these statistics, it would appear that the graph method is the worst of those mentioned, but, in reality, by manual inspection it is the one that gives the best results.

7.1 Future Work

Although done with the utmost effort, this thesis work could be extended and improved in several directions.

- **Additional clustering features** - A study on the development of additional metrics could improve the proposed work, allowing clustering algorithms to perform a more robust subdivision of anomalous posts into groups. For example, one could better analyze the content of posts, not limiting oneself exclusively to textual characteristics, but applying a Convolutional Neural Network on images and videos as a comparison/similarity metric.
- **Manually adding labels** - To better identify whether the posts that are detected anomalous have actually received more/less reactions than expected due to external events in the real world, you could manually add a label to each post, indicating the type of event or whether it does not depend on any "offline" events. To do this, you could also compare the results obtained by extracting the trending topics corresponding to the publication period of each post from Google Trend. The problem would thus go from unsupervised to supervised.

- **Different evaluation metrics** - For this work, only the Silhouette Score and a manual inspection of the data are used as evaluation metrics for the goodness of the clusters (by sample, considering the huge amount of data to be checked, or even total in some cases). Different metrics could more specifically assess the performance of the algorithms.
- **In-depth analysis of the LDA algorithm** - In our work, the LDA model is cited only as an example, without carrying out a more in-depth analysis of the results, also because it is difficult to compare with other clustering methods due to the nature of the algorithm and its output. Since the first results seem promising, further checks could be made and investigated if the results obtained are better than the other proposed techniques.
- **Deepening on underperforming posts** - Given the nature of our research, it seems more sensible to focus more on anomalous posts that have been scored overperforming, as an external event is more likely to "positively" affect the popularity of an influencer or a certain post. Intuitively, it is probable instead that, again with reference to events exogenous to the OSN, a fake news negatively affects a post, or that the causes for which a post receives fewer reactions than expected are rather endogenous to the OSN. Another possible reason could be a too optimistic evaluation by CrowdTangle's predictive algorithm for the expected reactions of a post. For this reason, the manual inspection and also the examples shown were not thorough for posts with an underperforming score, but this could be done in the future.
- **Applying the best methods to Facebook** - The best Anomaly Detection - Clustering method pair could also be applied to the Facebook dataset (similarly downloaded via CrowdTangle) to compare the results from the Instagram dataset. it is necessary, however, to change the name of some parameters (it depends on the dataset) and decide whether to deepen the variety of reactions existing on Facebook, unlike Instagram in which reactions are limited to a simple like or comment.

Bibliography

[1] Michael Ray. *Social Network*. In: *Encyclopaedia Britannica*. 2021. URL: https://www.britannica.com/technology/social-network (cit. on p. 2).

[2] *Leading Online Social Networks logos*. URL: https://www.robertabruzzone.com/intervista-a-roberta-bruzzone-ragazzi-attenti-perche-i-social-non-perdonano (cit. on p. 2).

[3] *Instagram round logo*. URL: https://pngset.com/transparent-png#gyltb (cit. on p. 2).

[4] *Whatsapp logo*. URL: https://it.wikipedia.org/wiki/File:Whatsapp_logo_svg.png (cit. on p. 2).

[5] Terry Flew. *Media Convergence - Social Media*. In: *Encyclopaedia Britannica*. 2021. URL: https://www.britannica.com/topic/media-convergence (cit. on p. 2).

[6] *Number of people using social media platforms, 2004 to 2019*. URL: https://ourworldindata.org/grapher/users-by-social-media-platform?time=2004..latest&country=Facebook~Instagram~MySpace~Snapchat~TikTok~Twitter~Whatsapp~YouTube (cit. on p. 4).

[7] *Number of users of leading social networks in Italy in March 2021(in millions)*. URL: https://www.statista.com/statistics/787390/main-social-networks-users-italy (cit. on p. 5).

[8] Jan Kietzmann, Kristopher Hermkens, Ian McCarthy, and Bruno Silvestre. «Social Media? Get Serious! Understanding the Functional Building Blocks of Social Media». In: *Business Horizons* 54 (May 2011), pp. 241–251. DOI: 10.1016/j.bushor.2011.01.005 (cit. on p. 6).

[9] Vitaly Gorbachev. *Avatar icons*. URL: https://www.flaticon.com/authors/vitaly-gorbachev (cit. on p. 8).

[10] Kyrylo Petrenko. *Instagram Stories mockup*. URL: https://www.dreamstime.com (cit. on p. 8).

[11] *Photocamera flat icon*. URL: https://icon-library.com/icon/camera-flat-icon-2.html (cit. on p. 8).

[12] Koosha Zarei, Damilola Ibosiola, Reza Farahbakhsh, Zafar Gilani, Kiran Garimella, Noël Crespi, and Gareth Tyson. «Characterising and detecting sponsored influencer posts on Instagram». In: Dec. 2020, pp. 327–331. DOI: 10.1109/ASONAM49781.2020.9381309. URL: https://hal.archives-ouvertes.fr/hal-03044105/document (cit. on pp. 10–12, 36).

[13] Rana Tallal Javed, Mirza Elaaf Shuja, Muhammad Usama, Junaid Qadir, Waleed Iqbal, Gareth Tyson, Ignacio Castro, and Kiran Garimella. «A First Look at COVID-19 Messages on WhatsApp in Pakistan». In: *CoRR* abs/2011.09145 (2020). arXiv: 2011.09145. URL: https://arxiv.org/abs/2011.09145 (cit. on pp. 12, 13, 16).

[14] Derek Weber and Frank Neumann. «Who's in the Gang? Revealing Coordinating Communities in Social Media». In: *CoRR* abs/2010.08180 (2020). arXiv: 2010.08180. URL: https://arxiv.org/abs/2010.08180 (cit. on pp. 13, 14).

[15] Derek Weber, Mehwish Nasim, Lewis Mitchell, and Lucia Falzon. «A method to evaluate the reliability of social media data for social network analysis». In: *CoRR* abs/2010.08717 (2020). arXiv: 2010.08717. URL: https://arxiv.org/abs/2010.08717 (cit. on p. 14).

[16] Matteo Cardaioli, Pallavi Kaliyar, Pasquale Capuozzo, Mauro Conti, Giuseppe Sartori, and Merylin Monaro. «Predicting Twitter Users' Political Orientation: An Application to the Italian Political Scenario». In: *2020 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. 2020, pp. 159–165. DOI: 10.1109/ASONAM49781.2020.9381470 (cit. on p. 15).

[17] Md. Shafiur Rahman, Md. Ashraf Uddin, Sajal Halder, and Uzzal Acharjee. «An efficient hybrid system for anomaly detection in social networks». In: *Cybersecurity* 4 (Mar. 2021). DOI: 10.1186/s42400-021-00074-w. URL: https://doi.org/10.1186/s42400-021-00074-w (cit. on pp. 16, 17).

[18] Volodymyr Miz, Benjamin Ricaud, Kirell Benzi, and Pierre Vandergheynst. «Anomaly Detection in the Dynamics of Web and Social Networks Using Associative Memory». In: *The World Wide Web Conference*. WWW '19. San Francisco, CA, USA: Association for Computing Machinery, 2019, pp. 1290–1299. ISBN: 9781450366748. DOI: 10.1145/3308558.3313541. URL: https://doi.org/10.1145/3308558.3313541 (cit. on p. 17).

[19] Mustafa H. Hajeer, Alka Singh, Dipankar Dasgupta, and Sugata Sanyal. «Clustering online social network communities using genetic algorithms». In: *CoRR* abs/1312.2237 (2013). arXiv: 1312.2237. URL: http://arxiv.org/abs/1312.2237 (cit. on p. 18).

[20] Bhaskar Biswas Kuldeep Singh Harish Kumar Shakya. «Clustering of people in social network based on textual similarity». In: *Elsevier GmbH* 8 (Sept. 2016), pp. 570–573. DOI: https://doi.org/10.1016/j.pisc.2016.06.023 (cit. on p. 19).

[21] Elizabeth Williams, Jeff Gray, Edwin Morris, Ben Bradshaw, Keegan Williams, and Brandon Dixon. «A comparison of two methods for the topical clustering of social media posts». In: *2016 IEEE 7th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*. Oct. 2016, pp. 1–7. DOI: 10.1109/UEMCON.2016.7777863 (cit. on pp. 20, 21).

[22] Narumol Prangnawarat, Ioana Hulpus, and Conor Hayes. «Event Analysis in Social Media Using Clustering of Heterogeneous Information Networks». In: *FLAIRS Conference*. 2015 (cit. on pp. 21, 22).

[23] Lei-Lei Shi, Lu Liu, Yan Wu, Liang Jiang, and James Hardy. «Event Detection and User Interest Discovering in Social Media Data Streams». In: *IEEE Access* 5 (2017), pp. 20953–20964. DOI: 10.1109/ACCESS.2017.2675839 (cit. on p. 22).

[24] Patrick Liu Shuhua and Jansson. «Topic Modelling Analysis of Instagram Data for the Greater Helsinki Region». In: *Arcada Working Papers* (2017). URL: http://www.theseus.fi/handle/10024/140608 (cit. on p. 23).

[25] Martino Trevisan, Luca Vassio, Idilio Drago, Marco Mellia, Fabricio Murai, Flavio Figueiredo, Ana Paula Couto da Silva, and Jussara M Almeida. «Towards Understanding Political Interactions on Instagram». In: *Proceedings of the 30th ACM Conference on Hypertext and Social Media*. 2019. URL: http://hdl.handle.net/11583/2752645 (cit. on p. 24).

[26] Carlos Henrique Gomes Ferreira, Fabricio Murai, Ana Paula Couto da Silva, Jussara Marques de Almeida, Martino Trevisan, Luca Vassio, Idilio Drago, and Marco Mellia. «Unveiling Community Dynamics on Instagram Political Network». In: *ACM Conference on Web Science*. 2020 (cit. on pp. 24, 25).

[27] Nicolo Russo Francesca Soro Marco Mellia. «Regular Pattern and Anomaly Detection on Corporate Transaction Time Series». In: *EDBT/ICDT Workshops*. 2020 (cit. on pp. 25, 26, 105).

[28] Martino Trevisan, Luca Vassio, and Danilo Giordano. «Debate on online social networks at the time of COVID-19: An Italian case study». In: *Online Social Networks and Media* 23 (2021), p. 100136. ISSN: 2468-6964. DOI: https://doi.org/10.1016/j.osnem.2021.100136 (cit. on p. 27).

[29] Luca Vassio, Michele Garetto, Carla Chiasserini, and Emilio Leonardi. «Temporal Dynamics of Posts and User Engagement of Influencers on Facebook and Instagram». In: *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ASONAM '21. Virtual Event, Netherlands: Association for Computing Machinery, 2021, pp. 129–133. ISBN: 9781450391283. DOI: 10.1145/3487351.3488340. URL: https://doi.org/10.1145/3487351.3488340 (cit. on p. 28).

[30] Fabio Bertone, Luca Vassio, and Martino Trevisan. «The Stock Exchange of Influencers: A Financial Approach for Studying Fanbase Variation Trends». In: *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ASONAM '21. Virtual Event, Netherlands: Association for Computing Machinery, 2021, pp. 431–435. ISBN: 9781450391283. DOI: 10.1145/3487351.3488413. URL: https://doi.org/10.1145/3487351.3488413 (cit. on p. 28).

[31] Varun Chandola, Arindam Banerjee, and Vipin Kumar. «Anomaly Detection: A Survey». In: *ACM Comput. Surv.* 41 (July 2009). DOI: 10.1145/1541880.1541882 (cit. on pp. 46, 51).

[32] Vic Barnett and Toby Lewis. *Outliers in statistical data*. 1984 (cit. on p. 46).

[33] Francesca Soro, Thomas Favale, Danilo Giordano, Luca Vassio, Zied Ben Houidi, and Idilio Drago. «The New Abnormal: Network Anomalies in the AI Era». In: *Communication Networks and Service Management in the Era of Artificial Intelligence and Machine Learning*. Wiley, John and Sons, Ltd, 2021. Chap. 11, pp. 261–288. ISBN: 9781119675525. DOI: https://doi.org/10.1002/9781119675525.ch11. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119675525.ch11. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119675525.ch11 (cit. on pp. 46, 48, 51).

[34] Ravneet Kaur and Sarbjeet Singh. «A survey of data mining and social network analysis based anomaly detection techniques». English. In: *Egyptian Informatics Journal* 17.2 (2016), pp. 199–216. DOI: 10.1016/j.eij.2015.11.004 (cit. on pp. 48–51).

[35] David Savage, Xiuzhen Zhang, Xinghuo Yu, Pauline Lienhua Chou, and Qingmai Wang. «Anomaly detection in online social networks». In: *CoRR* abs/1608.00301 (2016). arXiv: 1608.00301. URL: http://arxiv.org/abs/1608.00301 (cit. on p. 50).

[36] *Anomaly Detection Service*. URL: https://developer.mindsphere.io/apis/analytics-anomalydetection/api-anomalydetection-overview.html (cit. on p. 52).

[37] Adithya Krishnan. *Anomaly Detection with Time Series Forecasting*. URL: https://www.kaggle.com/adithya44/anomaly-detection-with-time-series-forecasting (cit. on p. 55).

[38] André Bauer. «Automated Hybrid Time Series Forecasting: Design, Benchmarking, and Use Cases». PhD thesis. Jan. 2021. DOI: 10.25972/OPUS-22025 (cit. on p. 56).

[39] Yousra Regaya, Fodil Fadli, and Abbes Amira. «Point-Denoise: Unsupervised outlier detection for 3D point clouds enhancement». In: *Multimedia Tools and Applications* 80 (July 2021), pp. 1–17. DOI: 10.1007/s11042-021-10924-x (cit. on p. 60).

[40] *IsolationForest example*. URL: https://scikit-learn.org/stable/auto_examples/ensemble/plot_isolation_forest.html#sphx-glr-auto-examples-ensemble-plot-isolation-forest-py (cit. on p. 61).

[41] *Detecting and Treating Outliers | Treating the odd one out!* May 2021. URL: https://www.analyticsvidhya.com/blog/2021/05/detecting-and-treating-outliers-treating-the-odd-one-out/ (cit. on p. 62).

[42] Itziar Frades and Rune Matthiesen. «Overview on Techniques in Cluster Analysis». In: *Methods in molecular biology (Clifton, N.J.)* 593 (Jan. 2010), pp. 81–107. DOI: 10.1007/978-1-60327-194-3_5 (cit. on p. 64).

[43] *Hierarchical Clustering / Dendrogram: Simple Definition, Examples*. Nov. 2021. URL: https://www.statisticshowto.com/hierarchical-clustering (cit. on p. 65).

[44] Pradeep Rai and Singh Shubha. «A Survey of Clustering Techniques». In: *International Journal of Computer Applications* 7 (Oct. 2010). DOI: 10.5120/1326-1808 (cit. on p. 65).

[45] Benedek Rozemberczki. *Awesome Community Detection Research Papers*. URL: https://github.com/benedekrozemberczki/awesome-community-detection (cit. on p. 66).

[46] Vladimir Estivill-Castro. «Why so Many Clustering Algorithms: A Position Paper». In: *SIGKDD Explor. Newsl.* 4.1 (June 2002), pp. 65–75. ISSN: 1931-0145. DOI: 10.1145/568574.568575 (cit. on p. 66).

[47] Luca Becchetti, Emilio Cruciani, Francesco Pasquale, and Sara Rizzo. «Step-by-Step Community Detection for Volume-Regular Graphs». In: *CoRR* (2019). arXiv: 1907.07149. URL: https://www.uniroma3.it/articoli/step-by-step-community-detection-in-volume-regular-graphs-160045 (cit. on p. 67).

[48] Mark Needham and Amy E. Hodler. *Graph Algorithms in Neo4j: Louvain Modularity*. 2019. URL: https://neo4j.com/blog/graph-algorithms-neo4j-louvain-modularity (cit. on p. 68).

[49] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. «Finding community structure in very large networks». In: *Physical Review E* 70.6 (Dec. 2004). ISSN: 1550-2376. DOI: 10.1103/physreve.70.066111. URL: http://dx.doi.org/10.1103/PhysRevE.70.066111 (cit. on p. 68).

[50] V. A. Traag, L. Waltman, and N. J. van Eck. «From Louvain to Leiden: guaranteeing well-connected communities». In: *Scientific Reports* 9 (Mar. 2019). DOI: 10.1038/s41598-019-41695-z (cit. on p. 69).

[51] *Girvan-Newman algorithm*. URL: https://networkx.guide/algorithms/community-detection/girvan-newman (cit. on pp. 71, 72).

[52] Chris Ernst. *DBSCAN in Python*. URL: https://github.com/chriswernst/dbscan-python (cit. on p. 73).

[53] *A demo of K-Means clustering on the handwritten digits data*. URL: https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_digits.html#sphx-glr-auto-examples-cluster-plot-kmeans-digits-py (cit. on p. 74).

[54] Diego Buenaño-Fernández, Mario Gonzalez, David Gil, and Sergio Luján-Mora. «Text Mining of Open-Ended Questions in Self-Assessment of University Teachers: An LDA Topic Modeling Approach». In: *IEEE Access* PP (Feb. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.2974983 (cit. on p. 78).

[55] *Cluster Mode Overview*. URL: https://spark.apache.org/docs/latest/cluster-overview.html (cit. on p. 80).

[56] *Introduction to Hadoop and MapReduce*. URL: https://dbdmg.polito.it/wordpress/wp-content/uploads/2020/10/03_Intro_HadoopAndMapReduce_BigData_NewStyle.pdf (cit. on p. 83).

[57] *sklearn.ensemble.IsolationForest*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html (cit. on p. 109).

[58] *sklearn.cluster.DBSCAN*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html (cit. on p. 119).

[59] Nadia Rahmah and Imas Sukaesih Sitanggang. «Determination of Optimal Epsilon (Eps) Value on DBSCAN Algorithm to Clustering Data on Peatland Hotspots in Sumatra». In: *IOP Conference Series: Earth and Environmental Science* 31 (Jan. 2016), p. 012012. DOI: 10.1088/1755-1315/31/1/012012. URL: https://doi.org/10.1088/1755-1315/31/1/012012 (cit. on p. 119).

[60] *sklearn.cluster.KMeans*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html (cit. on p. 129).

Ringraziamenti

Vorrei spendere qualche riga di ringraziamento nei confronti di tutti coloro che mi hanno sostenuto e aiutato durante i miei anni di studio e di crescita, personale e professionale, al Politecnico.

In primis, un grazie speciale va ai miei relatori, i Prof. Trevisan M. e Vassio L., per avermi guidata e supportata, giorno dopo giorno, nella fase più importante del mio percorso accademico. Grazie per per gli indispensabili consigli, l'infinita disponibilità, precisione e gentilezza, è stato un onore e un piacere per me poter scrivere il mio progetto di tesi con voi.

Grazie ai miei genitori, per non avermi mai fatto mancare il vostro dolce e instancabile sostegno, economico e morale, durante questi anni e per tutta la vita. Grazie per avermi sempre incoraggiata, anche a distanza, a raggiungere questo obiettivo, a sfruttare al massimo le mie capacità, a non perdermi mai d'animo in qualunque situazione. Grazie per tutti i consigli, per esservi sempre preoccupati per me, per aver sempre soddisfatto qualunque mia necessità, per aver vissuto con me ogni momento di gioia e ogni difficoltà, grazie per esserci sempre stati. Senza di voi tutto questo non sarebbe stato possibile, e senza i vostri insegnamenti oggi non sarei ciò che sono. Grazie, vi voglio bene.

Grazie a mio fratello Giovanni, per essersi sempre interessato al mio percorso universitario, per avermi sempre incoraggiata e sostenuta, anche nei momenti più duri, per avermi sempre difesa da qualsiasi svalutazione, anche quando io stessa non ho creduto in me. Grazie per avermi trattata da adulta quando gli altri mi consideravano ancora un'adolescente, grazie per esserti sempre preoccupato per me, per avermi capita quando mi sono sentita sola. Grazie per essere stato mio fratello, quando mi sentivo una figlia unica.

Grazie a zia Mina, per essere stata sempre la prima a telefonare ogni volta che dovevo affrontare un esame, per ogni tipo di bontà che hai sempre fatto trovare ad ogni mio ritorno a Taranto, per tutte le vecchie storie che mi hai raccontato sulla

mia famiglia, per avermi sempre sostenuta economicamente nonostante le ristrette possibilità, grazie per esserti sempre interessata ad ogni piccolo aspetto della mia vita. Avrei tanto voluto darti la gioia di poter assistere a questo mio traguardo, spero tanto che da lassù sarai fiera di me.

Grazie a Monica, per essere sempre stata al mio fianco durante gli ultimi 11 anni, per avermi dato la forza di affrontare i momenti più bui, per aver colmato i miei vuoti quando il mondo intorno a me crollava a pezzi, per avermi ascoltata senza mai giudicare, per avermi sempre spinta ad affrontare le mie paure e raggiungere i miei obiettivi. Grazie per tutte le risate insieme, per la tua onestà e lealtà, per aver sempre condiviso tutto con me, anche a distanza, ogni gioia, ogni dolore, ogni ansia, ogni rabbia, ogni piccola emozione. Grazie per essere cresciuta con me. Sono la persona più fortunata del mondo ad averti accanto, sei e sarai sempre la mia migliore amica, la mia "partner in crime", la sorella che non ho mai avuto, la mia eterna complice.

Grazie a Michele, per ogni nottata sui nostri magici fogli di studio, per tutte le ore passate a chiacchierare tra una sigaretta e l'altra, per le serate karaoke, per il Martini rosato a qualsiasi ora, per essere stato il mio compagno di studi e di svago durante la quarantena, per essermi stato accanto ad ogni singola lacrima versata, per tutte le piccole sorprese che hai sempre organizzato per me. Grazie per le sere alle nostre altalene, con una cuffia nell'orecchio, a ridere e cantare, a parlare di tutto e di niente, a guardare la luna riflessa su Palazzo Lancia. Grazie per le fughe di soppiatto dalle lezioni, per le veglie a guardare serie TV, per la gita a Moncalieri, per la notte a suonare *Wonderwall* alla chitarra, grazie per il nostro "silent-party", che non dimenticherò mai: "io e te, contro tutto e tutti come sempre, a ballare in silenzio, nel buio, nel mondo". Sei una delle persone più importanti della mia vita, ora e per sempre, senza di te tutto questo non sarebbe stato possibile.

Grazie ad Albo, il mio compagno di banco, di giochi e di vita, per ogni singola ora di lezione noiosa insieme, per i pomeriggi di studio, per ogni camminata verso il patibolo dell'esame, per le videochiamate a qualsiasi ora per KlapKlap, per ogni ansia universitaria (e non) affrontata insieme durante gli ultimi 5 anni. Grazie per le Dragoon al pub, per le chiacchierate infinite, le risate, gli abbracci, per la nostra complicità, per i sogni condivisi, per ogni nostra piccola avventura insieme. Grazie per essermi stato accanto e avermi sempre regalato un sorriso anche nei momenti di tristezza, per avermi sempre incoraggiata e capita, per essere stato la mia boccata d'aria fresca, il mio arcobaleno nei giorni più grigi. Diventare tua amica è stata la cosa più semplice e naturale che mi sia mai capitata, come un riflesso involontario, non l'ho controllato, non l'ho premeditato. Sarai sempre il mio primo pensiero quando ripenserò a questi anni universitari, i miei ricordi più belli sono tutti con te. Sei e sarai sempre parte di me, e questo niente potrà mai cambiarlo.

Grazie a tutti i miei parenti, i miei amici di Taranto e di Torino, i miei compagni di università e a tutti coloro che in questi anni hanno incrociato la mia vita lasciandomi qualcosa di buono. Grazie per aver fatto parte, ognuno a suo modo, di questo percorso intenso ed entusiasmante, nel bene e nel male.

Sono così tanti i ricordi che mi tornano in mente che è impossibile trovare le parole giuste o sufficienti per onorarli, tutte le pagine di questa tesi non basterebbero.

Grazie di cuore a tutti voi.

Infine, dedico questa tesi a me stessa, ai miei sacrifici, al mio impegno e alla mia tenacia che mi hanno permesso di arrivare fin qui.

"We were together. I forget the rest."
Walt Whitman