
**UNIVERSITÀ DELLA CALABRIA**

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

**Dottorato di Ricerca in**

INFORMATION AND COMMUNICATION TECHNOLOGIES

*Con il contributo di (Ente finanziatore)*

POR Calabria FSE/FESR 2014 – 2020

CICLO XXXV

# Distributed Big Social Data Analysis: Advanced Techniques and Execution Strategies

**Settore Scientifico Disciplinare**: ING-INF/05 Sistemi di elaborazione delle informazioni

**Coordinatore**: Ch.mo Prof. Giancarlo Fortino

Firma ___ Firma oscurata in base alle linee guida del Garante della privacy

**Supervisore/Tutor**: Ch.mo Prof. Paolo Trunfio

Firma ____ Firma oscurata in base alle linee guida del Garante della privacy

Ch.mo Prof. Fabrizio Marozzo

Firma ___ Firma oscurata in base alle linee guida del Garante della privacy

**Dottorando**: Dott. Riccardo Cantini

Firma __ Firma oscurata in base alle linee guida del Garante della privacy

*“La borsa di dottorato è stata cofinanziata con risorse del Programma Operativo Regionale Calabria FSE/FESR 2014 – 2020 (CCI 2014IT16M2OP006)”*

# Contents

Chapter 1

# Introduction

In recent years, social media analysis is arousing great interest in various scientific fields, such as computational politics, sociology, linguistics, and computer science. Indeed, social media have now become part of our daily lives, leading to the generation of huge amounts of opinion-rich multi-modal data, effectively exploitable to investigate human dynamics and behaviors [1]. Such data, commonly referred to as *Big Social Data* [2], are intrinsically suited to a vast set of applications, aimed at understanding how information spreads within a network, analyzing user opinions, emotions, and political alignment, extracting user trajectories and identifying the main topics underlying social media conversation, just to name a few [1, 3–6].

This thesis mainly focuses on the analysis of politically-polarized Big Social Data, produced online during election campaigns and referenda, with the aim of outlining a detailed profile of social users, investigating their interests, opinions, and feelings, and shaping their perception of real-world facts and events. For this purpose, we developed and evaluated innovative methodologies and algorithms, discussed in depth in the next chapters, for the effective extraction of the valuable information hidden in such data, thus providing an effective data-driven approach to a thorough understanding of political phenomena.

Specifically, we designed effective solutions to investigate the political leaning of social users, taking into account several important aspects [1], such as the im-

pact of social bots on political discussion and their influence on legitimate users, as well as the importance of time-related aspects in determining the voting intention of social media users [7]. We also studied the relationships between user polarization and the sentiment expressed in referring to the different candidates, by modeling political support across a broad spectrum of emotions [8]. Moreover, we combined information diffusion and influence maximization with political polarization analysis, to identify the main influencers for the different factions and derive the main information diffusion strategies adopted during the political campaign [3]. Finally, to achieve a rich representation of social media conversation, we worked on topic detection and tracking techniques, aimed at identifying the main topics underlying the online discussion, following their evolution over time, and characterizing them from the viewpoint of political polarization [6, 8]. A further research area that this thesis has focused on is hashtag recommendation, with the aim of supporting Big Social Data analysis techniques, not only in the political sphere. Specifically, advanced natural language processing techniques have been exploited to determine the appropriate hashtags for a given post, leading to a double advantage: on the one hand, users are supported in choosing a hashtag in line with both the semantics of the text and the latest trends; on the other hand, hashtag-based techniques can benefit from a greater amount of representative and high-quality data [9].

Besides the development of innovative methodologies and algorithms for the analysis of Big Social Data, this thesis focuses on the design and implementation of novel techniques aimed at enabling their efficient execution in high-performance distributed environments. Indeed, while Big Data analysis offers great opportunities in numerous fields of application, from politics to economics and social sciences, their volume and speed continually challenge today's storage, processing, and analysis capabilities. Therefore, state-of-the-art tools for analyzing and learning from Big Data on scalable computers, including the main parallel programming paradigms (e.g., MapReduce, workflow, BSP, message passing, and SQL-like), and the most used systems for Big Data analysis (e.g., Apache Spark, Hadoop, and Storm) are expected to be constantly improved to effectively tackle

Big Data issues [10]. Starting from the above considerations, our research activity focused on the study of ad-hoc techniques and strategies aimed at enhancing the execution of data-intensive high-parallel applications. Specifically, we worked on: *i*) workflow task scheduling, to maximize the trade-off between task-parallelism and memory usage, by minimizing the risk of data spilling-to-disk [11]; *ii*) data partitioning, to estimate a suitable size for data blocks so as to speed-up parallel data-intensive applications and increase scalability and throughput [12].

## 1.1 Main issues and challenges

Every day millions of people use social media platforms, generating vast amounts of high-speed and heterogeneous data, which are by their very nature highly dynamic, noisy, and often polluted with harmful content. Therefore, extracting high-quality information from them is not an easy task, and there exist several issues and challenges that had to be faced during the realization of this thesis.

**Platform biases.** A first issue lies in different possible platform biases that can lead to the analysis of an unrepresentative data sample, whose statistical significance should always be investigated as a fundamental, preliminary step. As an example, when analyzing politically-polarized data, users' representativeness should be assessed by understanding to what extent they can be considered voters in the political event under consideration, evaluating how they are distributed in terms of gender, age, culture, and social status. Another source of bias is related to technical aspects such as platform policies about data availability and restrictions imposed in some areas of the world.

**Language barrier.** This issue arises when an analysis methodology is tied by design to a specific language, which makes it unable to handle posts written in different languages and not applicable in multilingual contexts. This may limit its generalizability since social media platforms are widespread all over the world

and the behavior of social users varies across different cultures and geographies. Due to this, language-agnostic approaches are often preferred, as well as the use of multilingual language representation models such as the *Google Multilingual Universal Sentence Encoder* [13, 14], which embeds text from 16 languages into a single semantic space, and the *Multilingual BERT* [15, 16], which covers 104 spoken languages from around the world.

**Misclassification.** Many techniques follow a supervised or semi-supervised approach, which implies a step in which social media posts are assigned a label or a score. As an example, they can be associated with a political party, an emotion, or a sentiment score. However, social media data are generally noisy, due to the use of slang terms and expressions, and often unbalanced, due to the overrepresentation of certain groups or classes. In such a setting, the risk of misclassification can be high [17]. As an example, an imbalance may be present in the number of posts in favor of the different candidates or parties published by social users during an election campaign, which can cause the learning process to be biased toward the most represented classes. Therefore, data balancing techniques should be used, like randomized sampling, and only high-confidence annotations should be accepted to ensure the final results' quality.

**Reliability.** As stated above, the quality of the final results heavily depends on the reliability of the data involved in the analysis process. Unfortunately, social media data are easily manipulated through spamming activities, misinformation campaigns, and the spread of fake and malicious content. In such a scenario, getting reliable and impartial data, discerning them from rumors, constructed reports, and fake news is not an easy task. Recent studies have shown that social bots are among the factors that most undermine the reliability of online news [18]. These are algorithmically-driven entities that try to emulate human behavior and automatically produce content on social media, to alter the popularity of users and influence discussions of any kind, including political issues. This malicious publishing activity can lower the quality of data extracted from social media, thus leading to the drawing of misleading conclusions [7].

**Dynamicity.** Big data extracted from user interactions on social media platforms are highly dynamic, causing the patterns of interest to change rapidly over time. Due to this, time-related aspects are key to a correct understanding of the extracted information. As an example, trending topics evolve through time, and the voting intentions of social users can fluctuate during an election campaign. This dynamicity reflects in the continuous evolution of hashtags over time, linking social media content to new topics or events that catalyze the attention of the online conversation. This makes the choice of a suitable hashtag not always easy for social users, which hinders those methodologies that exploit hashtags as the main source of information. This issue can be mitigated through hashtag recommendation models, which can be used to generate consistent hashtags, thus enhancing the data processed by hashtag-based techniques [9].

**Resource-intensive computation.** In the age of the Internet of Things and social media platforms, the ability to generate and gather data is increasing constantly and dramatically, which poses a series of challenges to the current solutions aimed at processing, storing, and analyzing Big Data. Due to this, current frameworks are expected to be constantly improved to tackle Big Data issues, allowing the effective extraction of useful knowledge on high-performance computing (HPC) systems, Clouds, and multi-clusters, through parallel and distributed algorithms and frameworks in several application domains [10]. Furthermore, the novel Exascale systems pose new requirements for addressing architectures composed of a huge number of cores. In particular, in the near future, existing frameworks will have to address a wide range of issues related to energy consumption, scheduling, data distribution and access, communication, and synchronization, to enable the scalable implementation of real-world Big Data analysis applications [19].

## 1.2 Structure of the Thesis

The remainder of this thesis is organized as follows.

- Chapter 2 presents *IOM-NN (Iterative Opinion Mining using Neural Networks)* [1], a semi-supervised methodology that leverages an iterative approach based on feed-forward neural networks to estimate the political polarization of public opinion on social media platforms during a political event of interest, such as an election or referendum.
- Chapter 3 presents an extension of IOM-NN which combines political polarization with other techniques, such as emotion analysis and topic discovery, with the aim of outlining an accurate representation of the 2020 US presidential election through a unified analysis workflow [8].
- Chapter 4 deals with data dynamicity and reliability by presenting *TIMBRE (Time-aware opInion Mining via Bot REmoval)* [7], an opinion mining methodology that estimates the voting intentions of users through a hashtag-based classification process, enhanced by considering time-related information and by filtering out content published by social bots.
- Chapter 5 presents *WABC (Weighted Artificial Bee Colony)* [3], a bio-inspired influence maximization algorithm aimed at identifying the main influencers in a politically-polarized network, deriving also the main information diffusion strategies leveraged by each faction during the political campaign.
- Chapter 6 presents a novel translation-based hashtag recommendation model, namely *HASHET (HAshtag recommendation using Sentence-to-Hashtag Embedding Translation)* [9], that relies on the semantic mapping between the embedded representation of a post and the latent representation of its hashtags. The recommendation process of HASHET relies on the concept of locality in the hashtag embedding space, which considers both its underlying topic-based clustering structure and the learned relationships among hashtags.
- Chapter 7 presents *IIWM (Intelligent In-memory Workflow Manager)* [11], a methodology that optimizes the execution of data-intensive workflows on parallel machines by balancing task parallelism and memory usage to minimize

data spilling. IIWM uses machine learning to predict task memory occupancy and execution time and leverages these predictions to determine an effective schedule using a heuristic solution to the bin backing problem.

- Chapter 8 presents a novel methodology that leverages a machine learning-based approach to determine a suitable estimate of data block size for hybrid partitioning, thus optimizing the execution of data-parallel applications on large-scale high-performance infrastructures (e.g., the MareNostrum supercomputer) by requiring minimal resources and domain knowledge [12].
- Chapter 9 summarizes the research context and highlights the key findings of this thesis, summarizing the main contributions through the lens of all the addressed issues and research challenges. Finally, future research lines are outlined, which concludes the thesis.

## 1.3 Publications

### Papers in journals

- R. Cantini, F. Marozzo, G. Bruno, P. Trunfio. "Learning sentence-to-hashtags semantic mapping for hashtag recommendation on microblogs". In: *ACM Transactions on Knowledge Discovery from Data*, vol. 16.2, pp. 1-26, 2022.
- L. Belcastro, R. Cantini, F. Marozzo, A. Orsino, D. Talia, P. Trunfio. "Programming Big Data Analysis: Principles and Solutions". In: *Journal of Big Data*, vol. 9, n. 4, 2022.
- R. Cantini, F. Marozzo, D. Talia, P. Trunfio. "Analyzing Political Polarization on Social Media by Deleting Bot Spamming". In: *Big Data and Cognitive Computing*, vol. 1, n. 6, 2022.
- L. Belcastro, F. Branda, R. Cantini, F. Marozzo, D. Talia, P. Trunfio. "Analyzing voter behavior on social media during the 2020 US presidential election campaign". In: *Social Network Analysis and Mining*, 2022.
- L. Belcastro, R. Cantini, F. Marozzo. "Knowledge Discovery From Large Amounts Of Social Media Data". In: *Applied Sciences*, vol. 12, n. 3, 2022.

- R. Cantini, F. Marozzo, A. Orsino, D. Talia, P. Trunfio. "Exploiting Machine Learning For Improving In-memory Execution of Data-intensive Workflows on Parallel Machines". In: *Future Internet*, vol. 13, n. 5, 2021.
- R. Cantini, F. Marozzo, S. Mazza, D. Talia, P. Trunfio. "A Weighted Artificial Bee Colony Algorithm for Influence Maximization". In: *Online Social Networks and Media*, vol. 26, pp. 100167, 2021.
- L. Belcastro, R. Cantini, F. Marozzo, D. Talia, P. Trunfio. "Learning Political Polarization on Social Media using Neural Networks". In: *IEEE Access*, vol. 8, n. 1, pp. 47177-47187, 2020.

## Papers in conference proceedings

- R. Cantini, F. Marozzo. "Topic Detection and Tracking in Social Media Platforms". In: *EAI International Conference on Pervasive knowledge and collective intelligence on Web and Social Media (PerSoM)*, November 2022.
- R. Cantini, F. Marozzo, A. Orsino, M. Passarelli, P. Trunfio. "A visual tool for reducing returns in e-commerce platforms". In: *6th International Research and Technologies for Society and Industry Innovation for a smart world (IEEE RTSI)*, pp. 474-479, September 2021.
- L. Belcastro, R. Cantini, F. Marozzo, D. Talia, P. Trunfio. "Discovering Political Polarization on Social Media: A Case Study". In: *15th International Conference on Semantics, Knowledge, and Grids (SKG)*, Guangzhou, China, pp. 182-189, September 2019.

## Other publications

- Riccardo Cantini, Fabrizio Marozzo, Alessio Orsino, Domenico Talia, Paolo Trunfio, Rosa M. Badia, Jorge Ejarque, Fernando Vázquez. "Block size estimation for data partitioning in HPC applications using machine learning techniques". In: *arXiv preprint arXiv:2211.10819*, 2022. Under submission.

Chapter 2

# Learning political polarization on social media using neural networks

Computational Politics is a research area that involves a set of techniques aimed at analyzing users' behavior during a political event of interest, informing political strategy and decision-making, and engaging with citizens and stakeholders. It is an interdisciplinary field that spans computer science, political science, and communication, which provides a more data-driven and quantitative approach to understanding and studying political phenomena. Its main applications include the use of machine learning to forecast election results and discover patterns in political behavior, natural language processing to analyze political discourse and track public opinion, network analysis to study the structure of political organizations and the pattern of information diffusion, and the development of interactive platforms for political decision-making and engagement [1, 7, 8, 20, 21].

This chapter presents *IOM-NN* (*Iterative Opinion Mining using Neural Networks*) [1], a methodology that we designed for measuring the polarization of social media users during election campaigns characterized by the competition of political factions or parties, with the final aim to estimate the outcome of the political event under consideration. This makes IOM-NN a suitable tool to support, enhance or even replace opinion polls, since it can capture the opinion of a larger number of people more quickly and at a lower cost, by providing relevant insights

useful to understand the dynamics of the election campaign. This methodology follows a semi-supervised iterative approach based on feed-forward neural networks for analyzing the posts published by social media users. Starting from a limited set of classification rules, created from a small subset of hashtags that are notoriously in favor of specific factions, IOM-NN iteratively generates new classification rules following a high-confidence pseudo-labeling process. Such rules allow determining the political leaning of a post based on the words/hashtags it contains. This information is then used to determine the polarization of social media users towards a faction, finally estimating the outcome of the political event. The methodology has been assessed on two case studies analyzing the polarization of a large number of Twitter users during the 2018 Italian general election and the 2016 US presidential election. The achieved results are very close to the real ones, which reveals the high accuracy and effectiveness of the proposed approach. Moreover, our approach has been compared to the most relevant techniques used in the literature (i.e., *NLP-based sentiment analysis* [22], *adaptive sentiment analysis* [23], *emoji-based polarization* [24], *hashtag-based polarization* [25]), achieving the best accuracy in estimating the polarization of social media users and forecasting the winning candidate in both binary and multi-class settings, in which two or more opposing candidates or factions are present.

During the design of IOM-NN, we dealt with some of the issues highlighted in Section 1.1. Firstly, our technique follows a language-agnostic approach, as it uses a hashtag-based bag-of-words representation. Thanks to this, language barrier issues can be overcome, by classifying social posts according to their political leaning, regardless of the language used to write them. Posts are classified through a strictly conservative process, in which a high threshold on the polarization probability is used for minimizing the risk of misclassification. In addition, random sampling is used to address data imbalance, avoiding the learning process to be biased toward the most represented candidates and factions. Furthermore, the statistical significance of the collected data has been evaluated for assessing the representativeness of users, by understanding whether they can be considered voters in the political event under analysis.

The remainder of the chapter is organized as follows. Section 2.1 reviews existing literature on computational politics, discussing more in-depth the main techniques for analyzing political polarization and predicting election outcomes. Section 2.2 describes the IOM-NN methodology. Section 2.3 presents the case studies. Finally, Section 2.4 concludes the chapter.

## 2.1 Background and related work

According to a recent survey [26], existing literature on computational politics can be categorized into five classes, as discussed in the following.

**Community and user modeling.** This class of works focuses on modeling the behavior of social media users from both an individual and collective viewpoint. Many works in this category are related to the analysis of homophily, i.e., the connection of groups of users driven by common interests, which leads to the formation of community structures of like-minded people [27–29]. Other works focus on modeling the political affiliation of social users, exploiting community information for predicting the results of a political event [1, 30, 31].

**Information flow.** These works investigate how information flows within the network. Most of them analyze the misinformation spread, trying to detect fake news thus limiting its distortion effects on public opinion [32–34]. Other works in this category are also aimed at identifying echo chambers, i.e., situations in which the repetition and sharing of information cause the strengthening of an opinion inside a community [35–37].

**Political discourse.** Works in this category model online discussion from different points of view, taking into account demographic aspects, community structure, and information diffusion patterns. Many works are aimed at extracting the main topics of discussion through topic modeling [38, 39], or identifying political crisis [40]. Opinion mining techniques can be also exploited for identifying the opinion

or mood of social media users about those topics, as users' interactions on social media can affect their political engagement [41–43].

**Election campaigns.** Research contributions in this class are aimed at measuring the engagement of the online audience, enabling large-scale opinion polls and the management of the political campaign. In fact, social media provide an effective platform for engaging users in political discussion, which is often used by politicians during political campaigns [44, 45]. Moreover, the analysis of the political engagement of social users can accurately forecast the final results of the political event under analysis [1, 46].

**System design.** Works in this category propose a full system design of computation politics systems. As an example, Cambre et al. [47] propose a system design that can help to break the echo chamber effect, moderating the online political discussion, while Dade et al. [48] discuss the relationship between political processes, urban environments, and situated technologies.

Focusing on those techniques aimed at estimating political consensus and predicting election outcomes, the following taxonomy can be individuated [17, 49].

**Volume-based.** These techniques basically count the number of mentions (e.g., posts, likes, retweets) related to a candidate/party for predicting the election results. In many cases, such techniques analyze social media data for predicting the outcome of an election. For example, Gaurav et al. [50] proposed a technique based on moving average aggregate probability, which infers the results of an election by counting how many times a candidate's name is mentioned in tweets. Tumasjan et al. [51] used micro-blogging data for understanding how people express their political orientation, showing a high closeness between the volume of tweets mentioning a party and the election results. Burnap et al. [52] analyzed the volume of mentions for calculating an overall score for each party.

**Network-based.** These techniques analyze the social network structure, which is often visualized through graphs or sociograms, to analyze the dynamics of public opinion. Relationships and interactions between social media users can provide useful insights for estimating the standing of political events or identifying the opinion leaders on a social media platform [53]. As an example, some studies have demonstrated a relationship between the centrality of political candidates on social networks and their electoral consensus [54, 55].

**Sentiment- and opinion-based.** These techniques are more advanced than those based on volume, as they consider the opinion of social users and their collective sentiment towards political candidates or parties. In particular, such techniques rely on natural language processing (NLP) and text mining algorithms for analyzing social media posts, extracting the semantic content and the hierarchical structure of the text.

*Text mining-based* techniques discover the sentiment of a text by considering only the words it contains without analyzing its structure. For example, El Alaoui et al. [23] proposed an adaptive sentiment analysis approach that generates dictionaries from tweets classified as positive/negative for the different factions. Such dictionaries are then used to calculate a score for each faction. Marozzo and Bessi [25] proposed a methodology that exploits the keywords contained in tweets for calculating the polarization of social media users and news sites during political campaigns. Chin et al. [24] exploited the emojis contained in a post to determine its sentiment (i.e., positive or negative). Other studies exploited machine learning techniques for discovering the political orientation of users, such as the Naive Bayes algorithm [56], or logistic regression [57].

*NLP-based* works go beyond the mere syntactic context of the text, exploring its semantics with the final aim of understanding its meaning and sentiment. Oikonomou et al. [22] used Textblob[1], a Python library for natural language processing, to predict the outcome of the US presidential election in three states of interest (i.e., Florida, Ohio and North Carolina). Wong et al. exploited [58]

[1] https://textblob.readthedocs.io/en/dev/

SentiStrength[2], a lexicon-based sentiment analysis tool, for modeling the political behaviors of users by analyzing tweets and retweets. Alashri et al. [59] analyzed Facebook posts about the 2016 US presidential election with CoreNLP[3] [60], one of the most popular tools for natural language processing, to examine the dynamics between candidate posts and comments they received on Facebook and calculate a score for each political candidate for measuring his/her credibility on a given issue. Singh et al. [61] carried out a comparison among four machine and deep learning algorithms (i.e., TextBlob, Naive Bayes, SVM, and BERT [15]) for sentiment analysis. The authors used the 2020 US presidential election as a case study, finding that the use of BERT, a transformer-based language representation model, leads to the best results. This is in line with the recent tendency in many fields of NLP to leverage huge pre-trained language representation models for a specific downstream task, via transfer learning. These models are often also fine-tuned, to readapt pre-trained features to work better with task-specific data.

## 2.2 Proposed methodology: IOM-NN

IOM-NN is a semi-supervised methodology that relies on neural networks to perform an iterative process aimed at estimating the polarization of public opinion during a political event characterized by the rivalry of different factions.

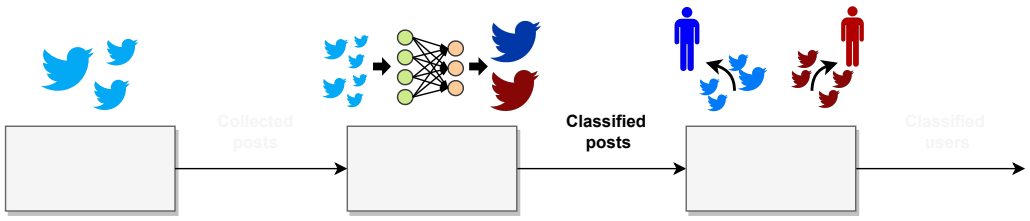



**Figure 2.1:** Main steps of IOM-NN.

[2]http://sentistrength.wlv.ac.uk/
[3]https://stanfordnlp.github.io/CoreNLP/

As shown in Figure 2.1, the proposed methodology consists of three main steps:

1. *Collection of posts*: posts are collected by using a set of keywords related to the selected political event (see Section 2.2.1).

2. *Classification of posts*: the collected posts are classified in a semi-supervised way, through an incremental procedure implemented by a feed-forward neural network (see Section 2.2.2).

3. *Polarization of users*: the classified posts are analyzed for determining the political polarization of users towards a faction, finally estimating the outcome of the political event (see Section 2.2.3).

For each step, a formal description and practical examples are provided in the following sections. For the sake of clarity, Table 2.1 reports the meaning of the main symbols used to describe the different steps.

| Symbol | Meaning |
|---|---|
| $\mathcal{E}$ | Political event |
| $F = \{f_1, f_2, ..., f_n\}$ | Factions |
| $K = K_{context} \cup K_F^{\oplus}$ | Context keywords and positive faction keywords |
| $K_F^{\oplus} = K_{f1}^{\oplus} \cup ... \cup K_{fn}^{\oplus}$ | Positive keywords grouped by factions |
| $P$ | All the posts in input |
| $C^i$ | Classified posts at the $i$-th iteration |
| $N^i$ | Not classified posts at the $i$-th iteration |
| $M^i$ | Classification model generated at the $i$-th iteration |
| $C$ | Classified posts |
| $U$ | Polarized users |
| $S$ | Faction score |

**Table 2.1:** Meaning of the most important symbols used in this chapter.

### 2.2.1 Collection of posts

A political event $\mathcal{E}$ is characterized by the rivalry of different factions, represented as the set $F = \{f_1, f_2, \ldots, f_n\}$. Examples of political events and relative factions are *i*) municipal election, in which a faction supports a mayor candidate;

*ii*) parliament election, in which a faction supports a party; *iii*) presidential election, in which a faction supports a presidential candidate. The posts are collected by using a set $K$ of keywords that people commonly use to refer to the political event $\mathcal{E}$ on social media. The selection of such keywords usually requires a small amount of domain knowledge, as they can be manually selected among the trending hashtags related to $\mathcal{E}$, or automatically generated through specific patterns, like "#vote + *candidate*", often used for labeling politically polarized posts. The keywords in $K$ can be divided into two groups:

- Context keywords, $K_{context}$, i.e., generic keywords that can be associated to $\mathcal{E}$ without referring to any specific faction in $F$.
- Positive faction keywords, $K_F^{\oplus} = K_{f1}^{\oplus} \cup ... \cup K_{fn}^{\oplus}$, where $K_{fi}^{\oplus}$ contains the keywords used for supporting $f_i \in F$ .

The keywords in $K$ are given as input to public APIs provided by social media platforms, to collect posts containing one or more keywords. Posts are not collected in real-time, but downloaded at a given time after their publication (e.g., 24 hours). In this way, we are able to get some statistics related to the popularity of a post (e.g., number of shares, number of likes). Since data collection is usually a continuous process, new keywords can be discovered and integrated into $K$ during the collection procedure. It is important to highlight that obtaining a representative collection of posts depends on two factors: *i*) the quality and the number of keywords used; *ii*) the amount of data that can be downloaded from social media. Regarding the latter factor, attributable to platform biases, it is increasingly difficult to obtain complete data from social media platforms due to the restrictions introduced for protecting the privacy of users.

Collected posts undergo the following preprocessing operations: *i*) the text of each post is lowercased and accented characters are normalized; *ii*) words are stemmed to reduce morphological variation (e.g., votes or voted → vot); *iii*) URLs, mentions, and stop-words are removed; *iv*) frequent bigrams are identified (e.g., Donald Trump → Donald_Trump); *v*) all the posts written in a language other than those spoken in the countries hosting the event $\mathcal{E}$ are filtered out.

The output of this step is a collection of posts $P$ to be labeled according to their political leaning in the next one. Figure 2.2 shows an example of how posts are collected using keywords about the 2016 US presidential election. Some of these keywords are generic (e.g., #*election*2016), and others are used to support a specific candidate (e.g., #*voteblue* for Clinton and #*maga* for Trump).

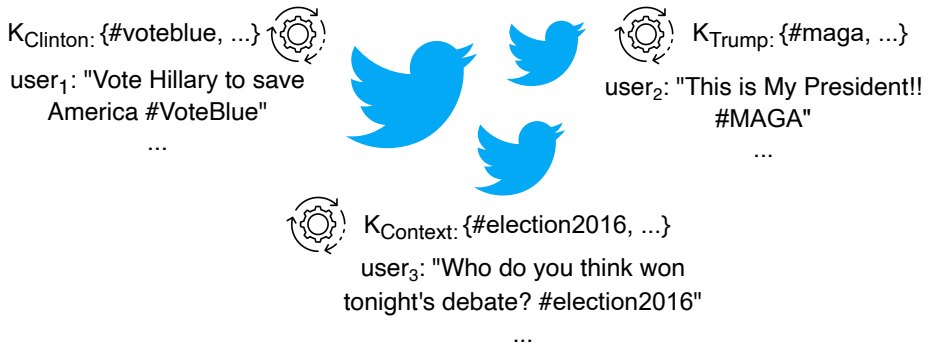



**Figure 2.2:** Example of how the collection of posts step works.

As stated in Section 1.1, the statistical significance of the collected data should always be evaluated before starting the analysis. In this case, we used a wide range of information obtained from statistical reports or directly extracted from users' metadata to study their age, gender, and geographical distribution. The aim is to assess users' representativeness by understanding whether they can be considered voters of the analyzed political event (more details in Sections 2.3.1 and 2.3.2).

### 2.2.2 Classification of posts

During this step, the posts in the set $P$ are classified in favor of a faction using a semi-supervised approach. Firstly, they are labeled based on the keywords in $K_F^{\oplus}$, which represent the initial knowledge that must be given to the algorithm to start the process. Specifically, posts containing keywords related to exactly one faction are polarized toward that faction, while the remaining ones are labeled as neutral. This set of high-confidence labeled data is then gradually expanded through an iterative classification process, during which the model tries to assign a label to neutral posts by exploiting the knowledge acquired at the previous iterations.

**ALGORITHM 1:** Classification of the posts.

**Input :** Set of posts $P$, set of positive faction keyword $K_F^{\oplus}$, threshold $th$, minimum increment $\varepsilon$, maximum number of iterations $max_{iters}$
**Output:** Classified posts $C$

1 $C^0 \leftarrow \varnothing$;
2 $M^0 \leftarrow textualModel.build(K_F^{\oplus})$;
3 **for** $p \in P$ **do**
4 $\quad v_b \leftarrow classify(M^0, p)$;
5 $\quad$ **if** $sum(v_b) = 1$ **then**
6 $\quad\quad f \leftarrow argmax(v)$;
7 $\quad\quad C^0 \leftarrow C^0 \cup \langle p, f \rangle$;
8 $C \leftarrow C^0$;
9 $N^0 \leftarrow P \backslash C^0$;
10 **for** $i = 1; i <= max_{iters}; i++$ **do**
11 $\quad C^i \leftarrow \varnothing$;
12 $\quad M^i \leftarrow neuralNetwork.train(C)$; // $C = C^0 \cup \ldots \cup C^{i-1}$
13 $\quad$ **for** $p \in N^{i-1}$ **do**
14 $\quad\quad v_p \leftarrow classify(M^i, p)$;
15 $\quad\quad$ **if** $max(v_p) > th$ **then**
16 $\quad\quad\quad f \leftarrow argmax(v)$;
17 $\quad\quad\quad C^i \leftarrow C^i \cup \langle p, f \rangle$;
18 $\quad C \leftarrow C \cup C^i$;
19 $\quad N^i \leftarrow N^{i-1} \backslash C^i$;
20 $\quad$ **if** $\frac{|C^i|}{|N^{i-1}|} < \varepsilon \vee \frac{|C^i|}{|N^{i-1}|} > 1 - \varepsilon$ **then**
21 $\quad\quad$ **break**
22 **return** $C$

Algorithm 1 shows the pseudo-code used for post-classification, which is divided into two main parts, described below.

In the first part (lines 1-9), the algorithm performs the initial iteration (iteration 0). Here, it initializes an empty set $C^0$ for storing the classified posts and builds a classification model $M^0$ based on the positive faction keywords in $K_F^{\oplus}$ (lines 1-2). The algorithm iterates (lines 3-7) on each post $p$ in $P$ classifying it using $M^0$, which produces a vector $v_b$ (line 4) such that $v_b[i]$ is 1 if $p$ contains a keyword from $K_{fi}^{\oplus}$, 0 otherwise. Afterward, if $p$ is in favor of a single faction $f$ (lines 5-6), the classified post (i.e., a pair $\langle p, f \rangle$) is added to $C^0$ (line 7). At the end of iteration 0,

the set of posts classified through $K_F^{\oplus}$ is assigned to $C$ (line 8), which will be used, in the subsequent iterations, to store all posts that the model was able to annotate with high confidence up to that point, according to its current knowledge. The set of unclassified posts ($N^0$) is obtained as the difference between $P$ and $C^0$ (line 9).

The second part (lines 10-21) iteratively generates new classification rules for expanding the set of labeled posts, performing at most $max_{iters}$ iterations. Specifically, at the $i$-th iteration, the following operations are performed:

- It initializes a set $C^i$ to store the classified posts at $i$-th iteration (line 11).
- It builds a classification model $M^i$ by training a neural network using the classified posts at previous iterations, stored in $C = C^0 \cup \ldots \cup C^{i-1}$ (lines 12). The training set is balanced using a random under-sampling approach to avoid a learning process biased towards the majority classes.
- For each unclassified post at the previous iteration $N^{i-1}$ (line 13), the algorithm classifies $p$ using $M^i$, which produces a vector of softmax probabilities $v_p$ (line 14), where $v_p[i]$ is the probability that $p$ supports $f_i$. If the maximum value of $v_p$ is greater than the given threshold $th$, the post is assigned to the most likely faction $f$ and added to $C^i$ (lines 15-17).
- The set of classified posts $C^i$ is added to $C$, and the remaining unclassified posts $N^i$ are obtained as difference between $N^{i-1}$ and $C^i$ (lines 18-19).
- If the ratio between the size of $C^i$ and $N^{i-1}$ is lower than $\varepsilon$ or greater than $1-\varepsilon$ (i.e., convergence is reached), or the maximum number of iterations is exceeded, the algorithm stops iterating and returns the dictionary $C$, containing all the posts classified at the various iterations (lines 20-22).

The neural networks used in our methodology were implemented using the framework *Keras*[4] with *TensorFlow*[5] as the back-end engine. These networks are *multilayer perceptrons* with one hidden layer that includes 2/3 of the input neurons and a ReLU activation. The choice of using a single hidden layer simplifies

[4]https://keras.io/
[5]https://www.tensorflow.org/

the learning process with respect to a deeper network, also limiting the risk of overfitting. In addition, we used a dropout layer after the fully-connected one, with a dropout rate fixed at 5%, as a regularization mechanism. Finally, the output layer has a number of neurons equal to that of the considered candidates/factions, and a softmax activation to distribute the probability of a post being assigned to them. The training phase has been carried out using early stopping callbacks to prevent overfitting, accuracy as the main metric, a batch size of 32, the categorical cross-entropy loss function, and the optimization algorithm ADAM [62]. In addition, since the parameters of the neural network are randomly initialized, IOM-NN repeats the post-classification phase with a new random seed for $n_{seeds}$ times, in order to get more stable results, by reducing the risk of getting stuck in local minima. Figure 2.3 shows a graphic representation of how the post-classification algorithm discussed above works, starting from a set of input posts $P$ and - for example purposes - converging in three iterations.

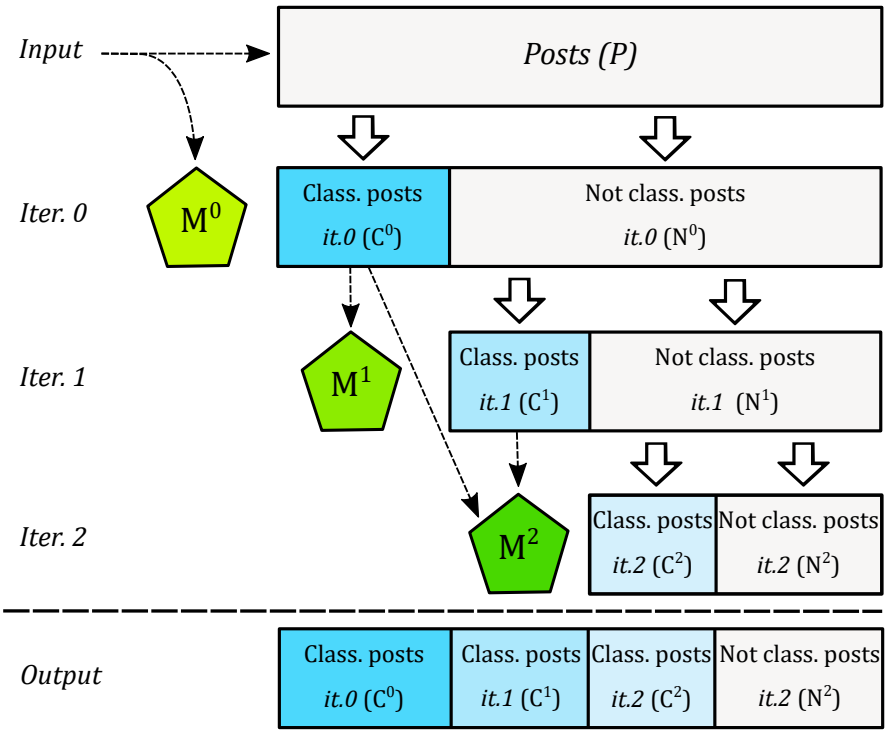



**Figure 2.3:** Representation of the classification of posts algorithm.

At iteration 0, the classification model $M^0$ is created using the faction keywords $K_F^{\oplus}$. This model is used to classify $P$, which generates two subsets for classified ($C^0$) and unclassified posts ($N^0$) respectively. At iteration 1, a new model $M^1$ is trained on $C^0$ and is used to label the unclassified posts from the previous iteration ($N^0$). The classification process splits $N^0$ into two new subsets: $C^1$ for high-confidence classified posts and $N^1$ for unclassified ones. Similarly, at iteration 2 the model $M^2$ is trained using $C^0 \cup C^1$, generating the sets $C^2$ and $N^2$. Generalizing, at the $i-th$ iteration, the model $M^i$ is trained using $C = C^0 \cup \ldots \cup C^{i-1}$, which is used to label posts in $N^{i-1}$. Classified posts at this point are added to the current knowledge ($C = C \cup C^i$), which will be used in the next iteration to train $M^{i+1}$ and label $N^i$. The process iterates until the ratio between the size of $C^i$ and the size of $N^{i-1}$ is lower than $\varepsilon$ or greater than $1-\varepsilon$, i.e., convergence is reached. In the end, two sets will be available: the whole set of classified posts $C$, and the remaining posts (in Figure 2.3, $N^2$), which are classified as neutrals.

It is worth noting that, due to the incremental nature of the annotation process, IOM-NN is not tied to a specific set of initial faction keywords and does not require an in-depth knowledge of the political event under consideration. In fact, even starting from a small but representative set of faction keywords, IOM-NN is able to iteratively discover new political-oriented topics of discussion and infer new classification rules. This implies good robustness and generalizability of our methodology. Moreover, the discovered knowledge increases monotonically iteration by iteration, as posts once classified above the minimum confidence threshold, are added into $C$ permanently. Although this approach ensures the convergence of the algorithm, it entails the risk of confirmation bias, a typical problem of many semi-supervised schemes. In particular, incorrectly classified posts in the first iterations could negatively impact the entire learning process. However, the effects of this issue are alleviated by the use of a high-confidence threshold and by the re-initialization of the neural network weights between successive iterations, which regularizes its convergence direction, leading to more robust results with respect to iterative fine-tuning.

### 2.2.3 Polarization of users

This step aims to analyze the set of previously classified posts to determine users' polarization toward a faction. Algorithm 2 shows the pseudo-code used in this step, which takes as input a collection of classified posts $C$ (i.e., the output of Algorithm 1), a filtering function $filter$ and its parameters $par_f$, and a polarization function *polarize* and its parameters $par_p$. The output is composed of a collection of classified users $U$ and a faction score $S$ containing the overall polarization percentages for each faction.

**ALGORITHM 2:** Prediction of user polarization.

**Input :** Classified posts $C$, filtering function $filter$, filtering function parameters $par_f$, polarization function *polarize*, polarization function parameters $par_p$.
**Output:** Classified users $U$, faction score $S$

```
1  C_U ← aggregateByUser(C);
2  U ← ∅;
3  S ← ∅;
4  for ⟨u, P_u⟩ ∈ C_U do
5      if filter(⟨u, P_u⟩, par_f) then
6          v_s^u ← polarize(P_u, par_p);
7          U ← U ∪ ⟨u, v_s^u⟩;
8  for ⟨u, v_s^u⟩ ∈ U do
9      S ← S + v_s^u;
10 S ← S/sum(S);
11 return U, S
```

As a first step, the classified posts are aggregated by user to produce a dictionary ($C_U$), which contains the list of classified posts $P_u$ for each user $u$ (line 1). Two empty variables are initialized for storing the output (lines 2-3). On each pair $\langle u, P_u \rangle$ of $C_U$, the algorithm performs the following operations (lines 4-7):

- It filters out all the pairs that do not match the criteria defined by the $filter$ function (line 5). For example, users who published a number of posts below a given threshold are skipped.

- Using the classified posts $P_u$, it computes $v_s^u$ a vector containing the score of

user $u$ for each faction (line 6). The score vector is calculated by using the function *polarize*.

- It adds the pair $\langle u, v_s^u \rangle$ to $U$ (line 7).

Then, the algorithm calculates the overall faction score $S$ as the normalized sum of the user vector scores $v_s^u$ (lines 8-10). Finally, the two outputs are returned (line 11). The *filter* and *polarize* functions that we used in the experimental evaluation (see Section 2.3), were configured as follows.

- *filter*: a user $u$ is considered only if he/she fulfills the following criteria: $i$) $u$ posted at least *minPosts* on the political event of interest; $ii$) there exists a faction $f$ for which $u$ has published more than 2/3 of his/her posts. This last condition entails that more than half of the posts published by $u$ were devoted to his/her preferred faction.

- *polarize*: for each user $u$, the vector score $v_s^u$ contains, in position $f$ (i.e., the preferred faction), the percentage of posts written in favor of it, 0 otherwise.

Figure 2.4 shows how the user polarization step is performed on a set of seven example posts, published by four different users and classified by Algorithm 1.

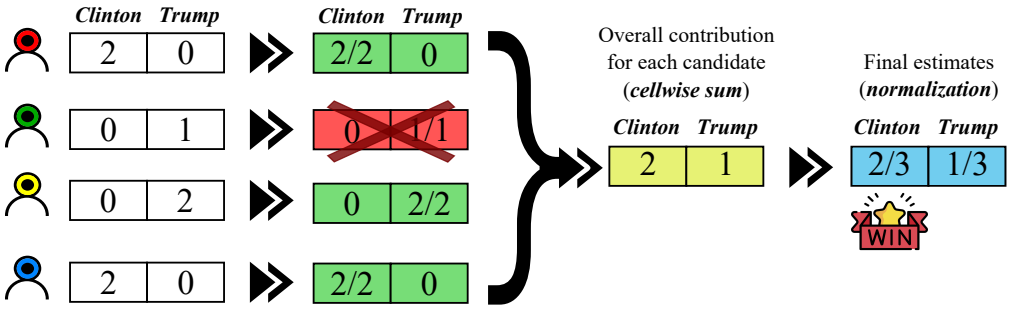



**Figure 2.4:** Representation of the user polarization algorithm.

For each user, the posts in favor of Clinton and Trump are counted, discarding those users who have published less than two tweets. Then, the polarization vector for each user is computed containing the percentage of posts published in favor of his/her preferred faction. Lastly, the final score vector $S$ is determined, which contains the overall polarization percentages for the two candidates.

## 2.3 Case studies

In this section, we describe and analyze two case studies: the 2018 Italian general election and the 2016 US presidential election. In both case studies, for each faction $f_i$ we defined three sets of keywords $K_{fi}^{\oplus}$, $K_{fi}^{\ominus}$ and $K_{fi}^{\bigcirc}$ that are respectively positive, negative, and neutral keywords for faction $f_i$. For example, for the *Movimento 5 Stelle* (*M5S*) faction in the 2018 Italian general election, $K_{M5S}^{\oplus}$ contains keywords used to clearly support *M5S* party (e.g., *#iovotoM5S*), $K_{M5S}^{\ominus}$ contains keywords to speak negatively about *M5S* (e.g., *#maiM5S*), $K_{M5S}^{\bigcirc}$ contains neutral keywords for *M5S* (e.g., *m5s* or *movimento5stelle*).

As described in Section 2.2, *IOM-NN* exploits only positive faction keywords ($K_{fi}^{\oplus}$) for classifying posts and then for determining the polarization of users. For evaluating the accuracy of IOM-NN, we carried out an extensive comparison with the most relevant techniques used in the literature:

1. *NLP-based sentiment analysis* [22][59]. For each post, we used CoreNLP for calculating a sentiment score that ranges from 0 (very negative) to 4 (very positive). The neutral keywords ($K_{fi}^{\bigcirc}$) are then used for grouping posts and calculating an overall score for each faction.

2. *Adaptive sentiment analysis* [23]. Starting from the positive and negative keywords of each faction ($K_{fi}^{\oplus}$ and $K_{fi}^{\ominus}$), this technique generates two word-polarity dictionaries, which are built from a set of posts containing such positive and negative keywords. Finally, a score for each faction is returned.

3. *Emoji-based polarization* [24]. This technique groups the posts of each faction by using keywords ($K_{fi}^{\bigcirc}$), then classifies their sentiment by using emojis and returns a score for each faction.

4. *Hashtag-based polarization* [25]. The posts are classified as in favor of a given faction based on the positive faction keywords ($K_{fi}^{\oplus}$). Then the posts are aggregated by user and the polarization of each user is computed, from which the overall faction scores are determined.

To allow a direct comparison with the real percentages, the results obtained by the different techniques have been normalized with respect to the sum of the real ones.

### 2.3.1 2018 Italian general election

Here we discuss the case study carried out to analyze the polarization of a large number of Twitter users during the 2018 Italian general election. Italians voted to elect 630 deputies and 315 senators of the XVIII legislature. The results decreed the center-right coalition as the most voted, with about 37% of votes, while the most voted list was the *Movimento 5 Stelle*, which received over 32% of votes. The electorate was composed of 50,782,650 voters for the Chamber of Deputies and 46,663,202 for the Senate[6], with a turnout of about 73%, the lowest in Italian republican history. The analysis we carried out focused on the four most successful political factions, in decreasing order of consensus: *M5S* (*Movimento 5 Stelle*), *PD* (*Partito Democratico*), *LEGA*, *FI* (*Forza Italia*). In the following, we show how the classification model has been trained, discussing also the main achieved results.

#### Models training and iteration-level results

IOM-NN was leveraged to classify 60,782 tweets posted by 21,883 users from February 1, 2018 to March 3, 2018 (i.e., the day before the election). We assessed the statistical significance of the data involved in the analysis as follows.

- All the tweets under analysis have been written in Italian, which means they have the *lang* field set to *it* (Italian). With very few exceptions, the Italian language is used only by Italian people who reside in Italy or abroad[7].
- 92% of the users who set a location in their profile specified a region in Italy. Moreover, there is a strong correlation between the number of users

[6] http://www.interno.gov.it/it/notizie/elezioni-2018-come-vota-corpo-elettorale-tessera-elettorale (in Italian)

[7] https://en.wikipedia.org/wiki/Italian_language

that can be assigned to a region and the number of people actually living in that region according to official statistics (the Pearson correlation coefficient $R$ is equal to 0.8, significant for $p - value < 0.01$).

- About 98% of the Italian social media users are adults and equally divided by gender (51.2 females and 48.8 males)[8].

We used the following positive faction keywords for analyzing the collected data:

- $K^{\oplus}_{M5S}$={#iovotom5s,#m5salgoverno,#dimaiopresidente}
- $K^{\oplus}_{PD}$={#sceglipd,#iovotopd,#pdvinci}
- $K^{\oplus}_{LEGA}$={#4marzovotolega,#iovotolega,#salvinipremier}
- $K^{\oplus}_{FI}$={#berlusconipresidente,#votoforzaitalia, #4marzovotoforzaitalia}

The threshold $th$ and the minimum increment $\varepsilon$ have been set to 0.9 and 5% respectively. In our test, the post-classification algorithm terminated in 4 iterations by annotating 23,997 tweets, which represents about 39.5% of the total. Table 2.2 shows the obtained results at each iteration by specifying the number of classified and unclassified tweets, the ratio $\frac{|C^i|}{|N^{i-1}|}$ and the accuracy of the neural network.

| Iteration | Tweet input | Classified ($C^i$) | Not classified ($N^i$) | Perc. of class. tweets | $\frac{\lvert C^i\rvert}{\lvert N^{i-1}\rvert}$ | Accuracy |
|---|---|---|---|---|---|---|
| 0 | 60,782 | 3,072 | 57,710 | 5.1% | 5.1% | - |
| 1 | 57,710 | 14,676 | 43,034 | 24.1% | 25.4% | 0.916 |
| 2 | 43,034 | 4,677 | 38,357 | 7.7% | 10.9% | 0.990 |
| 3 | 38,357 | 1,572 | 36,785 | 2.6% | 4.1% | 0.992 |
| Total | 60,782 | 23,997 | 36,785 | 39.5% | - | - |

**Table 2.2:** Partial results for each iteration achieved by IOM-NN (2018 Italian general election).

[8]https://wearesocial.com/it/digital-2019-italia

### Polarization of users and final results

The algorithm described in Section 2.2.3 has been used for analyzing the users who have written the 23,997 classified tweets so as to determine their polarization degree towards the considered factions. The main achieved results are summarized in Table 2.3.

| | **Tweets** | **Users** | **M5S%** | **PD%** | **LEGA%** | **FI%** | **LogAcc** | **MAPE** | **MAE** |
|---|---|---|---|---|---|---|---|---|---|
| Real percentages | - | - | *32.68* | *18.72* | *17.37* | *14.01* | - | - | |
| Averages of opinion polls | - | ≈1,000 | 28.10 | 22.80 | 13.40 | 16.40 | 0.81 | 0.19 | 3.74 |
| ***IOM-NN*** | ***23,997*** | ***9,942*** | ***31.64*** | ***19.89*** | ***18.45*** | ***12.80*** | ***0.94*** | ***0.06*** | ***1.13*** |
| NLP-based sent. analysis | 25,299 | - | 20.84 | 30.69 | 13.26 | 17.99 | 0.63 | 0.38 | 7.98 |
| Adaptive sent. analysis | 53,488 | - | 21.67 | 18.28 | 21.30 | 21.53 | 0.73 | 0.28 | 5.72 |
| Emoji-based polarization | 234 | - | 32.25 | 13.76 | 23.20 | 13.57 | 0.84 | 0.16 | 2.92 |
| Hashtag-based polarization | 3,053 | 1,589 | 21.03 | 28.78 | 6.70 | 26.28 | 0.39 | 0.60 | 11.16 |

**Table 2.3:** Obtained percentages and accuracy evaluation (2018 Italian general election).

The first three rows show a comparison between the official results, the average of the latest polls, and the percentages obtained by IOM-NN. We evaluated the accuracy through different statistical indexes, comparing the obtained results with the latest opinion polls published before the elections. Considering the four most supported parties, our methodology obtained the following approval percentages: M5S 31.64%, PD 19.89%, LEGA 18.45%, and FI 12.80%. These results are extremely close to the real ones (i.e., M5S 32.68%, PD 18.72%, LEGA 17.37%, FI 14.01%), even more than the average of polls. The obtained results are also characterized by a quite good log accuracy ratio (LogAcc), as well as a negligible value of mean percentage and absolute errors (MAPE and MAE). In particular, we achieved a mean average error of 1.13 percentage points and a log accuracy ratio very close to 1, while opinion polls achieved an MAE of 3.74 percentage points and a LogAcc of 0.81. This confirms the ability of the proposed methodology to correctly detect the political polarization of public opinion, correctly forecasting election results. For the sake of completeness, Figure 2.5 shows an infographic about the comparison of the real percentages, opinions polls, and obtained results.

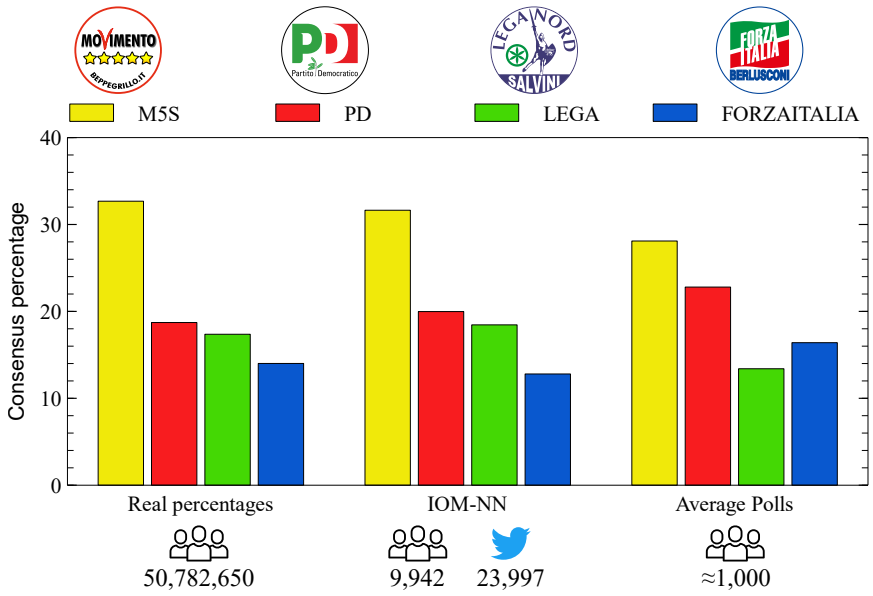



**Figure 2.5:** Comparison among real percentages, opinions polls, and IOM-NN results (2018 Italian general election).

Table 2.3 also presents the results obtained by the other techniques in the literature (rows 4-7). These techniques have been configured with the positive faction hashtags used by IOM-NN (see $K^{\oplus}_{M5S}$, $K^{\oplus}_{PD}$, $K^{\oplus}_{LEGA}$ and $K^{\oplus}_{FI}$) and the following negative and neutral faction keywords:

- $K^{\ominus}_{M5S}$={#nom5stelle, #rimborsopolim5s, #maim5s},
  $K^{\bigcirc}_{M5S}$={m5s, movimento5stelle, dimaio}
- $K^{\ominus}_{PD}$={#nonvotopd, #maipd, #bastapd},
  $K^{\bigcirc}_{PD}$={pd, partitodemocratico, renzi}
- $K^{\ominus}_{LEGA}$={#maiconsalvini, #iononvotolega},
  $K^{\bigcirc}_{LEGA}$={lega, salvini, leganord}
- $K^{\ominus}_{FI}$={#maipiuberlusconi, #stopberlusconi},
  $K^{\bigcirc}_{FI}$= {forzaitalia, berlusconi}

Compared to such techniques, IOM-NN turned out to be the most accurate in estimating the voting percentages, outperforming the competitors in terms of achieved LogAcc, MAPE, and MAE. Specifically, we found out what follows.

Compared to emoji- and hashtag-based techniques, IOM-NN was able to classify a much greater number of tweets and users, which ensures greater statistical representativeness of data and robustness of results. In particular, emoji-based techniques are strictly tied to the presence of emojis, which imposes a huge limit on the data considered in the analysis process. A similar issue affects hashtag-based methods, which can only analyze posts containing keywords from a predefined set. IOM-NN, due to its semi-supervised incremental nature, can leverage all the annotations from an initial set of keywords to gradually expand its knowledge by labeling new data iteratively, finally converging to a better solution.

Compared to sentiment analysis techniques (i.e., NLP-based and adaptive sentiment analysis), IOM-NN was able to achieve better results as it is not volume-based. Our methodology, in fact, is not influenced by users who published a large number of posts, as the overall polarization percentages are not directly computed from them. Instead, like opinion-based techniques, IOM-NN leverages all posts published by a user to estimate his/her degree of political polarization, i.e. his/her normalized contribution to the final percentages. Finally, the overall polarization percentages for each faction are obtained from all these contributions.

### 2.3.2 2016 US presidential election

After the presentation of the Italian use case, here we discuss the analysis we carried out on the 2016 US presidential election, which was characterized by the rivalry between Hillary Clinton and Donald Trump. The analysis has been performed on data collected for ten US Swing States: Colorado, Florida, Iowa, Michigan, Ohio, New Hampshire, North Carolina, Pennsylvania, Virginia, and Wisconsin. Swing states are those characterized by greater political uncertainty, in which neither major political party holds a lock on the outcome of presidential elections. These states are considered of strategic importance, as their votes have a high probability of being the deciding factor in a presidential election. For each state, data have been collected through the standard Search Twitter API, which allows for collecting tweets published in a given area or place. Overall about 2.5 million tweets, posted by 521,291 users, have been collected from October 10,

2016, to November 7, 2016 (i.e., the day before the election). From such data, we filtered out all the tweets posted by users with a not defined or inconsistent location or with a location that does not belong to any of the considered states. Filtered data (818,403 tweets posted by 141,959 users) were statistically representative since:

- All the tweets under analysis had the *lang* field set to *en* (English).
- For each state, we calculated a strong linear correlation between the number of analyzed users and the number of people actually living in that state that belong to the voting-eligible population, according to official statistics (we obtained a linear correlation $R = 0.95$, significant for $p < 0.01$).
- About 94% of the social media users in the USA are adults (at least 18 years old) and almost equally divided by gender (42.7% females and 57.3% males)[9].

The following keywords have been used in our experiments:

- $K^{\oplus}_{Clinton}$={#voteHillary, #imwithher, #strongertogether, #hillary2016},
  $K^{\ominus}_{Clinton}$={#neverhillary, #lockherup},
  $K^{\bigcirc}_{Clinton}$={clinton, hillary, democrats, dems}
- $K^{\oplus}_{Trump}$={#voteTrump, #maga, #americafirst, #wakeupamerica},
  $K^{\ominus}_{Trump}$={#nevertrump, #dumpfortrump} ,
  $K^{\bigcirc}_{Trump}$={trump, donald, republicans, gop}

Table 2.4 shows the results obtained using IOM-NN in comparison with the real voting percentages, the main opinion polls, and the other related techniques. For each state in the table, we reported the results obtained by the two candidates, where “C” stands for Clinton and “T” for Trump. In addition, when the winning candidate is correctly identified, the percentage is written in bold.

[9]https://www.statista.com/statistics/376128/facebook-global-user-age-distribution/

| State | Real percentages | | Average of opinion polls | | IOM-NN | | NLP-based sent. analysis | | Adaptive sent. analysis | | Emoji-based polarization | | Hashtag-based polarization | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *C* | *T* | *C* | *T* | *C* | *T* | *C* | *T* | *C* | *T* | *C* | *T* | *C* | *T* |
| Colorado | **48.2** | 43.3 | **43.3** | 40.4 | **53.3** | 38.2 | **49.1** | 42.4 | 45.4 | 46.1 | **52.1** | 39.4 | **49.9** | 41.7 |
| Florida | 47.8 | **49.0** | 46.4 | **46.6** | 49.9 | 46.9 | 46.1 | **50.7** | 49.3 | 47.5 | 49.3 | 47.5 | 48.4 | 48.4 |
| Iowa | 41.7 | **51.1** | 41.3 | **44.3** | 44.6 | **48.2** | 49.5 | 43.3 | 47.0 | 45.8 | 46.9 | 45.9 | 42.0 | **50.8** |
| Michigan | 47.3 | **47.5** | 45.4 | 42.0 | 43.5 | **51.4** | 49.8 | 45.0 | 47.5 | 47.3 | 48.2 | 46.6 | 38.1 | **56.7** |
| New Hamp. | **47.0** | 46.6 | **43.3** | 42.7 | **49.0** | 44.6 | **48.2** | 45.4 | 45.6 | 48.0 | 44.0 | 49.6 | 45.5 | 48.1 |
| North Car. | 46.2 | **49.8** | 46.4 | 46.4 | 47.8 | **48.2** | 50.6 | 45.4 | 49.2 | 46.8 | 48.7 | 47.3 | 40.7 | **55.3** |
| Ohio | 43.6 | **51.7** | 42.3 | **45.8** | 46.9 | **48.4** | 51.3 | 44.0 | 48.4 | 46.9 | 50.9 | 44.4 | 42.6 | **52.7** |
| Pennsylvania | 47.9 | **48.6** | 46.2 | 44.3 | 54.9 | 41.6 | **50.3** | 46.2 | 48.1 | **48.4** | 54.4 | 42.1 | 50.2 | 46.3 |
| Virginia | **49.8** | 44.4 | **47.3** | 42.3 | **53.2** | 41.0 | 49.0 | 45.2 | 46.3 | 47.9 | **48.2** | 46.0 | 45.1 | 49.1 |
| Wisconsin | 46.5 | **47.2** | 46.8 | 40.3 | 45.8 | **47.9** | 48.5 | 45.2 | 47.4 | 46.3 | 49.8 | 43.9 | 40.8 | **52.9** |
| Correctly pred. | - | | **6/10** | | **8/10** | | **4/10** | | **1/10** | | **2/10** | | **6/10** | |
| Tweets | - | | - | | 718,425 | | 775,277 | | 818,403 | | 23,937 | | 409,146 | |
| Users | - | | ≈10,000 | | 125,891 | | - | | - | | - | | 78,430 | |
| Mean LogAcc | - | | 0.97 | | 0.93 | | 0.93 | | 0.95 | | 0.93 | | 0.93 | |
| Mean MAPE | - | | 0.03 | | 0.07 | | 0.07 | | 0.05 | | 0.07 | | 0.07 | |
| Mean MAE | - | | 1.57 | | 3.19 | | 3.21 | | 2.37 | | 3.48 | | 3.25 | |

**Table 2.4:** Obtained percentages and accuracy evaluation on the 2016 US presidential election dataset.

Firstly, IOM-NN was able to correctly identify the winning candidate in 8 out of 10 cases, outperforming the opinion polls that correctly classifies 6 out of 10 states. This result is also represented in Figure 2.6, in which the *Democratic Donkey* symbolizes the party of Hillary Clinton, while the *Republican Elephant* that of Donald Trump.

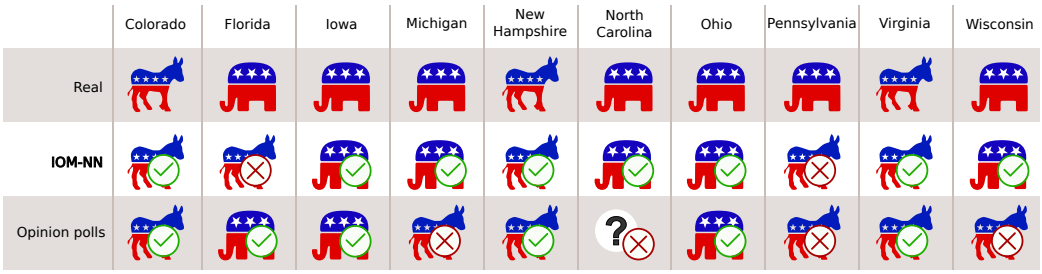



**Figure 2.6:** Comparison among the real winning candidate and that identified by IOM-NN and opinions polls.

Also in comparison with the other techniques, IOM-NN turned out to be the most accurate in predicting the winning candidate. In particular, emoji- and hashtag-based techniques classified a much smaller amount of tweets and correctly identified the winning candidate in 6 and 3 cases respectively. Moreover, the results of the adaptive sentiment analysis were very poor, with only one cor-

rect prediction, while the NLP-based technique produced correct predictions in 4 out of 10 cases. It is worth noting that, compared to IOM-NN, some techniques (i.e., adaptive sentiment analysis and opinion polls) achieved slightly better results in terms of log accuracy ratio, MAPE, and MAE, because their predictions are quite balanced by assigning almost the same score to the two candidates in the different states. However, as for the adaptive sentiment analysis, this behavior may be a symptom of higher model uncertainty, which leads to lower forecasting accuracy. In general, it is better to correctly forecast the winning candidate, even with a gap greater than the real one, than to shorten the distances by reversing the predicted polarity, which can be a crucial issue while analyzing these kinds of states characterized by a high degree of uncertainty.

## 2.4 Conclusions

This chapter discussed IOM-NN, a semi-supervised methodology that leverages an iterative approach based on feed-forward neural networks to estimate the political polarization of public opinion on social media platforms during a political event of interest. We evaluated its effectiveness in two case studies analyzing the polarization of a large number of Twitter users during the 2018 Italian general election and the 2016 US presidential election. It produced quite good estimates, very close to the real voting percentages, outperforming the opinion polls and the most relevant techniques used in the literature in terms of forecasting accuracy.

Our methodology can therefore be seen as a suitable alternative to empower or even replace traditional opinion polls, by providing relevant insights useful to understand the dynamics of the election campaign. Indeed, it can capture the opinion of a larger number of people more quickly and at a lower cost, also for topics perceived as embarrassing or offensive, for which people are reluctant to declare their true opinion during the polls. An open-source implementation of IOM-NN is available on Github[10].

[10] `https://github.com/SCAlabUnical/IOM-NN`

In future work, we can adapt our methodology to process real-time data, also coming from different sources (e.g, e-commerce and news sites blogs). Moreover, its effectiveness can also be evaluated in different scenarios, other than the political one, such as assessing the reputation of companies and analyzing product adoption. In fact, IOM-NN can be easily generalized to different use cases, as it is not tied to any specific application domain, and only relies on the representativeness of the analyzed posts. Furthermore, we can integrate it with other techniques, introducing new steps to improve the quality of the achieved results and the understanding of the analyzed political phenomena. As an example, a hashtag recommendation model can be used for enriching the information content of the analyzed data, since keyword-based approaches like IOM-NN are strongly dependent on the availability of consistent hashtags in social media posts [9]. In the next chapter, we will show how IOM-NN can be integrated with emotion analysis, topic discovery, and temporal analysis, with the final aim of outlining an accurate representation of a political event from different points of view, through a unified analysis workflow.

Chapter 3

# Analyzing voter behavior on social media: the case of the 2020 US presidential election

In the previous chapter, we presented *IOM-NN* (*Iterative Opinion Mining using Neural Networks*), a semi-supervised methodology that leverages an iterative approach based on feed-forward neural networks to estimate the political polarization of public opinion on social media platforms during a political event of interest. In this chapter, we show how IOM-NN can be integrated with other techniques to go beyond opinion mining with the ultimate goal of broadening the understanding of the analyzed political phenomenon, outlining an accurate representation from different viewpoints through a unified analysis workflow. We designed such workflow in [8], by jointly applying topic discovery, opinion mining, and emotion analysis techniques on social data related to the 2020 US presidential election, characterized by the rivalry between Donald Trump and Joe Biden.

We extracted the main discussion topics characterizing the 2020 US election campaign by leveraging the unsupervised approach proposed in [9], which relies on the density-based clustering of the latent representation of trending hashtags. Afterward, we studied the weekly evolution of the detected topics, which is useful to understand how online discussion evolves over time.

We also leveraged IOM-NN to understand which candidate or party public opinion is most in favor of in the weeks preceding the election day. Differently from the use cases presented in the previous chapter, here we analyzed real-time data produced on Twitter during the 2020 US election campaign, correctly determining Joe Biden's lead over Donald Trump before election day. The achieved results, publicly available through our university web portal[1], represent a remarkable step forward with respect to previous works present in the literature. In fact, to the best of our knowledge, experimental evaluations are carried out *ex-post*, i.e., after the end of the considered event, while in our case we have given proof of the real-time effectiveness of IOM-NN, which leads to the possibility of using it to empower or even replace traditional opinion polls. The polarization analysis was further extended by focusing on the main swing states, i.e., those states characterized by a high uncertainty about the winning candidate and for this reason by marked strategic importance. The obtained results confirm the great effectiveness of our approach, which outperformed the average of the latest opinion polls by correctly identifying the leading candidate in 10 out of 11 swing states. Furthermore, the polarization information achieved by IOM-NN was also leveraged to investigate the temporal dynamics of social media conversation, with the aim of studying how users' publishing behavior is related to their political leaning, and how it reflected the occurrence of external events like debates or rallies.

Finally, we analyzed the relationship between the emotional sphere of Twitter users and their political alignment. In particular, we exploited sentiment analysis and text mining techniques for extracting the sentiment of social users. Then, we combined this information with the polarization achieved by IOM-NN to investigate how users express their preferences on social media, by modeling their political leaning with respect to a broad spectrum of emotions. Specifically, they can praise, as in the case of pro-Trump users, their favorite candidate with positive content that shows emotions like joy and confidence. Alternatively, they may tend to discredit the opposing candidate, as in the case of pro-Biden users, by producing negative online content characterized by emotions like anger and sadness.

[1] `https://www2.unical.it/portale/portaltemplates/view/view.cfm?103993` (text in Italian)

The remainder of the chapter is organized as follows. Section 3.1 describes the different techniques combined in the proposed analysis workflow. Section 3.2 presents the use case, providing an in-depth discussion of the achieved results. Finally, Section 3.3 concludes the chapter.

## 3.1 Analysis workflow

The workflow we designed, shown in Figure 3.1, combines several techniques, with the aim of outlining an accurate representation of the analyzed political event from different perspectives, including users' publishing behavior, discussion topics, political alignment, and its relationships with the emotional sphere.

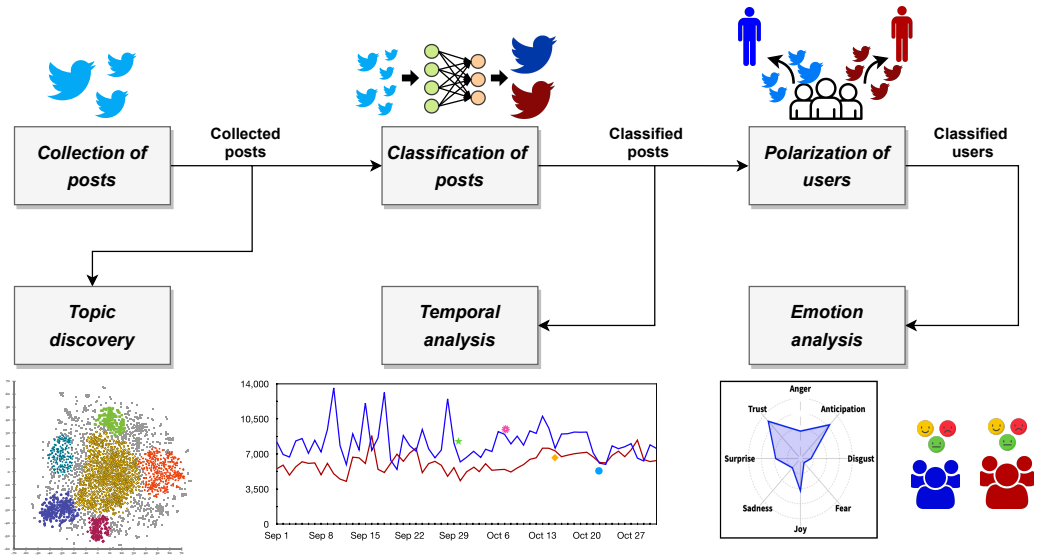



**Figure 3.1:** A graphic representation of the analysis workflow.

This workflow extends the one shown in the previous chapter (see Figure 2.1) by adding to IOM-NN three additional steps for the analysis of discussion topics, temporal dynamics, and user emotions, from a political perspective. In particular, its first three steps jointly constitute the IOM-NN methodology:

- *Collection of posts*: data are gathered from social media by using a set of keywords related to the considered political event.

- *Classification of posts*: the collected posts are classified in favor of a faction according to the detected political support.
- *Polarization of users*: the classified posts are analyzed for determining the polarization of users towards a faction.

The other steps, instead, make up the proposed extension and are presented in the following.

**Topic discovery.** In this step, the posts collected by IOM-NN (see Section 2.2.1) are used to identify the main politically-related discussion topics underlying the 2020 US election campaign on social media, by following the unsupervised approach we proposed in [9]. As a first step, a Word2Vec model is trained on the entire corpus of tweets, to get the latent representation of hashtags and words in a 150-dimensional embedding space. We selected the dimensionality of this space by conducting several experiments, finding out the smallest size for which a clear clustering structure emerged, i.e. the best trade-off between complexity and representativeness. Subsequently, all hashtags are embedded in this 150-dimensional space, whose dimensionality is then reduced by using the t-distributed stochastic neighbor embedding (t-SNE) technique, initialized through principal component analysis (PCA), to obtain a 2D projection of that space. Moreover, in order to reduce noise, all hashtags with a frequency lower than a given threshold are filtered out. Dimensionality reduction and noise filtering reduction bring a twofold benefit, making both the identification of the clustering structure within the embedding space and its visualization easier. Finally, the OPTICS algorithm is used for extracting the clustering structure originating from the topic-based separation of hashtags, induced by the projection of their semantic distribution. We have chosen this clustering algorithm due to its ability to discover clusters with arbitrary shapes at a selected density level, chosen against a reachability distance plot, which is useful to deal with hierarchical topic structures. We also experimented with the use of the HDBSCAN algorithm to deal with multi-density scenarios, analyzing micro-topics at different density levels.

**Temporal analysis.** In this step we leverage the posts classified by IOM-NN according to the detected political support (see Section 2.2.2), for investigating the relationship between users' publishing behavior and their political alignment. Specifically, temporal dynamics of social media conversation are analyzed, studying how online content is produced by the supporters of both candidates, and how this reflects the occurrence of external events such as debates and rallies.

**Emotion analysis.** In this step, we exploited the user classified by IOM-NN according to their political polarization (see Section 2.2.3), with the aim of modeling political support from an emotional viewpoint. In particular, we combined user polarization with the overall sentiment and emotions they expressed while talking about the two candidates. In order to extract the sentiment from online published content, we exploited SentiStrength [63], computing a positive $\mathcal{S}c^{+}(p)$ and negative $\mathcal{S}c^{-}(p)$ sentiment score, both ranging between 1 (neutral) and 5 (strongly positive/negative). Then, the overall sentiment score $\mathcal{S}c(p)$ of a polarized post was obtained as follows: $\mathcal{S}c(p) = \mathcal{S}c^{+}(p) - \mathcal{S}c^{-}(p)$. For detecting the emotions expressed by social users, instead, we used NRC-EmoLex [64], a publicly available emotion lexicon that has proven its performance in several sentiment and emotion classification tasks [65–67]. Specifically, NRC contains more than 14 thousand English terms labeled by the expressed polarity (i.e., positive or negative) and the eight basic emotion categories of Plutchik [68] (i.e., joy, trust, anticipation, sadness, surprise, disgust, fear, and anger). Through the joint use of the political leaning of users, provided by IOM-NN, and the extraction of sentiment and emotions from the contents they publish, it is possible to understand how users, in favor of a given faction, refer to the different candidates. Specifically, it is possible to investigate whether they follow a positive approach, supporting their preferred candidate, or proceed by discrediting the opposing candidate, showing a negative emotional profile.

## 3.2 Results and discussion

In this section, we provide an accurate description of the results coming from the analysis of the 2020 US presidential campaign, characterized by a strong rivalry between Joe Biden and Donald Trump. In particular, we analyzed election-related tweets with the aim of outlining a precise representation of this political event from different points of view, in terms of users' publishing behavior, sentiment, political alignment, and discussion topics. For this purpose, we combined several techniques in an analysis workflow, whose steps are accurately described in Section 3.1 and whose results are reported in the following sections.

### 3.2.1 Data description

The data used to perform the experimental evaluation comes from a public repository that contains a real-time collection of tweets related to the 2020 US presidential election from December 2019 to June 2021 [69]. From such repository we considered only the tweets published close to the election event (from 1 September to 31 October 2020), i.e., about 160 million of which 18 million are tweets (11%), 110 million are retweets (69%), and 32 million are replies (20%), posted by about 29 million users. Only 22% of filtered data contain hashtags (e.g., #*trump*2020, *#bidenharris2020*), useful to understand the arguments used in favor of the different candidates. In particular, the percentage of tweets published with at least one hashtag related to Trump (i.e., #*trump*, #*trump*2020, and #*maga*) and Biden (i.e., #*bidenharris*2020, #*biden*) is about 31% and 11%, respectively. However, 7% of tweets contain at least one negative hashtag about Trump (i.e., #*trumpknew*, #*pedotrump*, #*trumphascovid*, #*trumptaxreturns*), whereas only 1% of tweets contain a negative hashtag for Biden (i.e., #*crookedjoebiden*). In order to ensure the representativeness of the collected posts, we analyzed users' account information, filtering out content posted by users that show anomalous publishing activity or inconsistent profile information. This step allows for mitigating the negative effects caused by the presence of content published by new sites and social bots, which can introduce a heavy bias in social media data [7].

We further analyzed the publishing behavior of the users in the filtered dataset of election-related tweets by determining the Complementary Cumulative Density Function (CCDF) of shared tweets per user. Specifically, given the random variable $X$ representing the number of shared tweets, it is determined by the frequency of users publishing a number of posts greater than $x$, i.e. the probability $P(X > x)$. The scatter plot in log-scale shown in Figure 3.2, reveals a highly skewed distribution, which is a common pattern in social media platforms, with few active users posting a huge amount of posts and many users posting infrequently or not at all, the so-called social lurkers.

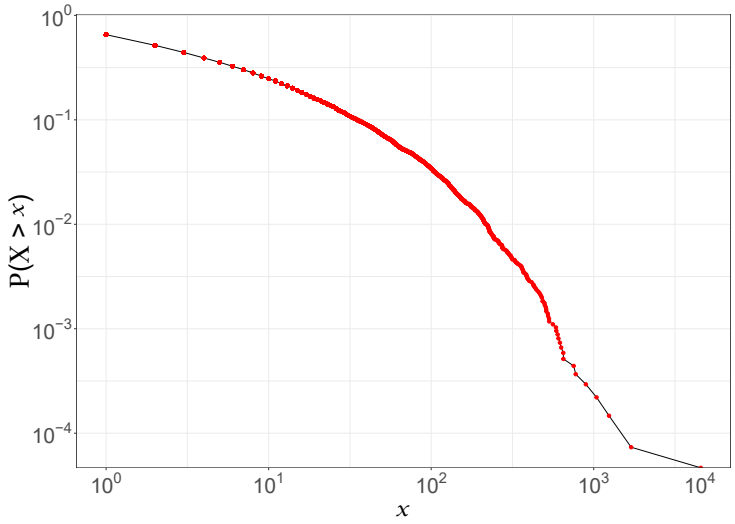



**Figure 3.2:** Complementary Cumulative Density Function (CCDF) of published tweets per user.

### Statistical significance analysis

As discussed in the previous chapters, the assessment of statistical significance is a crucial step, especially when analyzing data coming from social media platforms. Here we investigated user representativeness to understand to what extent they can be considered voters of the political event under analysis.

Firstly, from tweets metadata we extracted aggregate information on the used language of social media users, discovering that most tweets have the *lang* field

set to *English* (about 90%), whereas the remaining 10% is "undefined" or set to different languages. Secondly, we compared the number of Twitter users in our dataset, grouped by state, with the number of adult citizens actually living in that state, belonging to the voting-eligible population (VEP)[2]. Specifically, users were associated with states via Twitter metadata, by analyzing the *location* field present in each tweet, which indicates the location defined by the user in his/her Twitter account (e.g., Austin, TX). It is worth noting that, from the textual analysis of this field, it is not always easy to extract a meaningful city/state, as many users either left the field blank, or did not provide precise information (e.g. "USA"), or specified fictitious or non-existent locations (e.g. "the moon" or "NY, Italy"). We measured the strength of the relationship between analyzed users and the voting-eligible population through the Pearson correlation coefficient, finding a value $R = 0.97$, significant for $p < 0.01$. This relationship can also be easily seen in Figure 3.3, which depicts an interpolation of the related scatter plot, with a goodness-of-fit $R^2 = 0.93$. Notice that outlier states were not considered in this step in order to achieve meaningful results, by excluding data of different magnitudes.

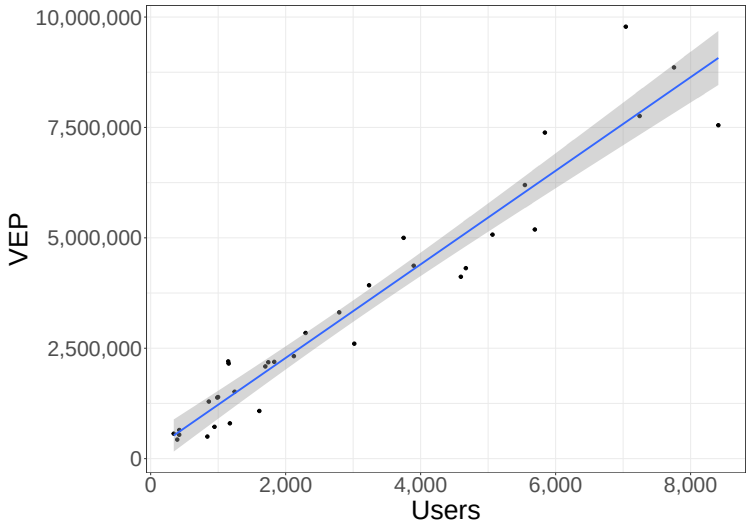



**Figure 3.3:** Linear interpolation: analyzed users vs. voting-eligible population grouped by US states.

[2]http://www.electproject.org/2020g

In addition, we explored the age and gender distribution of analyzed users, finding out that about 94% of them are adults (at least 18 years old)[3] and almost equally divided by gender (22% females and 25% males)[4]. Among all the available tweets, we selected those published by users located in the 11 main swing states (i.e., Arizona, Florida, Georgia, Michigan, Minnesota, Nevada, New Hampshire, North Carolina, Pennsylvania, Texas, and Wisconsin). We analyzed only these states as they are characterized by a marked political uncertainty and their outcomes have a high probability of being a decisive factor in the electoral event. We made these data, used in all the subsequent analysis steps, publicly available on Github[5]. Table 3.1 reports a comparison between the users we were able to capture for each swing state and the voting-eligible population.

| State | #Users | #VEP |
|---|---|---|
| Arizona | 5,692 | 5,189,000 |
| Florida | 16,921 | 15,551,739 |
| Georgia | 5,841 | 7,383,562 |
| Michigan | 8,411 | 7,550,147 |
| Minnesota | 4,596 | 4,118,462 |
| Nevada | 1,156 | 2,153,915 |
| New Hampshire | 1,610 | 1,079,434 |
| North Carolina | 7,245 | 7,759,051 |
| Pennsylvania | 7,040 | 9,781,976 |
| Texas | 19,119 | 18,784,280 |
| Wisconsin | 3,898 | 4,368,530 |

**Table 3.1:** Number of Twitter users vs. voting-eligible population grouped by swing states.

In conclusion, the high correlation between the number of analyzed users per state and the VEP leads to a significant set of social media data, effectively exploitable to determine the polarization of public opinion and extract useful insights on the analyzed political phenomenon. However, despite the representativeness of the considered posts, the results achieved by the analysis of the online conversa-

[3] https://www.statista.com/statistics/265647/share-of-us-internet-users-who-use-twitter-by-age-group/

[4] https://www.statista.com/statistics/265643/share-of-us-internet-users-who-use-twitter-by-gender/

[5] `https://github.com/SCAlabUnical/USA2020`

tion can be influenced by platform biases. Specifically, there may exist usage biases due to the distribution of users in terms of gender, age, culture, and social status, as well as technical biases related to data availability and restrictions imposed in some areas of the world.

### 3.2.2 Topic discovery

In this step, we identified the main politically-related discussion topics characterizing the 2020 US election campaign following the approach proposed in [9] and described in Section 3.1. Achieved results are shown in Figure 3.4, where 6 clusters are clearly visible, each one related to a different topic of discussion, as discussed in the following.

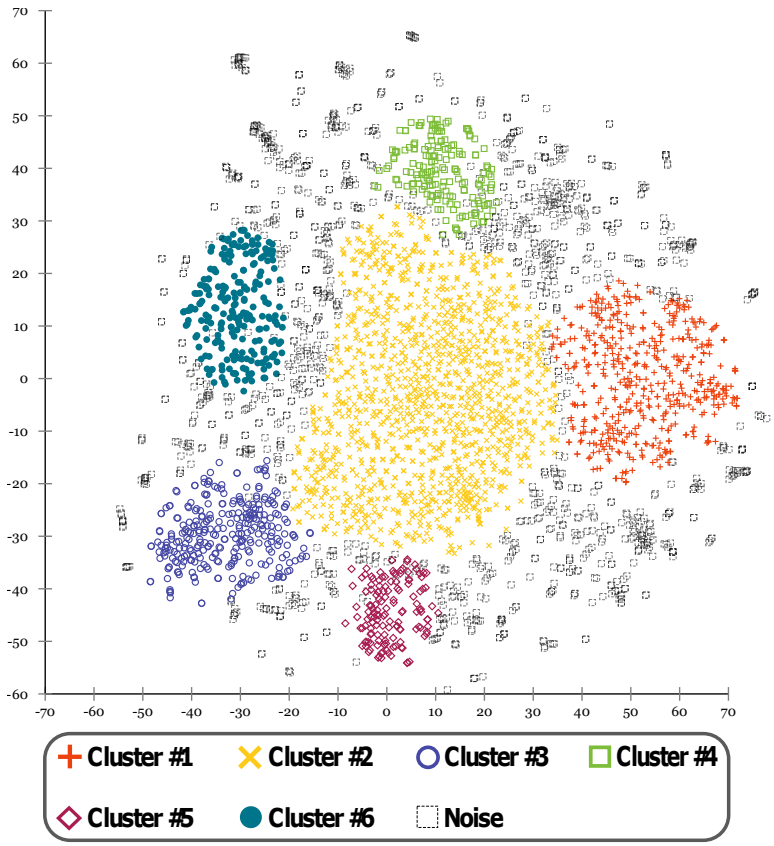



**Figure 3.4:** Unsupervised detection of the main topics underlying the online discussion.

The first topic is focused on the criticisms leveled at Donald Trump regarding the management of the health emergency in the United States caused by the Covid-19 pandemic. The second one is related to town hall meetings, covering different topics against Trump like discrimination, veterans, and climate crisis (e.g., he referred to climate change as a "*hoax*", and to veterans as "*human scum*"). The third one is a general topic about the presidential election. The fourth topic is related to the accusations of corruption and wrongdoing in regard to China and Ukraine leveled against Hunter Biden, i.e., the son of the democratic candidate Joe Biden. The fifth topic focuses on the nomination of the conservative Amy Coney Barrett for a seat on the Supreme Court as successor to the liberal Associate Justice Ruth Bader Ginsburg. Finally, the last topic is related to the online discussion of Trump's supporters, characterized by notorious hashtags like *#maga* and *#kag*. Discovered topics are summarized in Table 3.2 by reporting their corresponding top hashtags.

| Cluster ID | Topic | Top hashtags |
|---|---|---|
| #1 | Bad management of Covid-19 emergency | #trumpknew, #trumpvirus, #covid, #trumpisaloser, #trumpisanationaldisgrace, #trumpliedpeopledied |
| #2 | Town hall meetings; sub-topics: climate crisis, veterans, discrimination | #cnntownhall, #climatecrisis, #greennewdeal, #respectveterans, #hererightmatters, #stoptrumpsterror |
| #3 | Encouraging people to vote | #election2020, #voteearly, #vote2020, #votebymail, #voteready, #electionday |
| #4 | Accusations against Hunter Biden | #hunterbiden, #bidencrimefamily, #burisma, #ukraine, #hunterbidenemails, #china |
| #5 | US Supreme Court; nomination of Amy Coney Barrett | #scotus, #amyconeybarrett, #filltheseat, #supremecourt, #riprbg, #scotushearings |
| #6 | Support for Trump | #maga, #votetrump2020, #maga2020, #kag, #voteredtosaveamerica2020, #trumppence2020 |

**Table 3.2:** Brief description of the identified topics.

Once the major discussion topics were detected, we analyzed their overall impact on the online conversation related to the US election. Specifically, we calculated the volume of each hashtag-based topic by determining the percentage of tweets that contain hashtags belonging to the corresponding cluster. Considering

our overall observation period, the most relevant topic is about Covid-19 pandemic and it specifically refers to Trump's mismanagement of the health emergency. Other topics are related to the presidential election in general or arise from the publishing activity of Trump's supporters. Also, Biden's supporters significantly contributed to the online discussion, by leveraging anti-Trump sub-topics that have emerged from several town hall meetings, about discrimination, veterans, and the position of the Republican candidate about the climate crisis.

To get a deeper understanding of the impact of the extracted topic on social media conversation, we analyzed also their evolution in the eight weeks included in our observation period, as shown in Figure 3.5.

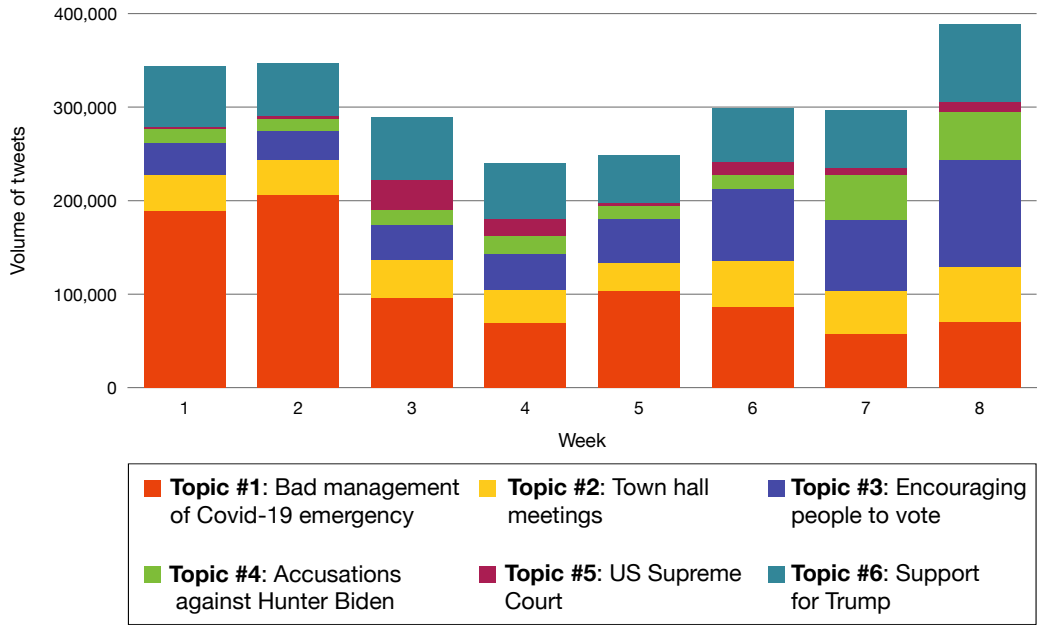



**Figure 3.5:** Weekly volume of tweets related to the detected topics from 1 September to 31 October 2020.

We found out that, in the early weeks, the online conversation focused on the relationship between Trump and Covid-19 pandemic. In addition, the discussion about the US Supreme Court showed a slight increase close to the nomination, announced by Donald Trump, of Judge Amy Coney Barrett as Associate Justice of the US Supreme Court to fill the vacancy left by the death of Ruth Bader Ginsburg. In the following weeks, the focus of the online conversation shifted to var-

ious topics related to the approach of election day and the importance of voting. We also observed an increase in the volume of tweets concerning the accusations leveled against Joe Biden's son (i.e., Hunter Biden), a topic discussed mostly by the Democratic candidate's detractors. Finally, other topics regarding the support voters expressed towards Trump and their criticisms leveled against him linked to town hall meetings showed an almost constant impact on the online conversation.

### 3.2.3 Temporal analysis

In this step we investigated the temporal dynamics of social media conversation, in order to analyze users' publishing behavior, studying how it is related to the detected polarity and how it reflected the occurrence of external events (e.g., debates or rallies). However, as described by the repository owners, there may be gaps in the dataset due to several issues. Firstly, the data collection step was highly contingent upon the stability of the network and hardware. Secondly, Twitter significantly limits the number of tweets that can be rehydrated. Finally, tweets may no longer be available as users have been removed, banned, or suspended [69].

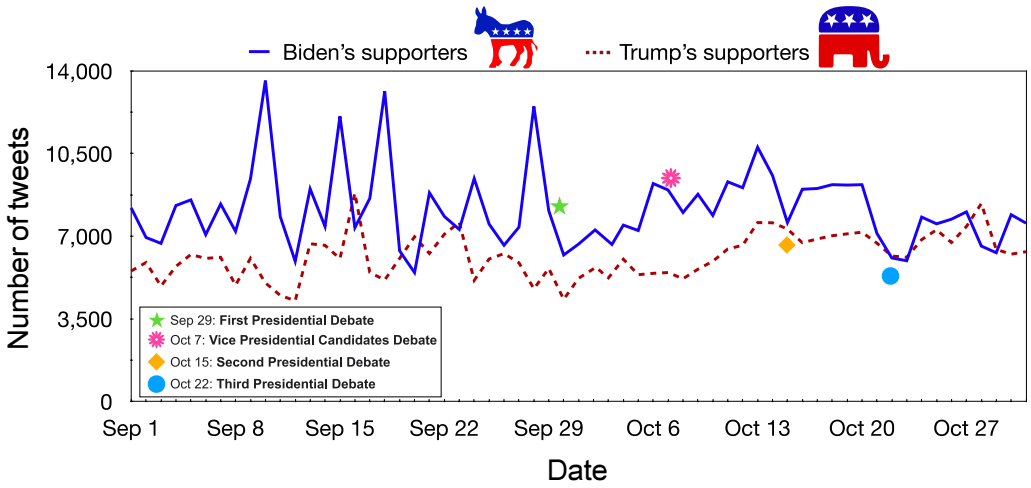



**Figure 3.6:** Time series of polarized tweets published from 1 September to 31 October 2020.

Figure 3.6 shows the timeline of polarized tweets volume annotated with the four main political debates occurring during the election campaign, i.e., between 1 September and 31 October. The first observation period (from 1 to 28 September) exhibits significantly different communication dynamics prior to the first debate. Interestingly, this image shows intense activity spikes of Biden's supporters, as a likely consequence of President Trump's actions:

- *10 September*: President Trump has attacked Democratic Vice Presidential candidate Kamala Harris.
- *15 September*: despite being banned by state authorities from holding rallies, President Trump still decided to hold one in Nevada.
- *18 September*: President Trump blamed blue states for the high number of US Covid-19 fatalities.
- *28 September*: during a rally in Pennsylvania, Trump called Biden "a dishonest politician and a puppet in the hands of the radical left".

The second and third observation windows (from 30 September to 31 October) show typical weekly cycles of social media chatter, with no particular explosion or shock-related spike from external events, except for the Vice Presidential debate (6 October) and the second Presidential debate (13 October).

### 3.2.4 Polarization analysis

In this step, we used IOM-NN to understand which candidate or party public opinion is most in favor of. As introduced at the beginning of this chapter, a remarkable result was obtained by analyzing real-time data collected during the two weeks before election day. Specifically, IOM-NN was able to determine Joe Biden's lead over Donald Trump, also correctly predicting the winning candidate in different US states, including Georgia, where a Democratic candidate had not won since 1992 with the election of Bill Clinton. This result, publicly available

through our university web portal[6], represents a step forward with respect to techniques in the literature, including our previous work, as it gives clear proof of the real-time effectiveness of IOM-NN. For what concerns the polarization analysis we carried out on the main swing states, we compared the estimates produced by IOM-NN with the average values of the latest opinion polls before the election[7]. Achieved results are reported in Table 3.3, in which the two candidates (i.e., Joe Biden and Donald Trump) are indicated with "B" and "T", respectively, and the winning candidate is written in bold when it is correctly identified. The results of the comparison are also summarized in Figure 3.7.

| **State** | **Real percentages** | | **Opinion polls** | | **IOM-NN** | |
|---|---|---|---|---|---|---|
| | ***B*** | ***T*** | ***B*** | ***T*** | ***B*** | ***T*** |
| Arizona | 49.4 | 49.1 | **48.0** | 45.8 | **50.2** | 48.3 |
| Florida | 47.9 | 51.2 | 48.7 | 46.0 | 48.0 | **51.1** |
| Georgia | 49.5 | 49.2 | **47.6** | 47.4 | **52.7** | 46.0 |
| Michigan | 50.6 | 47.8 | **49.9** | 44.4 | **55.4** | 43.0 |
| Minnesota | 52.4 | 45.3 | **51.6** | 41.8 | **55.1** | 42.6 |
| Nevada | 50.1 | 47.7 | **49.4** | 44.4 | **49.8** | 48.0 |
| New Hampshire | 52.7 | 45.4 | **53.4** | 42.4 | **50.9** | 47.3 |
| North Carolina | 48.6 | 49.9 | 47.8 | 47.5 | 56.6 | 41.9 |
| Pennsylvania | 50.0 | 48.8 | **49.4** | 45.7 | **55.7** | 43.1 |
| Texas | 46.5 | 52.1 | 47.5 | **48.8** | 46.1 | **52.5** |
| Wisconsin | 49.4 | 48.8 | **52.0** | 42.8 | **56.3** | 41.9 |
| Correctly classified | - | | **9/11** | | **10/11** | |
| Tweets | - | | - | | 670,451 | |
| Users | - | | $\approx$ 11,000 | | 57,116 | |
| Avg. Acc | - | | 0.82 | | 0.91 | |

**Table 3.3:** Comparison between voting percentages estimated by IOM-NN and the latest opinion polls.

Figure 3.7 shows that the estimates achieved by IOM-NN, related to the voting intentions of social media users, are more in line with the actual behaviors of voters with respect to the opinion polls, thus giving a clue to the final result in 10 out of 11 swing states with an average accuracy of 91%.

[6] `https://www2.unical.it/portale/portaltemplates/view/view.cfm?103993` (text in Italian)

[7] `https://www.270towin.com/2020-polls-biden-trump/`

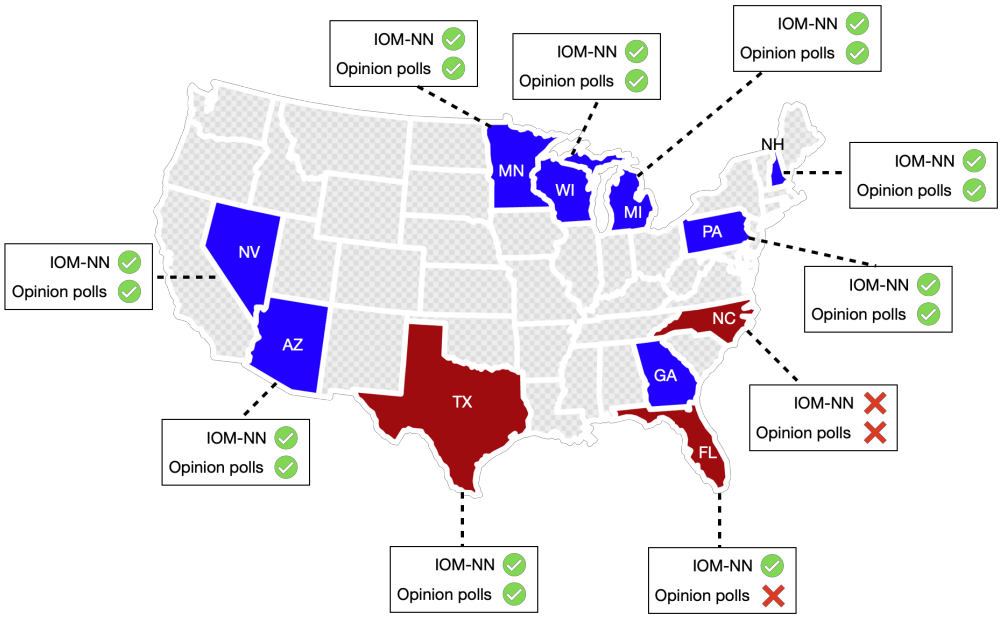



**Figure 3.7:** Comparison between IOM-NN and the latest opinion polls in identifying the winning candidate.

Using this metric we penalize the inversions of polarity which can be a crucial issue while analyzing these kinds of states characterized by a high degree of uncertainty. Notice that, for what concerns North Carolina, neither the estimates achieved by IOM-NN nor the opinion polls were in line with the actual outcome in this state. This is a common situation, as the results achieved by the polls and IOM-NN must be understood as an estimate of the ongoing polarization of public opinion in the weeks preceding the election day, not always in accordance with the actual behavior of voters. Moreover, a noteworthy advantage of IOM-NN with respect to traditional opinion polls is the ability to capture the opinion of a larger number of people more quickly and at a lower cost, even in relation to topics perceived as embarrassing or sensitive, on which people tend to discuss and express their opinions on social networks more freely than they do during polls. This makes IOM-NN a valid tool to support, enhance or even replace opinion polls, by providing relevant insights useful to understand the dynamics of the election campaign.

### 3.2.5 Emotion analysis

The goal of this last step is to model the political orientation of Twitter users from an emotional point of view. For this purpose, we first used SentiStrength for discovering the existing relationships between the political polarization of users and the sentiment expressed in referring to the two presidential candidates (positive and negative). Then, we extended this analysis with respect to a broader spectrum of emotions, including anger, anticipation, disgust, fear, joy, sadness, surprise, and trust. In this way, we achieved a full emotional profile of pro-Trump and pro-Biden users, in relation to the subject of their posts and their political polarization, previously discovered by IOM-NN. From the achieved results, shown in Figure 3.8 and 3.9, emerges that, on average, the tweets produced by Trump's supporters are significantly more positive than those produced by Biden's supporters, which devote a significant number of negative tweets to their opponent.

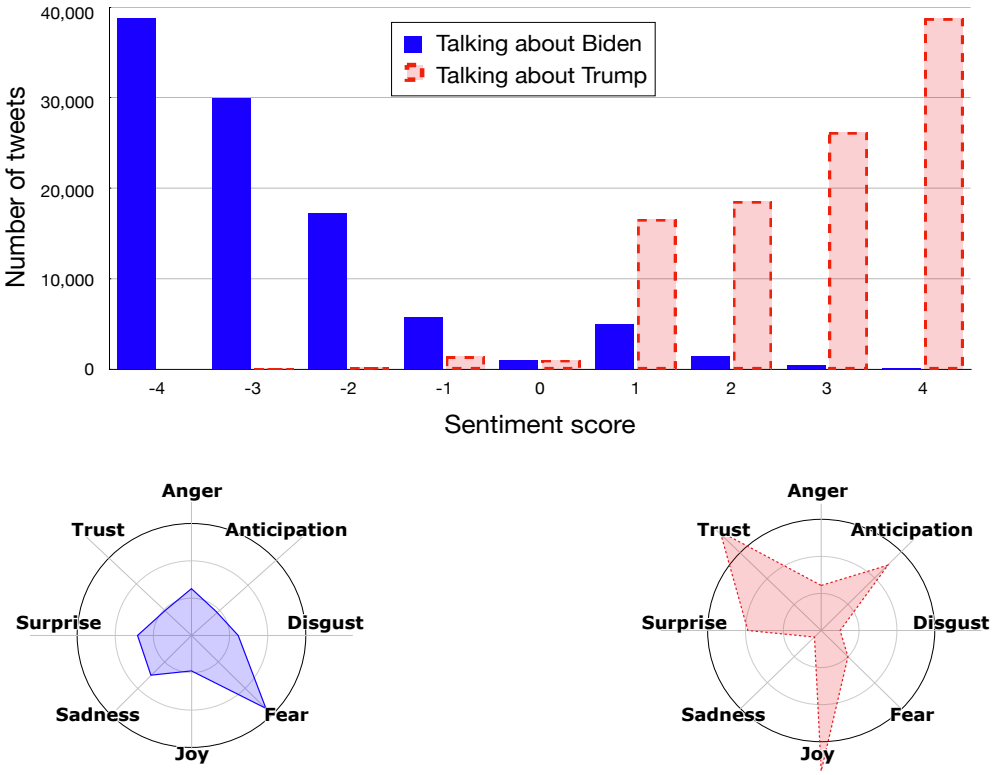



**Figure 3.8:** Distribution of sentiments and emotions of pro-Trump tweets.

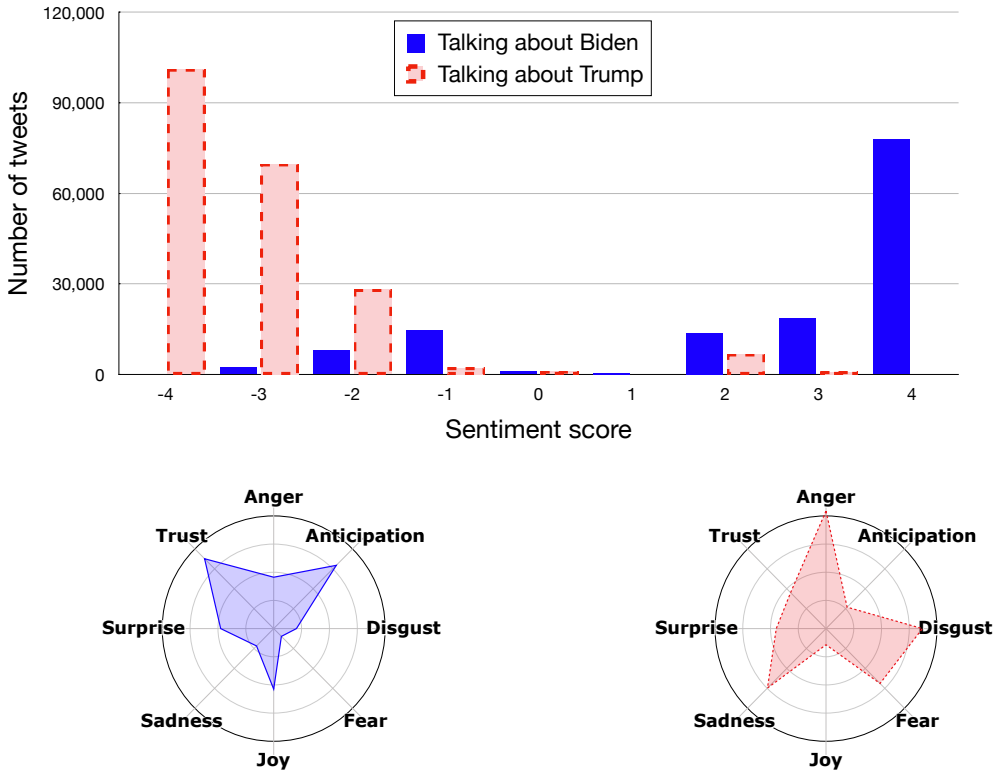



**Figure 3.9:** Distribution of sentiments and emotions of pro-Biden tweets.

For what concerns the detected emotions, Trump's supporters express *joy* and *confidence* about Trump, while *fear* about Biden's election. Biden's supporters, instead, show *trust* and *anticipation* in having Biden as the future president of the United States, with a more marked presence of negative emotions about Trump, like *anger*, *disgust* and *sadness*. Table 3.4 presents the results achieved on some example tweets included in the analysis.

| Tweet | About | Sentiment | Emotion |
|---|---|---|---|
| "#realDonaldTrump If anyone can do it, you can. Best President ever! #AmericaFirst #MAGA2020" | Trump | Positive | Trust |
| "#realDonaldTrump You are a racist and a loser. #TrumpIsALoser #RacistTrump" | Trump | Negative | Disgust |

**Table 3.4:** A sample of tweets showing different emotions.

## 3.3 Conclusions

This chapter presented an in-depth analysis of the posts published on Twitter during the 2020 US election campaign, in which different techniques for topic discovery, opinion mining, and emotion analysis are combined in a unified analysis workflow, which extends IOM-NN with the final aim of outlining an accurate representation of the political event from different viewpoints. In particular, we extracted the main discussion topics following a clustering-based approach, monitoring their weekly impact on the online conversation. We also leveraged IOM-NN to estimate the political polarization of Twitter users, both in real-time and by focusing on the main swing states. We investigated the temporal dynamics of the online discussion, combining it with the polarization information coming from IOM-NN, in order to study how users' publishing behavior reflected election-related external events. Finally, we exploited sentiment analysis and text mining techniques to discover the relationship between user polarization, determined by IOM-NN, and the sentiment expressed in referring to the different candidates, thus modeling the political support of Twitter users from an emotional viewpoint.

Experimental evaluation shows that in the early weeks the online conversation focused on the relationship between Trump and the Covid-19 pandemic and on the nomination of Judge Amy Coney Barrett as Associate Justice of the US Supreme Court. In the following weeks, instead, the focus shifted to other topics including the accusations leveled at Hunter Biden and the criticism against Trump linked to his position about the climate crisis and veterans. Regarding the political polarization of public opinion, IOM-NN was able to achieve meaningful estimates of the voting intentions of social media users, which makes it a valid alternative to go beyond traditional opinion polls. Finally, as for the analysis of the emotional state of social users, we found out that the tweets produced by Trump's supporters are significantly more positive than those produced by Biden ones. In particular, *i*) Trump's supporters express joy and confidence about Trump, while fear about Biden's election; *ii*) Biden's supporters show trust and anticipation in having Biden as the future president of the United States, with a more marked presence of negative emotions about Trump, like anger, disgust, and sadness.

Chapter 4

# Analyzing political polarization by deleting bot spamming

In the previous chapters, we have shown how data from social networks are a powerful tool for analyzing political phenomena from a wide range of perspectives. However, as stated in Section 1.1, the effectiveness of Big Social Data analysis techniques heavily depends on the reliability of the data gathered from social media. Unfortunately, these data are easily manipulated through spamming activities, misinformation campaigns, and the spread of fake and malicious content. In such a scenario, getting reliable and impartial data, discerning them from rumors, constructed reports, and fake news can be a quite challenging task. Recent studies have shown that social bots are among the factors that most undermine the reliability of online news. These are algorithmically-driven entities that appear as legitimate users by imitating human behavior, with the aim of altering the popularity of users and influencing discussions of any kind, including political issues [18]. Another issue that we highlighted in Section 1.1 concerns the high dynamicity of social media data, which causes the patterns of interest to change rapidly over time. Due to this, time-related aspects are key to a correct understanding of the extracted information. Regarding the political field, for example, the voting intentions of social users can fluctuate during an election campaign, independently or in response to external events.

Starting from the above considerations, this chapter presents *TIMBRE* (*Time-aware opInion Mining via Bot REmoval*) [7], a methodology that we specially designed to address these issues, aimed at measuring the polarization of social users during an election event, estimating its final outcome. This methodology follows a keyword-based approach to detect the political polarization of users, also considering time-related aspects. Specifically, for each classified post, it computes an importance weight that represents its relevance to the voting intentions of the user who published it. The idea behind this temporally-aware polarization process is that preferences expressed by social users closer to election day are more likely to reflect their true voting intentions. Another key aspect of TIMBRE is the bot removal step, aimed at avoiding the distortion effect introduced by the presence of automatically-generated data. Specifically, our methodology filters out the data produced by social bots, identifying them through the use of the *Botometer* [70] framework. Therefore, by jointly exploiting a bot detection system and a temporally-aware polarization technique, TIMBRE is able to accurately detect the real voting intentions on social media platforms, capturing only the polarization of legitimate users that belong to the voting-eligible population. To the best of our knowledge, this work is one of the few in the literature that focuses on the study of the joint influence of bots and temporal aspects in the analysis of electoral results.

To test the effectiveness of TIMBRE we applied it to a real-world case study that analyzes the polarization of a large number of Twitter users during the 2016 US presidential elections, which was characterized by the rivalry between Hillary Clinton and Donald Trump. This use case is particularly interesting since it was characterized by a marked use of Twitter to foster political debate along with a significant activity by social bots, which would have strongly influenced voter decisions [18, 71, 72]. As an example, in [18] authors analyzed the pervasive presence and activity of social bots involved in social media conversation related to the 2016 US presidential election. They found out that about 400,000 bots were engaged in the political discussion, responsible for roughly 3.8 million tweets. Our analysis focused in particular on the analysis of the main US Swing States, characterized by great political uncertainty. We evaluated the benefits brought by the temporal

weighting and bot removal steps, through an ablation analysis, finding that both are crucial in getting a correct estimate of users' voting intentions. As a last step, we studied how the presence of social bots may have affected political discussion around the 2016 US presidential election, focusing on two main aspects. On the one hand, we analyzed the publishing behavior of both real users and social bots, along with the differences between human and artificial political support. On the other hand, we exploited a competitive diffusion model to estimate the degree of influence of social bots on legitimate users.

The remainder of the chapter is organized as follows. Section 4.1 reviews the main social bot detection techniques present in the literature. Section 4.2 describes the proposed methodology. Section 4.3 presents the case study and obtained results. Finally, Section 4.4 concludes the chapter.

## 4.1 Social bot detection techniques

The last few years have been characterized by a marked growth of social media legitimate use and manipulation, fostering democratic conversation about socio-political issues [18] and, at the same time, a large spread of misinformation. This phenomenon has made social platforms one of the most used sources of information, exposing users to risks caused by the lack of veracity of the news. Moreover, political online discussion is often strongly polarized, leading to the formation of *echo chambers* that provide selective exposure to news sources, thus biasing the opinion of users. This effect sometimes is amplified by the priority policies of the main social media platforms, which tend to favor engaging rather than trustworthy posts [73]. In such a setting, distinguishing trustworthy and unbiased news from rumors and fake news may be difficult. In this regard, *social bots*, also known as sybil accounts, pose one of the most serious threats to the reliability of online available content. They can be defined as algorithmically-driven entities that automatically produce content and interact with humans on social media, trying to emulate and alter their behavior. In a political scenario, bots can be used illicitly to artificially increase the support for a candidate, influencing the outcome of the

election. Campaigns of this type are usually called *astroturf* or *Twitter bombs*. Many efforts were made by the research community towards developing social bot detection and classification systems, especially on Twitter, one of the most used microblogging platforms. According to [74], state-of-the-art techniques can be categorized into three classes, as discussed in the following.

**Graph-based detection.** Methods in this category exploit a graph-based representation of a social network to understand the relationships between edges or links across accounts, using this information for detecting bot activity. As described in [75], there are three main graph-based approaches aimed at detecting social bots and malicious accounts: *i*) *trust propagation* that quantifies the strength of the relationship among users; *ii*) *graph clustering* groups similar users according to their characteristics. *iii*) *graph analysis* that relies on several metrics and properties of the social graph, like degree distribution and centrality measures. SybilWalk [76] is a sybil detection method that exploits a random walk-based method on an undirected social graph. It proceeds by assigning a score to users in the social graph, which is then used to classify them as legitimate users or sybils. Mehrotra et al. [77] proposed a supervised method for fake followers detection based on several centrality measures and the Random Forest classification algorithm.

**Crowdsourcing.** This class of methods leverages human detection to identify social bot behaviors, seeking patterns across profile information or shared content. As an example, DARPA held a Twitter bot challenge competition [78] in which teams were asked to identify influential bots that supported pro-vaccination discussions on Twitter. A common use of human annotation in bot detection involves the generation of annotated datasets, which can be then used by supervised techniques. In [79] four annotators were employed for the classification of Twitter profiles as humans or bots, starting from a wide range of features such as the number of tweets or favorites. Similarly, in [80] ten volunteers were tasked with labeling 2000 random accounts, in order to build a ground truth dataset.

**Machine learning.** These methods are based on machine learning algorithms and statistical techniques for social bot detection. Kantepe et al. [81] proposed a supervised approach that relies on an extensive process of feature extraction. In particular, they used Apache Spark for data collection, categorizing features into three types, i.e. user, tweet, and periodic features. Afterward, users were labeled as humans or bots through a gradient-boosting classifier. Devis et al. [70] proposed Botometer (formerly BotOrNot), a classification system that leverages more than one thousand features to evaluate the extent to which a Twitter account exhibits similarity to the known characteristics of social bots. Specifically, such features are extracted from available metadata, shared content, and interaction patterns. Ersahin et al. [82] presented a supervised method for fake account detection on Twitter which leverages a naive Bayes classifier and an entropy minimization discretization technique. Cai et al. [83] proposed a behavior-enhanced deep learning model (BeDM) for social bot detection. In particular, they jointly exploited a convolutional neural network and a long short-term memory network to capture temporal patterns in user behavior.

## 4.2 Proposed methodology: TIMBRE

TIMBRE is a temporally-aware methodology that exploits a keyword-based classification for determining the political polarization of social media users and the Botometer framework to distinguish legitimate users (i.e., voters) from social bots. Its final aim is to forecast the election results by discerning the political leaning of legitimate users, analyzing also how the presence of social bots may have affected politically-related online discussion, potentially altering public opinion.

Given a political event $\mathcal{E}$, a set of the factions $F$, and a set the keywords $K$ associated to $\mathcal{E}$, the proposed methodology consists of four main steps:

1. *Post collection*: posts are collected by using the set of keywords $K$ related to the political event $\mathcal{E}$.

2. *Post classification and weighting*: for each post we determine its political orientation, *neutral* or in favor of a specific faction $f \in F$, and a weight $w_p^u$ indicating the importance of the post $p$ in estimating the voting intentions of the user $u$ who published it.

3. *User polarization and classification*: starting from classified posts and related weights, we determine the political leaning of each user in our dataset, classifying it as a real user or a social bot. This information is then used to forecast the outcome of the event $\mathcal{E}$.

4. *Bot influence analysis*: during this step we analyze information production patterns, estimating also the degree of influence of social bots on real users.

Figure 4.1 shows a graphical representation of the main steps of TIMBRE introduced above, which will be discussed in detail in the following sections.

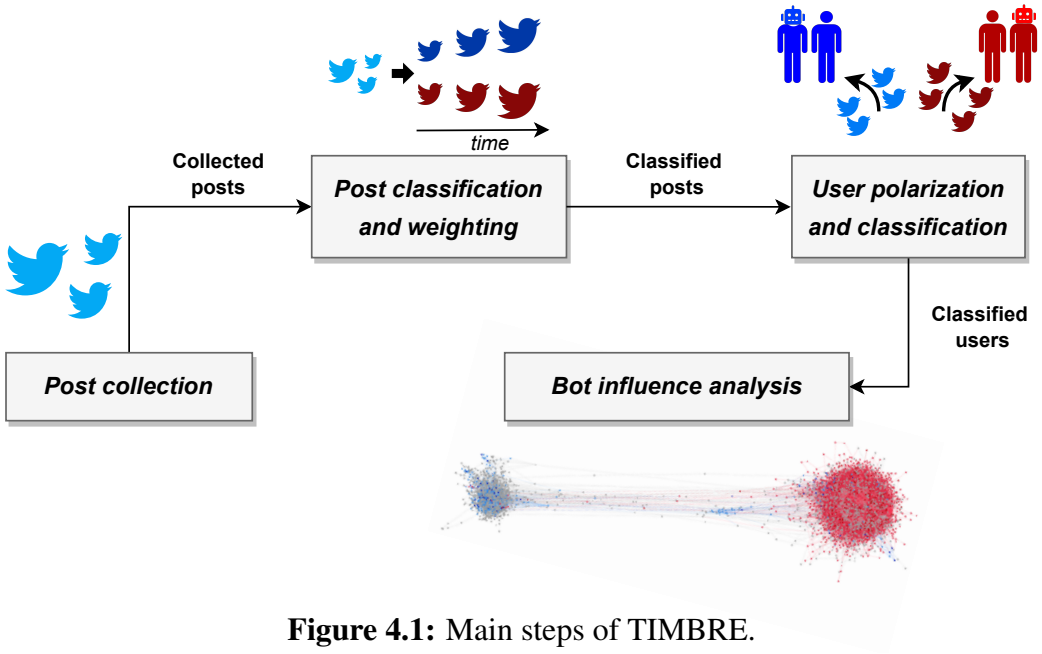



**Figure 4.1:** Main steps of TIMBRE.

### 4.2.1 Post collection

A political event $\mathcal{E}$ is characterized by the rivalry of different parties or factions $F = \{f_1, f_2, ..., f_n\}$. Following the same approach we used in [1], posts are

collected by using the keywords that people commonly use to refer to the political event $\mathcal{E}$ on social media. Such keywords $K$ are divided into two classes:

- $K_{neutral}$, which contains generic keywords that can be associated with $\mathcal{E}$ without referring to any specific faction in $F$.
- $K_F = K_{f1} \cup ... \cup K_{fn}$, where $K_{fi}$ contains the keywords used for supporting the faction $f_i \in F$.

The keywords in $K$ are given as input to public APIs provided by social media platforms, which permit collecting posts containing one or more keywords. Moreover, since data collection is usually a continuous process, new keywords can be discovered and integrated into $K$ during the collection procedure. The collected posts are pre-processed before the analysis as follows:

- Hashtags are normalized removing non-alphanumerical characters and transforming them to lowercase. This way we can avoid differences between different versions of the same hashtag, e.g., *#voteTrump*, *#vote_trump*, or *#votetrump!* becomes *#votetrump*.
- Data representativeness is further improved by filtering out all the posts having a language different from the one spoken in the nation hosting the considered political event.

As the proposed method relies on a hashtag-based analysis without exploiting other textual information, no further preprocessing like stopwords removal or lemmatization is needed. The output of this step is a collection of posts $P$ related to the event $\mathcal{E}$.

### 4.2.2 Post classification and weighting

In this step, we assign each post included in $P$ to a specific faction in $F$ by analyzing the keywords it contains. Moreover, we determine a weight $w_p^u$ indicating the importance of the post $p$ in estimating the voting intentions of the user $u$ who published it. The intuition behind this is that more recent posts are more suited

for deriving useful information about the voting intentions of a user. In fact, the polarization of users can vary over time as they can influence each other or be influenced by external events, such as political debates or scandals. Algorithm 3 shows the pseudo-code of the classification procedure, whose output $S$ consists of a set of triple containing the post $p$, the faction $f_p$ associated with it, and its importance weight $w_p^u$.

**ALGORITHM 3:** Post classification and weighting

**Input :** Set of posts $P$, set of faction keyword $K_F$, decay rate $\lambda$
**Output:** Set of Classified posts $S$

$S \leftarrow \emptyset$;
/* Given the post $p$, $v_F$ is a binary vector containing a 1 in position $f \in F$, if $p$ contains a keyword in $K_F$ (i.e., $K_p \cap K_F \neq \emptyset$) */
**for** $p \in P$ **do**
  $v_F \leftarrow [\,]$; // the vector of candidate factions to which the post $p$ can be assigned.
  **for** $f \in F$ **do**
    **if** $K_p \cap K_F \neq \emptyset$ **then**
      $v_F[f] \leftarrow 1$;
  /* The post $p$ is assigned to the faction $f_p \in F$ if it contains only keywords in favor of that faction (i.e., $sum(v_F) = 1$) */
  **if** $sum(v_F) = 1$ **then**
    $f_p \leftarrow argmax(v_F)$; // the faction to which the post $p$ is assigned.
    $u \leftarrow p.user$; // the user who wrote the post $p$.
    $d \leftarrow p.day$; // the day in which $p$ was written.
    $P^u \leftarrow \{\bar{p} \in P \mid \bar{p}.user = u\}$; // the set of posts written by $u$.
    $d^u_{max} \leftarrow max_{\bar{p}.day} P^u$; // the day user $u$ published his/her last post.
    $\delta_p \leftarrow d^u_{max} - d$; // the distance between $d^u_{max}$ and $d$ measured in days.
    $w^u_p \leftarrow e^{-\lambda \delta_p}$; // the importance weight assigned to $p$.
    $S \leftarrow S \cup \langle p, f_p, w^u_p \rangle$;
**return** $S$

As shown by the classification algorithm, a post $p$ is assigned to a faction $f$ only if it contains keywords that are exclusively in favor of that faction; otherwise, $p$ is classified as neutral. This is a very strict and conservative partisanship assignment, which leads to a small but high-confidence annotated dataset, likely less prone to misclassification than automatic machine-learning techniques. As regards the importance weight $w_p^u$, it is computed as follows. Given a user $u \in U$ and the set of his/her posts $P^u$, we determine $d_{max}^u$ as the day the user $u$ published his/her last post $p \in P^u$ before the end of $\mathcal{E}$. Given a post $p$ published by user $u$ the day $d$, and $\delta_p = d_{max}^u - d$, we define the importance weight as $w_p^u = e^{-\lambda \delta_p}$. This weight undergoes exponential decay according to a constant $\lambda$ (*decay rate*): larger values of this constant make the quantity vanish much more rapidly.

### 4.2.3 User polarization and classification

Starting from the set $S$ containing classified and weighted posts, we use a *one-vs-all* strategy to determine the political leaning of each user in our dataset. Specifically, given the set of opposing factions $F$ and a user $u \in U$, let $P^u$ be the set containing all of his/her posts, and $P_f^u \subseteq P^u$ the subset containing only post published by $u$ classified as in favor of $f$. For each faction $f$, we determine the support of $u$ towards $f$ as:

$$s_f^u = 2 \times \frac{\sum_{p \in P_f^u} w_p^u}{\sum_{p \in P^u} w_p^u} - 1$$

As the above formula is normalized in the interval $[-1, 1]$, positive values of $s_f^u$ means that user $u$ tends to be polarized towards the faction $f$, and the polarization becomes stronger as $s_f^u$ approaches the value of 1. Negative values, instead, suggest a polarization towards the set of all the remaining factions. Therefore, given a threshold $th$, political partisanship $f^u$ of $u$ is determined as follows:

- $f^u \leftarrow argmax(s_f^u)$, if $max(s_f^u) \geqslant th$
- $f^u \leftarrow neutral$ otherwise

Besides determining user partisanship, we also exploited the Botometer framework for the automatic classification of social media users into real or fake accounts, related to potential electors and automatic entities respectively. Given a user $u$ Botometer determines a real-valued score $s \in [0,1]$ which measures the likelihood that user $u$ is a social bot. According to prior studies [18, 70], we selected a threshold value for $l$ equal to 0.5, for the classification process. At the end of the entire procedure, two dictionaries $B$ and $R$ are obtained, related to bots and real users respectively, composed by $\langle u, f^u \rangle$ key-value pairs.

**ALGORITHM 4:** User polarization and classification

**Input :** Set $S$ of triples $\langle p, f_p, w_p^u \rangle$, set of users $U$, threshold $th$, set of factions $F = \{f_1, f_2, ..., f_n\}$, function $bot\_score : U \rightarrow [0,1]$ from Botometer which computes the likelihood $l$ for the user $u$

**Output:** Dictionary $B$ of polarized bots, dictionary $R$ of polarized real users

$W \leftarrow \varnothing$;
**for** $\langle p, f_p, w_p^u \rangle \in S$ **do**
    /* Compute the sum of the importance weights of posts grouped by the corresponding faction $f_p$ and user $u$. */
    $W[f_p, u] \leftarrow W[f_p, u] + w_p^u$;
$B \leftarrow \varnothing$;
$R \leftarrow \varnothing$;
**for** $u \in U$ **do**
    **for** $f \in F$ **do**
        $s_f^u \leftarrow 2 \times \frac{W[f,u]}{\sum_{f' \in F} W[f',u]} - 1$;  // polarization of user $u$ toward $f$
    **if** $max(s_f^u) \geqslant th$ **then**
        $f^u \leftarrow argmax(s_f^u)$;
    **else**
        $f^u \leftarrow neutral$;
    **if** $bot\_score(u) \geqslant 0.5$ **then**
        $B \leftarrow B \cup \langle u, f^u \rangle$;
    **else**
        $R \leftarrow R \cup \langle u, f^u \rangle$;
**return** $B$, $R$

Once the *user polarization and classification* step is completed, the outcome of the political event $\mathcal{E}$ can be determined starting from the $R$ set, containing the

polarity of legitimate users. Let $R_f$ be the subset of $R$ containing all legitimate users polarized in favor of $f$; the final consensus $c_f$ for each faction $f \in F$ is determined as follows:

$$c_f = \frac{|R_f|}{\sum_{f \in F} |R_f|}$$

### 4.2.4 Bot influence analysis

During this step, we analyze how the presence of social media bots may affect political discussion around the event $\mathcal{E}$ under analysis. After having built the set $P$ of classified posts and the sets $R$ and $B$, which specify the political leaning of bots and real users, the proposed methodology analyzes them by exploiting different algorithms and techniques, focusing on the following aspects.

- *Information production patterns*. During this step, the publishing behavior of both real users and social bots is analyzed, focusing on the differences between human and artificial political support.

- *Influence spread*. This step is aimed at estimating the degree of influence of social bots, clustered according to their partisanship, on real social users. To achieve that, TIMBRE builds a graph based on repost relationships, analyzing the spread of influence through a competitive version of the Linear Threshold diffusion model. Specifically, we adapted the *Separated-Threshold Model for Competing Technologies* [84] to our purposes, as described below.

First of all, we built the repost graph $G = (V, E)$, a directed graph where $V \subseteq B \cup R$ is the set of bots and real users involved in repost relationships and $E$ is the set of edges $(u, v)$ where $v$ reposted $u$, with $u, v \in V$. For each edge $(u, v) \in E$ we assigned a unique real-valued weight $w_{u,v}$ corresponding to the impact of node $u$ on $v$, computed as follows. Let $N_{u,v}$ be the number of times node $v$ reposted $u$ and $N_u$ the number of total reposts made by $v$; the weight of the edge $(u, v)$ is defined as: $w_{u,v} = \frac{N_{u,v}}{N_u}$, with $w_{u,v} \in (0, 1]$. Therefore, a node $u$ has a high influence on $v$ if $v$ shows a high tendency in reposting $u$'s posts more than the others.

Once the network is built, given the set $F = \{f_1, f_2, ..., f_n\}$ of factions involved in the political event $\mathcal{E}$, and the set of polarized bots $B \subseteq V$, we partitioned this set in $n$ disjoint subsets $B_1, B_2, ..., B_n$, such as $B_f$ contains only social bots polarized towards the faction $f$. For remaining users (i.e., neutral bots and real users $\in R \subseteq V$), a threshold value $\theta_f^u$ for each faction is selected, picked uniformly at random in the interval $[0, 1]$, representing the resistance of user $u$ to be influenced in favor of the faction $f$. At the step $t$, for each faction $f \in F$, let $I_f^{t-1}$ be the set of nodes influenced by faction $f$. During this step, a neutral node $v$ becomes polarized towards $f$ if $\sum_{u \in I_f^{t-1}} w_{u,v} \geqslant \theta_f^v$, which means that the influence exercised on $v$ in favor of $f$ is higher than its resistance to that faction. If for the node $v$ more than one threshold is exceeded during step $t$, then this node will be polarized in favor of the faction that exercises the highest influence. This process ends when all neutral nodes become influenced, returning $n$ disjoint sets, containing the users (both real and bot) polarized towards one of the factions and an additional set containing unpolarized nodes.

## 4.3 Results and discussion

In the following, we discuss a case study related to the 2016 US presidential election characterized by the rivalry between Hillary Clinton and Donald Trump. Our analysis focused on 10 US Swing States: Colorado, Florida, Iowa, Michigan, Ohio, New Hampshire, North Carolina, Pennsylvania, Virginia, and Wisconsin. These states are deemed of significant strategic importance because of their considerable political unpredictability. Therefore, manipulating information and amplifying propaganda in those states, thus influencing the political leaning of social users, can have significant effects on the election outcome.

As explained in Section 4.2.1, posts were collected using a set of neutral keywords and two sets of faction keywords, one for each candidate. An extract of these sets is shown in the following:

- $K_{Neutral}$={#election2016, #uselection, #earlyvote, #ivoted, ... }

- $K_{Hillary}$={#voteblue, #imwithher, #nevertrump, #strongertogether, ... }
- $K_{Trump}$={#votetrump, #maga, #neverhillary, #podestaemails, ... }

We analyzed about 4.7 million posts posted by 1.5 million users, finding a non-negligible impact of social bots on political discussion. As shown in Table 4.1, states like Colorado, Iowa, and Ohio, are characterized by a high rate of bot posts, from 20.6% to 24.6%. Furthermore, 7% of total user accounts have been identified as social bots, which produced about 15% of the total posts related to the 2016 US presidential election coming from the analyzed swing states. This last result is in line with [18], which found a percentage of posts published by bots equal to 20%, albeit using a different sample of tweets and analysis methodology.

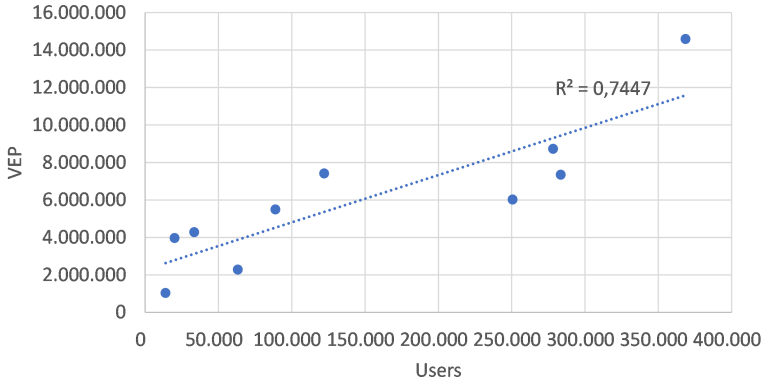
| State | #Users | %Bots | #Posts | %Bot Posts |
|---|---|---|---|---|
| Colorado | 20,029 | 9.57% | 45,197 | 22.15% |
| Florida | 368,593 | 2.73% | 604,482 | 13.89% |
| Iowa | 63,264 | 6.82% | 162,567 | 20.52% |
| Michigan | 122,141 | 2.40% | 444,321 | 19.79% |
| New Hampshire | 13,920 | 9.39% | 30,523 | 20.58% |
| North Carolina | 283,419 | 12.88% | 1,108,556 | 12.77% |
| Ohio | 88,896 | 6.11% | 293,150 | 24.55% |
| Pennsylvania | 278,255 | 8.89% | 978,913 | 11.45% |
| Virginia | 250,622 | 7.63% | 955,821 | 12.65% |
| Wisconsin | 33,446 | 2.30% | 72,197 | 19.60% |
| **Total** | **1,522,585** | **7.03%** | **4,695,727** | **14.52%** |

**Table 4.1:** Bot incidence in posts and users collected by state

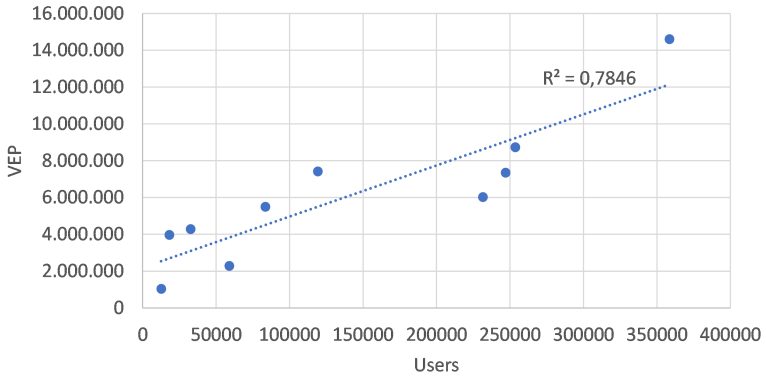
Collected data are representative of the analyzed event as:

- All the posts under analysis have the lang field set to *en* (i.e. English).
- About 94% of the social media users in the USA are adults and almost equally divided by gender (42.7% females and 57.3% males).
- For each state, we measured the correlation between collected users and the voting-eligible population (VEP). We observed a strong linear correlation, with a Pearson coefficient $r = 0.86$, which improved after removing bots reaching 0.89. Both results are significant at $p < .01$.

The high correlation and statistical significance achieved in this exploratory step, indicate that collected users can most likely be considered voters in the related swing state. Moreover, a significant improvement in the goodness-of-fit due to the removal of bots suggests a higher concentration of bot-driven campaigns in some states, which may amplify election-related online content, likely for illicit purposes. Figure 4.2 summarizes these results by showing a linear interpolation, along with the goodness-of-fit measured through the determination coefficient ($R^2$).




(a) All users




(b) Bots excluded

**Figure 4.2:** Linear interpolation: analyzed users vs. voting-eligible population.

In the next sections, we analyze the polarization of users during the 2016 US presidential election campaign, also investigating how the presence of bots may have affected the political discussion on Twitter.

### 4.3.1 Polarization analysis and election forecasting

In this step, we exploited Algorithm 3 and 4, described in Section 4.2.2 and 4.2.3, for determining the political orientation of the collected posts and the corresponding users. Furthermore, posts are assigned an importance weight and users are classified as real accounts or social bots. The decay rate $\lambda$ and the threshold $th$ have been set to 0.3 and 0.7 respectively. Table 4.2, shows how the support detected for the different factions is distributed among real users and bots. Specifically, with pro-$x$ bots, we indicate Twitter accounts classified as bots, which have mainly published tweets in favor of candidate $x$.

| Polarization | #Users | %Bots | #Posts | %Bot Posts |
|---|---|---|---|---|
| Pro-Trump | 94,124 | 26.70% | 194,428 | 17.86% |
| Pro-Clinton | 78,900 | 10.00% | 128,154 | 8.27% |

**Table 4.2:** Supporting posts and users per candidate

We found a greater presence of pro-Trump bots, which had a more marked impact on the online discussion, producing almost 18% of the contents classified as in favor of Trump. This suggests a greater use of social bots supporting the Trump political positioning, compared to the other faction, which however shows a quite high volume of bot-generated content, in line with work [18].

Once posts and users were classified according to their political polarization and social bot were detected using Botometer, we determined the outcome of the 2016 US election as explained in Section 4.2.3. The achieved results are summarized in Table 4.3, which shows a comparison among the real voting percentages, the average values of the latest opinion polls before the election, and the results obtained by using TIMBRE. The winning candidate is written in bold when it is correctly identified.

Compared to the latest opinion polls, which gave a correct forecast for only 6 out of 10 swing states, the proposed methodology was able to correctly identify the winning candidate in 8 out of 10 states, confirming its ability to accurately determine the polarization of social media users. TIMBRE outperformed the latest opinion polls even in terms of average absolute error, improving it from 1.2 to

| State | Real | | Polls | | TIMBRE | |
|---|---|---|---|---|---|---|
| | *Clinton* | *Trump* | *Clinton* | *Trump* | *Clinton* | *Trump* |
| Colorado | **48.2** | 43.3 | **43.3** | 40.4 | **47.7** | 43.8 |
| Florida | 47.8 | **49.0** | 46.4 | **46.6** | 48.1 | **48.7** |
| Iowa | 41.7 | **51.1** | 41.3 | **44.3** | 34.1 | **58.7** |
| Michigan | 47.3 | **47.5** | 45.4 | 42.0 | 41.7 | **53.1** |
| New Hampshire | **47.0** | 46.6 | **43.3** | 42.7 | **56.8** | 36.9 |
| North Carolina | 46.2 | **49.8** | 46.4 | 46.4 | 44.7 | **51.2** |
| Ohio | 43.6 | **51.7** | 42.3 | **45.8** | 43.9 | **51.4** |
| Pennsylvania | 47.9 | **48.6** | 46.2 | 44.3 | 51.5 | 45.0 |
| Virginia | **49.8** | 44.4 | **47.3** | 42.3 | **49.9** | 44.3 |
| Wisconsin | 46.5 | **47.2** | 46.8 | 40.3 | 52.0 | 41.7 |
| ***Correctly classified*** | - | | **6/10** | | **8/10** | |
| ***Posts*** | - | | - | | 277,181 | |
| ***Users*** | - | | $\approx$ 10,000 | | 140,003 | |
| ***Avg. accuracy*** | - | | 0.6 | | 0.8 | |
| ***Avg. absolute error*** | - | | 1.2 | | 0.9 | |

**Table 4.3:** Voting percentages estimates of the 2016 US presidential election.

0.9. We computed this metric only focusing on wrong predictions by using the following formula:

$$avg.\ absolute\ error = \frac{1}{|F|}\sum_{f\in F}\frac{1}{|S|}\sum_{s\in S} w(s) * |real_{f,s} - pred_{f,s}|$$

where $F$ and $S$ are the set of considered factions and states, $real_{f,s}$ and $pred_{f,s}$ are the real and predicted voting percentages related to the faction $f$ in the state $s$, and $w(s)$ is a binary function which outputs 1 if the predicted polarity is wrong, 0 otherwise (i.e. the winning candidate is correctly identified). Using this metric we both penalized the absolute error in terms of percentage points and the inversions predicted polarity, which can be a crucial issue while analyzing these states, characterized by a high degree of uncertainty. Another noteworthy advantage is related to the number of polarized users, which is much larger than that of the people interviewed, making the proposed approach a viable alternative to traditional opinion polls.

### Ablation analysis

We further extended our experimental evaluation by performing an ablation analysis, aimed at analyzing the benefits brought by each of the two key steps introduced by the proposed methodology, i.e. *temporal weighting* and *bot removal*.

| State | Real | Polls | Baseline | Bot removal | Temporal weighting | TIMBRE |
|---|---|---|---|---|---|---|
| Colorado | **C** | **C** | T | **C** | T | **C** |
| Florida | **T** | **T** | C | C | **T** | **T** |
| Iowa | **T** | **T** | **T** | **T** | **T** | **T** |
| Michigan | **T** | C | **T** | **T** | **T** | **T** |
| New Hampshire | **C** | **C** | **C** | **C** | **C** | **C** |
| North Carolina | **T** | Tie | **T** | **T** | **T** | **T** |
| Ohio | **T** | **T** | **T** | **T** | **T** | **T** |
| Pennsylvania | **T** | C | C | C | C | C |
| Virginia | **C** | **C** | **C** | **C** | **C** | **C** |
| Wisconsin | **T** | C | C | C | C | C |
| ***Correctly classified*** | - | 6/10 | 6/10 | 7/10 | 7/10 | 8/10 |

**Table 4.4:** Ablation analysis of the contribution brought by each step of TIMBRE in terms of election forecasting accuracy.

The achieved results are reported in Table 4.4, where "C" and "T" stand for Clinton and Trump respectively. What emerges is that both the temporal weighting of posts and bot removal steps are crucial to get a correct estimate of users' voting intentions. In particular, the *baseline* version of our methodology, which does not leverage either the removal of bots or the temporal weighing of posts, achieved the same accuracy as the latest polls, correctly identifying the winning candidate in 6 out of 10 states. By adding the bot removal step, the resulting methodology was able to correctly predict the final outcome in Colorado, increasing its accuracy to 7 out of 10 states correctly classified. Similarly, by only adding the time-based weighting mechanism, we observed an increase in forecasting accuracy, with a correct prediction for the state of Florida. Finally, TIMBRE was able to maintain the benefits coming from both steps, combining them and correctly determining the winning candidate in 8 out of 10 states. Furthermore, it is worth noting that the results for Pennsylvania and Wisconsin, incorrectly predicted by TIMBRE, were not correctly predicted even by opinion polls.

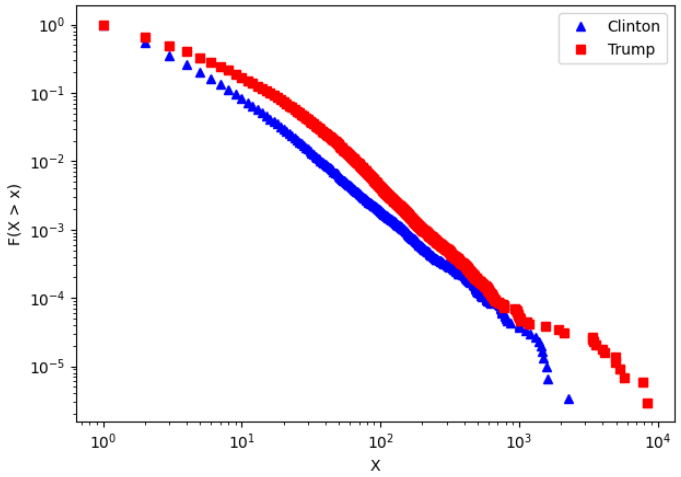
### 4.3.2 Bot influence on election-related discussion

In this section, we analyze how the presence of social bots may have affected the political online discussion around the 2016 US presidential election. Specifically, we first analyzed the publishing behavior of both real users and social bots focusing on the patterns of information production. Then, we studied the main differences in supporting the two candidates between human-driven and artificial accounts. Finally, we estimated the degree of influence of social bots on legitimate users using a competitive information diffusion model.

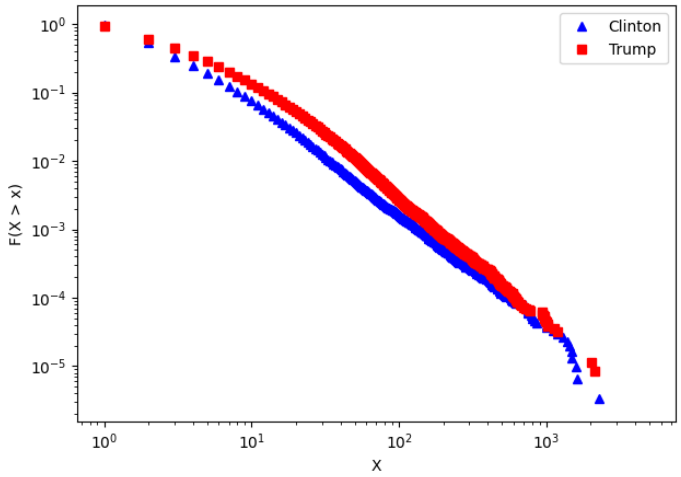
#### Information production patterns

In order to extract the publishing behavior of social media users involved in the political discussion, we used the information about their political orientation coming from the user polarization step, computing a publishing model for each candidate. In particular, such models are represented by the complementary cumulative distribution function (CCDF) of the number of posts published by users supporting Clinton and Trump respectively. We computed two different models, one that considers all accounts and another that excludes bot accounts, only retaining legitimate users based on the user classification previously performed.

Obtained results are shown in Figure 4.3, which provides a log-log scale scatter plot for each publishing model. By analyzing the publishing behavior of all polarized users (both real and fake accounts), shown in Figure 4.3(a), we observed a greater publication tendency of pro-Trump accounts, which appear to be far more prolific than pro-Clinton ones. However, the role of polarized bots behind this phenomenon should be investigated: for this purpose Figure 4.3(b) shows the publishing behavior of legitimate users only. By excluding the bots from the CCDF, we observed a narrowing of the distance between the curves relating to pro-Trump and pro-Clinton users. Therefore the polarity does not seem to be a deciding factor affecting the volume of posts published by legitimate users. Hence, it can be deduced that the differences emerging in Figure 4.3(a) are due to an amplifying effect caused by social bots, which agrees with the higher activity of pro-Trump bots detected in the previous sections.




(a) All users




(b) Bots excluded

**Figure 4.3:** CCDF of published posts for real and bot users classified by supported faction

For completeness, in Table 4.5 we provide the description of the most prolific real accounts in our dataset, according to the detected polarity. In particular, for each candidate, we selected the user labeled as real by Botometer that published the highest number of posts, i.e. the rightmost point in Figure 4.3(b). Despite the high number of published posts, Botometer gave the two accounts a BotScore score far below 0.5, which suggests that they are truly managed by prominent users or news sites, but not by automatic entities.

| Polarity | Screen name | Bot score (Botometer) | #Posts | Example post |
|---|---|---|---|---|
| Pro-Trump | @TheJonFerns | 0.18 | 3650 | "*Not even Hillary Clinton's campaign chief believes her. #podestamails*" |
| Pro-Clinton | @Kaliburger | 0.16 | 4004 | "*Think we should always have a woman as President. #imwithher*" |

**Table 4.5:** Most prolific real accounts supporting each candidate.

### Influence spread

This last step is aimed at estimating the degree of influence of social bots on legitimate users, following the approach described in Section 4.2.4. For this purpose, we built a graph $G$ based on repost relationships, characterized by 437,854 nodes and almost 1.5 million edges. From this graph, we removed self-loops, duplicated edges, and isolated nodes. Afterward, we analyzed the spread of influence by adapting the *Separated-Threshold Model for Competing Technologies* (see Section 4.2.4) to our case study, characterized by the rivalry of two candidates. Due to this, the diffusion process starts from two distinct seed sets containing respectively the bots polarized for the Democratic and the Republican party. When convergence is reached, we end up with a list of influenced nodes labeled with the related political polarization. We conducted 20 simulations varying the initial assignment of the random threshold for each faction, which represents the resistance of the users in the network to be influenced by social bots supporting that faction. Starting from the output of the diffusion process, we computed the following quantities:

- The expected spread for each candidate, determined as the average number of influenced nodes across the 20 simulations by pro-Trump and pro-Clinton nodes.

- The set of influenceable nodes, obtained through the voting technique. In particular, all the nodes activated at least once during the different simulations were assigned to the faction that influenced them the greatest number of times.

The final results obtained after the different simulations of the diffusion process are shown in Table 4.6. Both the expected number of influenced nodes and the total number of influenceable nodes confirmed the greatest activity of pro-Trump bots, which had a more marked impact on social media conversation compared to pro-Clinton ones. In particular, the expected number of nodes influenced by the seed set of pro-Trump bots was 12.4 times greater, while the number of influenceable nodes was 7.8 times greater.

| | **Expected number of influenced nodes** | **Total number of influenceable nodes** |
|---|---|---|
| **Pro-Trump bots** | 31,629 (2.4%) | 99,833 (7.5%) |
| **Pro-Clinton bots** | 2,547 (0.2%) | 12,775 (1.0%) |

**Table 4.6:** Obtained results after 20 simulations of the diffusion process.

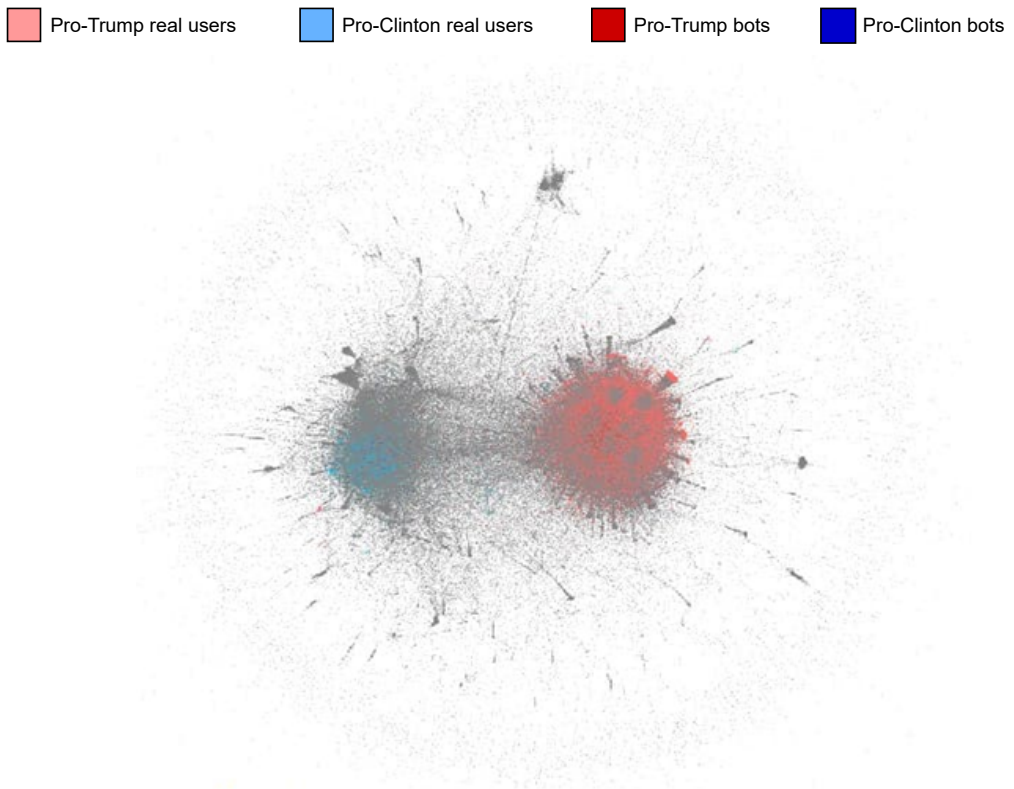



**Figure 4.4:** Visualization of the diffusion process on the complete repost graph.

These results are summarized by the graph shown in Figure 4.4, whose nodes are colored according to their polarity and characteristics. In particular, the polarized bots belonging to pro-Trump and pro-Clinton seed sets are colored in dark red and dark blue, influenceable nodes assigned to Trump are represented in light red, those assigned to Clinton in light blue, and neutral nodes in gray.

Finally, in Figure 4.5, we reduced the initial graph by 90% while keeping the top-$k$ nodes with the highest degree (right graph). In this way we maintained almost unchanged the polarity-based clustering structure that emerged in the total

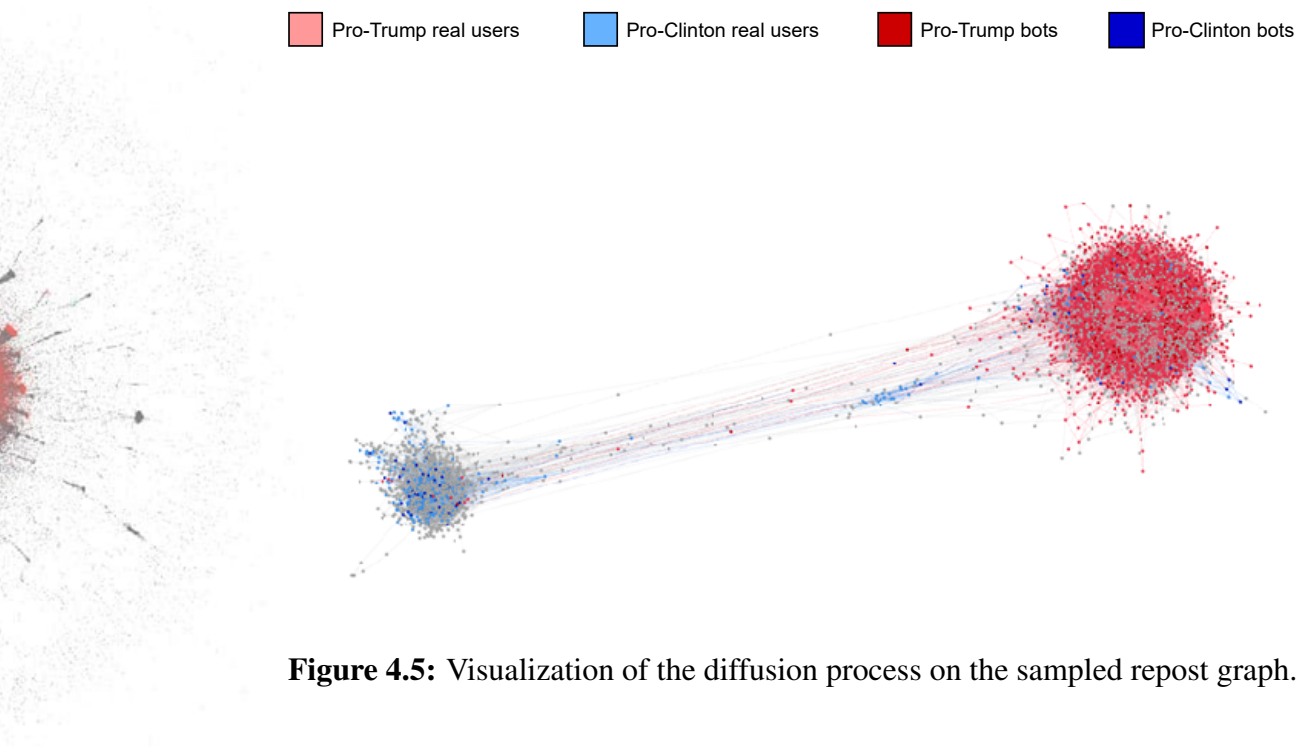



**Figure 4.5:** Visualization of the diffusion process on the sampled repost graph.

## 4.4 Conclusions

This chapter presented TIMBRE (Time-aware opInion Mining via Bot REmoval), a methodology we proposed with the aim of discovering the political polarization of social media users during election campaigns, coping with the presence of social bots, and the temporal oscillation of users' political polariza-

tion. Our methodology exploits a keyword-based classification to determine the political orientation of social media posts and users. It is also temporally aware, as it considers time-related aspects in deciding how much a post can be helpful to determine the voting intentions of the user who published it. Moreover, it recognizes and filters out data produced by social media bots, which participate in online discussions to amplify propaganda and alter public opinion.

In order to assess the effectiveness of TIMBRE, it was applied to a real-world case study related to the 2016 US presidential election. By leveraging Twitter metadata, we focused only on posts coming from 10 US Swing States, in particular: Colorado, Florida, Iowa, Michigan, Ohio, New Hampshire, North Carolina, Pennsylvania, Virginia, and Wisconsin. The achieved results showed the effectiveness of the proposed approach, along with the benefits brought on forecasting accuracy by its two key steps, i.e. temporal weighting and bot removal. Specifically, our methodology was able to correctly identify the winning candidate in 8 states out of 10, with an average absolute error of 0.9 percentage points, outperforming the latest opinion polls, which identified the winner in 6 out of 10 cases, with an average error of 1.2 points.

As a final step, we investigated how the presence of social bots may have affected political discussion around the 2016 US presidential election. In particular, we first analyzed the publishing behavior of both real users and social bots focusing on the patterns of information production. Then, we studied the main differences in supporting the two main candidates between human-driven and artificial accounts. Furthermore, we estimated the degree of influence of social bots on legitimate users finding out that in the analyzed scenario bots had a marked impact on social media conversation, showing a significant activity and influence on legitimate users. In conclusion, it is worth noticing that the obtained results are based on a politically neutral research analysis that produced accurate estimates, which are in accordance with related work. In addition, although our analysis discovered a high presence of social bots that may have affected online political discussion, it is impossible to know who was running those bots, as they can also be exploited for provocative campaigns or as part of an information war.

Chapter 5

# Influence maximization in politically polarized networks: a bio-inspired approach

Social media platforms are increasingly used to convey advertising campaigns for products or services, as well as to foster democratic debate on various issues, including political ones. As far as the political context is concerned, these platforms are gaining increasing success among prominent politicians of various parties around the world, who exploit various strategies to try to win the favor of public opinion on social networks, thus gaining an electoral advantage. An effective way to run a good social media campaign, such as one aimed at increasing support for a candidate or a political party, is to identify an appropriate subset of highly-followed users, investing resources in making them exploit their popularity in order to influence the preferences of a large number of voters who interact on the social network. However, finding such a set of users can be challenging, leading to the need for tailored techniques in the field of *influence maximization* [85]. Initially proposed as a stochastic optimization problem in [86], it formally consists in identifying a set of $k$ users that maximize the spread of influence in a social network, by analyzing the network structure, user interconnections, and user-specific features such as demographic properties [87]. Influence maximiza-

tion is an NP-Hard problem, with two sources of hardness: *i*) the complexity of computing the spread, i.e. the number of influenced users; *ii*) the combinatorial nature of identifying the best solution, that maximizes the influence, among all possible combinations.

This chapter presents *WABC (Weighted Artificial Bee Colony)*, a bio-inspired influence maximization technique that we designed for identifying a subset of users that maximizes the spread of influence in politically-polarized networks. Specifically, our analysis is aimed at identifying the main influencers of the different factions involved in a political event of interest, also deriving the main information diffusion strategies of each faction during the political campaign. Our work borrows ideas from the Artificial Bee Colony (ABC) [88] a swarm intelligence algorithm that was applied to the influence maximization task by Sankar et al. [89]. In WABC, we introduce an effective weighted approach to fitness evaluation, which can be considered as the resolution of a reachability problem centered on the paths of maximum probability. By following this approach we addressed the influence overlap problem of classical influence ranking-proxy algorithms, avoiding the negative effects caused by influence redundancy during the maximization process. Moreover, our algorithm is less sensitive to parameter tuning in comparison to related work, as it dynamically sets the depth at which to explore the graph, focusing more on the most promising paths. All of these factors contribute to making WABC able to produce an accurate estimate of the total spread for the final seed set, which is key to achieving a reliable estimate of the number of users that will actually be influenced.

To evaluate the effectiveness of the proposed algorithm in identifying the top influencers in a politically-polarized network, we applied it to a case study that analyzes the online discussion on Twitter during the Constitutional Referendum held in Italy in 2016. This analysis was aimed at identifying the main influencers of the *yes* and *no* factions and deriving the main information diffusion strategies leveraged by each faction during the political campaign. Specifically, we classified the identified influencers according to their profile (journalistic page, political activist, popular or normal user) to better determine the type of political campaign. Then,

we measured their influence strength by following a simulation-based approach. We also compared the obtained results with both the standard ABC algorithm and other related state-of-art techniques in terms of computing time, evaluated spread, and relative error on the expected spread. Our algorithm turned out to be more time-consuming than its classical version (ABC), but much more accurate in determining the expected spread, with an up to 24% decrease in the relative estimation error. Furthermore, it outperformed ranking-proxy techniques based on classical centrality measures like PageRank, with an up to 40% improvement.

The remainder of the chapter is organized as follows. Section 5.1 describes the most used information diffusion models. Section 5.2 discusses the related work. Section 5.3 describes the proposed algorithm. Section 5.4 presents the experimental evaluation. Finally, Section 5.5 concludes the chapter.

## 5.1 Information diffusion models

Interactions among users of a social network can be represented as a directed graph $G = (V, E)$, where $V$ is the set of users in the network and $E$ represents the relationship among them as edges directed from one user to another. The influence exercised by a user on the other members of the network is modeled as a function $p : E \rightarrow [0, 1]$ that associates a weight to each relationship $(u, v) \in E$. Given a user node $u \in V$, we define with $N^{in}(u)$ and $N^{out}(u)$ the sets of users $v \in V$ for which there exists a relationship $(v, u) \in E$ and $(u, v) \in E$ respectively.

In a diffusion model, nodes can be partitioned according to their current state: *influenced* (i.e., nodes that have been activated during the diffusion process), *active* (i.e., nodes that can propagate influence and activate others), and *idle* (i.e., nodes that have not yet been activated). The diffusion process starts from a small set of active nodes, called seed set $S \subseteq V$. Therefore, each node of the seed set can iteratively influence its out-neighbors and the process generally stops when there are no new active users. Diffusion models can be divided into two classes: *i*) *progressive* models, which do not allow a user to become idle once activated; *ii*) *non-progressive*, in which deactivation is allowed at any time.

The most used diffusion models in the literature are progressive, since the growth of the active set is monotonic, which ensures the termination of the propagation process in a finite number of steps when the number of users is finite. The main models in this category, described in the following, are the *Independent Cascade* and the *Linear Threshold*.

**Independent Cascade.** The independent Cascade model (IC) first described by Kempe et al. [90], is characterized by the independence of activation among nodes. Given the input network graph $G = (V,E)$ an initial set of active nodes, i.e. the seed set $S$, is chosen. Therefore, the IC model generates the active sets $A_t$ for each step $t \geqslant 1$ following randomized diffusion dynamics, with $A_0 = S$. Specifically at step $t$, for each inactive node $v \notin A_{t-1}$, each node $u \in N^{in}(v)$ activated at the previous step attempts to activate $v$ through a Bernoulli trial with a probability of success equal to $p(u,v)$. If the test is successful, the node $v$ is added to the active set of the current iteration.

**Linear Threshold Model.** Similar to IC, the Linear Threshold model (LT) [90], takes the network graph $G = (V,E)$ and the initial seed set $S_0$ as input. The probability on the in-edges is normalized so that $\sum_{u \in N^{in}(v)} p(u,v) \leqslant 1, \forall v \in V$. Therefore, LT generates the active sets $A_t$ for each step $t \geqslant 1$, according to the following mechanism. Initially, each node $v \in V$ independently selects a threshold $\theta_v$ by sampling a uniform distribution in the interval [0.1]. At step $t$, for each inactive node $v \in V$, if the sum of the weights of the active in-neighbors reaches the threshold $\theta_v$, i.e. $\sum_{u \in N^{in}(v) \cap A_{t-1}} p(u,v) \geqslant \theta_v$, then $v$ is activated and is included in the active set $A_t$. Intuitively, threshold $\theta_v$ models the likelihood with which $v$ is influenced by its active neighborhood: a high threshold value represents a greater resistance to the influence in the propagation process. The random choice of the threshold reflects the lack of information on users' tendency to be influenced and is the only source of non-determinism in the model.

The aforementioned propagation models need techniques for establishing the influence probability and therefore the weights to be assigned to the edges of the network. A widely used practice is weighted cascade (WC), where the probability of influence $p(u,v)$ is defined as $\frac{1}{|N^{in}(v)|}$, where $|N^{in}(v)|$ is the in-degree of $v$. The main idea behind this weighting scheme is that an important node (i.e., a public figure) is more likely to influence a user that tends to express interest only for its contents, assuming that edges are oriented according to a relationship of interest. Recent studies also proposed the estimation of the influence probabilities starting from logs. As an example, authors in [91] framed the acquisition of influence weights from existing logs as a likelihood maximization problem.

### 5.1.1 Spread function properties

Independent Cascade and Linear Threshold are progressive models that share two important properties in terms of influence spread, that is the function $\sigma$, which estimates the expected number of users who will be active at the end of the information diffusion process. For both models the spread function is:

1. *monotonic*, the inclusion of a new node $v$ in the active set $S$ can not lead to a decrement of the spread function: $\sigma(S \cup v) \geqslant \sigma(S), \forall v \in V, S \subseteq V$.

2. *submodular*, the marginal gain obtained by adding a new node $v$ to a set $S$ is at least equal to the marginal gain obtained by adding the same element to a superset $T$ of $S$: $\sigma(S \cup v) - \sigma(S) \geqslant \sigma(T \cup v) - \sigma(T),\ \forall v \in V, S \subseteq T$.

When used within an influence maximization algorithm, a spread function satisfying these two properties ensure the convergence on a $\varepsilon$-approximated solution of a greedy hill-climbing procedure, which selects at each iteration the most promising node in terms of influence spread.

**Theorem** [92]. For a non-negative, monotone submodular function $\sigma$, let $S$ be a set of size $k$ obtained by selecting elements one at a time, each time choosing the element that provides the largest marginal increase of $\sigma$. Let also $S^*$ be the optimal set that maximizes the value of $\sigma$ over all $k$-element sets.
Then $\sigma(S) \geqslant (1 - \frac{1}{e}) \cdot \sigma(S^*) \Longrightarrow S$ provides a $\varepsilon$-approximation, with $\varepsilon = 1 - \frac{1}{e}$

## 5.2 Related work

Despite the theoretical bound discussed above, the influence maximization task remains hard to solve. In fact, besides the complexity related to the maximization of the spread $\sigma$, which derives from the combinatorial nature of the problem, another crucial point is the calculation of $\sigma$ with respect to the addition of a node $v$ in the active set, which is a #P-hard counting problem. For this reason, different resolution techniques have been developed, which can be grouped as discussed in the following, according to the approach used in the evaluation of the spread function [93].

**Simulation-based.** The key idea of this approach is to perform a series of Monte Carlo simulations for evaluating the spread function for a given seed set. Considering the IC model, given a graph $G$, this approach consists in considering an initial seed set $S$ and removing the edges with probability $1 - p(u,v)$. This way a set of instances can be generated and the spread can be estimated on these sampled instances. The advantage of such models is their generality, as this process can be applied to any information propagation model, and also the bound provided by the greedy algorithm is preserved. However, the main problem here is the computational efficiency related to the large number of simulations needed to obtain a good estimate. Kempe et al. [90] extended the greedy algorithm using simulations for evaluating the marginal gain for a given node added to the active set. In particular, a seed set $S$ is built by considering the most promising nodes with respect to their marginal gain on the spread function, estimated after $r$ simulations, as the average cardinality of the active set. The number of simulations is a crucial parameter in such a mechanism, which affects computational complexity. For this reason, several methods have been proposed aimed at reducing $r$. The CELF technique [94] aims to estimate an upper bound of the marginal gain determined by adding a node to the current seed set. This avoids the evaluation of some nodes whose influence is considered insignificant, thus exploiting the submodularity of the spread function and a power law assumption on the degree distribution of the network graph. Another technique used to reduce the complexity of this kind

of approach is the Community-based greedy algorithm (CGA) [95]. This technique is based on the divide-and-conquer paradigm for reducing the complexity of the Monte Carlo simulations by partitioning the graph according to a community structure and evaluating the spread only within communities.

**Proxy-based.** The main idea behind this class of algorithms is to define proxy models such as PageRank or shortest path, for approximating the spread function $\sigma$. Therefore, its main advantage is the reduced complexity of the proxy model, but there are no guarantees of optimality. Proxy-based algorithms can be divided into *influence ranking proxy* and *diffusion model reduction proxy*.

*i) Influence Ranking Proxy* are models that provide a rank to each user in the graph $G$ in order to estimate a metric for their influence rate and subsequently generate the seed set directly from that ranking. There are different approaches based on a ranking that can be directly derived from the graph, such as degree, PageRank, and other centrality measures. However, these techniques are usually not very suitable to solve the problem of influence maximization, as the ranking defined on users often does not take into account any overlap of influence; two users with a high ranking could influence an almost identical set of users, providing an incorrect solution to the problem. To deal with this issue, the DegreeDiscount technique [96] has been proposed, which introduces a penalty on $\sigma$ for a given node $v$ proportional to the overlap of influence with the other nodes in the active set. Finally, the IRIE [97] algorithm, based on the Independent Cascade model, integrates the advantages of influence ranking (IR) and influence estimation (IE) methods for influence maximization.

*ii) Diffusion model reduction proxy* tries to reduce the complexity in computing the spread $\sigma$ following two main approaches: i) reducing the stochastic propagation model in a deterministic one; ii) estimating the influence from a local subgraph. The Shortest-Path Model (SPM) [98] considers the shortest path between two nodes in the activation process. In the MIA/PMIA [99] algorithm, for each pair of nodes $(u,v)$, $u$ can influence $v$ only along the maximum influence path (MIP), defined on a tree where unpromising paths are pruned using a threshold.

The LDAG algorithm [99] restricts the influence to acyclic graphs, by building a DAG for a node $v$, exploiting the Dijkstra algorithm for the shortest-path length. Goyal et al. [100] proposed the Sim-Path algorithm, where the influence of a set of nodes, propagated through the LT model, is calculated by enumerating all the simple paths starting from each node within the set. However, as this is a #P-Hard problem, the SimPath limits this enumeration to a restricted neighborhood, cutting those paths with a probability lower than a threshold. Lee et al. [101] proposed a fast greedy approximation algorithm for influence maximization that relies on the *2-hop influence spread*. It is based on the interesting observation that an item is generally diffused within a very small number of hops in a social network, and node influence gradually dissipates beyond two hops.

**Sketch-based.** The goal of sketch-based methods is to preserve the theoretical bounds provided by simulation-based methods while providing computational efficiency. In order to avoid the repetition of several Monte Carlo simulations, these techniques compute many deterministic sketches which can be viewed as many possible worlds, and aggregate the achieved results of each sketch, which capture the overall diffusion process. One of the most famous algorithms is NewGreIC [96], which extends the Independent Cascade model by building a given number of sketches via graph sampling, to evaluate the marginal gain of each node. Borges et al. [102] discovered that it is not necessary to estimate the influence using sketches generated starting from the entire graph. They developed the reverse reachable sketch approach, a technique in which the influence of each seed set $S$ is estimated by selecting a random subset of nodes and analyzing which of these can be reached. By creating multiple random RRs on different nodes, if a node $u$ has a great impact on the other nodes, then it will have a high probability of appearing within these RR sets. Similarly, if a seed set $S$ covers a maximum number of RR sets, it is likely to be the optimal seed set. Borges et al. [102] proposed the RIS algorithm, which generates random RRs until the total number of edges examined during the generation process does not reach a threshold.

**Context-aware.** The common factor which characterizes all the aforementioned algorithms, categorized in taxonomy [93], is that the influence propagation process is often modeled in an unrealistic way, never referring to a specific context. For this reason, other influence maximization algorithms have been proposed in the literature, for dealing with several tasks in specific contexts. In topic-aware influence maximization (TAIM) the IM problem is extended considering what are the topics to be propagated. TAIM introduces the topics to exploit the interests of users interacting in social networks while computing the spread. TAIM models are TIC (Topic-aware Independent Cascade) and TLT (Topic-aware Linear Threshold) [103]. The standard IM algorithm does not take the time dimension into account. Such an assumption could be unreasonable in some cases, time-aware diffusion models introduce the concept of step as a temporal measure and restrict the process of diffusion within these steps. Chen et al. [104] proposed IC-M the model, where for each edge between the nodes $u$ and $v$, a meeting probability $p(u,v)$ is defined, and the IM problem consists in identifying the optimal seed set capable of activating the greatest number of nodes in at the most $\tau$ steps. Kim et al. [105] proposed the CT-IC model, based on the concept of continuous time, by defining an activation delay of the nodes and a delay distribution.

**Data-driven.** The main goal of data-driven approaches is to exploit propagation traces available in historical data for learning how influence flows in the network, and thus estimating the expected spread of influence. As an example, Goyal et al. [106] proposed a data-driven approach based on the credit distribution model, which can learn different levels of influenceability of users, also taking into account the temporal aspects. Data-driven approaches are more flexible and able to adapt to different networks and application scenarios, compared to models that randomly assign influence probabilities, which may lead to large errors in spread prediction. However, they require more computational resources and a large number of propagation traces representative of user interactions.

### 5.2.1 Comparison

The *Weighted Artificial Bee Colony (WABC)* described in this chapter effectively exploits a weighted bio-inspired approach to deal with the influence maximization task. It can be classified as an influence ranking-proxy algorithm and it is characterized by several changes and improvements with respect to previous related work. Primarily, our algorithm exploits a more effective way of evaluating fitness, which can be considered as the resolution of a reachability problem, where the maximum probability path is considered among all possible ones connecting two distinct nodes. This feature leads to two main benefits:

- The total spread can be accurately estimated. It is a crucial result for an influence maximization task, as it measures the expected number of influenced users, without providing incorrect assessments.
- The influence overlap problem is addressed. This is a common issue of classical influence ranking-proxy algorithms, which can lead to negative effects caused by influence redundancy during the maximization process.

Moreover, WABC is less sensitive to parameter tuning in comparison to related approaches, as it does not use a fixed a-priori depth at which to explore the graph. In particular, it exploits a threshold on the influence probability, dynamically focusing more on the most promising paths. Furthermore, we combined information diffusion and influence maximization with a political polarization analysis and a user classification process, to identify the main influencers and derive the main information diffusion strategies in a scenario characterized by multiple opposing factions. In particular, we leveraged WABC to identify the top influencers in a politically-polarized network, classifying them according to their profile (journalistic page, political activist, popular or normal user) to determine the type of political campaign. Finally, we measured their influence strength by following a simulation-based approach.

## 5.3 Proposed algorithm: WABC

In recent years, nature has been a great source of inspiration for the development of different algorithms aimed at solving many real-world optimization problems [107]. These bio-inspired techniques are related to Swarm Intelligence (SI), a particular field of Artificial Intelligence (AI) based on observing the behavior of social animals such as ants and bees. Swarm Intelligence can be defined as the collective behavior of decentralized and self-organized systems, in which the interaction among components causes the emergence of complex behavior.

### 5.3.1 Artificial Bee Colony

Among the various swarm intelligence algorithms in the literature, Artificial Bee Colony (ABC) is one of the most studied and applied. It is a meta-heuristic algorithm, introduced in 2005 by Derviş Karaboğa [88] and applied to the influence maximization task by Sankar et al. [89], inspired by the food supply model of bee colonies. It consists of three main components: food sources, employed bees, and unemployed bees. In the colony system, the quality of a *food resource* depends on several factors, like the distance from the hive, the amount of food, and the ease of extraction. Each resource is assigned to a bee, whose task is to store the information related to that resource. The main objective of such a model is the search for a source rich in nectar and the abandonment of a poor source. *Employed bees* collect nectar from a flower and bring it to the hive, carrying details about the source of food and sharing this information with other bees. *Unemployed bees* are those bees that are not currently picking up nectar from any flower and can be divided into two types: *scout bees* whose job is to search for new nearby sources of food; *onlooker bees* which wait for choosing a food source based on information brought to the hive by employed bees, then selecting the most promising one. This exchange of information takes place in the hive through a particular technique called *waggle dance* [108]. It consists in a specific physical movement of bees, whose duration is proportional to the goodness of the food source.

The ABC algorithm can be adapted to explore a social network for identifying a subset of nodes with maximum influence, based on the waggle dance mechanism. Each node of the social network is considered a source of food. The employed bees, used to identify the opinion leaders of the network, are initially assigned on the basis of a ranking vector. Scout bees are used for exploring the neighborhood of employed bees for obtaining better solutions, while onlooker bees are used to indicate influenced nodes. For the sake of clarity, Table 5.1 reports the meaning of the main symbols used throughout this chapter.

| Symbol | Meaning |
|---|---|
| $V$ | Set of graph nodes |
| $E$ | Set of graph edges |
| $S$ | Seed set |
| $A_t$ | Set of active nodes at time $t$ |
| $EB$ | Set of employed bees $eb$ |
| $SB$ | Set of scout bees $sb$ |
| $fit_x$ | Fitness value of the node $x$ |
| $F(t)$ | Global fitness value at time $t$ |
| $p(x,u)$ | Probability of the path between $x$ and $u$ |
| $\sigma(S)$ | *Expected spread*, i.e. an estimate defined by the algorithm starting from $S$ |
| $\tilde{\sigma}(S)$ | *Evaluated spread*, i.e. an estimate defined via simulation starting from $S$ |
| $\omega$ | Convergence distance |
| $\theta$ | Cutting threshold |
| $\mathcal{N}_x$ | Set of all neighbors of the node $x$ |
| $\mathcal{N}_x^{in}$ | Set of in-neighbors of the node $x$ |
| $\mathcal{N}_x^{out}$ | Set of out-neighbors of the node $x$ |
| $\mathcal{N}_x^{d}$ | Set of distinct nodes reachable from $x$ in $d$ steps |

**Table 5.1:** Meaning of the most important symbols used in this chapter.

Algorithm 5 shows the pseudo-code of the ABC algorithm. The input is composed of: a graph $G = (V,E)$, a ranking vector $R$, with $|R| = |V|$, and an integer $k$ representing the seed set size. The produced output is twofold, and consists of:

- a set of nodes $S$ with $S \subset V$ and $|S| = k$, which maximizes the spread (i.e., the number of influenced users);
- the expected spread $\sigma(S)$.

**ALGORITHM 5:** Artificial Bee Colony (ABC)

**Input :** Graph $G = (V, E)$, a ranking vector $R$, an integer $k$
**Output:** Seed set $S$, Expected spread $\sigma(S)$

1 $EB \leftarrow$ *top-k nodes ordered by ranking*
2 $SB \leftarrow \varnothing$
3 **for** $eb \in EB$ **do**
4 $\quad SB \leftarrow SB \cup \mathcal{N}_{eb}^{out}$
5 $Fit_{EB} \leftarrow \varnothing$
6 **for** $eb \in EB$ **do**
7 $\quad fit_{eb} \leftarrow evalFitness(\{eb\}, G)$
8 $\quad Fit_{EB} \leftarrow Fit_{EB} \cup fit_{eb}$
9 `/* Local optimum search */`
10 **while** *not convergence reached* **do**
11 $\quad SB \leftarrow orderByRanking(SB)$
12 $\quad$ **for** $sb \in SB$ **do**
13 $\quad\quad fit_{sb} \leftarrow evalFitness(\{sb\} \cup EB, G)$
14 $\quad\quad$ **if** $\exists\ eb \mid fit_{sb} \geqslant fit_{eb}$ **then**
15 $\quad\quad\quad EB \leftarrow EB \setminus \{argmin_{eb} Fit_{EB}\} \cup \{sb\}$
16 $\quad SB \leftarrow \varnothing$
17 $\quad$ **for** $eb \in EB$ **do**
18 $\quad\quad SB \leftarrow SB \cup \mathcal{N}_{eb}^{out}$
19 `/* Estimate global optimum */`
20 $S \leftarrow EB$
21 **for** $s \in S$ **do**
22 $\quad \sigma(S) \leftarrow \sigma(S) + fit_s$
23 **return** $S, \sigma(S)$

The algorithm starts by defining two sets:

- The set of employed bees $EB \subset V$ is initialized with the best $k$ nodes of the input ranking vector $R$, identifying the initial seed set (line 1).
- The set of scout bees $SB \subset V$ is initialized with an empty set and filled by joining the out-neighborhood $\mathcal{N}_{eb}^{out}$ of each employed bee (lines 2-4).

Then the vector $Fit_{EB}$ is obtained evaluating the fitness function, for each employed bee in $EB$ (lines 5-8), whose goal is to iteratively determine local optima during the diffusion process. To that end, the algorithm starts an iterative phase

(lines 10-18), performing at each iteration the following operations:

- The set of scout bees *SB* is ordered by ranking value (line 11).
- For each scout bee in descending order of ranking the fitness $fit_{sb}$ is evaluated (lines 12-13).
- If $fit_{sb}$ exceeds the fitness value of one of the scout bees then the roles are exchanged (lines 14-15).

This phase is repeated until the evaluation of the whole set *SB*. The set of scout bees is therefore repopulated with the out-neighborhood of the new employed bees (lines 16-18) and the process iterates until a termination criterion is reached. Once this criterion is reached the final seed set $S$ is filled with the employed bees *eb* in *EB* (line 20). The expected spread $\sigma(S)$ is evaluated by summing up the fitness $fit_s$ of each seed $s \in S$ (rows 21-22). Finally, the algorithm returns the final seed set $S$ and the expected spread $\sigma(S)$ (line 23).

### 5.3.2 Weighted Artificial Bee Colony

Weighted Artificial Bee Colony (WABC) is the extension of the classical ABC algorithm [89] we designed in [3]. The main advantages are related to how the fitness function is calculated. Specifically, in ABC the fitness evaluation is based on a difference between sets. The input is composed of the graph $G = (V, E)$, a distance $d$, and the set of nodes $X$ with respect to which fitness is evaluated. For each node $x \in X$, the $covered_x$ set is filled with each node $u \in V$ reachable by $x$ in $d$ steps, i.e. the $d_{out}$–neighborhood of $x$ ($\mathcal{N}_x^d$), where $d$ is generally equal to one (lines 3-4). Therefore, each node $x \in X$ evaluates its fitness ($fit_x$) as the difference between its own $covered_x$ set and the $covered_z$ for each node $z \in X$ with $z \neq x$. In other words, it finds how many nodes it can reach in $d$ steps that are not reachable from any other node (lines 6-10). Finally, the global fitness value $fit_X$ is obtained by summing up $fit_x \forall x$ and returned (lines 11-12). This pseudo-code of this procedure is shown by Algorithm 6.

**ALGORITHM 6:** ABC fitness evaluation

**Input :** Graph $G = (V,E)$, a distance $d$, a set of nodes $X$
**Output:** Fitness value $fit_X$

1 $covered \leftarrow \varnothing$
2 /* For each $x \in X$ store each node $u \in V$ reachable by $x$ in $d$ steps, i.e., the $d_{out}$-neighborhood of $x$ ($\mathcal{N}_x^d$) */
3 **for** $x \in X$ **do**
4  $covered_x \leftarrow \mathcal{N}_x^d \subset X$
5 /* Each node evaluates its fitness as the number of unique nodes that compose its $d_{out}$-neighborhood $\mathcal{N}_x^d$ */
6 **for** $x \in X$ **do**
7  $fit_x \leftarrow 0$
8  **for** $u \in covered_x$ **do**
9  **if** $!\exists\ z \in X | u \in covered_z$ **then**
10  $fit_x \leftarrow fit_x + 1$
11 $fit_X \leftarrow \sum_{x \in X} fit_x$
12 **return** $fit_X$

Differently, in WABC (see Algorithm 7), each node evaluates the weighted sum with respect to the unique nodes it can activate. Similarly, the input is composed of: the graph $G = (V,E)$, a threshold $\theta$, and the set of nodes $X$. For each employed node $x \in X$, the $covered_x$ set is filled with the pair $\langle u, p(x,u) \rangle$ for each node $u$ covered by $x$, where $p(x,u)$ represents the maximum influence probability of $x$ on $u$. The evaluation of the fitness can be considered as the resolution of a reachability problem where the maximum probability path is considered among all the paths $\mathcal{P} \in Paths(u,v)$, connecting $x$ and $u$. However, we must note that considering all the possible influence paths between an employed bee and the other nodes is often computationally infeasible. For this reason, the algorithm takes as input positive threshold $\in \Re, \theta \geqslant 0$, which defines the minimum probability of influence, i.e., a cutting value for those paths having a negligible activation probability. Therefore, given this threshold, each employed $x$ determines the set of nodes reachable along a path of total probability at least equal to $\theta$, where the probability of a path $\mathcal{P}$ from $x$ to $u$, i.e. $p(x,u)$, is given by the product of the weights associated to each edge $(i,j) \in \mathcal{P}$ (line 3-6).

At this point, the fitness value $fit_x$ for each $x \in X$ is computed. Specifically, for each pair $\langle u, p(x,u) \rangle \in covered_x$, $fit_x$ is incremented of $p(x,u)$ if $x$ has the highest influence on the node $u$, i.e. there not exists another node $z$ such that $p(z,u) > p(x,u)$. So, when two nodes reach the same target, only the most influential will increase its fitness by a value of $p(x,u)$ (lines 8-12). Finally, the global fitness value $fit_X$ is obtained by summing up all the obtained $fit_x$ and returned (lines 13-14).

**ALGORITHM 7:** WABC fitness evaluation

**Input :** Graph $G = (V,E)$, a threshold $\theta$, a set of nodes $X$
**Output:** Fitness value $fit_X$

1 $covered \leftarrow \varnothing$
2 `/* Find the best activation path for each pair` $\langle x \in X, u \in V \rangle$ `*/`
3 **for** $x \in X$ **do**
4  **for** $u \in V$ **do**
5  $p(x,u) \leftarrow max_{\mathcal{P} \in Paths(x,u)} \prod_{(i,j) \in \mathcal{P}} p(i,j)$
6  $covered_x \leftarrow covered_x \cup \langle u, p(x,u) \mid p(x,u) \geqslant \theta \rangle$
7 `/* The employed bee with the highest influence probability increases its fitness */`
8 **for** $x \in X$ **do**
9  $fit_x \leftarrow 0$
10  **for** $\langle u, p(x,u) \rangle \in covered_x$ **do**
11  **if** $!(\exists \langle u, p(z,u)\ z, u \in X \rangle \in covered_z \mid p(z,u) > p(x,u))$ **then**
12  $fit_x \leftarrow fit_x + p(x,u)$
13 $fit_X \leftarrow \sum_{x \in X} fit_x$
14 **return** $fit_X$

The described fitness function introduces two main improvements with respect to the classical ABC approach. Firstly, the weighted activation, with a strength equal to the influence probability, leads to a more accurate estimate of the final spread, compared to the classical binary activation used in ABC, also addressing the influence overlap issue. Moreover, the algorithm dynamically focuses more on the most promising paths, leading to more effective and efficient exploration of the social graph compared to ABC, which considers the neighborhood of each employer node within a fixed small number of hops, often equal to one.

## 5.4 Experimental evaluation

In this section, we evaluate the performances of the proposed algorithm, by applying it to a politically-polarized case study involving real-world data and complex influencing behaviors, related to the 2016 Italian constitutional referendum. On 4th December 2016, Italian voters were asked whether they approve a constitutional law that amends the Italian Constitution to reform the composition and powers of the Parliament of Italy, as well as the division of powers between the State, regions, and administrative entities[1]. The main supporter of the referendum (i.e., in favor of *yes*) was the Democratic Party (in Italian Partito Democratico or PD) and its leader, also Italian prime minister, Matteo Renzi. On the other side, in favor of *no* were the main opposition parties (e.g., Movimento 5 Stelle, Forza Italia) and several citizen committees. The referendum saw a high voter turnout (approximately 65% of voters) and a majority of the votes opposed to the reform (i.e., voting *no*), which exceeded 59% of the expressed preferences. A political effect of the referendum's result was the resignation of the Italian prime minister.

The political event under analysis $\mathcal{E}$ is a two-faction event $F = \{yes, no\}$. In order to investigate the information diffusion processes involved in the political campaign of both factions, also identifying the main influencers, we build two polarity-based subgraphs, $G_{yes}$ and $G_{no}$. The subgraphs generation process is based on: *i*) the identification of retweet relationships among users and *ii*) the binary classification of tweets based on a set of faction keywords. As a first step, we collected the main keywords $K$ used as hashtags in tweets related to $\mathcal{E}$. Such keywords have been grouped as follows:

- $K_{neutral} = \{$#referendumcostituzionale, #siono, #referendum, #4dicembre, #riformacostituzionale, #referendum4dicembre$\}$
- $K_{yes} = \{$#bastaunsi, #iovotosi, #leragionidelsi, #italiachedicesi, #iodicosi, $\}$
- $K_{no} = \{$#iovotono, #iodicono, #bastaunno, #famiglieperilno, #ragionidelno$\}$

[1] http://www.interno.gov.it/it/italiani-voto-referendum-costituzionale

Based on these keywords, we collected 338,592 tweets posted from 23rd October (5 weeks before the voting day) to 3rd December 2016 (one day before). Collected tweets were pre-processed as described in the previous chapters, and classified as follows: if a tweet $t$ contains only keywords that are in favor of a specific faction $f \in F$, then $t$ is classified as in favor of $f$; otherwise, $t$ is classified as neutral. For instance, Table 5.2 shows some examples of tweets we collected with their classification (translated in English for the Reader's convenience).

| Text | Keywords | Class |
|---|---|---|
| Why it is important to be well informed on #referendumcostituzionale. | #referendumcostituzionale | neutral |
| #IoVotoNO: all the reasons to vote against this reform. | #iovotono | no |
| For a stronger Italy in Europe! #iovotosi #referendum #democrazia | #iovotosi, #referendum, #democrazia | yes |

**Table 5.2:** Examples of tweets about the Italian constitutional referendum.

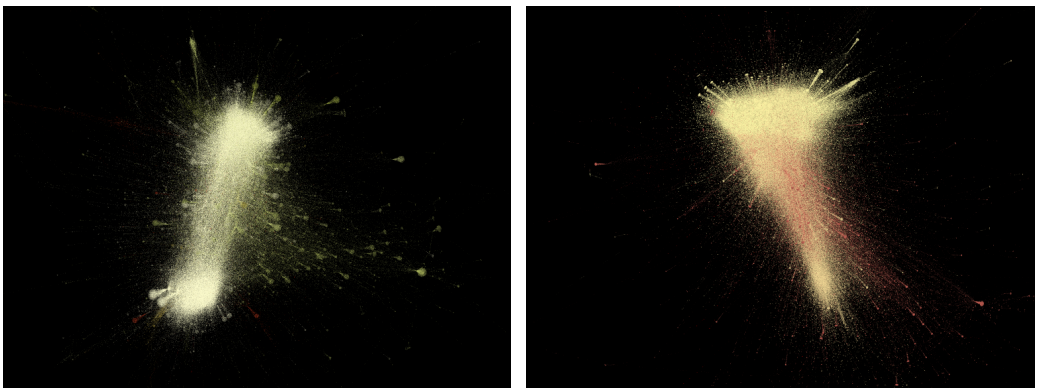
Starting from the set of classified tweets, we generated two subgraphs $G_{yes}$ and $G_{no}$, relating to the users who supported the *yes* and *no* faction, respectively. Specifically, the graph $G_{yes} = (V, E)$ was obtained as follows:

- The set of nodes $V$ is represented by users who have at least published a tweet or a retweet classified in favor of *yes*.
- The set of direct edges $E$ is determined using the *retweet* relationship. Particularly, there is an edge $(u_1, u_2)$ from user $u_1$ to user $u_2$ if there exists at least one tweet classified in favor of *yes* posted by $u_1$ and retweeted by $u_2$.
- The influence weight is established by using a weighted cascade criterion. In particular, the probability of influence $p(u, v)$ associated to the edge $(u, v)$ is defined as $\frac{1}{|\mathcal{N}_v^{in}|}$, where $|\mathcal{N}_v^{in}|$ is the in-degree of $v$, i.e., the number of retweets published by user $v$.

The generation process for the $G_{no}$ graph is analogous. Once having extracted the $G_{yes}$ and $G_{no}$ graphs, we determined the leading influencers and the main information diffusion strategies of the *yes* and *no* factions.

### 5.4.1 Graph properties

The *yes* graph $G_{yes}$ has 117,000 nodes and 270,000 edges, with a density of $1.95 \cdot 10^{-5}$, while the *no* graph $G_{no}$ has 130,000 nodes and 440,000 edges, with a density of $2.72 \cdot 10^{-5}$. The observed low densities are related to an accentuated sparsity of the graphs, induced by the presence of many isolated nodes (i.e., users publishing tweets that have not caught the attention of any user). For this reason, the analysis we carried out only focused, for both graphs, on the *Giant Component* (*GC*), i.e. their largest connected component. Figure 5.1 shows the *GC* subgraph for the *yes* and *no* factions, whose main properties are reported in Table 5.3.



(a) Giant component of graph of *yes* ($GC_{yes}$). (b) Giant component of graph of *no* ($GC_{no}$).

**Figure 5.1:** Representation of the giant component of the two graphs.

| *Feature* | $\mathbf{GC_{yes}}$ | $\mathbf{GC_{no}}$ |
|---|---|---|
| Num. of nodes | 72,225 | 78,899 |
| Num. of edges | 269,218 | 437,608 |
| Diameter | 18 | 18 |
| Average in-degree | 3.98 | 5.67 |
| Clustering coefficient | 0.05 | 0.06 |
| Average path length | 5.92 | 5.28 |

**Table 5.3:** Giant components properties of the two graphs

Table 5.4 shows the top-5 most influential nodes computed through the PageRank scoring strategy [109]. The underlying assumption of this algorithm is that more important users are likely to receive more links from others.

| *Pos.* | $G_{yes}$ | $G_{no}$ |
|---|---|---|
| 1 | bastaunsi | Mov5Stelle |
| 2 | matteorenzi | matteosalvimini |
| 3 | davidefaraone | marionecomix |
| 4 | fanpage | beppe_grillo |
| 5 | repubblicait | bastaunsi |

**Table 5.4:** Top-5 most influential nodes calculated using PageRank

### 5.4.2 Parameter sensitive analysis

Here we investigate the effect of different parameters on WABC performance. The algorithm requires as input the network graph $G = (V,E)$, an integer $k$ which represents the seed set cardinality, a threshold $\theta$ on the minimum path probability, a real number $\omega$ which controls convergence, and a diffusion model. In our case study, we used the following parameters: $\omega = 2 \cdot 10^{-2}$, $k = 10$, and Weighted Cascade as a diffusion model. As with the hyperparameter optimization of many algorithms, we set the threshold $\theta$ by performing several experiments, varying its value within a given range. This process follows a grid search approach and requires the optimization of a score function, $f(\theta)$, derived as described below.

We analyzed the behavior of the algorithm varying $\theta$ with respect to the expected spread given by the algorithm ($\sigma_\theta(S)$), an estimate of the real spread achieved through $20,000$ simulations ($\tilde{\sigma}_\theta(S)$), and the overall execution time ($T_\theta$). These different metrics are jointly modeled in the following score function:

$$f(\theta) = Err_{rel}(\theta) + Time_{rel}(\theta) + Cov_{rel}(\theta)$$

with $\Theta = \{\theta_1, ..., \theta_n\}$ the set of all considered thresholds.

The above formula takes three factors into account:

- The relative error of the expected spread with respect to its estimate achieved by simulating the diffusion process starting from the $k$ seeds identified by the algorithm:

$$Err_{rel}(\theta) = \frac{|\sigma_\theta(S) - \tilde{\sigma}_\theta(S)|}{\tilde{\sigma}_\theta(S)}$$

- The relative execution time, i.e. the ratio between the overall execution time with the current threshold $\theta$ and the maximum time taken by the other instances executed with different values of $\theta$:

$$Time_{rel}(\theta) = \frac{T^\theta}{max_{\theta \in \Theta} T^\theta}$$

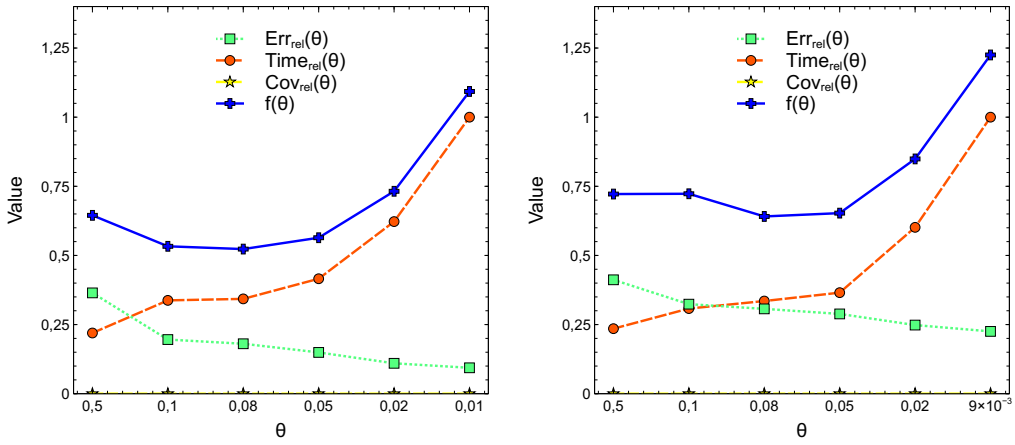
- The percentage decrement between the number of covered nodes with the current threshold and the maximum number of nodes covered with different values of $\theta$:

$$Cov_{rel}(\theta) = (1 - \frac{\tilde{\sigma}_\theta(S)}{max_{\theta \in \Theta} \tilde{\sigma}_\theta(S)})$$

We estimated the optimal value of the threshold as:

$$\hat{\theta} = argmin_{\theta \in \Theta} f(\theta)$$

Figure 5.2 shows the trend of $f(\theta)$ (in blue) varying $\theta$ for *yes* and *no* graphs, in which threshold values are sorted in descending order. For decreasing values of $\theta$, the relative error of the expected spread (in green) decreases, while the overall execution time (in red) increases. Thus, through the minimization of $f(\theta)$, we can find a suitable value for $\theta$, reached at $\hat{\theta} = 8 \cdot 10^{-2}$ for both graphs, which provides a good trade-off between accuracy and complexity. Subsequently, this configuration can be used to investigate interesting aspects of the obtained results, such as comparing the members of the final seed set with the major activists and journalistic pages affiliated with the corresponding faction.




(a) Graph of *yes* ($G_{yes}$). (b) Graph of *no* ($G_{no}$).

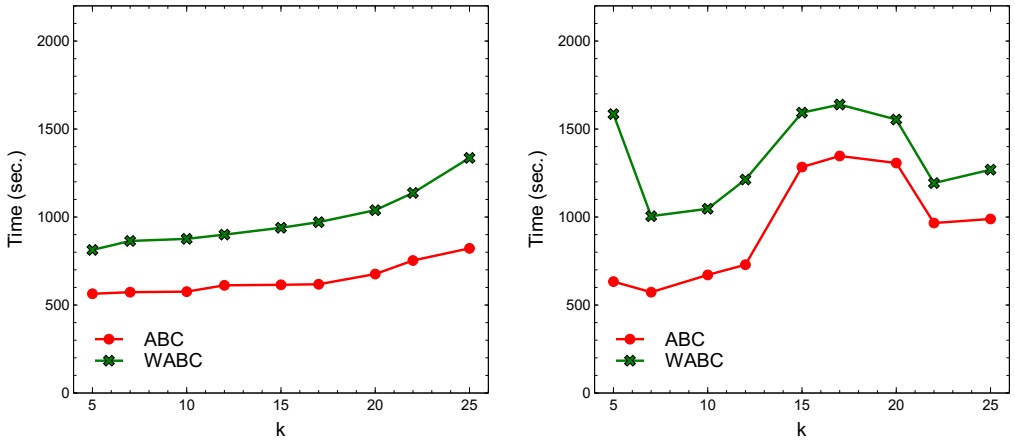
**Figure 5.2:** Trend of the $f(\theta)$ score function and its components.

### 5.4.3 WABC vs. ABC

In this section, we investigate the main advantages of the WABC algorithm with respect to its original version (ABC), by varying the cardinality of the seed set $k$. Our analysis focuses on evaluating: $i$) the *execution time*; $ii$) the *evaluated spread* $\tilde{\sigma}(S)$, i.e., the estimate of the real spread achieved via simulation; and $iii$) the *expected spread* $\sigma(S)$, i.e., the estimate given by the algorithm. The computing system used for the experimental evaluation is a cluster node equipped with 4 CPUs (AMD 6376), each one with 16 cores of 2.3GHz, and 256 GB of memory.

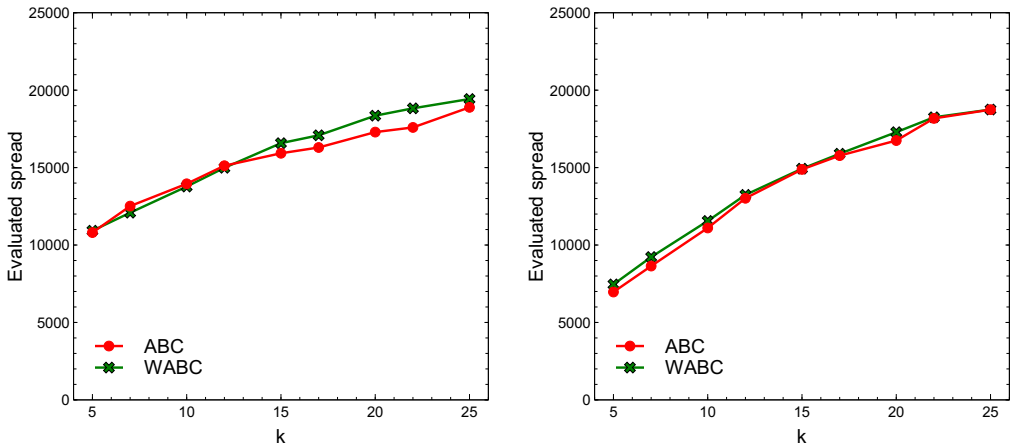
Figure 5.3 shows the execution time for the two algorithms by varying the value of $k$. The ABC algorithm provides better performance thanks to the greater simplicity in computing the fitness function that analyzes only the neighbors of the seeds at a fixed distance. Besides this factor, greater simplicity emerges in terms of convergence, as ABC generally converges after a single iteration. Figure 5.4 shows the number of influenced nodes, estimated by simulating the information diffusion process, starting from the seed set identified by the two algorithms. In this case, WABC achieved similar results with respect to ABC, finding sets of seeds that allow reaching almost the same number of nodes in all the considered configurations. The average in-degree of the subgraph induced by the set of influ-

enced users for $G_{yes}$ and $G_{no}$ graphs is equal to 3.86 and 6.15 respectively. These values highlight the tendency of users, especially in the $G_{no}$ graph, to retweet content from different sources. This results in a small number of exclusive retweet relationships with individual users, with a dilution of influence probability.




(a) Graph of *yes* ($G_{yes}$). (b) Graph of *no* ($G_{no}$).

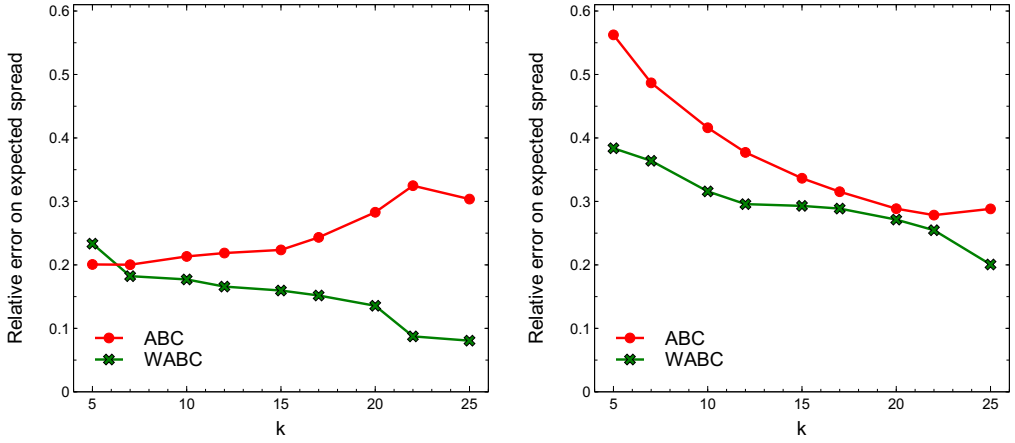
**Figure 5.3:** Comparison between WABC and ABC in terms of execution time.




(a) Graph of *yes* ($G_{yes}$). (b) Graph of *no* ($G_{no}$).

**Figure 5.4:** Comparison between WABC and ABC in terms of evaluated spread.

A third aspect to analyze concerns the quality of the expected spread. The majority of ranking-proxy models, such as Degree, PageRank, Rank, and IRIE are unable to provide an estimate of the expected spread, highlighting the need for identifying appropriate solutions for this issue. The ABC algorithm provides an estimate of the spread, returning the number of unique nodes reachable in one hop, starting from the identified seed set. Figure 5.5 shows a comparison between WABC and ABC in terms of relative error on the expected spread. Specifically, for each cardinality $k$ of the seed set, the error was computed by comparing the expected spread $\sigma_k(S)$ given by the algorithm with the evaluated spread $\tilde{\sigma}_k(S)$, an estimate of the real value achieved through $20,000$ simulations. It can be clearly observed that the WABC algorithm provides more accurate spread estimates compared to ABC, with an up to 24% decrease in the relative estimation error.




(a) Graph of *yes* ($G_{yes}$). (b) Graph of *no* ($G_{no}$).

**Figure 5.5:** Comparison between WABC and ABC in terms of relative estimation error on the expected spread.

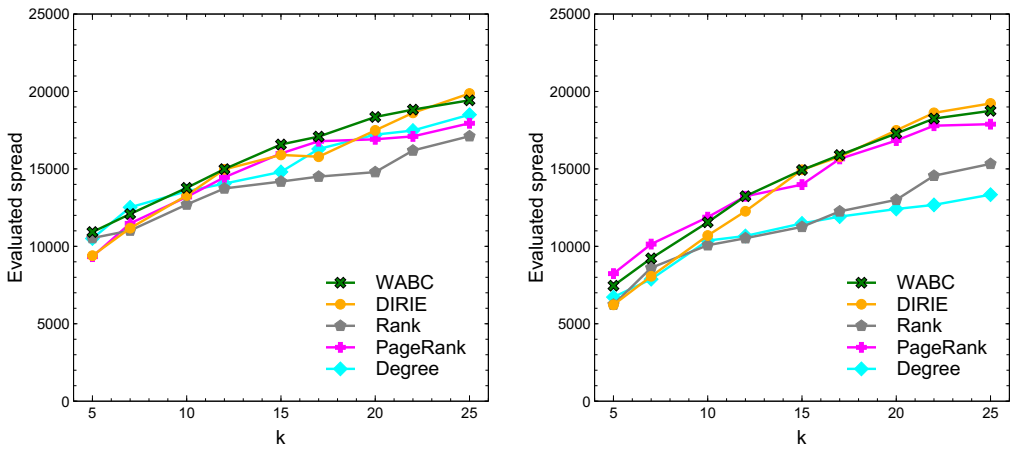
Summing up, the two algorithms achieved quite similar results in terms of the number of influenced users. WABC showed to be more time-consuming as it exploits a more sophisticated approach for fitness evaluation. However, it was able to obtain much more precise estimates of the expected spread, which is a crucial aspect that makes our algorithm more suitable for use in real contexts.

### 5.4.4 WABC vs ranking-proxy models

To better assess the effectiveness of WABC, we carried out a comparison, in terms of evaluated spread, with the most relevant ranking-proxy models used in the literature:

- *Degree*: uses the out-degree of each node as the seed set selection criterion.
- *PageRank*: uses the pagerank [109] of each node, evaluated on the reversed graph, as the seed set selection criterion.
- *Rank*: uses the rank, proposed in [89] of each node as the seed set selection criterion.
- *DIRIE*: is a distributed version of the IRIE algorithm [97].

Figure 5.6 shows the results obtained by WABC in comparison with these ranking-proxy techniques by varying $k$.




(a) Graph of *yes* ($G_{yes}$). (b) Graph of *no* ($G_{no}$).

**Figure 5.6:** Comparison between WABC and the most relevant state-of-art ranking-proxy techniques in terms of evaluated spread.

Compared to the aforementioned techniques, WABC turned out to be the most effective, providing the best solution in almost any configuration. In particular, WABC outperformed ranking-proxy techniques based on simple classical centrality measures, i.e. PageRank, Rank, and Degree, with an up to 40% improvement over the latter. Compared to DIRIE, which is based on the Independent Cascade model and exploits a more complex algorithm, WABC achieved quite similar results, with a slight improvement for low values of $k$, i.e., small seed sets. In addition, a noteworthy advantage of WABC compared to both DIRIE and all the other state-of-the-art approaches is the ability to give an accurate estimate of the spread at the end of the influence propagation process.

### 5.4.5 Diffusion strategies of politically-polarized information

Starting from the seed set determined by WABC, we classified each influencer according to its profile, in order to investigate the information diffusion strategies followed by each faction. Moreover, we simulated the diffusion process to visualize their estimated influence within the analyzed politically-polarized network. As a first step, we compared the two seed sets generated by WABC and ABC to measure the degree of overlap and their political-based coherence.

| $G_{yes}$ | | $G_{no}$ | |
|---|---|---|---|
| ABC | WABC | ABC | WABC |
| lucatelese | serracchiani | dukana2 | dukana2 |
| bastaunsi | fnicodemo | beppe_grillo | beppe_grillo |
| ArsenaleKappa | TwitterItalia | matteosalvinimi | matteosalvinimi |
| antonio_bordin | matteorenzi | marionecomix | comitatono |
| GiorgiaMeloni | pdnetwork | figprov | ale_dibattista |
| nonleggerlo | ArsenaleKappa | antonio_bordin | antonio_bordin |
| matteorenzi | repubblicait | tuseivitaearia | luigidimaio |
| TwitterItalia | Tgcom24 | ArsenaleKappa | ArsenaleKappa |
| tuseivitaearia | tuseivitaearia | arsenaletv | Mov5Stelle |
| molumbe | bastaunsi | ComitatoDelNo | GiorgiaMeloni |
| overlap: 50% | | overlap: 50% | |

**Table 5.5:** Comparison between the seed sets identified by WABC and ABC.

As shown in Table 5.5, the two algorithms produced quite different results, with a seed set overlap equal to 50% for both graphs. It is also worth noticing that the results generated by WABC are more accurate from the point of view of political polarization. As an example, the seed set related to the *yes* faction determined by ABC contains leading politicians, like Giorgia Meloni, that were notoriously against the referendum objectives. On the contrary, by analyzing the seed set produced by WABC, a correct correspondence emerges between the members of each seed sets and his/her actual political leaning.

Afterward, we divided influencers identified by WABC into four categories, *news pages* (information or satire), *political activist*, *popular user* and *normal user*, finding out the composition shown in Table 5.6.

| | **news pages** | **political activist** | **popular user** | **normal user** |
|---|---|---|---|---|
| $G_{yes}$ | 50% | 20% | 20% | 10% |
| $G_{no}$ | 20% | 60% | 10% | 10% |

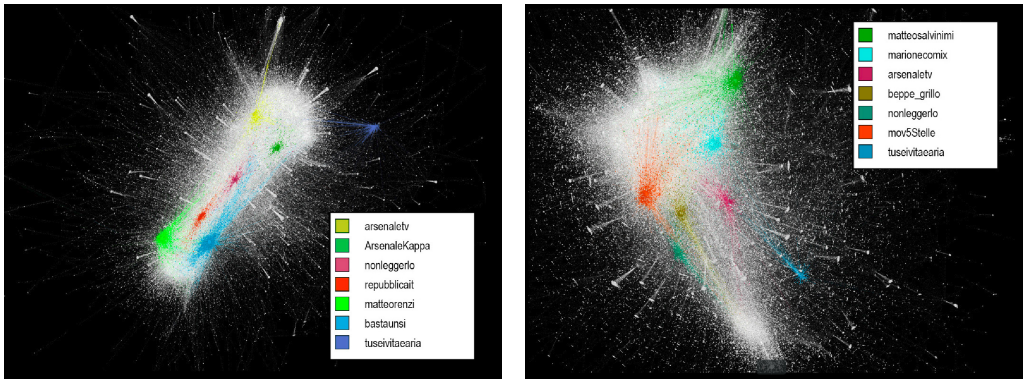
**Table 5.6:** Classification of the influencers for $G_{yes}$ and $G_{no}$ graphs.

By observing this categorization, we can determine the type of political campaign adopted by the two factions during the political campaign:

- The *no* faction saw a greater effort from leading politicians, such as Matteo Salvini, Alessandro Di Battista, Luigi Di Maio, Beppe Grillo, and Giorgia Meloni, actively opposing the constitutional referendum on social media platforms.

- The *yes* faction, instead, saw Matteo Renzi as the main leader, along with news pages such as La Repubblica and Tgcom24, which confirms the communication strategy of *yes* faction, centralized on the head of government.

Lastly, Figure 5.7 shows the results of a simulation executed starting from the seed set identified for the two graphs. By coloring each node according to the seed node (influencer) that determined its activation, we can see that the aforementioned political activists and news page can activate a remarkable portion of the politically-polarized networks analyzed in this case study.




(a) Graph of *yes* ($G_{yes}$). (b) Graph of *no* ($G_{no}$).

**Figure 5.7:** Simulation of the influence diffusion process starting from the seed set identified for the two graphs.

## 5.5 Conclusions

This chapter presented *Weighted Artificial Bee Colony (WABC)*, a bio-inspired influence maximization algorithm that introduces an improved weighted approach to fitness evaluation. It has been applied to a real case study related to the Constitutional Referendum held in Italy in 2016, to analyze the propagation of information in a real-world politically-polarized network. In particular, we identified the main influencers for the *yes* and *no* factions, deriving also the main information diffusion strategies of each faction during the political campaign. Experimental results confirmed the effectiveness of the proposed algorithm, which outperformed ranking-proxy techniques based on standard centrality measures, i.e., PageRank, Rank, and Degree. Compared to DIRIE, WABC achieved quite similar results, with a slight improvement in identifying small seed sets. In addition, compared to the classical ABC, it proved to be more time-consuming but much more accurate, with an up to 24% improvement in the accuracy of spread estimation.

As a future work, the relationship between the diffusion of influence and political polarization can be further investigated, analyzing how the tendency of users to polarize in favor of a faction affects the dynamics of information diffusion and vice versa, and identifying patterns in the evolution of the political leaning of users interacting on social media.

Chapter 6

# Hashtag recommendation on social media platforms: a BERT-based translation approach

In the previous chapters, we provided an accurate description of a wide range of hashtag-based methodologies that we designed to extract high-quality information from politically-polarized Big Social Data, with the aim of modeling users' behavior, opinion, emotions, and interactions, thus providing a data-driven approach to understanding and studying political phenomena. Due to the hashtag-based approach they follow, these methodologies are strongly conditioned by the availability of representative hashtags in the online content published by social users. Unfortunately, because of the high dynamicity of social media conversation, choosing up-to-date and appropriate hashtags for a post is not always easy for users, and therefore posts are often published without hashtags or with hashtags not well defined [110]. To mitigate this issue, as highlighted in Section 1.1, hashtag recommendation models can be effectively used to generate consistent hashtags for a given social media post, leading to a double advantage: on the one hand, users are supported in choosing a hashtag in line with both the semantics of the text and the latest trends; on the other hand, hashtag-based techniques can benefit from a greater amount of representative and high-quality data.

This chapter presents *HASHET* (*HAshtag recommendation using Sentence-to-Hashtag Embedding Translation*) [9], a methodology that we designed with the aim of suggesting a relevant set of hashtags for a given post. HASHET is based on two independent latent spaces for embedding the text of a post and the hashtags it contains. A mapping process based on a multilayer perceptron is then used for learning a translation from the semantic features of the text to the latent representation of its hashtags. Specifically, the semantic space of sentences is obtained by using a pre-trained sentence embedding model, such as the *Google Universal Sentence Encoder* (*GUSE*) [13], or the *Bidirectional Encoder Representations from Transformers*(*BERT*) [15], which are very effective in capturing semantic and syntactic features of microblog texts. Differently, the latent space of hashtags comes from the training of a Word2Vec model [111] based on a Continuous Bag of Words (CBOW) architecture, aimed at discovering contextual relationships between words and hashtags.

Similarly to neural attention-based models present in the literature, HASHET exploits a semantic representation of a microblog generated by a transformer-based encoder. The key difference is in how it uses this representation to recommend hashtags. Neural-based solutions frame the recommendation task as a multi-class classification problem [112–114], using a softmax activation and minimizing a cross-entropy loss. Differently, in HASHET, we translate the latent representation of a post into a target vector lying in the hashtags embedding space. Then, the top-k nearest hashtags in this space are found and the resulting set is enriched using semantic expansion, a process based on semantic similarity in the hashtags embedding space. The obtained output is composed of semantically similar hashtags, reflecting the semantic relationships learned among hashtags and the underlying topic-based clustering structure. This inspection process thus exploits the concept of *locality* in the hashtags embedding space, which introduces a marked improvement in predicting hashtags with respect to other techniques.

The effectiveness of HASHET has been investigated in two real-world case studies related to the 2016 US presidential election and the COVID-19 pandemic. We evaluated the performance of two language representation models for sentence

embedding and tested different search strategies for semantic expansion, finding out that the combined use of BERT and a global expansion strategy leads to the best recommendation results, with an average F-score up to 0.82 and a recommendation hit rate of up to 0.92. We also compared HASHET to the most relevant techniques used in the literature (generative models, unsupervised models, and attention-based supervised models) by achieving an up to 15% improvement in F-score for the hashtag recommendation task. For usability and reproducibility purposes, an open-source version of HASHET is available on Github[1].

The remainder of the chapter is organized as follows. Section 6.1 presents the embedding techniques used in the proposed model, discussing its application in multilingual contexts. Section 6.2 discusses related work on hashtag recommendation. Section 6.3 describes the HASHET model. Section 6.4 presents the case studies. Finally, Section 6.5 concludes the chapter.

## 6.1 Embedding techniques

The HASHET model is based on a translation between two independent embedding spaces: *i*) the semantic space of sentences; *ii*) the latent space of hashtags. For what concerns the first embedding space ($S_{emb}$) we compared two of the most used state-of-art solutions for sentence encoding, published by Google, described in the following.

**Google Universal Sentence Encoder (*GUSE*) [13].** It consists of a deep sentence embedding model with two available implementations: one makes use of the transformer architecture [115], while the other is formulated as a deep averaging network (DAN) [116]. In this work, the first solution was chosen for generating the latent representation of a given sentence. It is pre-trained on a variety of web sources and Stanford Natural Language Inference (SNLI) corpus [117]. The encoding model is designed by using multi-task learning, according to which a single encoder is used for multiple downstream tasks, which are: a Skip-Thought-

[1] `https://github.com/scalabunical/HASHET`

like unsupervised task [118], a conversational input-response task [119], and a classification task for supervised learning. The latest transformer-based large version available on Tensorflow-Hub[2] has been used. Starting from a lowercase PTB tokenized string, it computes context-aware representations of the input words, taking into account both their ordering and identity. These representations are then converted to a single 512-dimensional sentence encoding vector, computed as their element-wise sum.

**Bidirectional Encoder Representations from Transformers (*BERT*) [15].** It is based on a multi-layer bidirectional Transformer [115], pre-trained on two unsupervised tasks, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP), using a large cross-domain corpus. Unlike *OpenAI GPT* [120], which uses a unidirectional (left-to-right) language model or *ELMo* [121], which uses a shallow concatenation of independently trained left-to-right and right-to-left language models, BERT is deeply bidirectional. In fact, the use of the MLM objective enables the representation to fuse the left and right contexts, allowing the pre-training of a deep bidirectional language representation model. BERT outperformed many task-specific architectures, advancing the state of the art in a wide range of Natural Language Processing tasks, such as textual entailment, text classification, and question answering. In this work, we used the *bert-base-uncased* implementation from Huggingface[3], characterized by 12 Transformer blocks, a hidden dimensionality of 768, 12 attention heads and 110M parameters. We also experimented with both the average pooling of the last hidden states and the hidden representation of the *CLS* token as the raw sentence embedding to be fed to our translation head, responsible for the mapping of this condensed representation in the hashtag embedding space.

[2] `https://tfhub.dev/google/universal-sentence-encoder-large/5`
[3] `https://huggingface.co/bert-base-uncased`

The second embedding space ($W_{emb}$) comes from the training of a *Word2Vec* model [111] based on a Continuous Bag of Words (CBOW) approach, aimed at learning a dense vector representation of the words in a given set of documents. The embedding process is based on semantic and syntactic similarity and both statistical and co-occurrence relationships with other words. Word2Vec is one of the most popular techniques to train a word embedding model using shallow neural networks. The training of such a model leads to the definition of a multidimensional latent space that reflects the semantic distribution of the words in the corpus. Words are represented uniquely as latent vectors and will be close to each other if recognized as semantically similar, through notions such as cosine similarity. There are two approaches to obtaining an embedding with Word2Vec:

- *Continuous Bag-of-Words (CBOW)*: given a fixed number of context words, this model tries to predict the word related to this context by distributing the probability on all the input terms with a single softmax output layer.
- *Skip-Gram*: starting from an input word, this model tries to predict the context, generating many probability distributions in the softmax output layer for how many context words are considered.

Since the HASHET model relies on the semantic mapping between the two aforementioned embedding spaces, it can be applied in the presence of social media posts in different languages as well as multilingual posts, which is a desirable property, as microblogging platforms are widespread across different cultures and geographies, as stated in Section 1.1. In particular, the latent space of hashtags $W_{emb}$ is language-agnostic, as it comes from a CBOW Word2Vec model trained from scratch on the given corpus of posts. For what concerns the pre-trained language representation models used for sentence embedding in the $S_{emb}$ space, both present a multi-lingual version. the *Google Multilingual Universal Sentence Encoder* embeds text from 16 languages into a single semantic space, while the *Multilingual BERT* covers 104 spoken languages from around the world [14, 16].

## 6.2 Related work

In recent years, Natural Language Processing (NLP) has been attracting increasing interest from the scientific community. With the fast growth of microblog services, several NLP techniques have been developed to learn the representation of microblog posts and recommend pertinent hashtags. Existing techniques can be grouped into three main categories, described in the following.

**Generative models.** Godin et al. [110] proposed a method for suggesting the top hashtags for a given post. They exploited Latent Dirichlet Allocation for finding out the underlying topic distribution, used for recommending general hashtags. Gong et al. [114] proposed a generative model for recommending hashtags in multimodal microblog posts that combines textual and visual information. In [122] authors proposed a supervised topic model-based solution for hashtag recommendation on Twitter (TOMOHA). They treated hashtags as labels of topics, developing a supervised topic model for discovering relationships among words, hashtags, and topics of tweets. Then, by inferring the probability that a hashtag will be contained in a new tweet, the $k$ most probable ones are recommended.

**Unsupervised models.** Pang et al. [123] investigated methods from the perspective of similarity diffusion, proposing a clustering-based method that exploits similarity cascades (SCs). SCs are a series of sub-graphs generated by truncating a similarity graph with a set of thresholds, where maximal cliques are used to capture topics. Topics are then identified through a process of similarity diffusion. In [124] authors proposed a hashtag recommendation methodology based on the embedded representation of Twitter microblog posts. They performed the following steps: i) a given tweet is represented as the weighted average of its word embeddings; ii) latent representations of tweets are clustered according to their syntactic and semantic similarity using a density-based approach; iii) top-k hashtags are found by computing the similarity between the entered tweet and the centroids of the obtained clusters. Huang et al. [125] proposed a high utility pattern clustering (HUPC) framework over microblog streams. Starting from a group of repre-

sentative patterns from the microblogging stream, patterns that perform better in describing topics are grouped into clusters. In this way, the proposed framework can detect coherent and new emerging topics simultaneously. Otsuka et al. [126] proposed a hashtag recommendation system for Twitter data streams, based on a novel ranking scheme, called Hashtag Frequency-Inverse Hashtag Ubiquity (HF-IHU), which is a variation of TF-IDF that considers hashtag relevancy and microblog data sparseness.

**Attention-based supervised models.** In recent years, attention-based models proved to be very effective in a wide range of NLP tasks including summarization of sentences [127], or text entailment [128]. The basic idea behind the attention mechanism is to allow the model to focus on the relevant parts of the input sequence as needed. This goal is accomplished by determining a weight for each position that indicates the amount of attention that should be paid to it [129, 130]. The first contribution came from [129], in which an attention-based neural machine translation (NMT) approach was leveraged to jointly translate and align words. This model differs from a standard encoder-decoder model, as the input sentence is encoded into a sequence of vectors, weighted through the attention mechanism in order to generate the translation. In [131] authors proposed a novel co-attention model for Visual Question Answering (VQA) that jointly reasons about image and question attention. Feng et al. [132] proposed a context-attention-based Long Short-Term Memory network (CA-LSTM) for modeling a sequence of microblogging posts and classifying the related sentiment. Many efforts have been also made in the hashtag recommendation field. As an example, Gong et al. [114] proposed a novel architecture based on convolutional neural networks enhanced with an attention mechanism for incorporating the trigger words. The authors formulated the hashtag recommendation task as a multi-class classification problem. They adopted an attention mechanism to scan input microblogs and select trigger words, which are combined with the whole microblog to perform the recommendation task. Li et al. [113] used an attention-based neural network to learn the representation of a microblog post. Specifically, they proposed

a novel Topical Co-Attention Network (TCAN) that models content attention and topic attention simultaneously. Finally, in [133] authors compared the performances of various deep learning architectures, such as recurrent neural networks or transformer-based architectures. They evaluated various state-of-art Zero-Shot Learning methods like a Convex combination of Semantic Embedding (ConSE), the Embarrassingly Simple ZSL model (ESZSL), and a Deep Embedding Model for ZSL (DEM-ZSL), based on a joint embedding space in which either tweets or hashtags are represented.

### 6.2.1 Comparison

The HASHET model, described in this chapter, effectively exploits the state-of-art techniques and recent deep learning architectures for natural language processing, such as embedding models and transformer networks, but follows a different approach. In particular, neural-based solutions frame the hashtag recommendation task as a multi-class classification problem [112–114], using a softmax activation and minimizing a cross-entropy loss. Differently, in HASHET, we reformulated the hashtag recommendation task as a sentence-to-hashtag translation, by learning a mapping from the embedded representation of a sentence to the latent representation of its hashtags. Moreover, these hashtags are jointly represented by a single vector, called target, which can be seen as a summarizing concept about them.

Since semantically similar hashtags lie close together in the embedding space, the recommendation process performed by HASHET exploits a concept of *locality* that relies on the semantic relationships within this space and its underlying topic-based clustering structure. On the contrary, the aforementioned neural models do not consider this structure, thus diluting the output probability among all possible hashtags, through a plain softmax activation. Therefore, being fully aware of the latent space structure, which reflects the semantic distribution of the hashtags in the corpus, HASHET can produce more reliable recommendations consisting of semantically similar hashtags, by leveraging the learned relationships among them and the semantic mapping of the input post into this space.

Moreover, recent works often rely on the construction of a common multi-modal embedding space in which data from multiple modalities (e.g., sentences and hashtags) could be projected. By inspecting this space, relative distances can be measured in order to find the most relevant matching sentence-hashtag pairs. As an example, Kumar et al. [133] proposed a Zero-Shot Learning (ZSL) architecture based on a joint embedding model, where hashtags are projected in the embedding space of the sentences through an end-to-end learning process. Differently, in HASHET, instead of relying on a single multi-modal joint embedding space, we used two independent embedding spaces, i.e., the semantic space where sentences are embedded, and the latent space of hashtags. Then, we learned a translation between them through semantic mapping based on a multilayer perceptron. We also inverted the direction of the projection with respect to the most recent deep embedding models, by learning a semantic mapping from the latent representation of a sentence to the embedding space of its hashtags.

Finally, we introduced two different strategies for semantic expansion, a process that allows the enrichment of the set of recommended hashtags, based on semantic similarity in the hashtag embedding space.

## 6.3 Proposed model: HASHET

The HASHET model is based on the embedded representation of a post in the semantic space $S_{emb}$ and its projection in the latent space of its hashtags $W_{emb}$. The projection is performed by learning a translation between these independent spaces, through semantic mapping based on a multilayer perceptron.

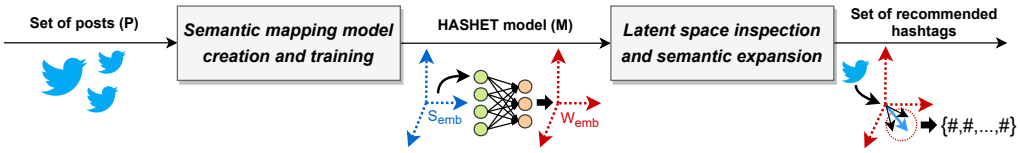



**Figure 6.1:** Execution flow of HASHET.

As shown in Figure 6.1, the execution flow of HASHET consists of two main steps:

1. Semantic mapping model creation and training.
2. Latent space inspection and semantic expansion.

A formal description of each step is provided in the following sections. For the Reader's convenience, Table 6.1 reports the meaning of the main symbols used throughout the sections.

| Symbol | Meaning |
|---|---|
| $P$ | Corpus of posts. |
| $E$ | The pre-trained encoder model (*GUSE* or *BERT*) exploited for sentence embedding. |
| $S_{emb}$ | Sentence embedding space. Dimensionality is 512 for *GUSE* and 768 for *BERT*. |
| $W2V$ | The words/hashtags embedding model, based on CBOW Word2Vec. |
| $W_{emb}$ | 150-dimensional latent space of word embedding. |
| $MLP$ | The mapper $S_{emb} \rightarrow W_{emb}$, based on a Multi-layer Perceptron. |
| $SM$ | The semantic mapping model, obtained by stacking the mapper on top of the encoder. |
| $\mathcal{M}$ | The HASHET model, defined as $\langle W2V, SM \rangle$. |
| $S_{emb}(p)$ | Embedded representation of a post $p$ in $S_{emb}$. |
| $W_{emb}(w)$ | 150-dimensional representation of a word $w$ in $W_{emb}$. |
| $H(p)$ | Set of the hashtags of the post $p$. |
| $target(p)$ | Arithmetic mean of all the $W_{emb}(h), \forall\ h \in H(p)$. |
| $h^*(p)$ | Projection of $S_{emb}(p)$ in the latent space $W_{emb}$. It is the output of $SM$ given $p$ as input. |
| $N^k(h)$ | Ordered set of k nearest hashtags of $h$ in $W_{emb}$. |
| $T^{k,n}(p)$ | Set of top-k recommended hashtags for a post $p$ expanded according to the factor $n$. |

**Table 6.1:** Meaning of the main symbols used throughout the chapter.

### 6.3.1 Semantic mapping model creation and training

HASHET is based on the hidden relationships between the sentences and words/hashtags embedding spaces, $S_{emb}$ and $W_{emb}$, learned by a semantic mapping process based on a multilayer perceptron. The main workflow of this step is shown in Figure 6.2 and described by Algorithm 8. The input of this step is composed of the set of posts $P$ and the selected encoder $E$ exploited for computing a representation of a given post $p \in P$ within the space $S_{emb}$. We tested two different pre-trained models for sentence embedding, described in Section 6.1, namely GUSE and BERT. The output is the HASHET model $\mathcal{M}$.

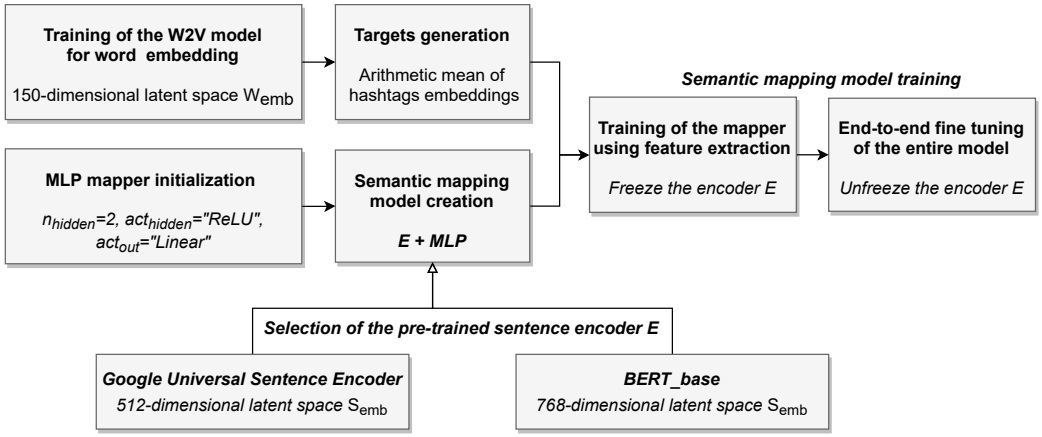



**Figure 6.2:** Training of the *W2V* model for word embedding and target generation. Creation of the semantic mapping model (encoder ($E$) + mapper ($MLP$)) and two-step training: training of the mapper using feature extraction and fine-tuning of the entire model.

The first step of the process (line 1) is to clean up data in order to prepare the corpus $P$ for the embedding process. In particular, the input posts are modified and filtered by using a function $preprocess\_data(P)$, which performs the following operations:

- Posts with no hashtags are removed.
- Posts are cleaned using regular expressions for standardizing the text encoding into UTF-8, solving the problems related to the presence of characters of different encodings, and filtering out URLs.
- The text of each post is normalized by transforming it to lowercase and replacing accented characters with regular ones.
- Words are lemmatized and stemmed for reducing morphological variation.
- Stopwords are removed from the text by using preset lists.
- Bigrams are found in the corpus for better-capturing semantics using the Phrases module of the Gensim library (San Francisco → San_Francisco).

**ALGORITHM 8:** Semantic mapping model creation and training.

**Input :** Set of posts $P$, selected encoder $E$
**Output:** HASHET model $\mathcal{M}$

1 $P \leftarrow preprocess\_data(P)$;
2 /* Word2Vec training and target vectors generation */
3 $W2V \leftarrow Word2Vec.train(P)$;
4 $targets \leftarrow \varnothing$;
5 **for** $p \in P$ **do**
6  $target(p) \leftarrow \varnothing$;
7  **for** $h \in H(p)$ **do**
8  $W_{emb}(h) \leftarrow W2V.embed(h)$;
9  $target(p) \leftarrow target(p) + W_{emb}(h)$;
10  $targets \leftarrow targets \cup \frac{target(p)}{|H(p)|}$;
11 /* Semantic mapping model creation */
12 $E \leftarrow init\_from\_pretrained()$;
13 $MLP \leftarrow init\_from\_scratch(n_{hidden} = 2,\ act_{hidden} = \text{“}ReLU\text{”},\ act_{out} = \text{“}Linear\text{”})$;
14 $SM \leftarrow stack(E, MLP)$;
15 /* Training of the MLP mapper using feature extraction */
16 $SM.E.freeze()$;
17 $opt \leftarrow ADAM(learning\_rate = 1e^{-3})$;
18 $SM.train(x = P,\ y = targets,\ loss = \text{“}cosine\_distance\text{”},\ optimizer = opt)$;
19 /* End-to-end fine-tuning of the entire model */
20 $SM.E.unfreeze()$;
21 $opt \leftarrow ADAM(learning\_rate = 3e^{-5})$;
22 $SM.train(x = P,\ y = targets,\ loss = \text{“}cosine\_distance\text{”},\ optimizer = opt)$;
23 $\mathcal{M} \coloneqq \langle W2V, SM \rangle$;
24 **return** $\mathcal{M}$

Afterward, a Word2Vec model is trained on the pre-processed corpus $P$ following the Continuous Bag-of-Words (CBOW) approach (line 3). Given a certain word $w$ of the corpus, the $W2V$ model outputs a 150-dimensional vector $W_{emb}(w)$, which is a latent representation of the input word in the latent space $W_{emb}$. In this way, Word2Vec is exploited to capture the semantic relationships between hashtags and words, and hashtags themselves. As semantically similar hashtags are used in similar contexts, lying close together in the latent space, this induced clustering structure is exploited by HASHET for increasing its recommendation abilities. After the training of the Word2Vec model, the target vectors for the se-

mantic mapping phase are generated with respect to the embedding learned by the $W2V$ model in the $W_{emb}$ space (lines 4-10). Specifically, an empty list $targets$ is initialized (line 4) and filled with the target vector of each post $p$ of $P$. Given the current post $p$ and the set of its hashtags $H(p)$, the latent representation of each hashtag $h_p$ in $H(p)$, $W_{emb}(h_p)$ is computed (line 8). Then, the target vector for $p$, $target(p)$, is obtained as the arithmetic mean of all $W_{emb}(h_p)$ and added to the list (lines 9-10). It can be seen as a summarizing concept about the hashtags in $H(p)$, and can be written as follows:

$$target(p) = \frac{1}{|H_p|} \sum_{h_p \in H_p} W_{emb}(h_p) \tag{6.1}$$

HASHET is based on the translation between sentences and words/hashtags domain, i.e. the mapping between the latent representation of the entire post in the semantic space $S_{emb}$ and its hashtags condensed in the corresponding target vector embedded in $W_{emb}$. Therefore, a crucial point of the model is the projection of the embedded sentences lying in $S_{emb}$ into the words/hashtags latent space $W_{emb}$. Specifically, this mapping of the semantic vectors is learned using a semantic mapping model $SM$, composed by stacking two main blocks: the pre-trained sentence encoder $E$ and the mapper $MLP$ (line 14). The encoder $E$ is initialized by loading its pre-trained weights (line 12), while the $MLP$ mapper is created from scratch (line 13), by initializing a multi-layer perceptron with two hidden layers. In particular, $\mathcal{H}^{(1)} = 350$ and $\mathcal{H}^{(2)} = 250$ are the number of neurons in the first and the second hidden layer, while the output layer has 150 neurons, as it determines a 150-dimensional vector lying in $W_{emb}$. For what concerns the activation functions we used the Rectified Linear Unit ($ReLU$), defined as $ReLU(x) = x^+ = max(0,x)$, in the two hidden layers and a linear activation ($lin$), defined as the identity function, for the output layer. The $ReLU$ activation was used to introduce non-linearity in the mapping process; this choice was driven also by its interesting properties, such as sparse activation, scale invariance, and efficiency.

Given a post $p \in P$, its semantic representation $S_{emb}(p)$ is computed by the first block of the *SM* model, i.e. the encoder $E$, obtaining a $\sigma$-dimensional semantic representation vector, where $\sigma$ is the dimensionality of the sentence embedding space $S_{emb}$. Then, this latent representation of the input post $p$ is fed to the *MLP* mapper, which outputs a 150-dimensional embedding vector lying in $W_{emb}$. The mapping process is driven by a cosine distance loss aiming at minimizing the angle between the projection into $W_{emb}$ of the semantic vector $S_{emb}(p)$ and the condensed representation of its hashtags, $target(p) \in W_{emb}$. The loss $\mathcal{L}$ can be derived as follows:

$$h_j^{(1)} = ReLU\left(\sum_{i=1}^{\sigma} w_{ij}^{(1)} s_i + b_j^{(1)}\right),\ j = 1,...,\mathcal{H}^{(1)},\ Semb(p) = s_1...s_\sigma \tag{6.2}$$

$$h_j^{(2)} = ReLU\left(\sum_{i=1}^{\mathcal{H}^{(1)}} w_{ij}^{(2)} h_i^{(1)} + b_j^{(2)}\right),\ j = 1,...,\mathcal{H}^{(2)} \tag{6.3}$$

$$out_j = lin\left(\sum_{i=1}^{\mathcal{H}^{(2)}} w_{ij}^{(out)} h_i^{(2)} + b_j^{(out)}\right),\ j = 1,...,150 \tag{6.4}$$

$$\mathcal{L}(S_{emb}(p), target(p)) = cosine\_distance(target(p), out) \tag{6.5}$$

where:

- $h^{(1)}$ and $h^{(2)}$ are the outputs of the first and the second hidden layer, while *out* is the result of the output layer, which determines the 150-dimensional predicted vector.
- $W^{(1)} \in \mathcal{R}^{|S_{emb}| \times \mathcal{H}^{(1)}}$, $W^{(2)} \in \mathcal{R}^{\mathcal{H}^{(1)} \times \mathcal{H}^{(2)}}$, $W^{(out)} \in \mathcal{R}^{\mathcal{H}^{(2)} \times 150}$ are the weights to be learned in the first and the second FC-layer, and the output linear layer respectively.

The training of the *SM* model, implemented in Python using the high-level framework *Keras*[4] with *TensorFlow*[5] back-end, is divided in two steps:

1. *Training of the mapper using feature extraction*. In this step, we used transfer learning for training the *SM* model, by freezing the encoder $E$ (line 16), which means that its weights will not be changed during training. This way, only the mapper *MLP*, composed of the top layers of *SM*, will be trained with the pairs $\langle S_{emb}(p), target(p)\rangle \; \forall p \in P$ (line 18), while the encoder $E$ is used as a feature extractor for computing $S_{emb}(p)$ for a given $p$. We used the ADAM optimizer [62], initialized with the default learning rate $1e^{-3}$ (line 17).

2. *End-to-end fine tuning of the entire model*. After the mapper was trained to convergence, we incrementally adapted the pre-trained features of the encoder $E$ to our translation task. This was achieved by fine-tuning the entire *SM* model on the pairs $\langle p, target(p)\rangle \; \forall p \in P$ (line 22), after having unfreezed the encoder $E$ (line 20). In this step, we used a very low learning rate of $3^{e-5}$ (line 21), as we only want to readapt the pre-trained features to work with our task and therefore large weight updates are not desirable at this stage, which also lowers the risk of overfitting.

At the end of the described process, the algorithm returns the HASHET model $\mathcal{M}$, defined as the pair $\langle W2V, SM\rangle$ (lines 23-24), used for the recommendation step.

### 6.3.2 Hashtags recommendation by latent space inspection and semantic expansion

In this step, the HASHET model, defined as the pair $\langle W2V, SM\rangle$, is used for recommending a consistent set of hashtags for a given post $p$. The different steps involved in this process are shown in Figure 6.3 and described by Algorithm 9.

[4] `https://keras.io/`
[5] `https://www.tensorflow.org/`

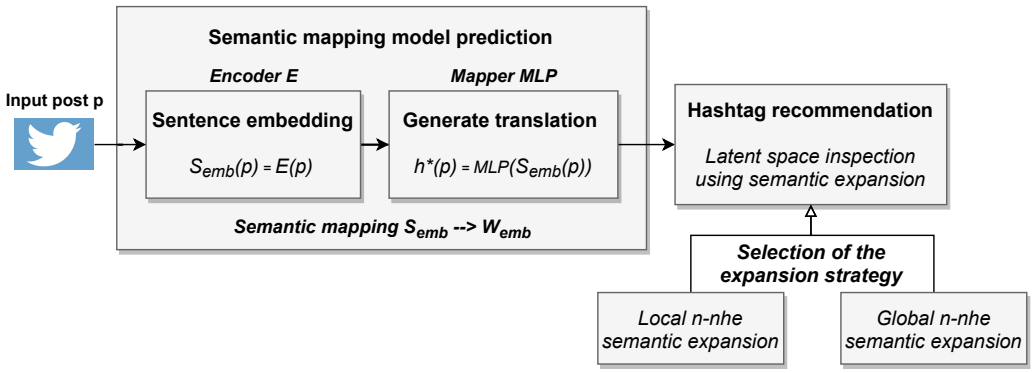



**Figure 6.3:** Hashtag recommendation for a given post $p$, composed of two steps: $i$) Semantic mapping of $p$ exploiting the $SM$ model to obtain the target vector $h^*(p)$; $ii$) latent space inspection using a selected semantic expansion strategy.

The input is composed of: the post $p$, the HASHET model $\mathcal{M}$, the number of hashtags to recommend $k$, the expansion factor $n$, and the expansion strategy $nhe$. The output is the set of recommended hashtags $T^{k,n}(p)$ for input post $p$.

**ALGORITHM 9:** Hashtag recommendation by latent space inspection.

**Input :** post $p$, HASHET model $\mathcal{M} := \langle W2V, SM \rangle$, ranked output limit $k$, expansion factor $n$, expansion strategy $nhe$
**Output:** set of recommended hashtags $T^{k,n}(p)$ for input post $p$

1 $h^*(p) \leftarrow \mathcal{M}.SM.predict(p)$;
2 $T^{k,n}(p) \leftarrow \varnothing$;
3 **if** *nhe is local* **then**
4 $\quad N^k(h^*(p)) \leftarrow \mathcal{M}.W2V.nearest\_hashtags(h^*(p), k)$;
5 $\quad T^{k,n}(p) \leftarrow N^k(h^*(p))$;
6 $\quad$ **for** $h \in N^k(h^*(p))$ **do**
7 $\quad\quad N^n(h) \leftarrow \mathcal{M}.W2V.nearest\_hashtags(h, n)$;
8 $\quad\quad T^{k,n}(p) \leftarrow T^{k,n}(p) \cup N^n(h) \setminus T^{k,n}(p) \cap N^n(h)$;
9 **else if** *nhe is global* **then**
10 $\quad N^{k+n}(h^*(p)) \leftarrow \mathcal{M}.W2V.nearest\_hashtags(h^*(p), k+n)$;
11 $\quad T^{k,n}(p) \leftarrow N^{k+n}(p)$;
12 **return** $T^{k,n}(p)$

Given the input post $p$, the target vector $h^*(p)$ is obtained by using the semantic mapping model $SM$. In particular, when the *predict* function is called (line 1), the encoder block $E$ of $SM$ is exploited for computing the sentence embedding

$S_{emb}(p)$ of the input post $p \in P$, which is then translated into the corresponding 150-dimensional target vector $h^*(p) \in W_{emb}$ using the mapper block *MLP*. After the mapping, an empty set $T^{k,n}(p)$ is initialized (line 2), which will be filled with the recommended hashtags according to the selected expansion strategy (lines 3-11). In particular, if the *Local* strategy is used (line 3), the set $N^k(h^*(p))$ is computed as the top-k nearest neighbors of $h^*(p)$ (line 4) and is assigned to $T^{k,n}(p)$ (line 5). Then, the set $T^{k,n}(p)$ is filled with the nearest hashtags of each hashtag in $N^k(h^*(p))$, removing duplicates (lines 7-8). If instead the *Global* strategy is chosen (line 9), the set $N^{k+n}(h^*(p))$ is computed as the top-(k+n) nearest neighbors of $h^*(p)$ (line 10) and is assigned to $T^{k,n}(p)$ (lines 11). Finally, the algorithm returns the expanded set of vectors $T^{k,n}(p)$, which contains the hashtags recommended by the HASHET model.

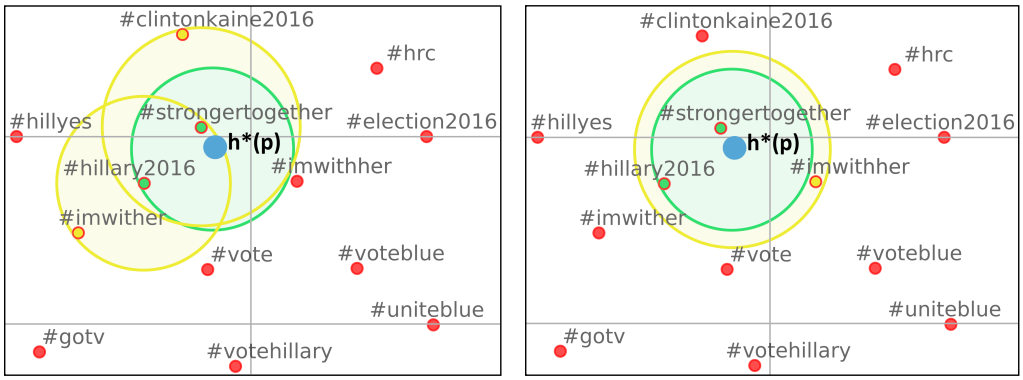
Since there are many semantically related hashtags that are almost interchangeable, as they share the same meaning (i.e., *#trumptrain* and *#maga* or *#imwithher* and *#votehillary*), a semantic expansion based on the n-nearest hashtags (*n-nhe*) has been introduced, in order to capture semantic equivalences and maximize the match with the target hashtags. In particular, two different strategies have been proposed:

- *Local n-nearest hashtag expansion*. Given the expansion factor $n$, the set $N^k(h^*(p))$ is extended with the top-n nearest neighbors of each hashtag it contains.

- *Global n-nearest hashtag expansion*. Given the expansion factor $n$, the set $N^k(h^*(p))$ is extended by considering the top-(k+n) neighbors of $h^*(p)$, obtaining the extended set $N^{k+n}(h^*(p))$.

Therefore, the two strategies are aimed at expanding the set $N^k(h^*(p))$ composed of the k-nearest neighbors of the vector $h^*(p)$ ordered by likelihood, which is defined as the cosine similarity with respect to $h^*(p)$. The local approach can be considered a sort of 2-hop decentralized process, where the distance is measured locally with respect to each hashtag in the non-expanded set. Thus we obtain the $n$-nearest hashtags for each neighbor of $h^*(p)$, where $n$ is the expansion factor.

Differently, when the global strategy is used, the nearest neighbor search process is extended by $n$ steps maintaining the same center ($h^*(p)$). From this derives the adjective global, as every new hashtag vector is included according to its distance from $h^*(p)$, obtaining its ($k$+$n$)-nearest hashtags in the embedding space $W_{emb}$.




(a) local n-nhe (b) global n-nhe

**Figure 6.4:** Local vs. global n-nhe expansion example ($k$=2 and $n$=1).

Figure 6.4 shows an example of how the two strategies work, with $k$=2 and $n$=1. The process starts from the 150-dimensional vector $h^*(p)$, obtained as the output of the mapping, represented by the blue point. Then, the two nearest neighbors of $h^*(p)$ are found obtaining the non-expanded set $N^k(h^*(p))$, containing the points highlighted in green, within the green circle. Starting from this set, the two strategies allow the inclusion of semantically related hashtags as described above, expanding by a factor $n$=1. These additive hashtags are represented by the points highlighted in yellow, within the yellow circles. The final set will be composed of the points located within the green and yellow circles. The resulting enriched set of vectors, referred to as $T^{k,n}(p)$, is the output of HASHET and contains the suggested hashtags. By following this approach, the output set will be composed of semantically similar hashtags reflecting the semantic relationships learned in $W_{emb}$ and the underlying topic-based clustering structure.

### 6.3.3 Why a translation approach? Exploit locality in the hashtag embedding space

In this section, we discuss the main reasons behind the translation approach followed by HASHET. Specifically, the choice of modeling the problem as a translation task is mainly related to how we modeled our target variable, a single 150-dimensional mean-pooled vector. This choice is based on the following assumptions:

**Assumption** $(1)$. Tweets are short and a single tweet is very likely to talk about a few topics (generally one). The same assumption can be found in [134], where authors proposed Twitter-LDA, a topic model specifically designed for Twitter that treats tweets as single-topic.

**Assumption** $(2)$. We assume that the hashtags embedding space is well-formed and the semantic relationships are well expressed inside it, which leads to the emergence of a topic-based clustering structure. In such a space, different hashtags which share semantic context will be highly clustered and will belong to the same topic.

Following the two assumptions above, instead of framing our problem as a multi-label classification, we used a translation-based approach, modeling our target variable using a fixed-length representation. This choice is a good trade-off between efficiency and loss of information, since:

- Using a fixed-length representation (i.e., 150 in our case) is much more efficient with respect to the use of multi-label classification. In fact, as the number of possible hashtags gets larger, the size of the output layer increases together with the number of weights to be learned which leads to a higher cost for the training step.
- The loss of information caused by the use of this kind of representation is acceptable considering the above assumptions. In fact, according to $(1)$, a given tweet is very likely to talk about a few related topics or even just one,

so the hashtags it contains will share semantic context. Moreover, according to (2), the latent representations of these hashtags result highly clustered in the embedding space, and taking their mean as a summarization will result in a vector lying in the same region.

At recommendation time, the system leverages the semantic distribution of latent hashtags relying on *i*) the learned relationships between the target vector and the candidate hashtags, and *ii*) the underlying topic-based clustering structure. Neither of these two factors, condensed in the concept of *locality* introduced by HASHET, is taken into account by neural methodologies that perform a multi-label classification. In addition, the recommendation abilities of HASHET are slightly improved by the semantic expansion process, further mitigating the negative effects of summarization.

## 6.4 Performance evaluation

In this section, we present the experiments carried out using the HASHET model on two different case studies. The first one concerns the 2016 US presidential election, characterized by the rivalry between Hillary Clinton and Donald Trump, while the second is related to the COVID-19 pandemic. In particular, for each case study, we present the following analysis:

- An in-depth analysis of the word embedding space for highlighting the topic-based clustering structure induced by the hashtag distribution.
- An evaluation of performance varying the pre-trained encoder model (GUSE vs. BERT) and the semantic expansion strategy (local vs. global n-nearest hashtags expansion).
- An extensive comparison with the most relevant state-of-art techniques, including generative models, unsupervised models, and attention-based supervised models.

Moreover, we investigated the ability of the HASHET model in discovering the main topic of a given tweet, starting from the set of recommended hashtags.

For evaluating the performance of the proposed model we used three different rank-based metrics: *precision* ($P@k,n$), *recall* ($R@k,n$), and $F_1$-*score* ($F@k,n$). Given a post $p$ and the set of its hashtags $H(p)$ (i.e., the target hashtags), the model outputs the set of recommended hashtags $T^{k,n}(p)$, where $k$ is set equal to $|H(p)|$ and $n$ is the expansion factor. We define a function $rel(t_i,p)$ for the $i^{th}$ recommended hashtag $t_i \in T^{k,n}(p)$ such that $rel(t_i,p) = 1$ if $t_i \in H(p)$ so it is relevant for that post, $rel(t_i,p) = 0$ otherwise. Using this definition, the metrics can be written as follows:

$$P@k,n(p) = \frac{1}{|T^{k,n}(p)|} \sum_{i=1}^{|T^{k,n}(p)|} rel(t_i,p) \tag{6.6}$$

it is the fraction of successfully recommended hashtags among those suggested by the model.

$$R@k,n(p) = \frac{1}{|H(p)|} \sum_{i=1}^{|T^{k,n}(p)|} rel(t_i,p) \tag{6.7}$$

it is the hit rate of the model, or rather the fraction of target hashtags that have been successfully recommended.

$$F@k,n(p) = \frac{2 \times P@k,n(p) \times R@k,n(p)}{P@k,n(p) + R@k,n(p)} \tag{6.8}$$

it is the harmonic mean of precision and recall weighted by a $\beta$ factor (i.e., $F_\beta$, $\beta = 1$). Furthermore, in our experiments, all tweets are grouped into five subsets, in relation to $|H(p)|$ (we impose the cardinality $k$ of the non-expanded set $N^k(h^*(p))$ equal to $|H(p)|$) and scores are shown in relation to the increment of $n$. When $n$ is equal to zero, no expansion is performed on neighbors, so $T^{k,n}(p)$ is equal to $N^k(h^*(p))$.

### 6.4.1 The 2016 US presidential election

In this section, we present the analysis carried out using HASHET on a corpus of about 2.5 million tweets, posted by 521,291 users regarding the 2016 US elections, published from October 10, 2016 to November 7, 2016. Our analysis focused on data collected for ten US swing states, i.e., Colorado, Florida, Iowa, Michigan, Ohio, New Hampshire, North Carolina, Pennsylvania, Virginia, and Wisconsin. As already discussed in the previous chapters, swing states are characterized by high political uncertainty, so they have been chosen in this analysis to capture a balanced corpus of posts with respect to the main topics of discussion, related to the support for the two candidates Hillary Clinton and Donald Trump.

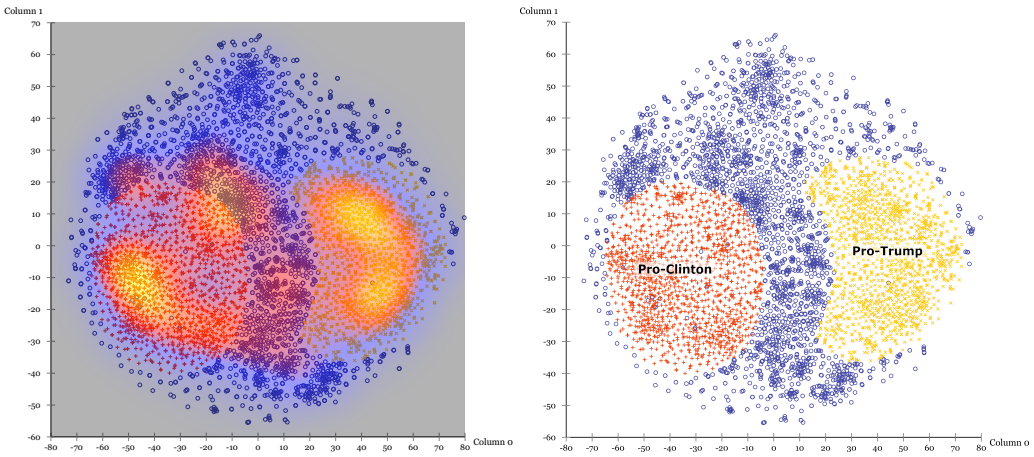
The words/hashtags embedding space $W_{emb}$ was obtained by training a CBOW Word2Vec model on the overall corpus, while the semantic mapping model $SM$ has been trained on a subset of 13,050 tweets published in New Hampshire: 9,787 of them have been used for learning the semantic mapping, while the remaining 3,263 make up the test set. We grouped test tweets in five classes according to the number of hashtags (from 1 up to 5) removing those containing more than 5 hashtags (115 tweets) for reducing noise. The obtained test set was composed of 1637, 780, 439, 202, and 90 tweets, with 1,2,3,4, and 5 hashtags respectively (3,148 in total). The average number of hashtags per tweet is equal to 2 ($weighted\ avg. = 1.83$), in line with Twitter guidelines which recommend using no more than 2 hashtags per tweet as best practice[6].

#### Word embedding space analysis

A peculiar characteristic of microblogging posts on Twitter is that of associating the most common hashtags to a topic. So the hashtag projection of the semantic space of the word embeddings is expected to be highly clustered around the main discussion topics. In the following, we show a series of representations of the latent space $W_{emb}$ obtained by training the Word2Vec CBOW model, by following the same process for topic discovery presented in Section 3.1.

[6] `https://help.twitter.com/en/using-twitter/how-to-use-hashtags`

Since the 150-dimensional latent space $W_{emb}$ can not be directly plotted, we firstly performed a dimensionality reduction using t-distributed stochastic neighbor embedding [135], initialized through principal component analysis (PCA + t-SNE), to obtain a 2D representation of $W_{emb}$. Then, in order to identify dense groups of hashtags, we retained only the hashtags among the totality of latent representations, filtering out those with a frequency lower than 20. The resulting 2-dimensional latent space counts almost 5,000 hashtag points. Then, the OPTICS cut clustering algorithm [136] has been used to identify density-based clustering structures in this space and the results are shown in Figure 6.5.




(a) Density Estimation Overlay (b) OPTICS Cut Clustering

**Figure 6.5:** OPTICS density-based cut-clustering structure of most frequent hashtags in the 2-dimensional representation of $W_{emb}$ obtained through PCA + t-SNE.

The cut-clustering algorithm was able to detect, consistently with the density estimation overlay (Figure 6.5(a)), two macro-clusters related to hashtags used for supporting the two major candidates Hillary Clinton and Donald Trump (Figure 6.5(b)). These clusters can be seen as the two macro-topics underlying the entire corpus. This topic-based separation of hashtags induced by the projection of their latent semantic distribution can be seen better in Figure 6.6. The proposed scatter plot shows the 2-dimensional latent representation of the top three most frequent hashtags for the two candidates and their nearest neighbors.

The considered hashtags are:

- Trump (yellow): *#maga, #trumptrain, #draintheswamp*
- Clinton (red): *#imwithher, #nevertrump, #strongertogether*

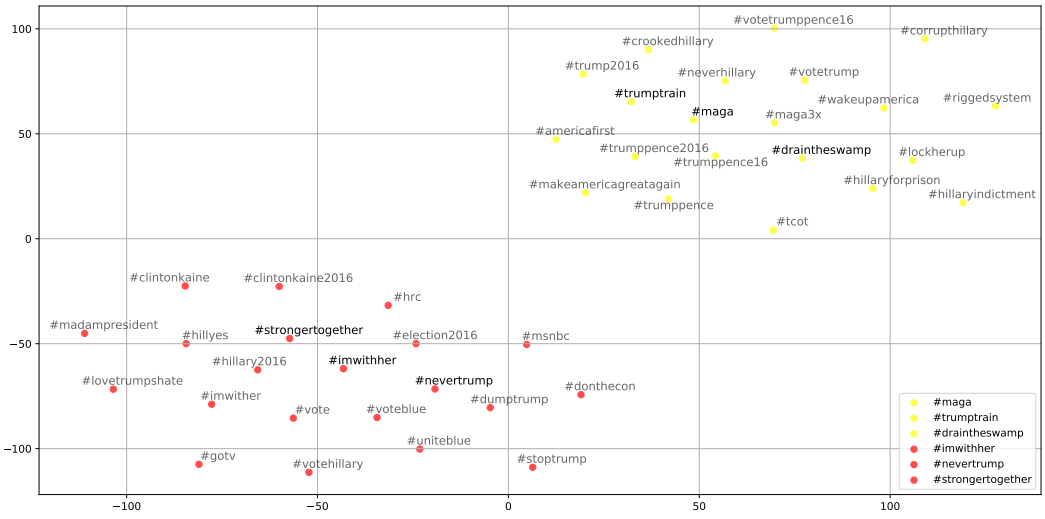
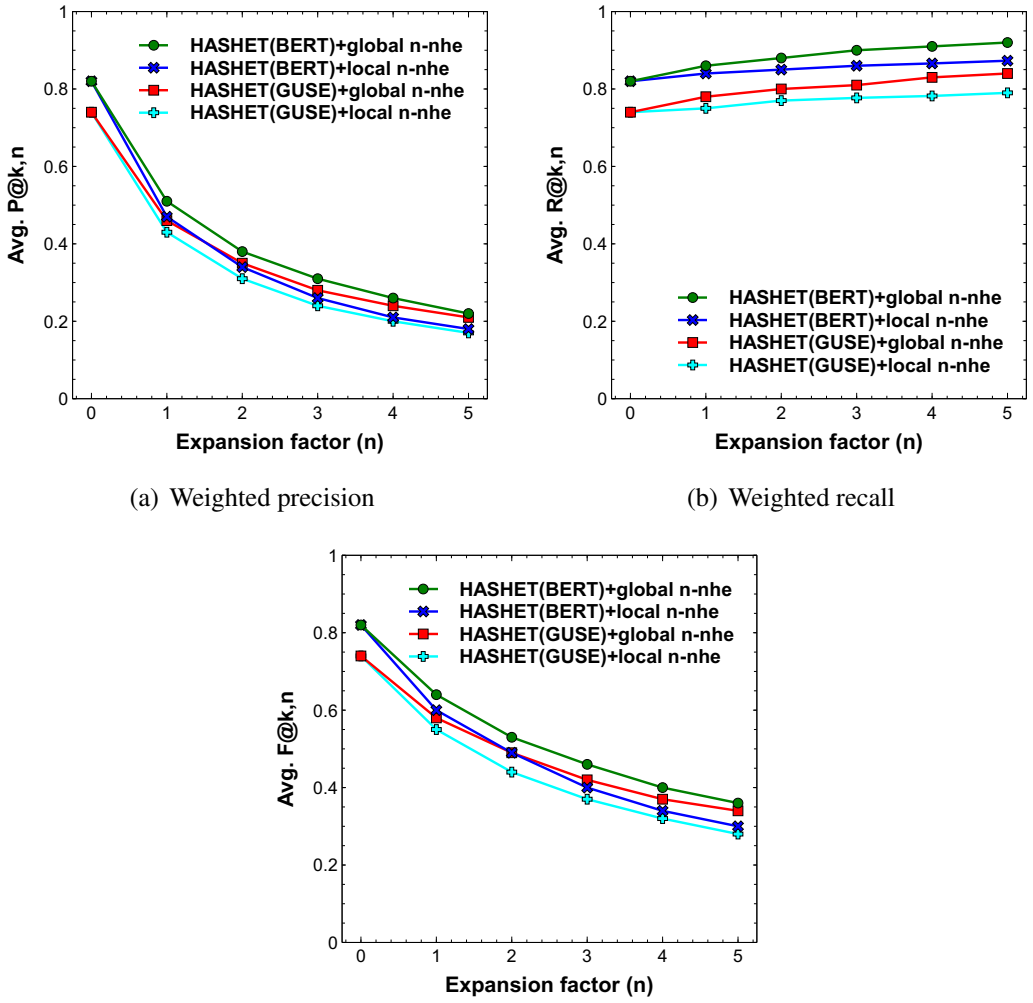



**Figure 6.6:** Top 3 most frequent hashtags per candidate with their nearest neighbors.

The plot shows clearly the separation of hashtags according to their political polarization, which reflects the underlying topic structure identified by the clustering algorithm.

### Encoding models and expansion strategies comparison

In this section we evaluated the use of different encoding models (*GUSE* vs. *BERT*) and semantic expansion strategies (*local* vs. *global*), showing the effects on the rank-based metrics. Figure 6.7 shows the benefits coming from the combined use of the BERT encoder and the global strategy, in terms of weighted precision, recall and F-score. Weighted averages are determined with respect to the scores achieved with different values of $k$ (ranging from 1 to 5) and shown in relation to the increment of $n$ (ranging from 0 (i.e., no expansion) to 5).




(a) Weighted precision

(b) Weighted recall

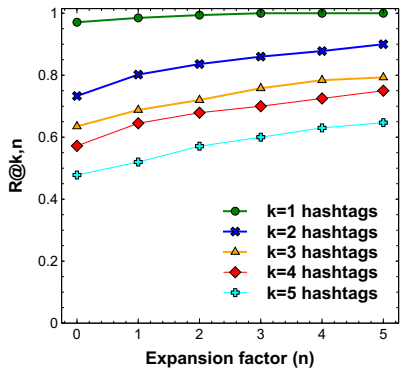
(c) Weighted F-score

**Figure 6.7:** Comparison of the two encoders (GUSE vs. BERT) and the two expansion strategies (global vs. local), in terms of precision, recall, and F-score, weighted on $k$ (number of target hashtags), varying $n$ (expansion factor).

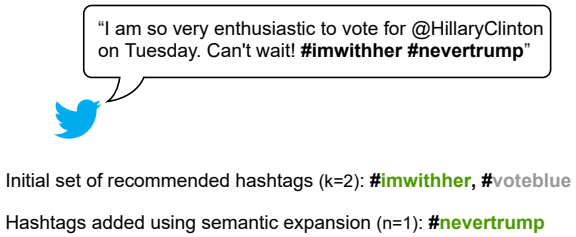
We can observe that, for both semantic expansion strategies, the use of BERT leads to better recommendation results, which means that it can better grasp, compared to GUSE, the semantic aspects of a given tweet, producing more representative embeddings. On the other hand, for both encoders, the global expansion performs better than the local approach. This behavior is due to the higher importance given to the translation $h^*(p)$, which allows the global strategy to better

exploit semantic relationships in the words/hashtags embedding space, by inspecting it with respect to a fixed center ($h^*(p)$). Differently, the local approach can lose this kind of information, focusing on the neighborhood of the top-k hashtags belonging to the non-expanded set $N^k(h^*(p))$. On the basis of these results, we selected the BERT encoder and the semantic expansion strategy as the best configuration to be used in the following experiments.

Afterward, we analyzed the effects of global semantic expansion on the performance of HASHET in terms of $R@k,n$, which measures the recommendation hit rate of the model (Figure 6.8(a)). Firstly, we observed that the recommendation hit rate depends on the number of target hashtags ($k$), and decreases for increasing values of $k$. This is a common behavior among rank-based recommendation systems, where the difficulty in recommending (or retrieving) a group of items increases as the cardinality of the target set gets larger. Moreover, the plot shows how the expansion mechanism allows the model to recommend a more rich set of hashtags, by including additional ones that share semantic context with those contained in the non-expanded set. This aspect can be seen better in Figure 6.8(b), where we show an example of a recommendation for a given tweet with two target hashtags: *#imwithher* and *#nevertrump*. The first target hashtag (*#imwithher*) is found among the top-$k$ hashtag initially recommended, while the second (*#nevertrump*) is obtained through semantic expansion with $n = 1$.




(a) Hit rate for different values of k




(b) Recommendation example ($k$=2, $n$=1)

**Figure 6.8:** Effects of semantic expansion on hit rate for different values of $k$, jointly using BERT and global n-nhe, with a recommendation example ($k$=2, $n$=1).

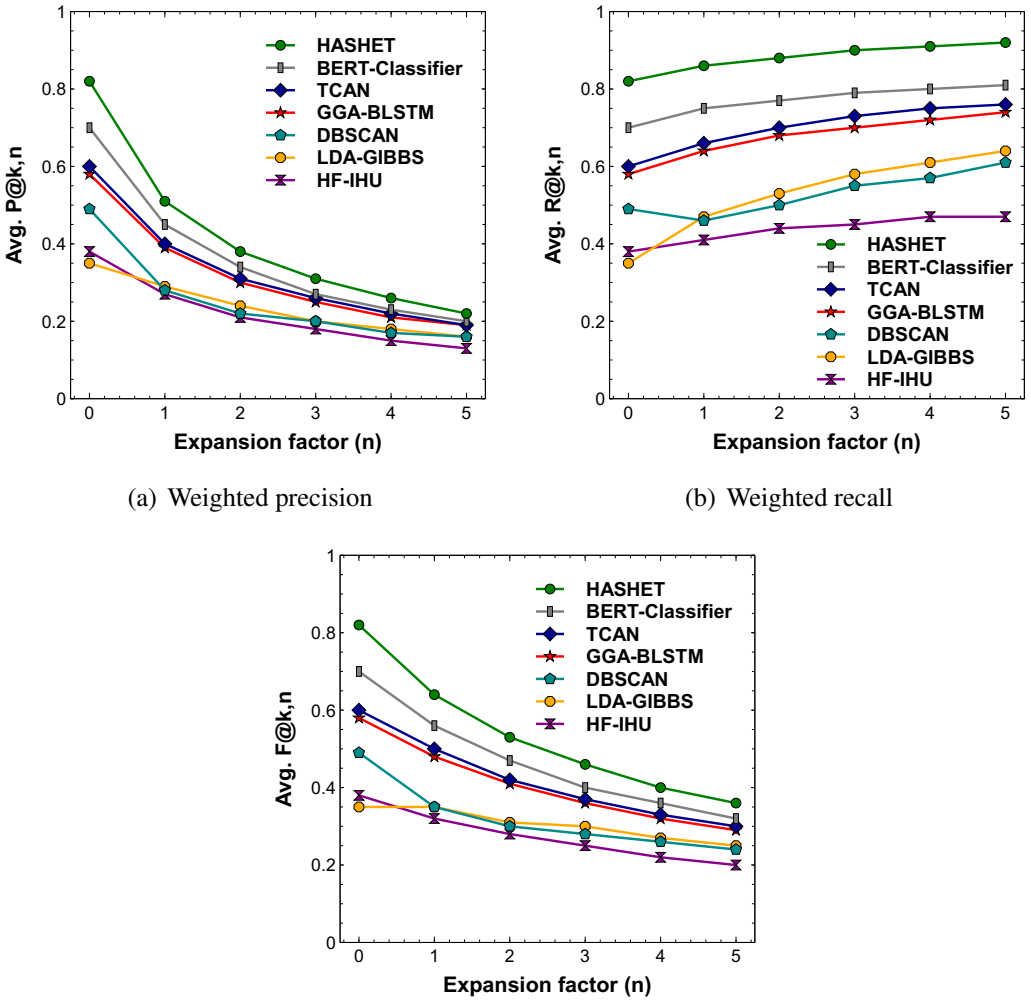
### Comparison to other methods

In order to evaluate the accuracy of HASHET, in both recommending a consistent set of hashtags and detecting the correct hashtag-based polarization of a given post, we carried out an extensive comparison with the most relevant techniques used in the literature:

- Generative models:
  - LDA-GIBBS [110]. This method exploits the Latent Dirichlet Allocation and Gibbs sampling for finding out the underlying topic distribution, used for recommending general hashtags.
- Unsupervised models:
  - DBSCAN [124]. This method is based on the embedded representation of Twitter microblog posts and performs the following steps: i) a given post is represented as the weighted average of its word embeddings; ii) latent representations of posts are clustered according to their syntactic and semantic similarity using a density-based approach; iii) top-k hashtags are found by computing the similarity between the entered post and the centroids of the obtained clusters.
  - HF-IHU [126]. The authors proposed a hashtag recommendation system for Twitter data streams based on a novel ranking scheme, the Hashtag Frequency-Inverse Hashtag Ubiquity (HFIHU). It consists of a variation of TF-IDF that considers hashtag relevancy and microblog data sparseness.
- Supervised models:
  - TCAN [113]. This method exploits an attention-based neural network to learn the representation of a microblog post. Specifically, the authors proposed a novel Topical Co-Attention Network (TCAN) that models content-based and topic-based attention simultaneously.

- GGA-BLSTM. It consists in a degenerate version of the aforementioned TCAN model, which takes into account only the content in the attention mechanism. It can be seen as a standard Bi-directional LSTM model enhanced with global general attention [130].
- BERT-Classifier. It consists of a fully fine-tuned BERT classifier obtained by stacking a softmax layer on top of the BERT-base transformer encoder. Since we are coping with an extremely sparse input, whereby only a few hashtags than those possible (on average two) are actually present in a single tweet, we have configured the model as follows. It was trained using the cross-entropy loss and each target vector has been normalized by scaling it with a factor $1/h$, where $h$ is the number of hashtags in the related post. This solution, already used in other works ([112, 137]), led to better performances for such a sparse input. We experimentally evaluated this aspect by testing BERT with a classical per-hashtag sigmoid output and a binary logistic loss, obtaining a significant performance degradation.

Figure 6.9 shows the results obtained by HASHET in comparison with the other related techniques for the hashtag recommendation task, in terms of weighted precision ($P@k,n$), recall ($R@k,n$) and F-score ($F@k,n$). Weighted averages are determined with respect to the scores achieved by each technique with different values of $k$ (ranging from 1 to 5) and shown in relation to the increment of $n$, ranging from 0 (no expansion) to 5. As explained in Section 6.4.1, we imposed the cardinality $k$ of the non-expanded set $N^k(h^*(p))$ equal to $|H(p)|$, that is the number of target hashtags, while $n$ is the expansion factor. When $n$ is equal to zero, any expansion is performed on neighbors, so $T^{k,n}(p)$ is equal to $N^k(h^*(p))$. We used the global n-nhe strategy for semantic expansion in HASHET, adapting this expansion strategy to the other techniques. In general, given $k$ equal to $|H(p)|$ and $n \geqslant 0$, every model outputs the top-($k$+$n$) hashtags.




(a) Weighted precision

(b) Weighted recall

(c) Weighted F-score

**Figure 6.9:** Comparison with the most relevant related works, in terms of precision, recall, and F-score, weighted on $k$ (number of target hashtags), varying $n$ (expansion factor).

Compared to the aforementioned techniques, HASHET turned out to be the most accurate in recommending the target hashtags, outperforming all state-of-the-art models in all configurations, in terms of precision, recall, and F-score.

Specifically, during the evaluation, we found out what follows. By considering the comparison with methods based on HF-IHU, DBSCAN, and LDA-Gibbs, we observed that HF-IHU performs worse than the others. This indicates that the clustering structure of tweet embeddings learned by the unsupervised approach as well as the topic structure identified by the generative model, capture more semantic information than the simple frequency-based scoring technique, leading to more representative suggested hashtags.

In comparing these more traditional techniques (i.e., HF-IHU, DBSCAN, and LDA) to the attention-based models based on neural networks, we observed a significant improvement in recommendation accuracy. The main reason behind the higher performance of neural models is the ability to learn an accurate latent representation of the microblog, rich in semantic information, also exploiting the attention mechanism. In particular, the comparison between GGA-BLSTM and TCAN shows that topic information is useful in learning this kind of representation. For this reason, the topical co-attention model achieved slightly better performance with respect to the GGA-BLSTM, by jointly modeling content attention and topic attention simultaneously. Furthermore, we noticed that the fine-tuned BERT classifier achieved even more accurate results, in line with the most recent improvements due to transfer learning with pre-trained language models in a broad set of NLP tasks [15].

Moreover, it is clear to observe that our model outperformed both traditional and attention-based models. Similarly to neural models, we exploited a semantic representation of the microblog, generated in our case by a transformer-based deep sentence embedding model. The key difference is in how this representation is used to recommend hashtags. In neural models, a softmax layer is generally used to output a probability distribution over all candidate hashtags. Then, top-k hashtags ordered by decreasing probability are recommended. Differently, in HASHET, starting from the latent representation of a post in $S_{emb}$ the target vector $h^*(p)$ in the hashtag space $W_{emb}$ is predicted using the neural-based semantic mapping. Then, the top-k nearest hashtags of $h^*(p)$ are found and enriched using semantic expansion, obtaining an output set composed of semantically similar

hashtags. This kind of inspection process, centered in $h^*(p)$, exploits a concept of locality in $W_{emb}$ that relies on the semantic relationships learned among hashtags and the underlying topic-based clustering structure.

It is also worth noting that HASHET is less dependent on tuning and parameters with respect to the majority of other techniques. The LDA-Gibbs model and the topical co-attention network are sensitive to the number of topics and the number of topical words for each topic. These two parameters control, in the two models respectively, the topic discovery process and the topic-based information used in the attention mechanism. A wrong setting of these parameters could lead to the identification of a poorly representative topic structure or the introduction of noise in topical information. Another parameter-sensitive technique is the density-based clustering of the embedded representation of training tweets. This model uses the DBSCAN algorithm that is highly dependent on $min_{pts}$ and $\varepsilon$ parameters. A wrong estimate of $min_{pts}$ or $\varepsilon$ could lead to the identification of an unrepresentative clustering structure, which hinders the recommendation performances of the model.

### 6.4.2 COVID-19 pandemic

After the presentation of the 2016 US presidential election, here we discuss the application of HASHET to a corpus of 704,867 tweets regarding the COVID-19 pandemic [138], published from December 23 to December 27, 2020.

As in the first case study, the words/hashtags embedding space $W_{emb}$ was obtained by training the CBOW Word2Vec model on the overall corpus, while the semantic mapping model *SM* has been trained on a subset of 24,903 tweets: 18,678 of them have been used for learning the semantic mapping, while the remaining 6,225 tweets have been used as the test set. Covid-related tweets have been grouped into five classes according to the number of hashtags (from 1 up to 5) removing those containing more than 5 hashtags (196 tweets) for reducing noise. The obtained test set was composed of 3011, 1591, 764, 375, and 288 tweets, with 1,2,3,4 and 5 hashtags respectively (6,029 in total) and an average number of hashtags per tweet equal to 2 ($weighted\ avg. = 1,89$).

Following the same approach for topic discovery described in the previous case study (see Section 6.4.1), we analyzed the topic-based separation of latent hashtags in the $W_{emb}$ space. We extracted a well-formed clustering structure, composed of 13 hashtag-based topics, as shown in Figure 6.10. In addition, Table 6.2 shows the top-5 most frequent hashtags for each cluster.

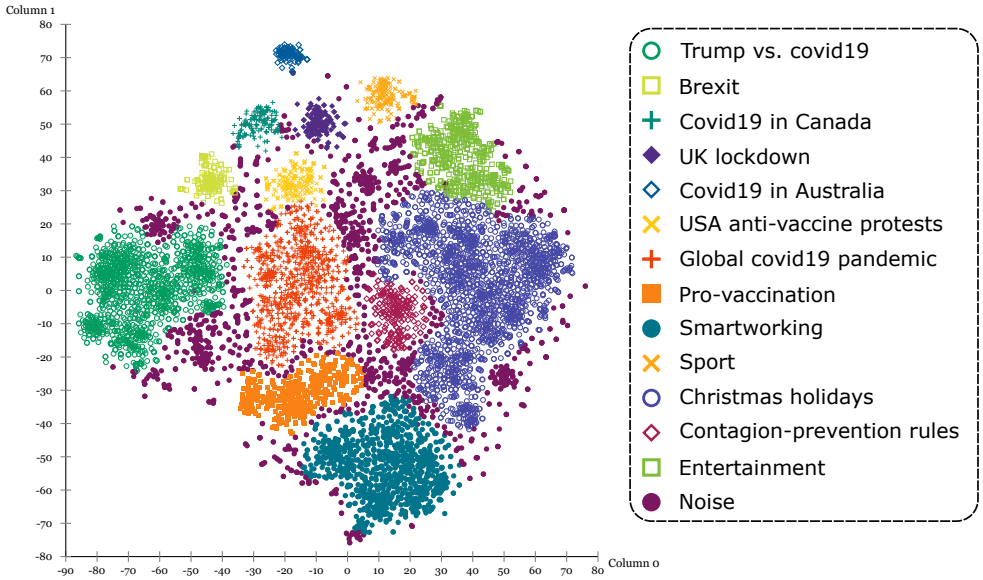
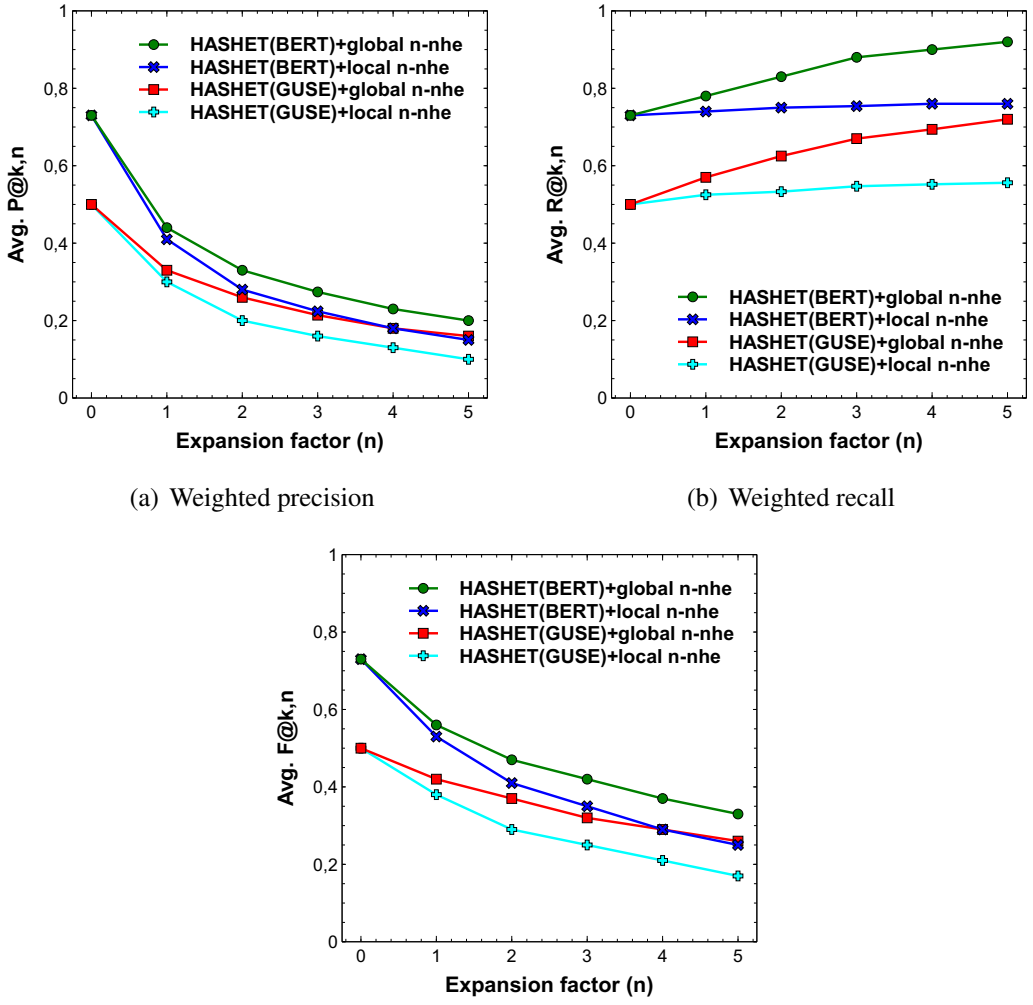



**Figure 6.10:** OPTICS density-based cut-clustering structure in the 2-dimensional representation of $W_{emb}$ obtained through PCA + t-SNE.

| Topic | Top-5 most frequent hashtags |
|---|---|
| *Global covid19 pandemic* | #covid19, #coronavirus, #covid_19, #coronaviruspandemic, #travel |
| *USA anti-vaccine protests* | #losangeles, #california, #protests, #2019ncov, #florida |
| *Christmas holidays* | #christmas, #merrychristmas, #christmas2020, #covidchristmas, #christmaseve |
| *Entertainment* | #wonderwoman1984, #ww84, #starwars, #lightsforlouis,#happybdaylouistomlinson |
| *Contagion-prevention rules* | #wearamask, #stayhome, #staysafe, #socialdistancing, #washyourhands |
| *Smartworking* | #workfromhome, #jobs, #business, #wfh, #remotejobs |
| *Pro-vaccination* | #vaccine, #covidvaccine, #healthcare, #covid_19, #sarscov2, #frontlineheroes |
| *UK lockdown* | #covid19uk, #tier4, #coronavirusuk, #londonlockdown, #uklockdown |
| *Covid19 in Canada* | #canada, #cdnpoli, #onpoli, #bcpoli, #ontariolockdown |
| *Sport* | #browns, #nba, #nfl, #football, #rockets |
| *Trump vs. covid19* | #trump, #trumpvirus, #republicans, #foxnews, #dopeydon |
| *Brexit* | #brexit, #brexitdeal, #wearenotgoingaway, #borishasfailedthenation, #boristheliar |
| *Covid19 in Australia* | #7news, #sydney, #covid19aus, #covid19vic, #gladyscluster |

**Table 6.2:** Top-5 most frequent hashtags per topic.

As for the first case study, we analyzed the performance of HASHET varying the pre-trained encoder model (GUSE vs. BERT) and the semantic expansion strategy (local vs. global n-nhe). The results, shown in Figure 6.11, confirm the benefits coming from the combined use of BERT and global semantic expansion.



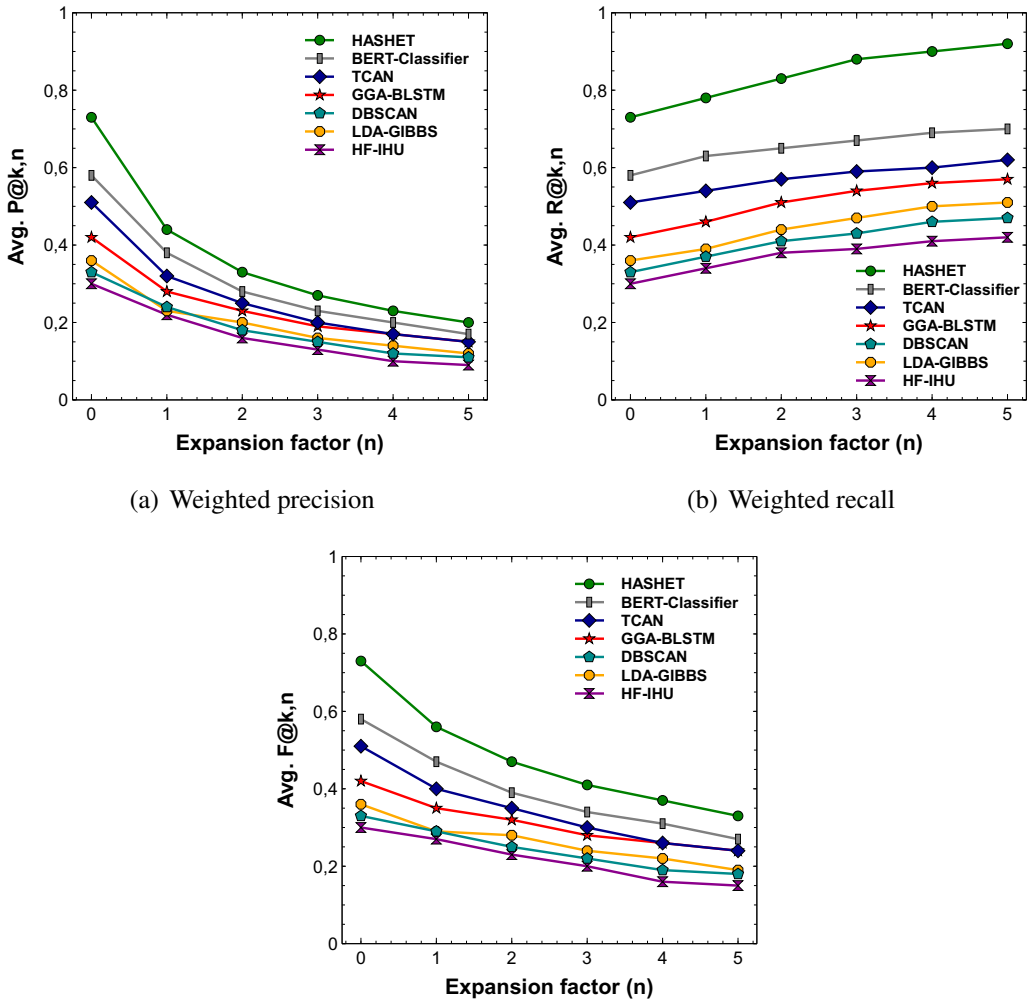

(a) Weighted precision

(b) Weighted recall

(c) Weighted F-score

**Figure 6.11:** Comparison of the two encoders (GUSE vs. BERT) and the two expansion strategies (global vs. local), in terms of precision, recall, and F-score, weighted on $k$ (number of target hashtags), varying $n$ (expansion factor).

Also in this case study, our model outperformed the most relevant related techniques described in Section 6.4.1, achieving the best recommendation results in all configurations, as shown in Figure 6.12.




(a) Weighted precision

(b) Weighted recall

(c) Weighted F-score

**Figure 6.12:** Comparison with the most relevant related works, in terms of precision, recall, and F-score, weighted on $k$ (number of target hashtags), varying $n$ (expansion factor).

By analyzing the experimental results, we observed that even if the performances achieved by the different techniques are characterized by trends similar to the previous case study, this case is intrinsically more difficult than the first one. The main reason behind this difference is the presence of a larger set of discussion topics, which leads to a more variegate set of hashtags. In such a scenario, the topic-based techniques, like LDA and TCAN, performed slightly better, albeit in proportion to the greater difficulty, thanks to the presence of richer topic information and their ability to effectively exploit it. The HASHET model also benefited from this aspect, as it exploits locality in the hashtag embedding space, which presents a well-formed topic-based clustering structure.

Compared to the other related techniques, HASHET turned out to be the most effective model for the hashtag recommendation task, outperforming either traditional techniques or neural-base models based on different types of attention mechanisms, such as topical co-attention, general global attention, or self-attention. These promising results fully confirm those achieved in the first case study and the effectiveness of HASHET, even in the presence of a variegated set of discussion topics, ranging from covid19 and vaccination to sport, entertainment, and socio-political issues like Brexit.

### 6.4.3 Assign topics using hashtag recommendation

In the previous sections, we showed how the semantic distribution of hashtags in the corpus generates a topic-based clustering structure in the hashtag embedding space $W_{emb}$ used by HASHET. Through this hashtag-based approach, the main topics underlying the analyzed corpus can be extracted, isolating densely-connected groups of latent hashtags. In this section, we investigate how the semantic mapping performed by the HASHET model can be leveraged to assign a topic to a given input tweet, assuming it is mono-topic and related to one of the topics that emerged in $W_{emb}$. Given $\mathcal{C} = c_1, \ldots, c_n$ the set of hashtag-based clusters, each of which represents a topic, the discussion topic for a given post $p$ is determined as follows. First, the recommendation model is exploited to get a set of $k$ hashtags suitable for that post.

Afterward, $p$ can be classified in three ways:

- *Assignable*, if recommended hashtags belong to one cluster only.
- *Ambiguous*, if recommended hashtags belong to two or more clusters.
- *Neutral*, if recommended hashtags do not belong to any cluster.

Then, if $p$ was labeled as assignable, it is assigned to the topic related to its corresponding cluster. Otherwise, if $p$ is labeled as neutral, i.e. none of the recommended hashtags belongs to one of the clusters in $\mathcal{C}$, semantic expansion is used to recommend $n$ additional hashtags and re-classify the post. This process is repeated until the post $p$ is classified - either as ambiguous or assignable - or a maximum number of steps is reached. In this case, it is finally classified as neutral. It is worth noticing that posts classified as ambiguous are not assigned to any specific topic, as we only focus here on mono-topic posts.

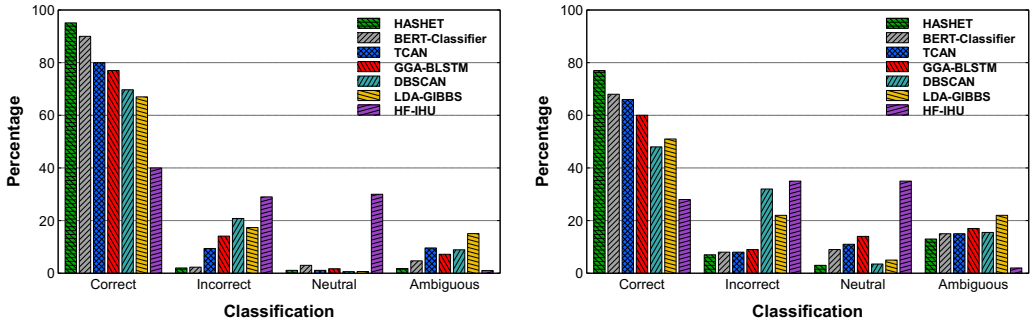
For testing purposes, we analyzed the corpus of tweets related to both the aforementioned case studies, characterized by discussion topics related to politics (US election, Clinton vs. Trump) and the COVID-19 pandemic, as described in the previous sections. Specifically, we determined the actual topic of each considered post, starting from its set of hashtags ($H(p)$). This topic, i.e. the assigned cluster, is expected to be equal to the classification computed with respect to the recommended hashtags for that tweet ($T^{k,n}(p)$); otherwise, the topic assignment is incorrect. This is an undesired situation, especially when dealing with politically-polarized data, as the post may be considered in favor of the opposite candidate. In addition, as an incorrect classification implies the assignment to a topic different from the real one, ambiguous and neutral posts - for which a topic could not be found - are considered separately.

We evaluated the effectiveness of using the recommendation abilities of the HASHET model for the topic assignment task through a comparison with the aforementioned related techniques. Comparing the HF-IHU, LDA-Gibbs, and DBSCAN-based models, the first achieved very poor results, with the lowest amount of correctly classified tweets and a large amount of incorrect and neutral

tweets, while the LDA-Gibbs and the DBSCAN-based models achieved better results, similar to each other, showing their ability in detecting an underlying topic and clustering structure respectively.

The attention-based neural models (GGA-BLSTM, TCAN, BERT-Classifier) achieved higher performances, thanks to the ability to learn a representative embedding of the analyzed microblogs, capturing a lot of semantic information. In particular, TCAN showed slightly better performances with respect to the GGA-BLSTM, exploiting the topical information within the co-attention mechanism. Moreover, the BERT-Classifier outperformed both TCAN and GGA-BLSTM models, thanks to a better understanding of the semantic content of a given tweet, which confirms the effectiveness of transfer learning from language representation models.

Finally, HASHET outperformed all the other recommendation models in the topic assignment task, achieving both the highest percentage of correct and the lowest amount of incorrect classifications. These results, detailed in Figure 6.13, confirm the ability of our model to determine a highly representative set of hashtags, which is beneficial both for the recommendation itself and for a wide range of alternative applications, such as the topic assignment - presented in this section - or the enrichment of social data analyzed by hashtag-based methodologies, discussed in Section 1.1.




(a) USA 2016: polarization discovery

(b) COVID19: topic discovery

**Figure 6.13:** Comparison with the most relevant related work in detecting the hashtag-based topic of discussion.

## 6.5 Conclusions

This chapter described HASHET, a hashtag recommendation model aimed at suggesting a proper set of hashtags for social media posts by leveraging a translation approach. Specifically, it exploits a semantic mapping process to model the hidden relationships that link a given post to the latent representation of its hashtags, learning how to map the embedded representation of a post into a vector lying in the hashtag space. Starting from this vector, the set of hashtags to be recommended is found using k-nearest neighbor search and semantic expansion. Therefore, unlike other state-of-the-art models based on deep learning architectures, HASHET is fully aware of the semantic distribution of hashtags in the embedding space. Due to this, the recommendation process relies on the concept of locality in this space that takes into account both the learned relationships among hashtags and its underlying topic-based clustering structure.

We assessed the effectiveness of HASHET through an extensive experimental evaluation focused on two real-world case studies related to the 2016 United States presidential election and the COVID-19 pandemic. In particular, we experimented with the use of two language models for sentence embedding and two different semantic expansion strategies, finding out that the combined use of BERT and global nearest-hashtag expansion leads to the best recommendation results, with a hit-rate up to 0.92. HASHET also outperformed the main hashtag recommendation techniques present in the literature (i.e., generative models, unsupervised models, and attention-based supervised models), with an up to 15% improvement in F-score. Finally, We analyzed the applicability of HASHET in multilingual contexts, discussing also how its recommendation abilities can be effectively leveraged for the topic assignment task, by achieving a 9% improvement compared to the other techniques.

In future work, we can adapt HASHET to work in real-time scenarios, to deal with the continuous evolution of hashtags in microblogging platforms. Moreover, we can investigate ways to tailor the recommendation process to specific tasks such as recommending personalized hashtags, based on user characteristics, or recommending hashtags aimed at maximizing post popularity.

Chapter 7

# Using machine learning for task scheduling in data-intensive parallel workflows

The applications of Big Data are countless, ranging from politics to economics, finance, healthcare, security, and social sciences. In particular, as shown in the previous chapters, the analysis of politically-polarized Big Social Data from different perspectives allows for achieving a thorough understanding of users' behavior, opinion, emotions, and interactions, thus providing a data-driven approach to understanding and studying political phenomena. However, as highlighted in Section 1.1, despite the great opportunities offered by the analysis of Big Data, their volume and speed continually challenge today's storage, processing, and analysis capabilities. Therefore, state-of-the-art tools for analyzing and learning from Big Data on scalable computers, including the main parallel programming paradigms (e.g., MapReduce, workflow, BSP, message passing, and SQL-like), and the most used systems for Big Data analysis (e.g., Apache Spark, Hadoop, and Storm) are expected to be constantly improved to effectively tackle Big Data issues [10].

Motivated by the above considerations, our research also focused on the study of ad-hoc techniques and strategies aimed at supporting the scalable implementation of real-world Big Data analysis applications, generally represented as data-intensive high-parallel workflows, in distributed environments and high-performance infrastructures. Our efforts in this direction led to the production of two interesting research results, in the following fields:

- *Workflow task scheduling*: in [11] we proposed *IIWM (Intelligent In-memory Workflow Manager)*, a methodology aimed at maximizing the trade-off between task-parallelism and memory usage, by minimizing the risk of data spilling-to-disk. It relies on a machine learning strategy for predicting memory occupancy and execution time of workflow tasks and leverages these predictions to determine an effective task schedule. The effectiveness of the machine learning-based predictor and the scheduling strategy were demonstrated experimentally using as a testbed, Spark, a high-performance Big Data processing framework that exploits in-memory computing to speed up the execution of large-scale applications.

- *Data partitioning*: in [12] we designed a machine learning-based approach to determine an effective data partitioning, which is crucial to speed-up parallel data-intensive applications and increase scalability. We framed the data partitioning task as a block size estimation problem, maximizing the trade-off between the degree of parallelism and the communication/synchronization overhead. This research will be discussed in the next chapter, which accurately describes the design of our solution and the main results achieved in different execution environments by using several real-world datasets.

The rest of this chapter will be devoted to the first of the research results mentioned above and is organized as follows. Section 7.1 introduces the main concepts and definitions related to the addressed problem. Section 7.2 discusses the main works in the literature aimed at improving the performance of data-intensive applications. Section 7.3 describes the proposed methodology. Section 7.4 presents and discusses the experimental results. Finally, Section 7.5 concludes the paper.

## 7.1 Background

A data-intensive workflow is the description of a process that usually involves a set of computational steps implementing complex scientific functions, such as data acquisition, transformation, analysis, storage, and visualization [139]. They can be defined as follows.

**Definition** [140]. A workflow $\mathcal{W}$ can be represented using a DAG, described by a set of tasks $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ (i.e., vertices) and dependencies among them $\mathcal{A} \subseteq (\mathcal{T} \times \mathcal{T}) = \{a_1, \ldots, a_m\}$: $a_i = (t_i, t_j), t_i \in \mathcal{T}, t_j \in \mathcal{T}$ (i.e., directed edges). Specifically, data dependencies (i.e., all the input data of a task have already been made available) have to be considered rather than control dependencies (i.e., all predecessors of a task must be terminated before it can be executed), as we refer to data-intensive workflows.

Parallel processing is often vital to reduce execution time when complex data-intensive workflows must be run efficiently, and at the same time, in-memory processing can bring important benefits to accelerate execution. It can be achieved by concurrently executing independent tasks by trying to make use of all computing nodes, even if, in many cases, it is necessary to execute multiple tasks on the same computing node [141]. For example, this occurs when the number of tasks is greater than the number of available nodes, or because multiple tasks use a dataset located on the same node. These scenarios are prone to memory saturation and moving data to disk may result in higher execution times, which leads to the need for a scheduling strategy able to cope with this issue [142, 143].

In most cases, distributed processing systems use a-priori policies for handling task execution and data management. For example, in the MapReduce programming model used by *Hadoop*, mappers write intermediate results after each computation so performing disk-based processing with partial use of memory [144] through the exploitation of the Hadoop Distributed File System (HDFS). On the other hand, *Apache Spark*[1], which is the state-of-the-art data analysis framework for large-scale data processing exploiting in-memory computing, relies on a Di-

[1] `https://spark.apache.org/`

rected Acyclic Graph (DAG) paradigm and is based on: *i*) an abstraction for data collections which enables parallel execution and fault-tolerance, named Resilient Distributed Datasets (RDDs) [145]; *ii*) a DAG engine, that manages the execution of jobs, stages, and tasks. Besides, it provides different storage levels for data caching and persistence, while performing in-memory computing with partial use of the disk. The Spark in-memory approach is generally more efficient, but a time overhead may be caused by spilling data from memory to disk when memory usage exceeds a given threshold [146]. This overhead can be significantly reduced if the memory occupancy of a task is known in advance, to avoid running in parallel two or more tasks that cumulatively exceed the available memory, thus causing data spilling. For this reason, memory is considered a key factor for the performance and stability of Spark jobs and Out-of-Memory (OOM) errors are often hard to fix. Recent efforts have been oriented towards developing prediction models for the performance estimation of Big Data applications, although most of the approaches rely on analytical models and only a few recent studies have investigated the use of supervised machine learning models [147–149].

Starting from the above considerations, we designed the *Intelligent In-memory Workflow Manager*, namely IIWM, which provides an effective way of scheduling a workflow that minimizes the probability of memory saturation while maximizing in-memory computing, which leads to an increase in application performance and throughput. Workflow scheduling can be defined as follows.

**Definition** [150]. Given a set of $q$ computing resources $\mathcal{R} = \{r_1, \ldots, r_q\}$, workflow scheduling can be defined as the mapping $\mathcal{T} \rightarrow \mathcal{R}$ from each task $t \in \mathcal{T}$ to a resource $r \in \mathcal{R}$, so as to meet a set of specified constraints $\mathcal{Z}$.

Workflow scheduling techniques are often aimed at optimizing several factors, including makespan and overall cost that in turn depend on data transfer and compute cost [150]. In this study, we applied multi-objective optimization by jointly minimizing execution time and memory saturation. This is achieved by using a scheduling strategy that exploits a regression model aimed at predicting the behavior of a given workflow, in terms of resource demand and execution time (see

Section 7.3). For the Reader's convenience, Table 7.1 shows the meaning of the main symbols used in the paper.

| Symbol | Meaning |
|---|---|
| $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ | Set of tasks. |
| $\mathcal{A} \subseteq (\mathcal{T} \times \mathcal{T}) = \{a_1, \ldots, a_m\}$ | Dependencies. $a_i = (t_i, t_j), t_i \in \mathcal{T}, t_j \in \mathcal{T}$. |
| $d_t$ | Description of the dataset processed by task $t$. |
| $\mathcal{W} = (\mathcal{T}, \mathcal{A})$ | Workflow. |
| $\mathcal{N}^{in}(t) = \{t' \in \mathcal{T} \mid (t', t) \in \mathcal{A}\}$ | In-neighbourhood of task $t$. |
| $\mathcal{N}^{out}(t) = \{t' \in \mathcal{T} \mid (t, t') \in \mathcal{A}\}$ | Out-neighbourhood of task $t$. |
| $\mathcal{M}$ | Regression prediction model. |
| $\mathcal{S} = \langle s_1, \ldots, s_k \rangle$ | List of stages. $s_i \subseteq \mathcal{T} \mid (t_x \parallel t_y) \forall t_x, t_y \in s_i$. |
| $C$ | Maximum amount of available memory (capacity). |
| $C_s = C - \sum_{t \in s} \mathcal{M}.predict\_mem(t, d_t)$ | Residual capacity of a stage $s$. |

**Table 7.1:** Meaning of the main symbols used throughout this chapter.

## 7.2 Related work

Recent studies have shown the effectiveness of machine learning in supporting code optimization, parallelism mapping, task scheduling, and processor resource allocation [149]. Moreover, predicting running times and memory footprint is important for estimating the cost of execution and better managing resources at runtime [151]. For instance, in-memory data processing frameworks like Spark can benefit from informed co-location of tasks [149]. In fact, if too many applications or tasks are assigned to a computing node, such that the memory used on the host exceeds the available one, memory paging to disk (i.e., swapping), data spilling to disk in Spark, or OOM errors can occur with consequential drops of performance.

Our work focuses on improving the performance of a Spark application using machine learning-based techniques. The challenge is to effectively schedule tasks in a data-intensive workflow for improving resource usage and application performance, by inferring the resource demand of each task, in terms of memory occupancy and time. State-of-the-art techniques aimed at improving the perfor-

mance of data-intensive applications can be divided into two main categories: *analytical-based* and *machine learning-based*. For each category, the main proposed solutions and their differences with respect to our technique are discussed.

**Analytical-based** These techniques use information collected at runtime and statistics to tune a Spark application, improving its performance as follows:

- Choosing the serialization strategy for caching RDDs in RAM, based on previous statistics collected on different working sets, such as memory footprint, CPU usage, RDDs size, serialization costs, etc. [152, 153].
- Dynamically adapting resources to data storage, using a feedback-based mechanism with real-time monitoring of application memory usage [154].
- Scheduling jobs by dynamically adjusting concurrency through a feedback-based strategy. Taking into account memory usage via garbage collection, network I/O, and Spark RDDs lineage information, it is possible to choose the number of tasks to assign to an executor [155, 156].

The aforementioned works improve the in-memory computing of Spark, by exploiting static or dynamic techniques that can inform the choice of configuration parameters. However, no prediction models are employed and this may lead to unpredicted behaviors. IIWM, instead, uses a prediction regression model to estimate a set of information about a running Spark application, exploiting it to optimize in-memory execution. Moreover, unlike real-time adapting strategies, which use a feedback-based mechanism by continuously monitoring the execution, the IIWM model is trained offline, achieving fast and accurate predictions while used for inferring the resource demand of each task in a given workflow.

**Machine learning-based.** These techniques are based on the development of learning models for predicting the performance (mainly memory occupancy and execution time) of a large set of different applications in several scenarios, on the basis of prior knowledge. This enables the adoption of a *performance-centric*

approach [147], which can be beneficial for the execution of data-intensive applications, especially in the context of HPC systems.

Several techniques use collaborative filtering to identify how well an application will run on a computing node. For instance, *Quasar* [147] uses classification techniques based on collaborative filtering to determine the characteristics of the running application in allocating resources and assigning tasks. When submitted, a new application is shortly profiled and the collected information is combined with the classification engine, based on previous workloads, to support a greedy scheduling policy that improves throughput. Application is monitored throughout the execution to adjust resource allocation and assignment if required, using a single model for the estimation. Adapting this technique to Spark can help to assign tasks to computing nodes within the memory constraints and avoid exceeding the capacity, thus causing the spilling of data to disk. Another approach based on collaborative filtering has been proposed by Llull et al. [148]. In this case, the task co-location problem is modeled as a cooperative game and a game-theoretic framework, namely *Cooper*, is proposed for improving resource usage. The algorithm builds pairwise coalitions as stable marriages to assign an additional task to a host based on its available memory, and the Spark default scheduler is adopted to assign tasks. In particular, a predictor receives performance information collected offline and estimates which co-runner is better, in order to find stable co-locations.

Moving away from collaborative filtering, Marco et al. [149] present a mixture-of-experts approach to model the memory behavior of Spark applications. It is based on a set of memory models (i.e., linear regression, exponential regression, Napierian logarithmic regression) trained on a wide variety of applications. At runtime, an expert selector based on k-nearest neighbor (kNN) is used to choose the model that best describes memory behavior, in order to determine which tasks can be assigned to the same host for improving throughput. The memory models and expert selector are trained offline on different working sets, recording the memory used by a Spark executor through the Linux command "/proc". Finally, the scheduler uses the selected model to determine how much memory is required for an incoming application, for improving server usage and system throughput.

### 7.2.1 Comparison

Similarly to machine learning-based techniques, IIWM exploits a prediction model trained on execution logs of previous workflows. However, it differs in two main novel aspects:

- It only uses high-level workflow features, without requiring any runtime information as done in [147] and [149], in order to avoid the overhead that could be not negligible for complex applications.
- It provides an algorithm for effectively scheduling a workflow in scenarios with limited computing resources.

As far as we know, no similar approaches in literature can be directly compared to IIWM in terms of goals and requirements. In fact, differently from IIWM, *Quasar* [147] and *Cooper* [148] can be seen as resource-efficient cluster management systems, aimed at optimizing QoS constraints and resource usage. With respect to the most related work, presented in [149], IIWM presents the following remarkable differences.

- It focuses on data-intensive workflows while in reference [149] general workloads are addressed.
- It uses high-level information for describing an application (e.g. task and dataset features), already available before the execution. Differently, in reference [149], low-level system features are exploited such as cache miss rate and the number of blocks sent, collected by running the application on a small portion (100 MB) of the input data.
- It proposes a more general approach since the solution proposed in [149] is only suitable for applications whose memory usage is a function of the input size.

## 7.3 Proposed methodology: IIWM

IIWM is based on three main steps, outlined below and described in the following sections:

1. *Execution monitoring and dataset creation*: starting from a given set of workflows, a dataset is generated by monitoring the memory usage and execution time of each task, specifying how it is designed, and giving concise information about the input.

2. *Prediction model training*: from the dataset of executions, a regression model is trained in order to fit the distribution of memory occupancy and execution time, according to the features that represent the different tasks of a workflow.

3. *Workflow scheduling*: taking into account the predicted memory occupancy and execution time of each task, provided by the trained model, and the available memory of the computing node, tasks are scheduled using an informed strategy. In this way, a controlled degree of parallelism can be ensured, while minimizing the risk of memory saturation.

### 7.3.1 Execution monitoring and dataset creation

The first step in IIWM consists of monitoring the execution of different tasks on several input datasets with variable characteristics, in order to build a dataset for training the regression model. The proposed solution was specifically designed for supporting the efficient execution of data analysis tasks, which are used in a wide range of data-intensive workflows. Specifically, it focuses on three classes of data mining tasks: *classification tasks* for supervised learning, *clustering tasks* for unsupervised learning, and *association rules discovery*. Using Spark as a testbed, the following data mining algorithms from the MLlib[2] library have been used:

[2] `https://spark.apache.org/mllib/`

Decision Tree, Naive Bayes, and Support Vector Machines (SVM) for classification tasks; K-Means and Gaussian Mixture Models (GMM) for clustering tasks; FPGrowth for association rules tasks.

### Execution monitoring within the Spark unified memory model

As far as execution monitoring is concerned, a brief overview of the Spark unified memory model is required. In order to avoid OOM errors, Spark uses up to 90% of the heap memory, which is divided into three categories: *reserved memory* (300 MB), used to store Spark internal objects; *user memory* (40% of heap memory), used to store data structures and RDDs computed during transformations and actions; *spark memory* (60% of heap memory), divided in *execution* and *storage*. The former refers to that used for computation during the shuffle, join, sort, and aggregation processes, while the latter is used for caching RDDs. It is worth noting that, when no execution memory is used, storage can acquire all the available memory and vice versa. However, storage may not evict execution due to complexities in implementation, while stored data blocks are evicted from main memory according to a Least Recently Used (LRU) strategy.

The occupancy of storage memory relies on the persistence operations performed natively by the algorithms. Table 7.2 shows some examples of data caching implemented in the aforementioned MLlib algorithms. In particular, the *cache()* call corresponds to *persist(StorageLevel.MEMORY_AND_DISK)*, where MEMORY_AND_DISK is the default storage level.

According to the Spark unified memory model, the execution monitoring was made via the Spark REST APIs, which expose executor-level performance metrics, collected in a JSON file, including peak occupancy for both execution and storage memory along with execution time.

### Dataset creation

Using the aforementioned Spark APIs, we monitored the execution of several MLlib algorithms on different input datasets, covering the main data mining tasks, i.e. classification, clustering, and association rules. The goal of this process is

| MLlib algorithm | Persist call |
|---|---|
| K-Means | *//Compute squared norms and cache them*<br>*norms.cache()* |
| DecisionTree | *//Cache input RDD for speed-up during multiple passes*<br>*BaggedPoint.convertToBaggedRDD(treeInput,...).cache()* |
| GMM | *instances.cache() ... data.map(_.asBreeze).cache()* |
| FPGrowth | *items.cache()* |
| SVM | *IstanceBlock.blokifyWithMaxMemUsage(...).cache()* |

**Table 7.2:** Examples of *persist* calls in MLlib algorithms.

the creation of a dataset for the regression model training, which contains the following information:

- The description of the task, such as its class (e.g., classification, clustering, etc.), type (fitting or predicting task), and algorithm (e.g., SVM, K-Means, etc.).
- The description of the input dataset in terms of the number of rows, columns, categorical columns, and overall dataset size.
- Peak memory usage (both execution and storage) and execution time, which represent the three target variables to be predicted by the regressor. In order to obtain more significant data, the metrics were aggregated on median values by performing ten executions per task.

Starting from 20 available datasets, we divided them into two partitions used for training and testing respectively. Afterward, an oversampling procedure was performed, aimed at increasing the number of datasets contained in both partitions. This procedure generates new synthetic datasets, starting from the original one, by sampling several subsets of its rows and columns. It is worth noticing that using a random sampling approach within this procedure can lead to unexpected behaviors. Specifically, if sampling dataset rows following a uniform distribution is usually not problematic, the same does not apply for columns, i.e. features. In this case, a wrong sampling may badly affect algorithm convergence, introducing

noise into the dataset used to build the regression model. To address this issue we used a more sophisticated strategy to sample datasets along columns. In particular, based on the specific task the dataset is used for, we proceeded as follows:

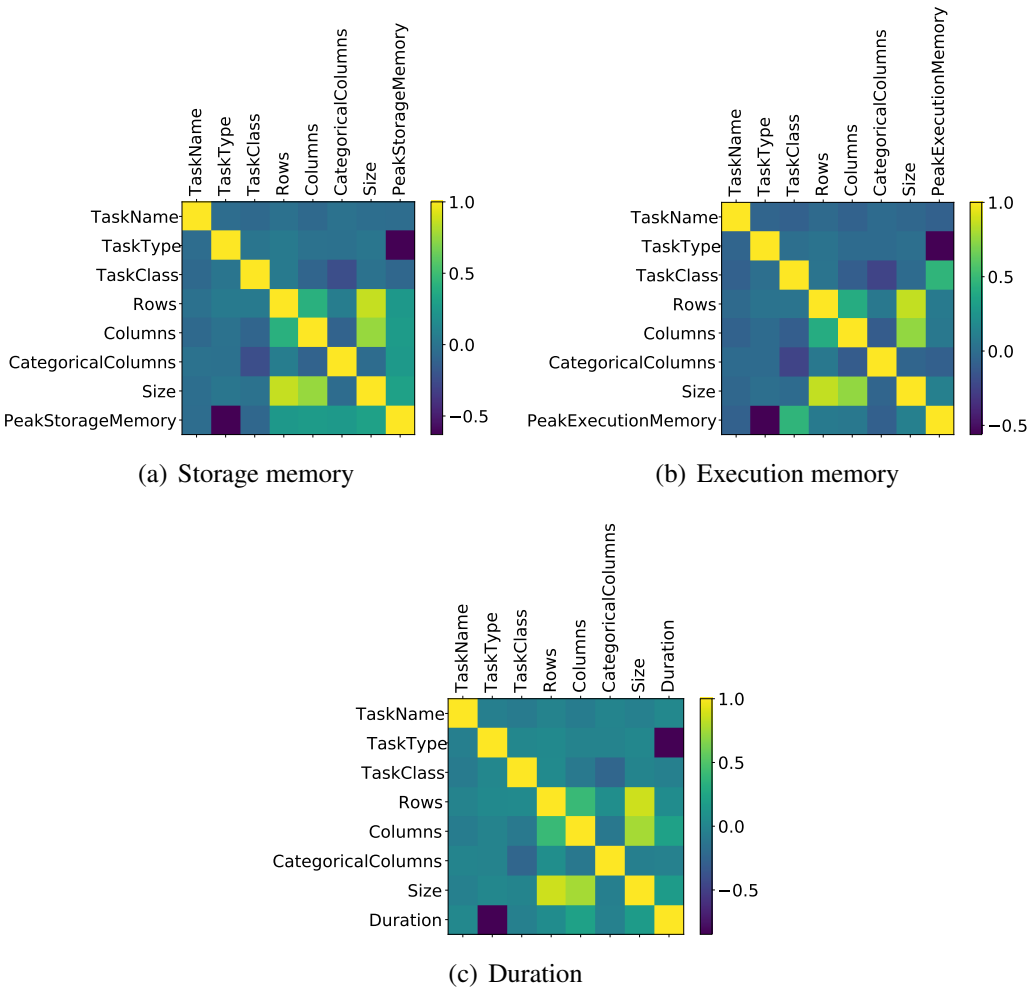
- For datasets used in classification or regression tasks we considered only the $k$ highest scoring features based on:
  - analysis of variance (F-value) for integer labels (classification problems);
  - correlation-based univariate linear regression test for real labels (regression problems).
- For clustering datasets we used a correlation-based test to maintain the $k$ features with the smallest probability to be correlated with the others.
- For association rules discovery datasets no feature selection is required, as the number of columns refers to the average number of items in the different transactions.

The described procedure has been applied separately on the training and test partitions, so as to avoid the introduction of bias into the evaluation process. Specifically, the number of datasets in the training and test partitions has increased from 15 to 260 and from 5 to 86 respectively. Subsequently, we fed these datasets to the MLlib algorithms, obtaining two final datasets of 1309 and 309 monitored executions, used for training and testing the regressor, respectively.

### 7.3.2 Prediction model training

Once the training and test datasets with memory and time information were built, a regression model can be trained with the goal of estimating peak memory occupancy and turnaround time of a task in a given workflow. As a preliminary step, we analyzed the correlation between the features of the training data and each target variable, using the Spearman index. We obtained the following positive correlations: a value of 0.30 between storage memory and the input dataset size,

0.46 between execution memory and the task class, and 0.21 between execution time and the number of columns. These results can be seen in detail in Figure 7.1.




(a) Storage memory

(b) Execution memory

(c) Duration

**Figure 7.1:** Correlation of target variables with the other features.

Afterward, we moved to the training of the regression model. Due to its complexity, the regression problem can not be faced with a simple linear regressor or its regularized variants (e.g. Ridge, Lasso, or ElasticNet), but a more robust model is necessary. We experimentally evaluated this aspect by testing the forecasting abilities of these linear models achieving poor results. For this reason, an ensemble learning model has been used in order to fit the nonlinear distribution of features. Specifically, the *stacking* technique (meta-learning) [157] has been used

by developing a two-layer model in which a set of regressors are trained on the input dataset and a *Decision Tree* is fitted on their predictions. The first layer consists of three tree-based regressors, able to grasp different aspects of input data: a *Gradient Boosting*, an *AdaBoost* and an *Extra Trees* regressor. The second layer exploits a single Decision Tree regressor, which predicts the final value starting from the concatenation of the outputs from the first layer. The described ensemble model has been set with the hyper-parameters shown in Table 7.3.

| **Hyper-parameter** | **Value** |
|---|---|
| *n_estimators* | 500 |
| *learning_rate* | 0.01 |
| *max_depth* | 7 |
| *loss* | *least squares* |

**Table 7.3:** Hyper-parameters.

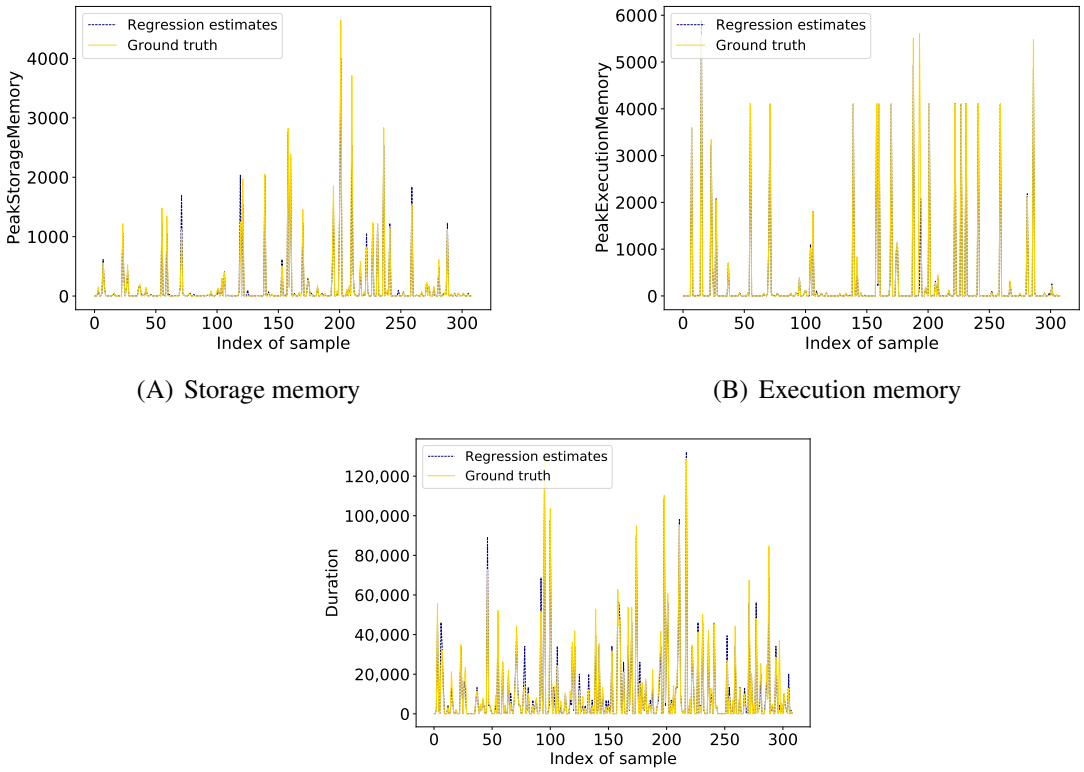
Among 20 trained models, initialized with different random states, we selected the best one by maximizing the following objective function:

$$\mathcal{O} = \bar{R}^2 - MAE$$

whose goal is to choose the model that best explains the variance of data, while minimizing the forecasting error. This function jointly considers the adjusted determination coefficient ($\bar{R}^2$), which guarantees robustness with respect to the addition of useless variables to the model compared to the classical $R^2$ score, and the mean absolute error ($MAE$), normalized with respect to the maximum.

The described model has been developed in *Python3* using the *scikit-learn*[3] library and evaluated against the test set of 309 unseen executions obtained as described in Section 7.3.1. Thanks to the combination of different models, the ensemble technique showed to be very well suited for this task, leading to good robustness against outliers and high regression accuracy. Achieved results are shown by Figure 7.2, in which ground truth and regression estimates are respectively depicted in yellow and blue.

[3] `https://scikit-learn.org/stable/`




(A) Storage memory

(B) Execution memory

(C) Duration

**Figure 7.2:** Meta-learner regression estimates for the different target variables.

These results are detailed in Table 7.4, which shows the evaluation metrics for each target variable. In particular, MAE and RMSE are represented in megabytes for storage and execution memory, and in milliseconds for the duration.

| | **RMSE** | **MAE** | **Adjusted $R^2$** | **Pearson Correlation** |
|---|---|---|---|---|
| **Storage Memory** | 108.23 | 26.66 | 0.96 | 0.98 |
| **Execution Memory** | 312.60 | 26.30 | 0.91 | 0.95 |
| **Duration** | 4443.17 | 2003.70 | 0.95 | 0.98 |

**Table 7.4:** Evaluation metrics on the test set.

### 7.3.3 Workflow scheduling

The prediction model described in Section 7.3.2 can be exploited to forecast the amount of memory that will be needed to execute a given task on a target computing node and its duration, based on the task features listed in Section 7.3.1. These predictions are then used within the scheduling strategy described in the following, whose goal is to avoid swapping to disk due to memory saturation in order to improve application performance and makespan through better use of in-memory computing. The results discussed below refer to a static scheduling problem, as the scheduling plan is generated before the execution. In typical static scheduling, the workflow system has to predict the execution load of each task accurately, using heuristic-based methods [158]. Likewise, in the proposed method the execution load of each task of a given workflow is predicted by the model trained on past executions. Moreover, we investigated how workflow tasks can be scheduled and run on a single computing node, but this approach can be easily generalized to a multi-node scenario. For example, a data-intensive workflow can be decomposed into multiple sub-workflows to be run on different computing nodes according to their features and data locality. Each sub-workflow is scheduled locally to the assigned node using the proposed strategy.

In IIWM, we modeled the scheduling problem as an *offline Bin Packing (BP)*. This is a well-known problem, widely used for resource and task management or scheduling, such as load balancing in mobile cloud computing architectures [159], energy-efficient execution of data-intensive applications in clouds [160], DAGs real-time scheduling in heterogeneous clusters [161] and task scheduling in multiprocessor environments [162]. Its classical formulation is as follows [163]. Let $n$ be the number of items, $w_j$ the weight of the $j$-th item and $c$ the capacity of each bin: the goal is to assign each item to a bin without exceeding the capacity $c$ and minimize the number of used bins. The problem is $\mathcal{NP}$-complete and a lot of effort went into finding fast algorithms with near-optimal solutions.

We adapted the classical problem to our purposes as follows:

- An item is a *task* to be executed.
- A bin identifies a *stage*, i.e. a set of tasks that can be run in parallel.
- The capacity of a bin is the maximum amount $C$ of available memory in a computing node. When assigning a task to a stage $s \in \mathcal{S}$, its residual available memory will be indicated with $C_s$.
- The weight of an item is the *memory occupancy* estimated by the prediction model. In the case of the Spark testbed, it will be the maximum of the execution and storage memory, in order to model a peak in the unified memory. For what concerns the estimated *execution time*, it is used for selecting the stage to be assigned when memory constraints hold for multiple stages.

With respect to the classical BP problem, two changes were introduced:

- All workflow tasks have to be executed, so the capacity of a stage may still be exceeded if a task takes up more memory than the available one.
- The assignment of a task $t$ to a stage $s$ is subjected to dependency constraints. Hence, if a dependency exists between $t_i$ and $t_j$, then the stage of $t_i$ has to be executed before the one of $t_j$.

To solve the BP problem, modeled as described above, thus producing the final scheduling plan, we used the *First Fit Decreasing* algorithm, a greedy procedure that assigns tasks sorted in non-increasing order of weight. However, the introduction of dependency constraints in the assignment process may cause the underusage of certain stages. To cope with this issue, we introduced a further step of consolidation, aimed at reducing the number of stages by merging together stages without dependencies according to the available memory. The overall procedure for task scheduling implemented by IIWM is described by Algorithm 10. In particular, given a data-intensive workflow $\mathcal{W}$, described as a DAG by its tasks and dependencies, and the prediction model $\mathcal{M}$ as input, a scheduling plan is generated in two steps: *i*) building of the stages and task assignment; *ii*) stage consolidation.

**ALGORITHM 10:** IIWM SCHEDULER

**Input:** Workflow $\mathcal{W} = (\mathcal{T}, \mathcal{A})$, Prediction model $\mathcal{M}$
**Output:** A list of stages $\mathcal{S}$

$\mathcal{S} \leftarrow \varnothing$
$\mathcal{Q} \leftarrow \langle t : |\mathcal{N}^{in}(t)|,\ \forall t \in \mathcal{T} \rangle$
$\mathcal{P}_{mem} \leftarrow \langle t : \mathcal{M}.predict\_mem(t, d_t),\ \forall t \in \mathcal{T} \rangle$ `// Memory prediction`
$\mathcal{P}_{time} \leftarrow \langle t : \mathcal{M}.predict\_time(t, d_t),\ \forall t \in \mathcal{T} \rangle$ `// Time prediction`
$\mathcal{T} \leftarrow sort\_decreasing(\mathcal{T}, \mathcal{P}_{mem})$
**while** $\mathcal{T} \neq \varnothing$ **do**
  $\mathcal{T}_{free} \leftarrow \{t \in \mathcal{T} \mid \mathcal{Q}[t] == 0\}$
  $t \leftarrow get\_first(\mathcal{T}_{free})$
  $mem_t \leftarrow \mathcal{P}_{mem}[t]$
  $\mathcal{S}_{sel} \leftarrow \{s_i \in \mathcal{S} \mid mem_t \leqslant C_{s_i}$ and $\nexists\, (t', t)^n \in \mathcal{A}, n > 0, \forall t' \in s_i \cup s_{i+1} \cup \cdots \cup s_k\}$
  **if** $\mathcal{S}_{sel} \neq \varnothing$ **then**
    $duration \leftarrow \langle s : \max_{t' \in s} \mathcal{P}_{time}[t'], \forall s \in \mathcal{S}_{sel} \rangle$
    $increase \leftarrow \langle s : \max\{\mathcal{P}_{time}[t], duration[s]\} - duration[s], \forall s \in \mathcal{S}_{sel} \rangle$
    $s \leftarrow argmin_{s' \in \mathcal{S}_{sel}} increase$
    $C_s \leftarrow C_s - mem_t$
    $s \leftarrow s \cup \{t\}$
  **else**
    $s \leftarrow \varnothing$
    $s \leftarrow s \cup \{t\}$
    $C_s \leftarrow C_s - mem_t$
    $\mathcal{S} \leftarrow \mathcal{S} \cup \{s\}$
  $\mathcal{Q}[t'] = \mathcal{Q}[t'] - 1,\ \forall t' \in \mathcal{N}^{out}(t)$
  $\mathcal{T} \leftarrow \mathcal{T} \backslash \{t\}$
`// Consolidation step`
$\mathcal{S}_{mov} \leftarrow \{s \in \mathcal{S} \mid |\mathcal{N}^{out}(t)| == 0,\ \forall t \in s\}$
**if** $\mathcal{S}_{mov} \neq \varnothing$ **then**
  **for** $s_i \in \mathcal{S}_{mov}$ **do**
    **for** $s_j \in \mathcal{S} \mid j > i$ **do**
      $mem_{s_i \cup s_j} \leftarrow \sum_{t \in s_i \cup s_j} \mathcal{P}_{mem}[t]$
      **if** $mem_{s_i \cup s_j} \leqslant C$ **then**
        $s_j \leftarrow s_i \cup s_j$
        $\mathcal{S} \leftarrow \mathcal{S} \backslash s_i$
        **break**
**return** $\mathcal{S}$

The algorithm is divided into two main parts: in the first part (lines 1-23), the stages are built by iteratively assigning each task according to the estimates of the prediction model; in the second part (lines 25-34), a consolidation process is performed, trying to minimize the number of stages.

The first part (lines 1-23) starts with the initialization of an empty list of stages $\mathcal{S}$, which will be filled according to a dictionary $\mathcal{Q}$ that stores the in-degree of each task in the DAG, which is used for identifying the free tasks which can be scheduled. The prediction model $\mathcal{M}$ is exploited to estimate the memory occupancy and execution time of each task in $\mathcal{T}$, according to their dataset description (lines 3-4). The dictionary $\mathcal{P}_{mem}$, which collects the predicted memory occupancies, is then used to sort tasks according to the First Fit Decreasing strategy (line 5). At each iteration, tasks that can be scheduled (i.e., assigned to a stage) are collected in the $\mathcal{T}_{free}$ set. In particular, they are identified by a zero in-degree, as their execution does not depend on others (line 7). By virtue of the acyclicity of the DAG-based workflow representation, there will always exist a task $t \in \mathcal{T}$ with a zero in-degree not yet scheduled, unless set $\mathcal{T}$ is empty. Afterward, the task with the highest memory occupancy is selected from $\mathcal{T}_{free}$ in order to be scheduled (line 8). At this point, a list of candidate stages ($\mathcal{S}_{sel}$) for the selected task is identified according to the peak memory occupancy forecasted by the prediction model $\mathcal{M}$ (lines 9-10). In particular, a stage $s_i$ belongs to $\mathcal{S}_{sel}$ if it satisfies the following conditions:

- The residual capacity $C_{s_i}$ of the selected stage $s_i$ is not exceeded by the addition of the task $t$.
- There not exists a dependency between $t$ and any task $t'$ belonging to $s_i$ and every subsequent stage ($s_{i+1} \cup \cdots \cup s_k$), where a dependency $(t', t)^n$ is identified by a path of length $n > 0$.

If there exist one or more candidate stages $\mathcal{S}_{sel}$ (line 11), the best one is chosen based on the minimum *marginal increase*. Specifically, for each of these stages, the expected increase of the execution time is estimated (lines 12-13), assigning the task $t$ to the stage $s$ with the lowest value (lines 14-16). Otherwise (line 17), a newly created stage is allocated for $t$ and added to the list $\mathcal{S}$ (lines 18-21). Once

the task $t$ is assigned to the stage $s$, the residual capacity $C_s$ is updated (lines 15, 20). Then, the residual in-degree for every task in the out-neighborhood of $t$ (line 22) is decremented by updating the dictionary $\mathcal{Q}$, so as to allow the assignment of these tasks in the next iterations. Finally, the assigned task $t$ is removed from the set of workflow nodes $\mathcal{T}$ (line 23).

The second part of the algorithm (lines 25-34) performs a consolidation step with the goal of reducing the number of allocated stages by merging some of them if possible, with a consequential improvement in the global throughput. The stages involved in the consolidation step, namely the movable stages ($\mathcal{S}_{mov}$), are those containing tasks with a zero out-degree (line 25). This means that no task in such stages blocks the execution of another one, so they can be moved forward and merged with subsequent stages if the available capacity $C$ is not exceeded. For each movable stage $s_i$ (line 27), another stage $s_j$ from $\mathcal{S}$ is searched among the subsequent ones, such that its residual capacity is enough to enable the merging with $s_i$ (lines 28-30). The merging between $s_i$ and $s_j$ is performed by assigning to $s_j$ each task of $s_i$ (line 31), finally removing $s_i$ from $\mathcal{S}$ (line 32). In the end, the list of stages $\mathcal{S}$ built by the scheduler is returned as output. Given this scheduling plan, the obtained stages will be executed in sequential order, while all the tasks in a stage will run concurrently.

Compared to a blind strategy where the maximum parallelism is achieved by running in parallel all the tasks not subjected to dependencies, which will be referred to as *Full-Parallel* in our experiments, IIWM can reduce both delays of parallelization ($\varepsilon_p$), due to context switch and process synchronization, and swapping/spilling to disk ($\varepsilon_s$), due to I/O operations. Delay $\varepsilon_p$ is always present in all scheduling strategies when two or more tasks are run concurrently, while $\varepsilon_s$ is present only when a memory saturation event occurs. Given $\varepsilon = \varepsilon_p + \varepsilon_s$, IIWM mainly reduces $\varepsilon_s$, which is the main factor behind the drop in performance in terms of execution time, due to the slowness in accessing secondary storage.

As far as the Spark framework is concerned, the proposed strategy is effective for making the most of the default storage level, i.e. *MEMORY_AND_DISK*: at each internal call of the *cache()* method, data is saved in memory as long as this resource is available, using disk otherwise. In this respect, IIWM can reduce the actual persistence of data on disk by better exploiting in-memory computing.

Finally, to further facilitate understanding, Figure 7.3 graphically summarizes the main execution flow of the IIWM scheduler described in this section.

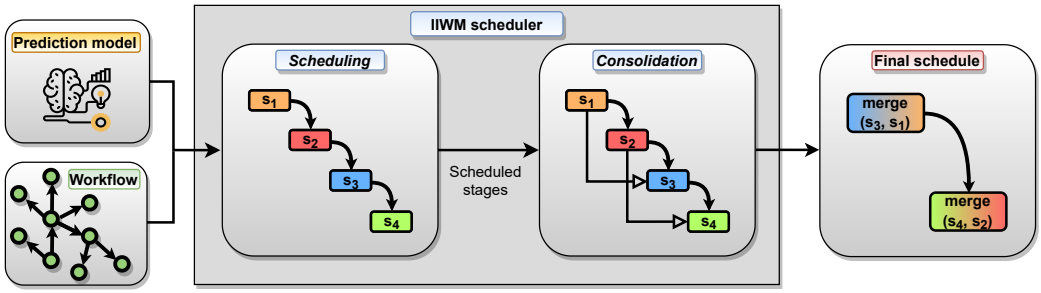



**Figure 7.3:** Execution flow of the IIWM scheduler. Given a workflow and a prediction model as input, a scheduling plan is generated in two steps: *i*) building of the stages and task assignment; *ii*) stage consolidation.

## 7.4 Results and discussion

This section presents an experimental evaluation of the proposed system, specially designed to optimize the in-memory execution of data-intensive workflows. We experimentally assessed the effectiveness of IIWM using Apache Spark 3.0.1 as a testbed. In particular, we generated two synthetic workflows for analyzing different scenarios, by assessing also the benefits coming from the use of IIWM using a real data mining workflow as a case study.

In order to provide significant results, each experiment was executed ten times and the average metrics with standard deviations are reported. In particular, for each experiment, we evaluated the accuracy of the regression model in predicting memory occupancy and execution time.

We evaluated the ability of IIWM to improve application performance taking into account two different aspects:

- *Execution time*: let $m_1$ and $m_2$ be the makespan for two different executions. If $m_2 < m_1$ we can compute the improvement on makespan ($m_{imp}$) and application performance ($p_{imp}$) as follows:

$$m_{imp} = \frac{m_1 - m_2}{m_1} \times 100\% \qquad p_{imp} = \frac{m_1}{m_2}$$

- *Disk usage*: we used the *on-disk usage* metric, which measures the amount of disk usage, jointly considering the volume and the duration of disk writes. Formally, given a sequence of disk writes $w_1, ..., w_k$ let $\tau_i^{'}$, $\tau_i^{''} \in \mathbb{T}$ be the start and end time of the $w_i$ write respectively. Let also $W : \mathbb{T} \rightarrow \mathbb{R}$ be a function representing the number of megabytes written to disk over time $\mathbb{T}$. We define on-disk usage as:

$$\textit{on-disk usage} = \sum_{i=1}^{k} \frac{1}{\tau_i^{''} - \tau_i^{'}} \int_{\tau_i^{'}}^{\tau_i^{''}} W(\tau) d\tau$$

Specifically, for each workflow we reported: *i*) a comparison between Full-Parallel and IIWM in terms of disk usage over time; *ii*) a detailed description of the scheduling plan generated by both strategies; *iii*) the average improvement on makespan and application performance with IIWM; *iv*) statistics about the use of disk, such as the time spent for I/O operations and the *on-disk usage* metric; *v*) the execution of the workflow by varying the amount of available memory, in order to show the benefits of the proposed scheduler in different limited memory scenarios.

### 7.4.1 Synthetic workflows

We first evaluated our approach against two complex synthetic data analysis workflows, where the Full-Parallel approach showed its limitations due to a high degree of parallelism. The dependencies in these workflows should be un-

derstood as execution constraints. For instance, clustering has to be performed before classification for adding labels to an unlabelled dataset, or a classification task is performed after the discovery of association rules for user classification purposes.

The first test has been carried out on a synthetic workflow with 42 nodes. Table 7.5 provides a detailed description of each task in the workflow, while their dependencies are shown in Figure 7.4.

| Node | Task Name | Task Type | Task Class | Rows | Columns | Categorical Columns | Dataset size (MB) |
|---|---|---|---|---|---|---|---|
| $t_0$ | NaiveBayes | Estimator | Classification | 2939059 | 18 | 4 | 198.93904 |
| $t_1$ | FPGrowth | Estimator | AssociationRules | 494156 | 180 | 180 | 417.01007 |
| $t_2$ | NaiveBayes | Estimator | Classification | 5000000 | 27 | 0 | 321.86484 |
| $t_3$ | K-Means | Estimator | Clustering | 1000000 | 104 | 0 | 247.95723 |
| $t_4$ | DecisionTree | Estimator | Classification | 4000000 | 53 | 0 | 505.45 |
| $t_5$ | DecisionTree | Estimator | Classification | 4000000 | 27 | 0 | 257.4918 |
| $t_6$ | DecisionTree | Estimator | Classification | 5000000 | 129 | 0 | 1542.5775 |
| $t_7$ | K-Means | Estimator | Clustering | 2000000 | 53 | 0 | 252.72397 |
| $t_8$ | NaiveBayes | Estimator | Classification | 2000000 | 104 | 0 | 495.9038 |
| $t_9$ | NaiveBayes | Estimator | Classification | 1000000 | 129 | 0 | 307.56625 |
| $t_{10}$ | SVM | Estimator | Classification | 2000000 | 53 | 0 | 252.72397 |
| $t_{11}$ | K-Means | Estimator | Clustering | 2049280 | 9 | 2 | 122.02598 |
| $t_{12}$ | GMM | Estimator | Clustering | 2458285 | 28 | 0 | 145.00838 |
| $t_{13}$ | K-Means | Estimator | Clustering | 9169 | 5812 | 1 | 101.88691 |
| $t_{14}$ | SVM | Estimator | Classification | 2000000 | 27 | 0 | 128.74657 |
| $t_{15}$ | K-Means | Estimator | Clustering | 3000000 | 104 | 0 | 743.87286 |
| $t_{16}$ | SVM | Estimator | Classification | 3000000 | 53 | 0 | 379.08823 |
| $t_{17}$ | SVM | Estimator | Classification | 14410 | 1921 | 0 | 127.3811 |
| $t_{18}$ | K-Means | Estimator | Clustering | 5000000 | 53 | 0 | 631.8128 |
| $t_{19}$ | K-Means | Estimator | Clustering | 5000000 | 104 | 0 | 1239.7812 |
| $t_{20}$ | K-Means | Estimator | Clustering | 2000000 | 78 | 0 | 371.93442 |
| $t_{21}$ | SVM | Estimator | Classification | 3000000 | 104 | 0 | 743.87286 |
| $t_{22}$ | K-Means | Estimator | Clustering | 2939059 | 18 | 4 | 198.93904 |
| $t_{23}$ | SVM | Estimator | Classification | 19213 | 1442 | 0 | 123.28475 |
| $t_{24}$ | DecisionTree | Estimator | Classification | 3000000 | 129 | 0 | 922.6897 |
| $t_{25}$ | K-Means | Estimator | Clustering | 1959372 | 26 | 4 | 189.5505 |
| $t_{26}$ | DecisionTree | Estimator | Classification | 4898431 | 18 | 4 | 331.56735 |
| $t_{27}$ | NaiveBayes | Estimator | Classification | 4898431 | 18 | 4 | 331.56735 |
| $t_{28}$ | K-Means | Estimator | Clustering | 2939059 | 34 | 4 | 334.91486 |
| $t_{29}$ | K-Means | Estimator | Clustering | 4898431 | 18 | 4 | 331.56735 |
| $t_{30}$ | K-Means | Estimator | Clustering | 1966628 | 42 | 0 | 170.48729 |
| $t_{31}$ | NaiveBayes | Estimator | Classification | 1959372 | 18 | 4 | 132.62437 |
| $t_{32}$ | K-Means | Estimator | Clustering | 3000000 | 78 | 0 | 557.9056 |
| $t_{33}$ | DecisionTree | Estimator | Classification | 3000000 | 53 | 0 | 379.08823 |
| $t_{34}$ | DecisionTree | Estimator | Classification | 14410 | 2401 | 0 | 159.71497 |
| $t_{35}$ | K-Means | Estimator | Clustering | 2939059 | 42 | 4 | 386.53033 |
| $t_{36}$ | DecisionTree | Estimator | Classification | 2939059 | 34 | 4 | 334.91486 |
| $t_{37}$ | DecisionTree | Estimator | Classification | 4000000 | 129 | 0 | 1230.2445 |
| $t_{38}$ | NaiveBayes | Estimator | Classification | 1000000 | 53 | 0 | 126.36286 |
| $t_{39}$ | GMM | Estimator | Clustering | 1000000 | 53 | 0 | 126.36286 |
| $t_{40}$ | DecisionTree | Estimator | Classification | 2939059 | 18 | 4 | 198.93904 |
| $t_{41}$ | K-Means | Estimator | Clustering | 4898431 | 18 | 4 | 331.56735 |

**Table 7.5:** Task and dataset descriptions (workflow 1).

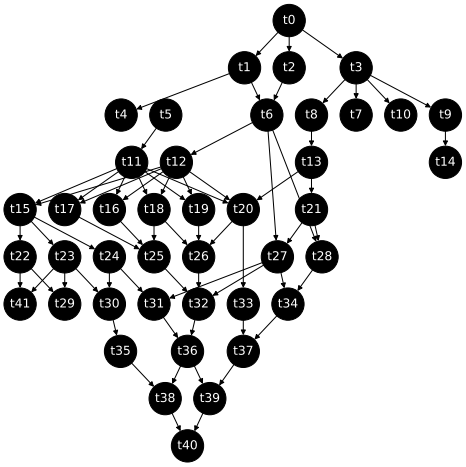



**Figure 7.4:** Task dependencies (workflow 1).

The first step involves predicting the memory occupancy and execution time of each task of the workflow. In this case, the regression model was able to accurately estimate the peaks on storage and execution memory, and the duration. Achieved results are shown in Table 7.6.

| | RMSE | MAE | Adjusted $R^2$ | Pearson Correlation |
|---|---|---|---|---|
| **Storage Memory** | 246.63 | 95.6 | 0.96 | 0.98 |
| **Execution Memory** | 4.7 | 1.6 | 0.99 | 0.99 |
| **Duration** | 20354.38 | 7877.72 | 0.80 | 0.91 |

**Table 7.6:** Performance evaluation of the prediction model.

We first considered a configuration characterized by 14 GB available for running the workflow, which will be used up to 60% due to the Spark unified memory model. Table 7.7 shows an execution example with IIWM, focusing on its main steps: *i*) the scheduling of tasks based on their decreasing memory weight; *ii*) the allocation of a new stage; *iii*) the use of the estimated execution time while computing the marginal increase.

| Iteration | State | Stages |
|---|---|---|
| *It.* 0 | $\mathcal{T}^0_{free} = \{t_0\}$<br>Create $s_0$ and assign $t_0$<br>Unlock $\{t_1, t_2, t_3\}$ | $s_0 = \{t_0\}$ |
| *It.* 1 | $\mathcal{T}^1_{free} = \{t_1, t_3, t_2\}$<br>Create $s_1$ and assign $t_1$<br>Unlock $\{t_4\}$ | $s_0 = \{t_0\}$, $s_1 = \{t_1\}$ |
| *It.* 2 | $\mathcal{T}^2_{free} = \{t_3, t_4, t_2\}$<br>Create $s_2$ and assign $t_3$<br>Unlock $\{t_7, t_8, t_9, t_{10}\}$ | $s_0 = \{t_0\}$, $s_1 = \{t_1\}$,<br>$s_2 = \{t_3\}$ |
| *It.* 3 | $\mathcal{T}^3_{free} = \{t_7, t_4, t_{10}, t_2, t_8, t_9\}$<br>Create $s_3$ and assign $t_7$ | $s_0 = \{t_0\}$, $s_1 = \{t_1\}$,<br>$s_2 = \{t_3\}$ , $s_3 = \{t_7\}$ |
| *It.* 4 | $\mathcal{T}^4_{free} = \{t_4, t_{10}, t_2, t_8, t_9\}$<br>$\mathcal{S}_{sel} = \{s_2, s_3\}$<br>$increase = \{0, 0\}$<br>Assign $t_4$ to $s_2$ | $s_0 = \{t_0\}$, $s_1 = \{t_1\}$,<br>$s_2 = \{t_3, t_4\}$, $s_3 = \{t_7\}$ |
| ... | ... | ... |
| *It.* 17 | $\mathcal{T}^{17}_{free} = \{t_{17}, t_{23}, t_8, t_9\}$<br>$\mathcal{S}_{sel} = \{s_6, s_7\}$<br>$increase = \{12496.363, 0\}$<br>Assign $t_{17}$ to $s_7$<br>Unlock $\{t_{25}\}$ | $s_0 = \{t_0\}$, $s_1 = \{t_1, t_2\}$,<br>$s_2 = \{t_3, t_4, t_5\}$,<br>$s_3 = \{t_7, t_{10}, t_6\}$,<br>$s_4 = \{t_{12}, t_{11}\}$, $s_5 = \{t_{15}, t_{18}\}$,<br>$s_6 = \{t_{19}, t_{22}\}$, $s_7 = \{t_{24}, t_{16}, t_{17}\}$ |
| ... | ... | ... |

**Table 7.7:** Example of execution of algorithm 10 at iteration level.

The role of estimated time, used to compute the marginal increase, can be clearly observed in iteration 17, where task $t_{17}$ is assigned to stage $s_7$, which presents a marginal increase equal to zero. This is the best choice compared to the other candidate stage ($s_6$), whose execution time would be increased by 12496 milliseconds by the assignment of $t_{17}$, with a degradation of the overall makespan. At the end of the process, a consolidation step is exploited for optimizing throughput and execution time, by merging two stages with zero out-degree with some tailing stages, so as to avoid the sequential execution of the two stages in favor of a parallel one.

Figure 7.5 shows disk occupancy throughout the execution. As a consequence of memory saturation, the execution of Full-Parallel resulted in a huge amount of disk writes, while IIWM achieved a null disk usage since no swapping occurred thanks to intelligent task scheduling. Thus, this translates into better use of in-memory computing.

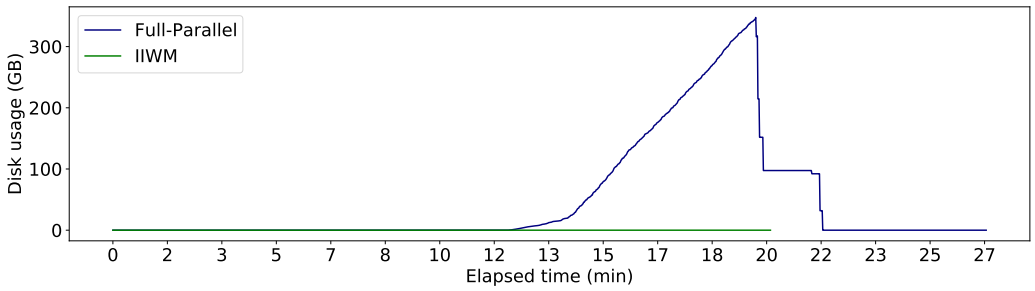


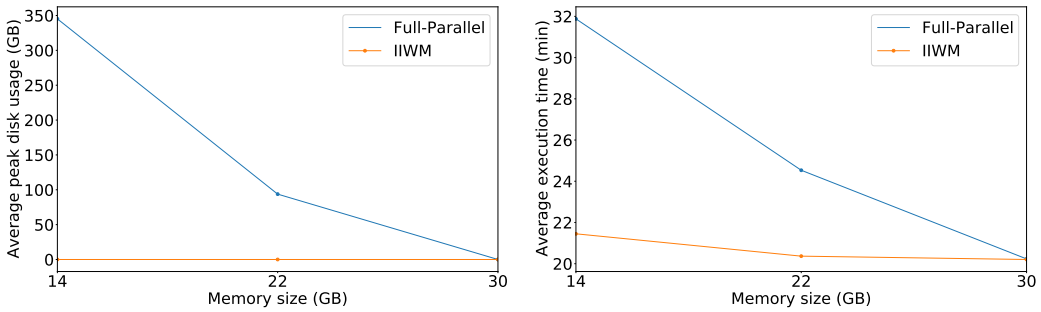

**Figure 7.5:** Disk usage over time for Full-Parallel and IIWM.

These results can be clearly seen also in Table 7.8, which shows the scheduling plan produced by the IIWM scheduler, together with some statistics about execution times and the use of the disk. In particular, given the curves representing disk writes over time shown in Figure 7.5, *on-disk usage* graphically represents the sum, for each disk write, of the ratio between the area under the curve identified by a write and its duration. Compared to the Full-Parallel strategy, IIWM achieved better execution times and an improvement in application performance, with a boost of 1.45$x$ ($p_{imp}$) and a 31.15% reduction in time ($m_{imp}$) on average.

| Strategy | Task-scheduling plan | Number of stages | Time (min.) | Peak disk usage (MB) | Writes duration (min.) | On-disk usage (MB) |
|---|---|---|---|---|---|---|
| **Full-Parallel** | $(t_0)$, $(t_1 \parallel t_2 \parallel t_3)$, $(t_4 \parallel t_5 \parallel t_6 \parallel t_7 \parallel t_8 \parallel t_9 \parallel t_{10})$, $(t_{11} \parallel t_{12} \parallel t_{13} \parallel t_{14})$, $(t_{15} \parallel t_{16} \parallel t_{17} \parallel t_{18} \parallel t_{19} \parallel t_{20} \parallel t_{21})$, $(t_{22} \parallel t_{23} \parallel t_{24} \parallel t_{25} \parallel t_{26} \parallel t_{27} \parallel t_{28})$, $(t_{29} \parallel t_{30} \parallel t_{31} \parallel t_{32} \parallel t_{33} \parallel t_{34} \parallel t_{41})$, $(t_{35} \parallel t_{36} \parallel t_{37})$, $(t_{38} \parallel t_{39})$, $(t_{40})$ | 10 | $31.52 \pm 0.6$ | 356106.601 | 11.56 | 126867.065 |
| **IIWM** | $(t_0)$, $(t_1 \parallel t_2)$, $(t_3 \parallel t_4 \parallel t_5)$, $(t_7 \parallel t_{10} \parallel t_6 \parallel t_8 \parallel t_9)$, $(t_{12} \parallel t_{11} \parallel t_{13})$, $(t_{15} \parallel t_{18})$, $(t_{19} \parallel t_{22} \parallel t_{23})$, $(t_{24} \parallel t_{16} \parallel t_{17} \parallel t_{29})$, $(t_{25} \parallel t_{41})$, $(t_{30} \parallel t_{20})$, $(t_{35} \parallel t_{21} \parallel t_{14})$, $(t_{28} \parallel t_{26} \parallel t_{27})$, $(t_{33} \parallel t_{32} \parallel t_{34} \parallel t_{31})$, $(t_{37} \parallel t_{36})$, $(t_{39} \parallel t_{38})$, $(t_{40})$ | 16 | $21.70 \pm 0.63$ | 0 | 0 | 0 |

**Table 7.8:** Scheduling plan and statistics about execution times and disk usage with 14 GB of RAM.

With different sizes of available memory, the Full-Parallel approach showed higher and higher execution times and disk writes as memory decreased, while IIWM was able to adapt the execution to available resources as shown in Figure 7.6, finding a good trade-off between the maximization of the parallelism and the minimization of the memory saturation probability. At the extremes, with unlimited available memory, or at least greater than that required to run the workflow, IIWM will perform as a full concurrent strategy, producing the same scheduling of Full-Parallel.




(a) Average peak disk usage

(b) Average execution time

**Figure 7.6:** Average peak disk usage and execution time, varying the size of available memory.

The second synthetic workflow consists of the 27 tasks described by Table 7.9 and their dependencies, shown in Figure 7.7. This scenario is characterized by highly heavy tasks and very low resources, where the execution of a single task can exceed the available memory. In particular, task $T_{18}$ has an estimated peak memory occupancy higher than Spark's available unified memory of 5413.8 MB (i.e., corresponding to a heap size of 9.5 GB): this will bring the IIWM scheduling algorithm to allocate the task to a new stage, but memory will be saturated anyway.

| Node | Task Name | Task Type | Task Class | Rows | Columns | Categorical Columns | Dataset size (MB) |
|---|---|---|---|---|---|---|---|
| $t_0$ | K-Means | Estimator | Clustering | 3918745 | 34 | 4 | 446.54932 |
| $t_1$ | DecisionTree | Estimator | Classification | 4000000 | 27 | 0 | 257.4918 |
| $t_2$ | GMM | Estimator | Clustering | 2458285 | 28 | 0 | 145.008 |
| $t_3$ | DecisionTree | Estimator | Classification | 3000000 | 53 | 0 | 379.08823 |
| $t_4$ | DecisionTree | Estimator | Classification | 4000000 | 129 | 0 | 1230.24 |
| $t_5$ | DecisionTree | Estimator | Classification | 3918745 | 18 | 4 | 265.2537 |
| $t_6$ | DecisionTree | Estimator | Classification | 4898431 | 42 | 3 | 648.887 |
| $t_7$ | DecisionTree | Estimator | Classification | 2939059 | 42 | 4 | 386.53033 |
| $t_8$ | K-Means | Estimator | Clustering | 2458285 | 56 | 0 | 278.75266 |
| $t_9$ | GMM | Estimator | Clustering | 3000000 | 53 | 0 | 379.08823 |
| $t_{10}$ | SVM | Estimator | Classification | 4000000 | 53 | 0 | 505.45 |
| $t_{11}$ | K-Means | Estimator | Clustering | 2939059 | 42 | 4 | 386.53033 |
| $t_{12}$ | SVM | Estimator | Classification | 2000000 | 53 | 0 | 252.72397 |
| $t_{13}$ | K-Means | Estimator | Clustering | 1639424 | 9 | 2 | 93.6976 |
| $t_{14}$ | NaiveBayes | Estimator | Classification | 260924 | 3 | 0 | 10.3273 |
| $t_{15}$ | K-Means | Estimator | Clustering | 2000000 | 78 | 0 | 371.93442 |
| $t_{16}$ | DecisionTree | Estimator | Classification | 3918745 | 26 | 4 | 379.1089 |
| $t_{17}$ | DecisionTree | Estimator | Classification | 3918745 | 34 | 4 | 446.54932 |
| $t_{18}$ | FPGrowth | Estimator | AssociationRules | 823593 | 180 | 180 | 697.0 |
| $t_{19}$ | DecisionTree | Estimator | Classification | 2939059 | 26 | 4 | 284.33032 |
| $t_{20}$ | SVM | Estimator | Classification | 5000000 | 27 | 0 | 321.86484 |
| $t_{21}$ | FPGrowth | Estimator | AssociationRules | 164719 | 180 | 180 | 139.871 |
| $t_{22}$ | GMM | Estimator | Clustering | 3000000 | 27 | 0 | 193.119 |
| $t_{23}$ | K-Means | Estimator | Clustering | 4898431 | 26 | 4 | 473.88403 |
| $t_{24}$ | DecisionTree | Estimator | Classification | 2000000 | 104 | 0 | 495.9038 |
| $t_{25}$ | K-Means | Estimator | Clustering | 2458285 | 69 | 0 | 344.6047 |
| $t_{26}$ | FPGrowth | Estimator | AssociationRules | 494156 | 180 | 180 | 417.01007 |

**Table 7.9:** Task and dataset descriptions (workflow 2).

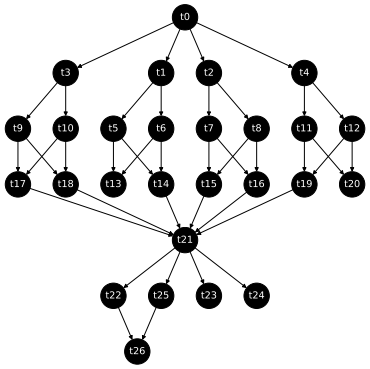


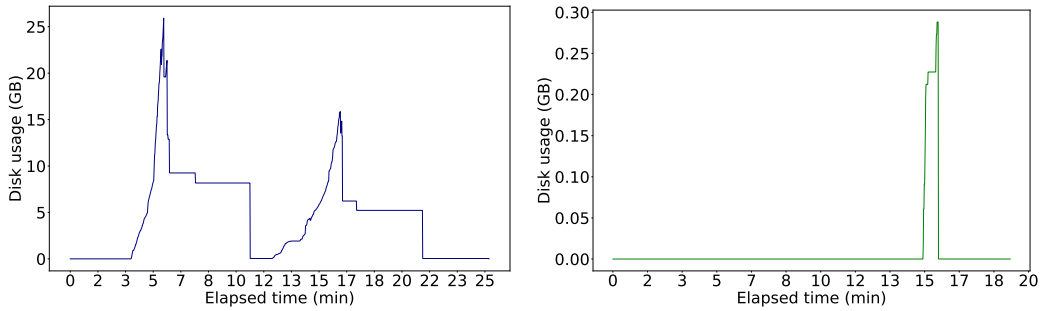

**Figure 7.7:** Task dependencies (workflow 2).

In such a situation, data spilling to disk cannot be avoided, but IIWM tries to minimize the number of bytes written and the duration of I/O operations. Even in this scenario, the prediction model achieved very accurate results, shown in Table 7.10, confirming its forecasting abilities.

| | RMSE | MAE | Adjusted $R^2$ | Pearson Correlation |
|---|---|---|---|---|
| **Storage Memory** | 213.81 | 78.92 | 0.98 | 0.99 |
| **Execution Memory** | 29.86 | 11.56 | 0.98 | 0.99 |
| **Duration** | 20086.80 | 9925.13 | 0.82 | 0.94 |

**Table 7.10:** Performance evaluation of the prediction model.

Figure 7.8 shows disk occupancy during the execution. As we can see, even IIWM cannot avoid data spilling, even though its disk usage was much lower considering peak value and writes duration compared to Full-Parallel.




(a) Disk usage of Full-Parallel (b) Disk usage of IIWM

**Figure 7.8:** Disk usage over time for Full-Parallel and IIWM.

| Strategy | Task-scheduling plan | Number of stages | Time (min.) | Peak disk usage (MB) | Writes duration (min.) | On-disk usage (MB) |
|---|---|---|---|---|---|---|
| **Full-Parallel** | $(t_0)$, $(t_1 \parallel t_2 \parallel t_3 \parallel t_4)$, $(t_5 \parallel t_6 \parallel t_7 \parallel t_8 \parallel t_9 \parallel t_{10} \parallel t_{11} \parallel t_{12})$, $(t_{13} \parallel t_{14} \parallel t_{15} \parallel t_{16} \parallel t_{17} \parallel t_{18} \parallel t_{19} \parallel t_{20})$, $(t_{21})$, $(t_{22} \parallel t_{23} \parallel t_{24} \parallel t_{25})$, $(t_{26})$ | 7 | $29.42 \pm 1.88$ | 27095.837 | 20.6 | 10593.79 |
| **IIWM** | $(t_0)$, $(t_4 \parallel t_2)$, $(t_{11} \parallel t_7)$, $(t_8 \parallel t_3)$, $(t_{15} \parallel t_{10} \parallel t_9 \parallel t_{16})$, $(t_{18})$, $(t_{17} \parallel t_1 \parallel t_{12})$, $(t_6 \parallel t_5)$, $(t_{14})$, $(t_{13} \parallel t_{20} \parallel t_{19})$, $(t_{21})$, $(t_{23} \parallel t_{24})$, $(t_{25})$, $(t_{22})$, $(t_{26})$ | 15 | $22.68 \pm 1.65$ | 304.5 | 3.6 | 60.82 |

**Table 7.11:** Scheduling plan and statistics about execution times and disk usage with 9.5 GB of RAM.

Finally, Table 7.11 shows the statistics about disk usage and execution times. Again, IIWM achieved better results with a boost in performance of almost $1.30x$ ($p_{imp}$) with respect to a Full-Parallel strategy and a 23% reduction in time ($m_{imp}$) on average. An interesting aspect that emerges from the behavior of the IIWM scheduler, in the task-scheduling plan, is the similarity with priority-based scheduling in assigning tasks based on decreasing weights. In fact, tasks characterized by low memory occupancy may be assigned to tailing stages even if they are close to the root (e.g., in Full-Parallel, $t_1$ is executed in the second stage, while in IIWM it is executed in the seventh one). Hence, in a dynamic scheduling scenario where

tasks can be added at runtime, IIWM may suffer from the *starvation* problem, as such tasks may experiment an indefinite delay as far as new tasks with a higher memory weight are provided to the scheduler. Nevertheless, in the proposed work we dealt with a static scheduling problem, where all tasks are known in advance and the task set is not modifiable at runtime.

### 7.4.2 Real case study

In order to assess the performance of the proposed approach against a real case study, we used a data mining workflow that implements a model selection strategy for the classification of an unlabelled dataset [164]. Figure 7.9 shows a

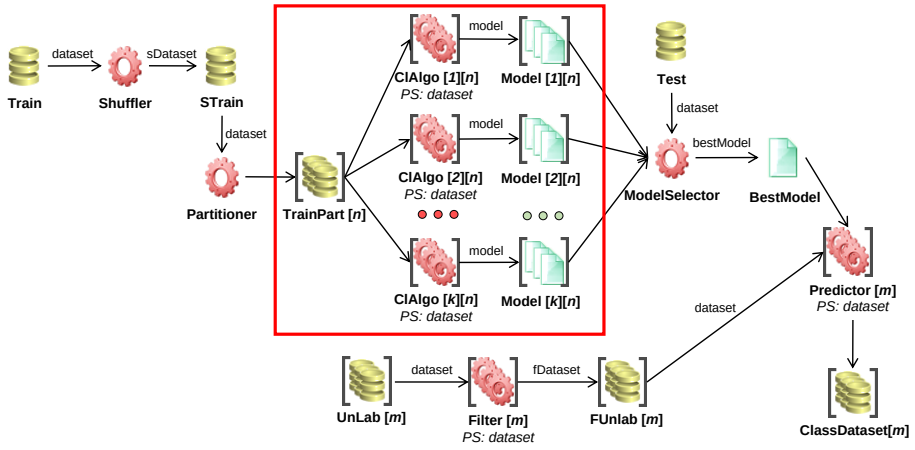



**Figure 7.9:** Ensemble learning workflow.

A training set is divided into $n$ partitions and $k$ classification algorithms are fitted on each partition for generating $k \times n$ classification models. The $k \times n$ fitted models are evaluated by a model selector on a test set to choose the best model. Afterward, the $n$ predictors use the best model to generate $n$ classified datasets. The following $k$ classification algorithms provided by the MLlib library were used: Decision Tree with C4.5 algorithm, Support Vector Machines (SVM), and Naive Bayes. The training set, test set and unlabelled dataset provided as input for the workflow have been generated from the *Physical Unclonable Functions*

*(PUFs)* [165] simulation through a $n$-fold-cross strategy. In this scenario, IIWM can be used to optimize the data processing phase regarding the execution of the $k \times n$ classification algorithms (estimators first, transformers then) concurrently. The other phases, such as data acquisition and partitioning, are out of our interest. The red box in Figure 7.9 shows the tasks of the workflow that will be analyzed.

Figure 7.10 shows disk occupancy over time with 14 GB of RAM. Even in this case, IIWM avoided disk writes, while Full-Parallel registered a high level of disk usage. In particular, during the training phase, the parallel execution of the $k \times n$ models (with $k = 3$ and $n = 5$) saturates memory with 15 concurrent tasks and generates disk writes up to 124 GB.

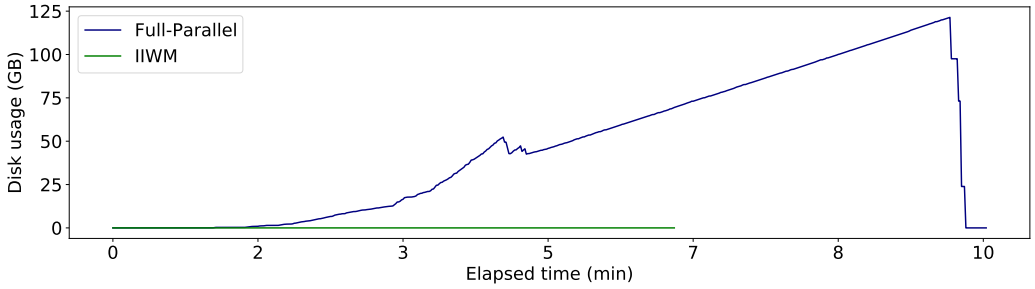


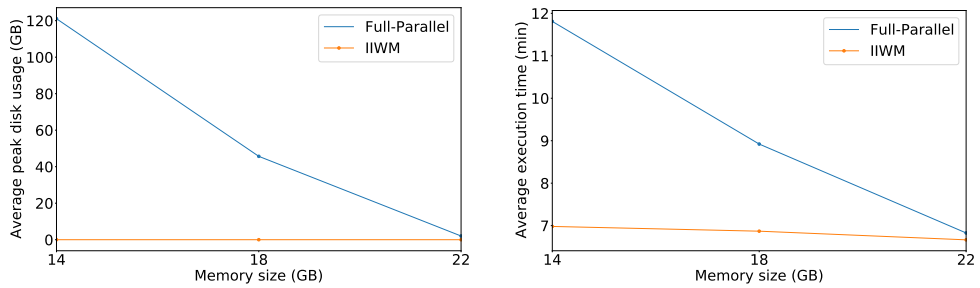

**Figure 7.10:** Disk usage over time for Full-Parallel and IIWM.

The results are detailed in Table 7.12, which shows a boost in execution time of almost $1.66x$ ($p_{imp}$) and a 40% time reduction ($m_{imp}$) with respect to Full-Parallel.

| Strategy | Task-scheduling plan | Number of stages | Time (min.) | Peak disk usage (MB) | Writes duration (min.) | On-disk usage (MB) |
|---|---|---|---|---|---|---|
| **Full-Parallel** | $(t_0 \parallel t_2 \parallel t_4 \parallel t_6 \parallel t_8 \parallel t_{10} \parallel t_{12} \parallel t_{14} \parallel t_{16} \parallel t_{18} \parallel t_{20} \parallel t_{22} \parallel t_{24} \parallel t_{26} \parallel t_{28})$, $(t_1 \parallel t_3 \parallel t_5 \parallel t_7 \parallel t_9 \parallel t_{11} \parallel t_{13} \parallel t_{15} \parallel t_{17} \parallel t_{19} \parallel t_{21} \parallel t_{23} \parallel t_{25} \parallel t_{27} \parallel t_{29})$ | 2 | $11.42 \pm 0.27$ | 124731 | 9.6 | 54443 |
| **IIWM** | $(t_{10} \parallel t_{12} \parallel t_{14} \parallel t_{16} \parallel t_{20} \parallel t_{22} \parallel t_{24} \parallel t_{26} \parallel t_{28})$, $(t_{18} \parallel t_0 \parallel t_2 \parallel t_4 \parallel t_6 \parallel t_8 \parallel t_{11} \parallel t_{13} \parallel t_{15} \parallel t_{17} \parallel t_{21} \parallel t_{23} \parallel t_{25} \parallel t_{27} \parallel t_{29})$, $(t_{19} \parallel t_1 \parallel t_3 \parallel t_5 \parallel t_7 \parallel t_9)$ | 3 | $6.88 \pm 0.1$ | 0 | 0 | 0 |

**Table 7.12:** Scheduling plan and statistics about execution times and disk usage with 14 GB of RAM.

The general trends varying the number of available resources are also confirmed with respect to the previous examples, as shown in Figure 7.11.




(a) Disk usage (b) Average execution time

**Figure 7.11:** Average peak disk usage and execution time, varying the size of available memory.

## 7.5 Conclusions

Currently, data-intensive workflows are widely used in several application domains, such as bioinformatics, astronomy, and engineering. This chapter introduced IIWM, a methodology aimed at optimizing the in-memory execution of data-intensive workflows on high-performance computing systems. Experimental results, achieved by using Apache Spark as a testbed, suggested that by jointly leveraging a machine learning model for performance estimation and a suitable scheduling strategy, the execution of data-intensive workflows can be significantly improved with respect to state-of-the-art blind strategies. In particular, the main benefits of the IIWM resulted when it was applied to workflows having a high level of parallelism. In this case, a significant reduction of memory saturation was obtained. Therefore, it can be used effectively when multiple tasks have to be executed on the same computing node, for example when they need to be run on multiple immovable datasets located on a single node or due to other hardware constraints. In these cases, an uninformed scheduling strategy will likely exceed the available memory, causing disk writes and, therefore, a drop in performance. The proposed approach was also shown to be a very suitable solution in scenarios

characterized by a limited amount of memory reserved for execution, thus finding possible applications in data-intensive IoT workflows, where data processing is performed on constrained devices located at the network edge.

In future work, additional aspects of the performance estimation will be investigated. For example, IIWM can be extended also to consider other common stages in workflows besides data analysis (e.g., data acquisition, integration, and reduction), and additional information about tasks, input data, and hardware platform features.

Chapter 8

# Enhance data partitioning in HPC applications: a machine learning approach to block size estimation

The extensive use of HPC infrastructures and frameworks for running data-intensive applications has led to a growing interest in data partitioning techniques and strategies. Indeed, finding an effective partitioning is crucial to speed-up parallel data-intensive applications and increase scalability.

Data partitioning refers to splitting a dataset into small and fixed-size units, called blocks or chunks, to enable efficient data-parallel processing and storing in distributed-memory-based systems. Several issues related to data partitioning must be addressed to reduce execution times and ensure the good scalability of applications. For example, when a dataset is mapped on a set of nodes of a parallel/distributed computing system, two critical problems arise, i.e. $(i)$ the choice of the destination node for a given block, and $(ii)$ the selection of an appropriate block size. The first problem has been addressed in several studies [166–168], in which scheduling algorithms have been proposed to minimize the movement of data at run-time. The second problem, less studied in the literature, requires to take a decision before the application is running as it strongly depends on the features of the input dataset, the algorithm, and the execution environment.

The block size can heavily affect the trade-off between single-node efficiency and parallelism in data-intensive applications. Specifically, a larger size reduces parallelism (fewer blocks) but makes tasks larger. Although this can lead to an overhead reduction, it must be ensured that the block size does not exceed the memory available on the individual nodes, so as to avoid memory saturation. On the other hand, a smaller size leads to finer exploitation of parallelism, while introducing a larger overhead due to communication, synchronization, and task management, which can negatively impact performance. Typically, block size estimation is not an easy task for programmers. In fact, they usually proceed by following a trial and error approach, only supported by simple heuristics and domain knowledge (i.e., the awareness of the behavior of the algorithm in a given distributed environment). As a result, this tuning process is often time-consuming and resource-intensive, especially when large datasets and complex hardware infrastructures are used.

This chapter describes a novel methodology we specially designed to address this issue, aimed at determining a good estimate for data block size in High-Performance Computing (HPC) applications, quickly and requiring few resources and domain knowledge [12]. The presented methodology relies on machine learning techniques and follows a supervised approach, by leveraging a cascade of tree-based classifiers to determine the most suitable value for the block size, given an execution to be performed. The model is trained on a log of past executions, represented by a set of descriptive features, including algorithm and dataset characteristics, performed in a specific execution environment. Consequently, predictions are applicable to systems having the same infrastructure features, which makes our methodology effectively usable on large-scale homogeneous HPC systems.

The effectiveness of our methodology was assessed through an extensive experimental evaluation, using as a testbed *dislib*, a distributed computing library that provides a wide range of machine learning algorithms. The experiments were performed on different execution environments, including MareNostrum 4 supercomputer (MN4) [169], located at the Barcelona Supercomputing Center (BSC), and involved several real-world datasets. The achieved results show the ability

of the proposed methodology to efficiently predict a suitable value for the block size, thus supporting programmers in determining the correct data partitioning (on both rows and columns), which enables the efficient execution of data-parallel applications in high-performance environments.

The remainder of this chapter is organized as follows. Section 8.1 discusses the main methods for data partitioning present in the literature. Section 8.2 describes the proposed methodology. Section 8.3 describes the use case and testbed. Section 8.4 presents the experimental evaluation and the achieved results. Finally, Section 8.5 concludes the paper.

## 8.1 Related work

In high-performance computing, data partitioning is key to speed-up parallel data-intensive applications and increase scalability. In this section, we describe the main methods proposed in the literature and used in HPC infrastructures.

The data partitioning problem is generally addressed by using three main alternative techniques: horizontal, vertical, and hybrid partitioning [170].

- *Horizontal partitioning.* This technique, also called sharding, divides the rows of the dataset into disjoint subsets so that each subset has the same number of columns as the whole dataset. This approach is used in many state-of-the-art frameworks for big data processing, such as Hadoop and Spark [171, 172].

- *Vertical partitioning.* In this approach, the columns of the dataset are divided into subsets, usually based on the columns on which data querying is more frequently performed, or a heuristic approach [173, 174].

- *Hybrid partitioning.* It combines the horizontal and vertical approaches, by aggregating data according to how it is used by the target application and/or system [175, 176]. It is also called functional partitioning.

Among all described strategies, horizontal partitioning is the one that is mostly used in big data applications and HPC systems, while the other two strategies are less used and therefore explored in the literature. Specifically, the horizontal partitioning can be further categorized into *hash-*, *range-*, and *random-* based. The hash-based approach divides records into subsets by hashing the record key and then mapping the hash value of the key to a partition. A common method to do this mapping is using a round-robin algorithm, which guarantees a balanced partitioning among nodes and partitions of equivalent size. In hash-based partitioning records having the same key value must have the same hash value, and consequently, they will be mapped to the same partition. The range-based partitioning approach, instead, partitions a dataset according to a given range and distributes records having the keys within the same range on the same node. In distributed environments, how to set this range is often challenging, especially when dealing with large-scale data analysis. Finally, the random-based partitioning approach divides the records randomly into subsets using a random number generator to determine where to distribute each record, producing approximately subsets of equal size. As an example, Salloum et al. proposed the Random Sample Partition (RSP) data model to support distributed big data analysis [177]. In particular, this model represents a big dataset as a set of non-overlapping blocks, where each block is a random sample of the whole dataset. In addition, Wei et al. [178] proposed a two-stage algorithm to generate RSP data blocks from the Hadoop Distributed File System (HDFS). A drawback of random-based approaches is that they do not consider the correlation between records, which instead can be leveraged to avoid unnecessary computations. As an example, in [179] the authors designed a context-based multi-dimensional partitioning technique that relies on data correlation to determine how records should be split.

Main approaches for data partitioning can be further categorized into *static* or *dynamic* [180]. Static approaches use a fixed-size when a block is selected, often defined in MB, and the partitioning is computed before starting the execution. As an example, data in HDFS is divided into fixed-size blocks, obtained from horizontal partitioning. Particularly, a typical block size is 128 MB, which means

that if we have a 1 GB file, it will be partitioned into 8 blocks, each one of 128 MB. Similarly, in Spark the HDFS blocks need to be loaded into an in-memory data structure, called Resilient Distributed Dataset (RDD) [145], which can be then partitioned using either the aforementioned strategies, such as hash and range, or a custom partitioning. Specifically, Spark runs a single task for every partition of an RDD, up to $2-3$x times the number of cores in the cluster. Hence, a heuristic can be derived that determines the number of partitions as a small multiple of the total number of available cores. On the contrary, in the dynamic approaches, the dimension and shape of the blocks in which to partition the dataset are not chosen a priori, but at runtime. As an example, several dynamic strategies can be found in [180], specially designed for graph data partitioning. Both approaches present some issues. Static approaches define a priori how data should be partitioned, without taking into account any dataset characteristic or algorithm feature. On the other hand, dynamic approaches support adaptation to the actual workload at runtime, but they can introduce significant overhead in dynamically adjusting data partitioning.

To overcome these issues, in [12] we proposed a hybrid and static data partitioning approach, able to determine how to adequately partition the dataset before the execution, thus avoiding the overhead introduced by dynamic approaches, while also taking into account dataset and algorithm characteristics and infrastructure features. Specifically, we determine the best size of data blocks by using a supervised machine learning technique, focusing on the estimation of two quantities, i.e. the number of partitions in which to divide dataset rows and columns. This allows for determining a suitable partitioning in order to make the most of the computational resources of the execution environment, thus improving the performance and scalability of the application.

## 8.2 Proposed methodology

Our methodology addresses the problem of data partitioning by finding a suitable estimate of data block size, allowing an effective hybrid partitioning of a given dataset. In particular, we formulated the block size estimation task as follows. Let $d$ be a dataset composed of $n$ rows and $m$ columns, and $a$ the algorithm to be run on the execution environment $e$. We must determine the best number of rows and column partitions $(p_r, p_c)$, from which to derive the best block size $S = (n/p_r, m/p_c)$ that minimizes the execution time of algorithm $a$.

In our solution, the optimal partitioning is estimated through a classification-based approach. Particularly, the target classes are represented by the two discrete variables to be predicted, i.e. $p_r$ and $p_c$, which range from 1 (i.e., a single partition) to a maximum number of partitions, usually defined as a small multiple of the total number of cores available. We selected this kind of approach as it results to be more stable compared to a regression-based one, in which the block size is directly predicted by identifying the number of elements of each block. In fact, the main problem of the regression-based approach is that its output is generally unconstrained, which may lead to a set of blocks with a non-uniform size. The classification-based approach, instead, is less affected by this problem, as it selects the number of partitions against a finite number of possible values. However, the ability to generalize heavily depends on the representativeness of the training data, which implies that the produced estimates are reliable for HPC systems having similar infrastructure features and datasets whose size is of the same order of magnitude as those seen during training.

In the following, we provide a detailed description of the three main steps that make up the proposed methodology, i.e. *i*) the analysis of the execution environment, *ii*) the analysis of execution logs from which to extract training data, and *iii*) the training of the machine learning model for block size estimation. For the sake of completeness, Figure 8.1 shows the main execution flow of our methodology.

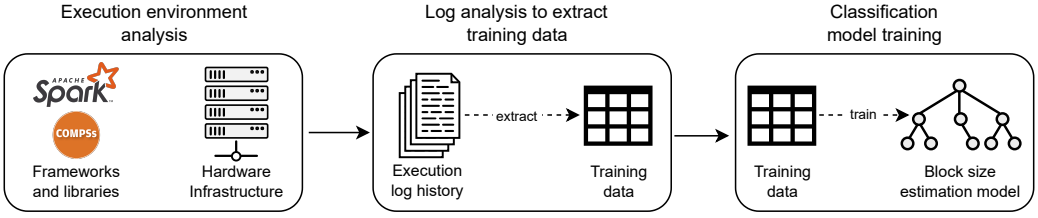



**Figure 8.1:** Execution flow of the proposed methodology for block size estimation.

### 8.2.1 Execution environment analysis

Given a distributed environment in which data analysis applications can be run, the proposed methodology aims to enable their efficient execution by identifying a proper size for data blocks. This can help programmers to make the most of all the computing and storage resources that are available in the environment, as they can efficiently obtain a suitable estimate for the block size, without the need for heavy tuning processes or domain knowledge. As a preliminary step, the execution environment must be carefully analyzed. Generally, it is characterized by a set of software features, such as the available frameworks and libraries, and infrastructure features, such as the number of nodes, cores per node, available memory, and disk space.

### 8.2.2 Log analysis to extract training data

The proposed methodology leverages a log of past executions to extract the patterns that link a specific execution to the best block size, by training a supervised machine learning model. However, in order to learn effective patterns, raw logs must be adequately processed to extract an appropriate set of training data. The log $\mathcal{L}$ consists of a collection of executions, performed by both standard users and domain experts, in which a single execution is described by the characteristics of the dataset ($d$), the algorithm ($a$), the execution environment ($e$), the applied partitioning along rows ($p_r$) and columns ($p_c$), the overall execution time ($t$), and other measurements such as main memory/disk usage. Formally, an execution in $\mathcal{L}$ is represented by the tuple $\langle d, a, e, p_r, p_c, t \rangle$.

In order to generate the training dataset $\mathcal{D}$, all executions in $\mathcal{L}$ are grouped by the triple $\langle d, a, e \rangle$. In this way, we can observe how, given a fixed configuration, execution time is affected by different block sizes. Afterward, for each group, the best partitioning ($p_r^*$, $p_c^*$) that led to the minimum execution time is found and the tuple $\langle d, a, e, p_r^*, p_c^* \rangle$ is added to $\mathcal{D}$. At the end of the process, the dataset $\mathcal{D}$ will contain the best partitioning found for each triple $\langle d, a, e \rangle$, specifically:

- Features related to algorithm $a$, dataset $d$ (dimension in MB, number of rows, etc.), and execution environment $e$ (number of cores, number of nodes, etc.).
- The optimal partitioning $(p_r^*, p_c^*)$, i.e. the target variable.

Table 8.1 shows an example of an excerpt of $\mathcal{D}$ obtained from the training data extraction step.

| **Algorithm** | **Dataset rows** | **Dataset columns** | **Dataset size (GB)** | **Infrastructure features** | | | **Best partitioning** | |
|---|---|---|---|---|---|---|---|---|
| | | | | *# nodes* | *# cores* | *RAM* | $p_r^*$ | $p_c^*$ |
| K-means | 500,000 | 1000 | 2.39 | 4 | 64 | 256 | 32 | 4 |
| RF | 1000 | 500,000 | 2.92 | 4 | 64 | 256 | 32 | 8 |
| SVM | 10,000 | 10,000 | 1.1 | 4 | 64 | 256 | 16 | 16 |

**Table 8.1:** Excerpt of the training set extracted by the log of executions.

Although many frameworks provide accurate instrumentation tools for collecting a wide range of information, effectively exploitable for application performance monitoring, it is not always possible to have a representative log. To face this issue, training data can be generated and/or enriched by arranging a set of executions, with the aim of finding the block size that optimizes execution time for the considered configurations. This process is characterized by several degrees of freedom, as different algorithms ($a$) must be taken into account, as well as a wide range of input data ($d$) and infrastructure features of the execution environment ($e$), which leads to the need for an efficient search strategy. For this purpose, a grid search technique can be leveraged, in which several triples $\langle d, a, e \rangle$ are generated and annotated with the best block size found during the search. Specifically, for each triple, the following operations are performed.

- Given $n_{cores}$ the number of available cores, a $k \times k$ grid $\mathcal{G}$ is built, with $k = log_s(n_{cores})$, where $s$ is a search step such that $log_s(n_{cores})$ is an integer number. The step $s$ (set to 2 by default) can be used to control the trade-off between the cost of the grid search and the representativeness of the generated training samples.

- Each element $g_{i,j}$ in the grid $G$, with $i$ and $j$ ranging from 1 to $k$, is determined as the time of executing algorithm $a$ on the dataset $d$ within the environment $e$, by splitting $d$ using the $(p_r = s^i,\ p_c = s^j)$ partitioning. This means that the rows and columns of $d$ will be divided into $s^i$ and $s^j$ blocks, respectively. The execution time in the case of failures (e.g., out-of-memory errors) is set to $\infty$.

- By exploring the grid, the best partitioning $(p_r^*, p_c^*)$ for the triple $\langle d, a, e \rangle$ can be found, which leads to the minimum execution time. Formally, it is computed as the pair $(p_r^*, p_c^*) = (s^{i^*}, s^{j^*})$, where $(i^*, j^*) = \arg\min_{i,j} \mathcal{G}$. Finally the triple $\langle d, a, e \rangle$ is labeled with $(p_r^*, p_c^*)$ and added to the training dataset $\mathcal{D}$.

This approach for training data generation via execution monitoring borrows ideas from the work described in the previous chapter, in which we proposed a machine learning-based approach to improving the in-memory execution of data-intensive workflows on parallel machines [11].

### 8.2.3 Classification model training

Given the dataset $\mathcal{D}$ obtained in the previous step, a classification model is trained to learn the patterns that relate the execution features/parameters and the best partitioning $(p_r^*, p_c^*)$. Thus, the output of this step is a classification model capable of estimating the optimal number of partitions in which to split the rows and the columns of a given dataset, based on its characteristics, the algorithm to be run, and the underlying execution environment. Since the target variable to be predicted, i.e. the best partitioning $(p_r^*, p_c^*)$, is two-dimensional, we used a

multi-output classification model based on a cascade of two different decision tree classifiers. The two classifiers, namely $DT_r$ and $DT_c$, are used to predict the best number of rows and column blocks, respectively. In addition, as the two target variables are likely to be dependent on each other, we used a chained model to condition the $DT_c$ predictions on the output of $DT_r$, as shown in Figure 8.2.

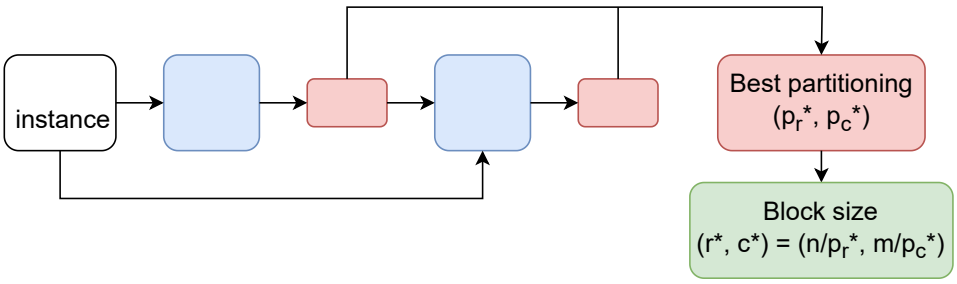



**Figure 8.2:** Chained multi-output classification model.

We followed this order in chaining the two decision tree models since partitioning along the rows is generally more relevant. The training step is performed as follows:

1. We train the first decision tree $DT_r$ with the training instances of $\mathcal{D}$ to learn the patterns underlying the number of blocks $p_r^*$ in which to partition dataset rows.

2. The second decision tree $DT_c$ is trained with the training instances concatenated with the output of $DT_r$, with respect to the second target variable $p_c^*$, to learn the number of blocks in which to partition dataset columns.

Afterward, the corresponding value of the block size can be determined as $(r^*, c^*) = (n/p_r^*, m/p_c^*)$, where $n$ and $m$ are the rows and columns of the considered dataset. For the sake of clarity, we report a simple example of computing the block size for a given input instance. Let $n = 51,200$ and $m = 256$ be the number of dataset rows and columns, respectively. Suppose that the result of the prediction is $(p_r^*, p_c^*) = (4, 16)$. Then the optimal block size can be computed as:

$$(r^*, c^*) = (n/p_r^*, m/p_c^*) = (12800, 16)$$

## 8.3 Block size estimation in dislib applications

The proposed methodology can be applied to a wide range of frameworks for distributed data processing. Indeed, the majority of these systems, such as Hadoop [171], Spark [172], DMCF [181] and PyCOMPSs [182], leverage a data-parallel approach that involves a data partitioning step for distributing the dataset across a set of working nodes [10]. Consequently, our methodology can bring huge benefits, by suggesting an adequate partitioning that allows to effectively run distributed applications, reducing overhead while ensuring a good level of parallelism and throughput. Among the main frameworks and libraries of interest for distributed data processing, we selected as a testbed PyCOMPSs, focusing on dislib [183], a distributed computing library built on top of it that provides distributed machine learning algorithms. The early implementation, based on PyCOMPSs and dislib, is publicly available on Github[1].

PyCOMPSs [182, 184] is a task-based programming model that enables the parallel execution of sequential Python code in distributed computing platforms. By means of Python decorators, the developer identifies the function/methods to be considered tasks. PyCOMPSs also offers a small API for synchronization. It is based on a runtime able to identify the data dependencies that exist among tasks building a data dependency graph. The task graph exposes the possible task concurrency that is exploited by the runtime, which manages the execution of the tasks in distributed infrastructures, scheduling them and performing all the necessary data transfers.

The Distributed Computing Library (dislib) [183], implemented on top of PyCOMPSs, provides various distributed algorithms for several machine learning tasks, including classification, clustering, and dimensionality reduction. Dislib is inspired by scikit-learn and NumPy, and it comes with two primary programming interfaces: an API to manage data in a distributed way and an estimator-based interface to work with different machine learning models.

[1] https://github.com/eflows4hpc/dislib-block-size-estimation

Its main data structure is the distributed array ($ds-array$) which enables it to distribute the datasets in multiple nodes of a distributed infrastructure. A ds-array is a matrix divided into blocks, which can be a NumPy array or a SciPy CSR (Compressed Sparse Row) matrix, depending on the kind of data used to create the ds-array. Dislib provides an API similar to NumPy to work with ds-arrays, but ds-arrays are stored remotely, allowing to store much more data than regular NumPy arrays. All operations on ds-arrays are internally parallelized with PyCOMPSs. The typical workflow in dislib consists of the following steps: *i*) reading input data into a ds-array; *ii*) creating an estimator object; *iii*) fitting the estimator with the input data; *iv*) getting information from the model's estimator or applying the model to new data. At each step, the level of parallelism is driven by the number of blocks of the ds-arrays that are operated, which in turn is controlled by the ds-array's block size, which defines the number of rows and columns of each block.

Choosing the right size of a block-array can be a quite challenging task: small blocks allow for higher parallelism as the computation is divided into more tasks. However, handling a large number of blocks can generate overhead that can negatively impact performance. Thus, the optimal block size will allow the full utilization of the available resources without adding too much overhead. In addition to this, block size also affects the amount of data that tasks load into memory. This means that block size should never be bigger than the amount of available memory per processor. Summing up, the choice of the optimal block size is often difficult but essential for exploiting the full potential of dislib, hence the possibility of effectively applying the proposed methodology.

## 8.4 Experimental evaluation

This section presents the extensive experimental evaluation we carried out to assess the effectiveness of the proposed methodology by using PyCOMPSs as the execution framework and the machine learning library dislib built on top of it.

In particular, we analyzed the impact on the execution time of the partitioning suggested by our methodology, performing several experiments on two distinct execution environments, i.e. a cluster node and the MareNostrum 4 supercomputer, in order to evaluate the methodology in a single-node and a multi-node scenario.

For what concerns the evaluation metrics, we used *makespan ratio* to measure the improvement in speed of execution brought by the predicted block size, with respect to other possible partitions. Specifically, given an algorithm $a$ to be run in a distributed environment $e$, let $t^*$ and $t_{other}$ be the execution times achieved by using the predicted block size and a different one, respectively. We compute the *makespan ratio* as follows:

$$makespan\ ratio = \frac{t_{other}}{t^*}$$

In addition, we measured the percentage *makespan reduction*, i.e. the percentage amount of execution time saved by running a given algorithm with the predicted block size, with respect to a different one. Formally:

$$makespan\ reduction = \frac{t_{other} - t^*}{t_{other}}$$

### 8.4.1 Single-node experiments

This first experimental evaluation was carried out on a cluster node equipped with an AMD processor, 64 cores, and 256 GB of RAM. The used log contains several executions performed on different datasets using a wide range of machine learning algorithms provided by dislib for classification and clustering, including Support Vector Machine (SVM), Random Forest (RF), Gaussian Mixture Model (GMM) and K-means. In the following sections, we describe the results achieved in the single-node scenario, by evaluating the benefits brought by the estimated block size with the use of both real-world and synthetic test datasets.

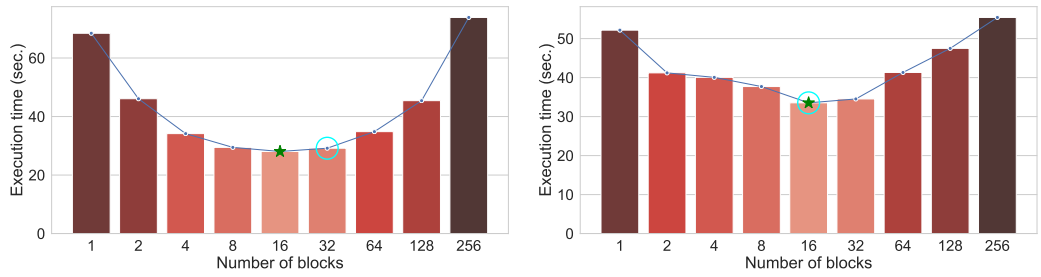
### Real-world datasets

The effectiveness of our methodology in suggesting a suitable block size value was evaluated on two real-world datasets used for clustering and classification:

- *HEPMASS* [185]: it is a high-energy physics dataset containing signatures of exotic particles, learned from Monte Carlo simulations of the collisions that produce them. The dataset contains 7 million training samples with 27 features that can be separated into two clusters, i.e. particle-producing collisions and background sources.

- *MNIST* [186]: it is a multi-class dataset used for image classification and pattern recognition, containing gray-scale images of handwritten digits, from 0 to 9, labeled with the represented number. In particular, the dataset contains 60 thousand training images in a $28 \times 28$ format, which can be represented by vectors of 784 features.

Since both test datasets are characterized by a big number of rows against a relatively small number of columns, the model predicted in both cases a block size that partitions both datasets only horizontally, that is just one block for the columns containing all features. For this reason, the number of blocks generated by creating the distributed arrays is equal to the number of partitions along rows. The results achieved by running K-means and Random Forest on HEPMASS and MNIST datasets are summarized in table 8.2. Specifically, the time $t^*$ achieved by running the two algorithms using the predicted block size was compared against the best, worst, and average times achieved by using all other possible partitionings, calculated using progressive powers of 2 from 2 to 256, i.e. 4x times the total number of cores available. Moreover, Figure 8.3 shows the measured execution time by using different gradations of red, where a greater intensity corresponds to a higher duration. In the proposed plots, the time obtained by using the predicted partitioning is marked by a cyan circle, while the best one is marked by a green star.




(a) K-means, HEPMASS dataset. (b) Random Forest, MNIST dataset.

**Figure 8.3:** Execution times achieved by K-means and Random Forest executed on two real-world datasets.

| *Algorithm* | *Dataset name* | *Dataset rows* | *Dataset columns* | *Metric* | *Best* | *Average* | *Worst* |
|---|---|---|---|---|---|---|---|
| K-means | HEPMASS | 7,000,000 | 27 | Makespan ratio | 0.96 | 1.48 | 2.53 |
| | | | | Makespan reduction | −3.80% | 32.6% | 60.5% |
| Random Forest | MNIST | 60,000 | 784 | Makespan ratio | 1.00 | 1.27 | 1.65 |
| | | | | Makespan reduction | 0% | 21.32% | 39.51% |

**Table 8.2:** Makespan ratio and percentage makespan reduction measured by running K-means and Random Forest algorithms on the HEPMASS and MNIST datasets.

By observing Figure 8.3(a), it can be noticed that the optimal number of blocks that led to the best execution time for the K-means algorithm was 16, while the model suggested partitioning rows in 32 blocks. However, the time measured by using the predicted block size, i.e $t^*$, is the second best time, and the difference with the best one is negligible ($\approx$ 1 second). By comparing $t^*$ with the average execution time, the predicted block size led to a 1.48 makespan ratio, with a percentage reduction of makespan equal to 32.6%. The worst execution time was observed in the case in which 256 blocks were used. This is caused by the excessively small size of the blocks, which causes the generation of a too large number of blocks and tasks. In fact, such a degree of parallelism produces too much overhead that results in a degradation of application performance. A similar execution time was measured when just one block was used, i.e. no partitioning is performed. This is the opposite case, in which parallelism is not exploited at all.

By comparing $t^*$ with the worst execution time, we measured a makespan ratio of 2.53 and a makespan reduction of 60.5%, which confirms how the block size suggested by the model allows determining a proper partitioning, which leads to a quite good trade-off between the degree of parallelism and the introduced overhead. The quality of the partitioning suggested by our model is further confirmed by the execution of Random Forest on the MNIST dataset (Figure 8.3(b)). In this case, the model predicts exactly the best possible partitioning, i.e. 16 blocks along rows. Also in this case, the worst values were measured at the extremes, where the level of parallelism is either zero (1 block) or too high (256 blocks). Furthermore, we measured a makespan ratio of 1.27 and 1.65 and a makespan improvement of 21.32% and 39.51%, compared to the average and worst execution times, respectively.

### Synthetic datasets

With the aim of further exploring the effectiveness of our methodology, the estimates provided by the model were evaluated against a set of synthetic test datasets, which is useful to observe how the algorithms behave in some specific cases. For this purpose, we generated a series of multiclass test datasets, by allocating one or more normally-distributed clusters of points to each class. Particularly, we used both isotropic and anisotropic Gaussian blobs. In addition, the obtained samples were augmented with random noise and redundant features, generated as a linear combination of the original ones.

Starting from a set of synthetic test datasets of varying shapes, generated following the aforementioned process, we measured the performance improvement achievable with the use of our methodology, relative to the execution of K-means and Random Forest algorithms. Specifically, for each test dataset, we compared the time $t^*$, achieved by using the predicted block size, against the best, worst, and average times obtained from all other possible partitionings. The different partitionings for this comparison were calculated using progressive powers of 2 from 2 to 64 for both the number of row and column blocks, which leads to 36 possible configurations. Furthermore, each test execution was repeated 10 times, taking

the median value, in order to get a robust measure of execution time, preventing the evaluation process from being biased by noisy measures. Achieved results, in terms of makespan ratio and percentage makespan reduction, averaged on all test datasets, are summarized in Table 8.3 and discussed in the following.

| *Metric* | **Best** | **Average** | **Worst** |
|---|---|---|---|
| Makespan ratio | 0.99 | 1.25 | 2.11 |
| Makespan reduction (%) | −0.79% | 24.71% | 55.06% |

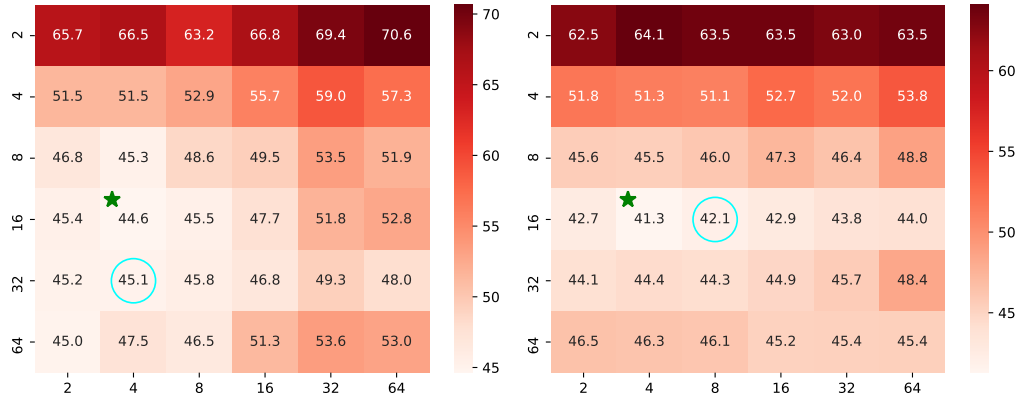
**Table 8.3:** Average values of makespan ratio and percentage makespan reduction obtained from executing K-means and Random Forest algorithms on the synthetic test datasets.

By comparing $t^*$ with the best time measured by trying all possible partitionings, it can be noticed that the data partitioning suggested by the learning algorithm is almost always the best one, i.e. it guarantees an execution time very close to the shortest obtainable time. In particular, we measured a very little difference compared to the best execution time, with a makespan ratio almost equal to 1, and a negligible increase of execution time less than 0.8%. Regarding the comparison with the average time, we obtained a good performance improvement, with a percentage reduction of makespan equal to 24.71% and a makespan ratio equal to 1.25. These results show how the choice of an unsuitable block size may lead to a degradation of performance, which can be avoided with the aid of the proposed methodology. We further stressed this aspect by comparing $t^*$ with the worst execution time, achieving a remarkable percentage reduction of makespan equal to 55% and a makespan ratio equal to 2.11. Measured values confirm the ability of our methodology in supporting the execution of machine learning algorithms in parallel and distributed environments.

As a last step, to make more detailed and clear the benefits brought by the use of our methodology, Figure 8.4 and 8.5 show the results achieved with K-means and Random Forest in three possible cases, in which the number of rows and columns can be equal or very imbalanced.

Specifically, a synthetic test dataset for each case was generated as follows:

- $n >> m$: 500,000 rows, 1000 columns.
- $n << m$: 1000 rows, 500,000 columns.
- $n \approx m$: 10,000 rows, 10,000 columns.

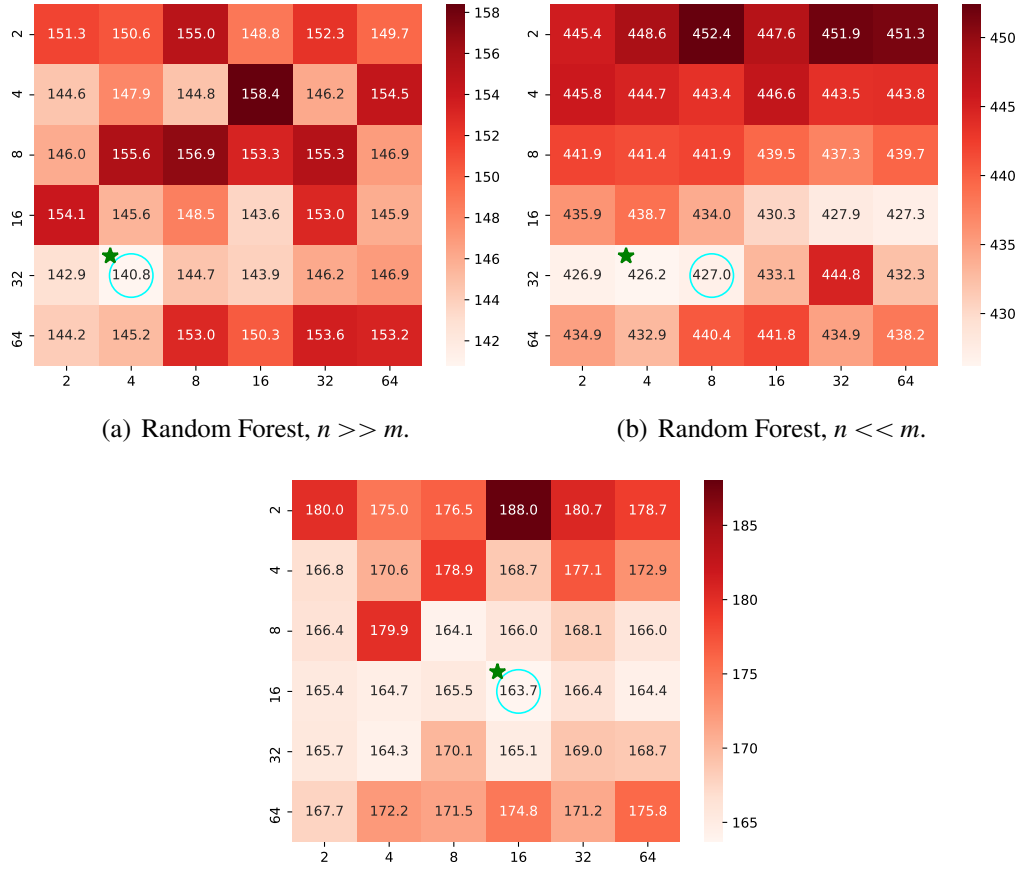



(a) K-means, $n >> m$.

(b) K-means, $n << m$.

(c) K-means, $n \approx m$.

**Figure 8.4:** Execution times achieved by running K-means on datasets of both balanced and imbalanced shape. The $x$ and $y$ axis show the number of blocks along rows and columns. The time obtained with the predicted block size is marked by the cyan circle, while the best one by the green star.




(a) Random Forest, $n >> m$.

(b) Random Forest, $n << m$.

(c) Random Forest, $n \approx m$.

**Figure 8.5:** Execution times achieved by running Random Forest on datasets of both balanced and imbalanced shape. The $x$ and $y$ axis show the number of blocks along rows and columns. The time obtained with the predicted block size is marked by the cyan circle, while the best one by the green star.

Even in this case, we compared execution times achieved by using the predicted partitioning and all other possible partitionings, set using progressive powers of 2 from 2 to 64 for both the number of row and column blocks.

By observing Figure 8.4, we can see that, even in the presence of a high imbalance, the algorithm always suggests a block size value very close or equal to the best one, thus allowing an efficient execution of the K-means algorithm. Particularly, the time obtained by using the predicted block size is marked by the cyan circle, while the best one is marked by the green star. Moreover, the heatmap is

useful to show how the variation of the block size affects the execution time in all the considered configurations. In particular, we used different gradations of red, where a greater intensity corresponds to a higher execution time, which in turn implies the choice of an unsuitable block size.

We observed that the time $t^*$ obtained by using the predicted block size leads to the second best time in the first two cases, and to the best time for the last one. The mean percentage difference between $t^*$ and the best time is almost equal to 1%, which shows how the partitioning predicted by the model is a very good estimate of the optimal one. Moreover, by comparing $t^*$ with the average and worst times, we measured an average makespan ratio of 1.17 and 1.53 and an average percentage improvement of makespan equal to 14.27% and 34.44%.

The good results achieved with K-means are confirmed by the experiments performed on Random Forest, shown in Figure 8.5. In this case, $t^*$ resulted in the best execution time in two cases out of three (i.e., the first and the third), and the third best time in the remaining case (i.e., the second). In particular, we measured a negligible difference of 0.56% between $t^*$ and the best execution time. Moreover, by comparing $t^*$ with the average and worst times we observed a makespan ratio of 1.03 and 1.10 and a percentage makespan reduction of 3.74% and 9.44%. All the described results, achieved by K-means and Random Forest, are summarized in Table 8.4.

| *Algorithm* | *Metric* | *Best* | *Average* | *Worst* |
|---|---|---|---|---|
| K-means | Makespan ratio | 0.99 | 1.17 | 1.53 |
| | Makespan reduction (%) | −1.03% | 14.27% | 34.44% |
| Random Forest | Makespan ratio | 0.99 | 1.03 | 1.10 |
| | Makespan reduction (%) | −0.56% | 3.74% | 9.44% |

**Table 8.4:** Average makespan ratio and percentage makespan reduction measured by running K-means and Random Forest on datasets of both balanced and imbalanced shape.

### 8.4.2 Multi-node experiments

We further investigated the effectiveness of our solution in a distributed execution environment. Specifically, we leveraged the MareNostrum 4 supercomputer (MN4) [169], located at the Barcelona Supercomputing Center. Its current peak performance is 11.15 Petaflops and it is composed of 3456 nodes, each of which has two Intel®Xeon Platinum 8160 (24 cores at 2,1 GHz each) and 96 GB of main memory. It has also 100 GB Intel®Omni-Path Full-Fat Tree Interconnection, and 14 PB of shared disk storage managed by the Global Parallel File System [187].

In this experimental evaluation, we focused on the execution of the Principal Component Analysis (PCA) algorithm. It is a dimensionality reduction algorithm, whose aim is to compute a meaningful low-dimensional representation of the input data by projecting each sample onto only the first few principal components. The log used for the extraction of the training data was enriched using several real-world datasets, listed in the following, which belong to different fields, ranging from medicine to particle physics.

- *Diabetes*: medical data, used for predicting whether or not a patient has diabetes, based on diagnostic measurements.
- *Cleveland*: medical data, used for predicting the heart disease risk based on clinical measurements.
- *Banknote*: high-resolution images, used for evaluating if a banknote is authentic or forgery.
- *Superconductors* [188]: superconductors data, used for predicting the critical temperature.
- *Accelerometer* [189]: accelerometer data, used for predicting motor failures.
- *1ubq.bck.10.crd*, *1ubq.bck.1.crd*, *1ubq.heavy.1.crd*: particle physics datasets, containing up to 1 million atom trajectories described by a varying number of features and obtained from GROMACS simulations [190].

For the experimental evaluation, we used three test datasets containing biomolecular simulation data, specifically information about atom trajectories in a *mdrcd*[2] format. These datasets are described in Table 8.5.

| Dataset name | # atoms | # rows | # columns |
|---|---|---|---|
| Traj_medium | 6912 | 60,000 | 20,736 |
| Traj_large | 19,848 | 100,000 | 59,544 |
| Traj_xlarge | 31,632 | 100,000 | 94,896 |

**Table 8.5:** Test datasets used to evaluate the benefits brought by the proposed methodology on the execution of the PCA algorithm on the MN4 supercomputer.

For the execution of our experiments, we employed 16 nodes of the MN4 supercomputer, with 96 GB of RAM per node. In addition, the number of used cores per task with the large and extra-large datasets was set to 24 due to the heavy computation and their big memory size, while for the medium dataset, we used 8 cores per task.

Unlike the experiments shown in section 8.4.1, we did not consider the large set of all possible partitionings, as the huge size of test datasets could have led to an excessively expensive process, due to time-consuming and resource-intensive computation. In this case, instead, we compared the time achieved by using the predicted partitioning $(p_r^*, p_c^*)$ against the best partitioning that was individuated by domain experts $(\hat{p}_r, \hat{p}_c)$, by following a trial and error approach. The obtained results are shown in Table 8.6.

| Dataset | Predicted partitioning | | | Manual partitioning | | |
|---|---|---|---|---|---|---|
| | $p_r^*$ | $p_c^*$ | *Time (s)* | $\hat{p}_r$ | $\hat{p}_c$ | *Time (s)* |
| Traj_medium | 4 | 16 | 270 | 6 | 21 | 484 |
| Traj_large | 8 | 40 | 1123 | 14 | 36 | 1096 |
| Traj_xlarge | 8 | 48 | 1770 | 14 | 48 | 1825 |

**Table 8.6:** Results obtained in MareNostrum 4 using model predictions and domain expert estimates.

[2] Amber trajectory format, https://ambermd.org/FileFormats.php

By comparing the time achieved by using the block size predicted by the model with that estimated by the domain experts, we achieved quite good results, with an average value of makespan ratio and makespan percentage reduction equal to 1.27 and 14.92%, respectively. In addition, as reported in Table 8.6, it can be seen that the data partitioning suggested by the model is the best one in two out of three test cases, resulting in the shortest execution time. Moreover, it is worth noticing that in the remaining case, corresponding to the *Traj_large* dataset, the relative difference between the two execution times is quite small ($\approx 2\%$). The prediction is indeed reasonably good, as it can be calculated quickly without involving any trial and error approach and requiring a quite small amount of resources and domain knowledge.

For the sake of completeness, in Figure 8.6 we provide the execution times measured by executing the PCA algorithm on the *Traj_medium* dataset with all possible partitionings, computed using progressive powers of 2 from 2 to 64, leading to a maximum number of blocks and tasks equal to $64^2$.


**Figure 8.6:** Execution times achieved in MareNostrum 4 by running PCA on the *Traj_medium* dataset. The *x* and *y* axis show the number of blocks along rows and columns. The time obtained with the predicted block size is marked by the cyan circle, while the best one by the green star.

The plot shows that the partition predicted by the model is the third best possible. Nevertheless, while not leading to the minimum execution time, the estimate is still an excellent approximation, obtainable in a very efficient way, and effectively exploitable to allow the efficient execution of PCA. Finally, we compared the time obtained by using the predicted partitioning with the average and worst times shown in Figure 8.6. Specifically, we measured a makespan ratio equal to 3.54 and 13.59 and a percentage reduction of makespan of 71.75% and 92.64% compared to the average and worst times. All of these results further confirm how crucial it is to choose a suitable partitioning for running data-intensive applications in high-performance distributed environments.

## 8.5 Conclusions

Data-intensive applications are widespread in several domains, such as bioinformatics, high-energy physics, and the modeling of natural phenomena. In such applications, an effective strategy for data partitioning is crucial to enable their efficient execution in distributed HPC environments. This chapter described a novel methodology, aimed at determining a good estimate for data block size quickly and requiring few resources and domain knowledge. Our methodology was evaluated on the dislib library of PyCOMPSs, considering different execution environments, including the MareNostrum 4 supercomputer, and different real-world datasets. Experimental results show how the proposed machine learning solution can lead to a significant improvement in application performance and a reduction in execution time. In future work, we plan to improve our methodology to make it even more generic and usable. In particular, we will extend it to support the choice of other parameters required to configure a distributed environment, such as the number of nodes and the RAM to be assigned to each node. Finally, to further validate the proposed methodology, it can be applied to frameworks and libraries other than PyCOMPSs and dislib, since it can be exploited in any case data partitioning is essential to improve application performance and scalability.

Chapter 9

# Conclusions and final remarks

Social media platforms are now became part of everyday life, allowing the interconnection of people around the world in large discussion groups relating to several topics, including important socio-political issues. Therefore, social media have become a valuable source of information-rich multi-modal data, effectively exploitable in several application fields, ranging from computational politics to sociology, linguistics, economics, finance, and computer science.

This thesis mainly focused on the analysis of politically-polarized data, with the aim of outlining a detailed profile of social users, investigating their interests, opinions, and feelings, and shaping their perception of real-world facts and events. During our research activity, we developed several innovative methodologies, described throughout this thesis, demonstrating how the analysis of these data can provide an effective data-driven approach to a thorough understanding of political phenomena, such as elections and referenda. Specifically, we designed effective solutions to investigate the political leaning of social users, also studying the relationships between user polarization and the sentiment expressed in referring to the different candidates, by modeling political support across a broad spectrum of emotions. Moreover, we combined information diffusion and influence maximization with political polarization analysis, to identify the main influencers for the different factions and derive the main information diffusion strategies adopted during the political campaign. Furthermore, to achieve a rich representation of

social media conversation, we worked on the extraction of the main topics underlying the online discussion, following their evolution over time, and characterizing them from a political perspective.

During the development of such methodologies, we addressed several issues and challenges intrinsically related to Big Social Data. In particular, we dealt with the statistical significance of election-related data, in which different kinds of biases may be present, due to the distribution of users in terms of gender, age, culture, and social status, as well as technical biases related to data availability policies. We addressed issues related to data reliability, dealing with the presence of social bots, whose aim is to manipulate and pollute online published content for altering the popularity of users. Specifically, we showed that by filtering out the spamming activity of social bots, it is possible to achieve a more reliable estimate of the true voting intentions of legitimate users. We coped with the high dynamicity of information extracted from social media, which continuously varies over time. In particular, we developed effective solutions that are aware of temporal aspects, such as the evolution of discussion topics through time and the fluctuation of voting intentions of social users during election campaigns. We addressed language barrier issues, by both developing language-agnostic models and exploiting the most recent multilingual transformer-based language representation models. These models were also leveraged to support hashtag-based Big Social Data analysis techniques, through the design of recommendation methodologies able to suggest high-quality hashtags, in line with both the semantics of posts and the latest trends. The last challenge addressed in this thesis concerned the resource-intensive computation involved in analyzing Social Big Data. Indeed, due to their high volume and speed, these data continually challenge today's storage, processing, and analysis capabilities. In this regard, our research focused on the study of ad-hoc techniques and strategies aimed at enhancing the execution of data-intensive high-parallel applications. Specifically, we designed novel machine learning-based solutions to enhance workflow scheduling and data partitioning in distributed and high-performance environments.

As regards the main future directions of the research presented in this thesis, we mention the investigation of more sophisticated time-adaptive models for Big Social Data analysis, the development of opinion mining models able to effectively learn from scarcely supervised datasets, and the design of distributed strategies to learn from Big Social Data at the network edge, thus ensuring low latency, privacy preservation, and scalability.

# List of Figures

# List of Tables

# References


[1] Loris Belcastro et al. “Learning Political Polarization on Social Media Using Neural Networks”. In: *IEEE Access* 8 (2020), pp. 47177–47187.

[2] Ekaterina Olshannikova et al. “Conceptualizing Big Social Data”. In: *Journal of Big Data* 4.1 (2017), p. 3.

[3] Riccardo Cantini et al. “A Weighted Artificial Bee Colony Algorithm for Influence Maximization”. In: *Online Social Networks and Media* 26 (2021), p. 100167.

[4] Massimo Stella, Valerio Restocchi, and Simon De Deyne. “# Lockdown: Network-enhanced Emotional Profiling in the Time of Covid-19”. In: *Big Data and Cognitive Computing* 4.2 (2020), p. 14.

[5] Loris Belcastro, Fabrizio Marozzo, and Emanuele Perrella. “Automatic Detection of User Trajectories from Social Media Posts”. In: *Expert Systems with Applications* 186 (2021), p. 115733.

[6] Riccardo Cantini and Fabrizio Marozzo. “Topic Detection and Tracking in Social Media Platforms”. In: *EAI International Conference on Pervasive knowledge and collective intelligence on Web and Social Media*. 2022.

[7] Riccardo Cantini et al. “Analyzing Political Polarization on Social Media by Deleting Bot Spamming”. In: *Big Data and Cognitive Computing* 6.1 (2022). ISSN: 2504-2289.

[8] Loris Belcastro et al. “Analyzing voter behavior on social media during the 2020 US presidential election campaign”. In: *Social Network Analysis and Mining* 12.1 (2022), pp. 1–16.

[9] Riccardo Cantini et al. “Learning Sentence-to-Hashtags Semantic Mapping for Hashtag Recommendation on Microblogs”. In: *ACM Transactions on Knowledge Discovery from Data* 16.2 (2022), pp. 1–26.

[10] Loris Belcastro et al. “Programming big data analysis: principles and solutions”. In: *Journal of Big Data* 9.1 (2022), pp. 1–50.

[11] Riccardo Cantini et al. “Exploiting Machine Learning For Improving In-memory Execution of Data-intensive Workflows on Parallel Machines”. In: *Future Internet* 13.5 (2021).

[12] Riccardo Cantini et al. “Block size estimation for data partitioning in HPC applications using machine learning techniques”. In: *arXiv preprint arXiv: 2211.10819* (2022).

[13] Daniel Cer et al. “Universal Sentence Encoder”. In: *arXiv preprint arXiv: 1803.11175* (2018).

[14] Yinfei Yang et al. “Multilingual universal sentence encoder for semantic retrieval”. In: *arXiv preprint arXiv:1907.04307* (2019).

[15] Jacob Devlin et al. “Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[16] Telmo Pires, Eva Schlinger, and Dan Garrette. “How multilingual is multilingual bert?” In: *arXiv preprint arXiv:1906.01502* (2019).

[17] Muhammad Bilal et al. “Predicting elections: Social media data and techniques”. In: *2019 international conference on engineering and emerging technologies (ICEET)*. IEEE. 2019, pp. 1–6.

[18] Alessandro Bessi and Emilio Ferrara. “Social Bots Distort the 2016 US Presidential Election Online Discussion”. In: *First Monday* 21.11 (2016).

[19] Domenico Talia. "A view of programming scalable data analysis: from clouds to exascale". In: *Journal of Cloud Computing* 8.1 (2019), pp. 1–16.

[20] L. Belcastro et al. "Discovering Political Polarization on Social Media: A Case Study". In: *2019 15th International Conference on Semantics, Knowledge and Grids (SKG)*. Sept. 2019, pp. 182–189.

[21] Stefan Spettel and Dimitrios Vagianos. "Twitter Analyzer—How to Use Semantic Analysis to Retrieve an Atmospheric Image around Political Topics in Twitter". In: *Big Data and Cognitive Computing* 3.3 (2019), p. 38.

[22] Lazaros Oikonomou and Christos Tjortjis. "A Method for Predicting the Winner of the USA Presidential Elections using Data extracted from Twitter". In: *2018 South-Eastern European Design Automation, Computer Engineering, Computer Networks and Society Media Conference*. IEEE. 2018, pp. 1–8.

[23] Imane El Alaoui et al. "A Novel Adaptable Approach for Sentiment Analysis on Big Social Data". In: *Journal of Big Data* 5.1 (2018), p. 12.

[24] Delenn Chin, Anna Zappone, and Jessica Zhao. "Analyzing Twitter sentiment of the 2016 presidential candidates". In: *American Journal Of Science and Research* 1 (Jan. 2016), pp. 128–137.

[25] Fabrizio Marozzo and Alessandro Bessi. "Analyzing Polarization of Social Media Users and News Sites during Political Campaigns". In: *Social Network Analysis and Mining* 8.1 (2018), p. 1.

[26] Ehsan Ul Haq et al. "A survey on computational politics". In: *IEEE Access* 8 (2020), pp. 197379–197406.

[27] Catherine Grevet, Loren G Terveen, and Eric Gilbert. "Managing political differences in social media". In: *Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing*. 2014, pp. 1400–1408.

[28] Marco Toledo Bastos, Cornelius Puschmann, and Rodrigo Travitzki. "Tweeting across hashtags: overlapping users and the importance of language, topics, and politics". In: *Proceedings of the 24th ACM conference on hypertext and social media*. 2013, pp. 164–168.

[29] Ophélie Fraisier et al. "Uncovering like-minded political communities on twitter". In: *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*. 2017, pp. 261–264.

[30] Shu-I Chiu and Kuo-Wei Hsu. "Predicting political tendency of posts on facebook". In: *Proceedings of the 2018 7th International Conference on Software and Computer Applications*. 2018, pp. 110–114.

[31] Hiroki Takikawa and Kikuko Nagayoshi. "Political polarization in social media: Analysis of the "Twitter political field" in Japan". In: *2017 IEEE International Conference on Big Data (Big Data)*. IEEE. 2017, pp. 3143–3150.

[32] Jooyeon Kim et al. "Leveraging the crowd to detect and reduce the spread of fake news and misinformation". In: *Proceedings of the eleventh ACM international conference on web search and data mining*. 2018, pp. 324–332.

[33] Giovanni Luca Ciampaglia et al. "Computational fact checking from knowledge networks". In: *PloS one* 10.6 (2015), e0128193.

[34] Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. "Combating web spam with trustrank". In: *Proceedings of the 30th international conference on very large data bases (VLDB)*. 2004.

[35] Kiran Garimella et al. "Political discourse on social media: Echo chambers, gatekeepers, and the price of bipartisanship". In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 913–922.

[36] Jisun An, Daniele Quercia, and Jon Crowcroft. "Fragmented social media: a look into selective exposure to political news". In: *Proceedings of the 22nd international conference on world wide web*. 2013, pp. 51–52.

[37] Kai Shu, H Russell Bernard, and Huan Liu. "Studying fake news via network analysis: detection and mitigation". In: *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*. Springer, 2019, pp. 43–65.

[38] Derek Greene and James P Cross. "Unveiling the political agenda of the european parliament plenary: A topical analysis". In: *Proceedings of the ACM web science conference*. 2015, pp. 1–10.

[39] Amine Trabelsi and Osmar R Zaiane. "Phaitv: A phrase author interaction topic viewpoint model for the summarization of reasons expressed by polarized stances". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 13. 2019, pp. 482–492.

[40] Yaser Keneshloo et al. "Detecting and forecasting domestic political crises: A graph-based approach". In: *Proceedings of the 2014 ACM conference on Web science*. 2014, pp. 192–196.

[41] Christian Pieter Hoffmann and Christoph Lutz. "Spiral of silence 2.0: Political self-censorship among young Facebook users". In: *Proceedings of the 8th international conference on social media & society*. 2017, pp. 1–12.

[42] Hosein Azarbonyad et al. "Words are malleable: Computing semantic shifts in political and media discourse". In: *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2017, pp. 1509–1518.

[43] Corrado Monti et al. "Modelling political disaffection from Twitter data". In: *Proceedings of the second international workshop on issues of sentiment discovery and opinion mining*. 2013, pp. 1–9.

[44] Volker Wulf et al. "Fighting against the wall: Social media use by political activists in a Palestinian village". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2013, pp. 1979–1988.

[45] Sounman Hong and Daniel Nadler. "Social media and political voices of organized interest groups: a descriptive analysis". In: *Proceedings of the 16th Annual International Conference on Digital Government Research*. 2015, pp. 210–216.

[46] Pedro Saleiro, Luis Gomes, and Carlos Soares. "Sentiment aggregate functions for political opinion polling using microblog streams". In: *Proceedings of the Ninth International C* Conference on Computer Science & Software Engineering*. 2016, pp. 44–50.

[47] Julia Cambre, Scott R Klemmer, and Chinmay Kulkarni. "Escaping the echo chamber: ideologically and geographically diverse discussions about politics". In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 2017, pp. 2423–2428.

[48] Martyn Dade-Robertson et al. "The political sensorium". In: *Proceedings of the 4th Media Architecture Biennale Conference: Participation*. 2012, pp. 47–50.

[49] Kokil Jaidka et al. "Predicting elections from social media: a three-country, three-method comparative study". In: *Asian Journal of Communication* 29.3 (2019), pp. 252–273.

[50] Manish Gaurav et al. "Leveraging candidate popularity on Twitter to predict election outcome". In: *Proceedings of the 7th workshop on social network mining and analysis*. 2013, pp. 1–8.

[51] Andranik Tumasjan et al. "Predicting elections with twitter: What 140 characters reveal about political sentiment". In: *Proceedings of the International AAAI Conference on Web and Social Media*. Vol. 4. 1. 2010.

[52] Pete Burnap et al. "140 Characters to Victory?: Using Twitter to Predict the UK 2015 General Election". In: *Electoral Studies* 41 (2016), pp. 230–233.

[53] Freimut Bodendorf and Carolin Kaiser. "Detecting opinion leaders and trends in online social networks". In: *Proceedings of the 2nd ACM workshop on Social web search and mining*. 2009, pp. 65–68.

[54] Emilie M Hafner-Burton and Alexander H Montgomery. "Centrality in politics: How networks confer power". In: (2010).

[55] Patrick R Miller et al. "Talking politics on Facebook: Network centrality and political discussion practices in social media". In: *Political Research Quarterly* 68.2 (2015), pp. 377–391.

[56] Muhammed K Olorunnimbe and Herna L Viktor. "Tweets as a Vote: Exploring Political Sentiments on Twitter for Opinion Mining". In: *International Symposium on Methodologies for Intelligent Systems*. Springer. 2015, pp. 180–185.

[57] Alexandre Bovet, Flaviano Morone, and Hernán A Makse. "Predicting election trends with Twitter: Hillary Clinton versus Donald Trump". In: *arXiv. org* (2016).

[58] Felix Ming Fai Wong et al. "Quantifying Political Leaning from Tweets, Retweets, and Retweeters". In: *IEEE Transactions on Knowledge and Data Engineering* 28.8 (2016), pp. 2158–2172.

[59] Saud Alashri et al. "An Analysis of Sentiments on Facebook during the 2016 US Presidential Election". In: *2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE. 2016, pp. 795–802.

[60] Christopher D Manning et al. "The Stanford CoreNLP natural language processing toolkit". In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60.

[61] Aditya Singh et al. "Predicting Elections Results using Social Media Activity A Case Study: USA Presidential Election 2020". In: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. Vol. 1. 2021, pp. 314–319.

[62] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[63] Mike Thelwall. “The Heart and soul of the web? Sentiment strength detection in the social web with SentiStrength”. In: *Cyberemotions*. Springer, 2017, pp. 119–134.

[64] Saif M Mohammad and Peter D Turney. “Crowdsourcing a word–emotion association lexicon”. In: *Computational intelligence* 29.3 (2013), pp. 436–465.

[65] Svetlana Kiritchenko, Xiaodan Zhu, and Saif M Mohammad. “Sentiment analysis of short informal texts”. In: *Journal of Artificial Intelligence Research* 50 (2014), pp. 723–762.

[66] Saif Mohammad. “Portable features for classifying emotional text”. In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2012, pp. 587–591.

[67] Preslav Nakov et al. “Developing a successful SemEval task in sentiment analysis of Twitter and other social media texts”. In: *Language Resources and Evaluation* 50.1 (2016), pp. 35–65.

[68] Robert Plutchik. “The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice”. In: *American scientist* 89.4 (2001), pp. 344–350.

[69] Emily Chen, Ashok Deb, and Emilio Ferrara. “# Election2020: the first public Twitter dataset on the 2020 US Presidential election”. In: *Journal of Computational Social Science* (2021), pp. 1–18.

[70] Clayton Allen Davis et al. “Botornot: A system to Evaluate Social Bots”. In: *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 273–274.

[71] Hunt Allcott and Matthew Gentzkow. “Social Media and Fake News in the 2016 Election”. In: *Journal of Economic Perspectives* 31.2 (2017), pp. 211–36.

[72] Zhiwei Jin et al. “Detection and Analysis of 2016 US Presidential Election Related Rumors on Twitter”. In: *International Conference on Social Computing, Behavioral-cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation*. Springer. 2017, pp. 14–24.

[73] Chengcheng Shao et al. “The Spread of Fake News by Social Bots”. In: *CoRR* abs/1707.07592 (2017).

[74] Eiman Alothali et al. “Detecting Social Bots on Twitter: A Literature Review”. In: *2018 International Conference on Innovations in Information Technology (IIT)*. 2018, pp. 175–180.

[75] Kayode Sakariyah Adewole et al. “Malicious Accounts: Dark of the Social Networks”. In: *Journal of Network and Computer Applications* 79 (2017), pp. 41–67.

[76] Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. “Random Walk Based Fake Account Detection in Online Social Networks”. In: *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2017, pp. 273–284.

[77] Ashish Mehrotra, Mallidi Sarreddy, and Sanjay Singh. “Detection of Fake Twitter Followers Using Graph Centrality Measures”. In: *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*. IEEE. 2016, pp. 499–504.

[78] Venkatramanan S Subrahmanian et al. “The DARPA Twitter Bot Challenge”. In: *Computer* 49.6 (2016), pp. 38–46.

[79] Zafar Gilani, Ekaterina Kochmar, and Jon Crowcroft. “Classification of Twitter Accounts into Automated Agents and Human Users”. In: *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 2017, pp. 489–496.

[80] Abdulrahman Alarifi, Mansour Alsaleh, and AbdulMalik Al-Salman. “Twitter Turing Test: Identifying Social Machines”. In: *Information Sciences* 372 (2016), pp. 332–346.

[81] Mücahit Kantepe and Murat Can Ganiz. “Preprocessing Framework for Twitter Bot Detection”. In: *2017 International Conference on Computer Science and Engineering*. IEEE. 2017, pp. 630–634.

[82] Buket Erşahin et al. “Twitter Fake Account Detection”. In: *2017 International Conference on Computer Science and Engineering (UBMK)*. IEEE. 2017, pp. 388–392.

[83] Chiyu Cai, Linjing Li, and Daniel Zengi. “Behavior Enhanced Deep Bot Detection in Social Media”. In: *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE. 2017, pp. 128–130.

[84] Allan Borodin, Yuval Filmus, and Joel Oren. “Threshold Models for Competitive Influence in Social Networks”. In: *International Workshop on Internet and Network Economics*. Springer. 2010, pp. 539–550.

[85] Suman Banerjee, Mamata Jenamani, and Dilip Kumar Pratihar. “A survey on influence maximization in a social network”. In: *Knowledge and Information Systems* (2020), pp. 1–39.

[86] Pedro Domingos and Matt Richardson. “Mining the network value of customers”. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 2001, pp. 57–66.

[87] A.-A. Stoica and A. Chaintreau. “Fairness in social influence maximization”. In: 2019, pp. 569–574.

[88] Dervis Karaboga. *An idea based on honey bee swarm for numerical optimization*. Tech. rep. Technical report-tr06, Erciyes university, engineering faculty, computer engineering department, 2005.

[89] C Prem Sankar, S Asharaf, and K Satheesh Kumar. “Learning from bees: An approach for influence maximization on viral campaigns”. In: *PloS one* 11.12 (2016).

[90] David Kempe, Jon Kleinberg, and Éva Tardos. "Maximizing the spread of influence through a social network". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2003, pp. 137–146.

[91] Kazumi Saito, Ryohei Nakano, and Masahiro Kimura. "Prediction of information diffusion probabilities for independent cascade model". In: *International conference on knowledge-based and intelligent information and engineering systems*. Springer. 2008, pp. 67–75.

[92] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. "An analysis of approximations for maximizing submodular set functions—I". In: *Mathematical programming* 14.1 (1978), pp. 265–294.

[93] Yuchen Li et al. "Influence maximization on social graphs: A survey". In: *IEEE Transactions on Knowledge and Data Engineering* 30.10 (2018), pp. 1852–1872.

[94] Jure Leskovec et al. "Cost-effective outbreak detection in networks". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2007, pp. 420–429.

[95] Arastoo Bozorgi et al. "Community-based influence maximization in social networks under a competitive linear threshold model". In: *Knowledge-Based Systems* 134 (2017), pp. 149–158.

[96] Wei Chen, Yajun Wang, and Siyu Yang. "Efficient influence maximization in social networks". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2009, pp. 199–208.

[97] Kyomin Jung, Wooram Heo, and Wei Chen. "Irie: Scalable and robust influence maximization in social networks". In: *2012 IEEE 12th International Conference on Data Mining*. IEEE. 2012, pp. 918–923.

[98] Masahiro Kimura and Kazumi Saito. "Tractable models for information diffusion in social networks". In: *European conference on principles of data mining and knowledge discovery*. Springer. 2006, pp. 259–271.

[99] Wei Chen, Yifei Yuan, and Li Zhang. "Scalable influence maximization in social networks under the linear threshold model". In: *2010 IEEE international conference on data mining*. IEEE. 2010, pp. 88–97.

[100] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. "Simpath: An efficient algorithm for influence maximization under the linear threshold model". In: *2011 IEEE 11th international conference on data mining*. IEEE. 2011, pp. 211–220.

[101] Jong-Ryul Lee and Chin-Wan Chung. "A fast approximation for influence maximization in large social networks". In: *Proceedings of the 23rd international conference on World Wide Web*. 2014, pp. 1157–1162.

[102] Christian Borgs et al. "Maximizing social influence in nearly optimal time". In: *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM. 2014, pp. 946–957.

[103] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. "Topic-aware social influence propagation models". In: *Knowledge and information systems* 37.3 (2013), pp. 555–584.

[104] Wei Chen, Wei Lu, and Ning Zhang. "Time-critical influence maximization in social networks with time-delayed diffusion process". In: *Twenty-Sixth AAAI Conference on Artificial Intelligence*. 2012.

[105] Jinha Kim, Wonyeol Lee, and Hwanjo Yu. "CT-IC: Continuously activated and time-restricted independent cascade model for viral marketing". In: *Knowledge-Based Systems* 62 (2014), pp. 57–68.

[106] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. "A data-based approach to social influence maximization". In: *arXiv preprint arXiv:1109.6886* (2011).

[107] Michalis Mavrovouniotis, Changhe Li, and Shengxiang Yang. "A survey of swarm intelligence for dynamic optimization: Algorithms and applications". In: *Swarm and Evolutionary Computation* 33 (2017), pp. 1–17.

[108] Joe R Riley et al. “The flight paths of honeybees recruited by the waggle dance”. In: *Nature* 435.7039 (2005), pp. 205–207.

[109] Sergey Brin and Lawrence Page. “The anatomy of a large-scale hypertextual web search engine”. In: *Computer networks and ISDN systems* 30.1-7 (1998), pp. 107–117.

[110] Fréderic Godin et al. “Using topic models for twitter hashtag recommendation”. In: *Proceedings of the 22nd International Conference on World Wide Web*. ACM. 2013, pp. 593–596.

[111] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[112] Dhruv Mahajan et al. “Exploring the limits of weakly supervised pretraining”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 181–196.

[113] Yang Li et al. “Topical Co-Attention Networks for hashtag recommendation on microblogs”. In: *Neurocomputing* 331 (2019), pp. 356–365.

[114] Yeyun Gong, Qi Zhang, and Xuanjing Huang. “Hashtag recommendation for multimodal microblog posts”. In: *Neurocomputing* 272 (2018), pp. 170–177.

[115] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[116] Mohit Iyyer et al. “Deep unordered composition rivals syntactic methods for text classification”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2015, pp. 1681–1691.

[117] Samuel R. Bowman et al. “A large annotated corpus for learning natural language inference”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[118] Ryan Kiros et al. “Skip-Thought Vectors”. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*. NIPS’15. Montreal, Canada, 2015, pp. 3294–3302.

[119] Matthew L. Henderson et al. “Efficient Natural Language Response Suggestion for Smart Reply”. In: *CoRR* abs/1705.00652 (2017). arXiv: 1705.00652.

[120] Alec Radford et al. “Improving language understanding with unsupervised learning”. In: (2018).

[121] Matthew E Peters et al. “Deep contextualized word representations”. In: *arXiv preprint arXiv:1802.05365* (2018).

[122] Jieying She and Lei Chen. “Tomoha: Topic model-based hashtag recommendation on twitter”. In: *Proceedings of the 23rd International Conference on World Wide Web*. ACM. 2014, pp. 371–372.

[123] Junbiao Pang et al. “Unsupervised web topic detection using a ranked clustering-like pattern across similarity cascades”. In: *IEEE Transactions on Multimedia* 17.6 (2015), pp. 843–853.

[124] Nada Ben-Lhachemi and El Habib Nfaoui. “Using tweets embeddings for hashtag recommendation in Twitter”. In: *Procedia Computer Science* 127 (2018), pp. 7–15.

[125] Jiajia Huang, Min Peng, and Hua Wang. “Topic detection from large scale of microblog stream with high utility pattern clustering”. In: *Proceedings of the 8th Workshop on Ph. D. Workshop in Information and Knowledge Management*. ACM. 2015, pp. 3–10.

[126] Eriko Otsuka, Scott A Wallace, and David Chiu. “A hashtag recommendation system for twitter data streams”. In: *Computational social networks* 3.1 (2016), p. 3.

[127] Alexander M Rush, Sumit Chopra, and Jason Weston. “A neural attention model for abstractive sentence summarization”. In: *arXiv preprint arXiv:1509.00685* (2015).

[128] Tim Rocktäschel et al. “Reasoning about entailment with neural attention”. In: *arXiv preprint arXiv:1509.06664* (2015).

[129] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).

[130] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. “Effective approaches to attention-based neural machine translation”. In: *arXiv preprint arXiv:1508.04025* (2015).

[131] Jiasen Lu et al. “Hierarchical question-image co-attention for visual question answering”. In: *Advances in neural information processing systems*. 2016, pp. 289–297.

[132] Shi Feng et al. “Attention based hierarchical LSTM network for context-aware microblog sentiment classification”. In: *World Wide Web* 22.1 (2019), pp. 59–81.

[133] Abhay Kumar et al. “From Fully Supervised to Zero Shot Settings for Twitter Hashtag Recommendation”. In: *arXiv preprint arXiv:1906.04914* (2019).

[134] Wayne Xin Zhao et al. “Comparing twitter and traditional media using topic models”. In: *European conference on information retrieval*. Springer. 2011, pp. 338–349.

[135] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[136] Mihael Ankerst et al. “OPTICS: ordering points to identify the clustering structure”. In: *ACM Sigmod record* 28.2 (1999), pp. 49–60.

[137] Armand Joulin et al. “Learning visual features from large weakly supervised data”. In: *European Conference on Computer Vision*. Springer. 2016, pp. 67–84.

[138] Rabindra Lamsal. *Coronavirus (COVID-19) Tweets Dataset*. 2020. URL: https://dx.doi.org/10.21227/781w-ef42.

[139] Domenico Talia, Paolo Trunfio, and Fabrizio Marozzo. *Data Analysis in the Cloud*. Elsevier, Oct. 2015.

[140] D. Talia. "Workflow Systems for Science: Concepts and Tools". In: *International Scholarly Research Notices* 2013 (2013), pp. 1–15.

[141] Georges Da Costa et al. "Exascale machines require new programming paradigms and runtimes". In: *Supercomputing Frontiers and Innovations* 2.2 (2015), pp. 6–27.

[142] Min Li et al. "SparkBench: A Comprehensive Benchmarking Suite for in Memory Data Analytic Platform Spark". In: *Proceedings of the 12th ACM International Conference on Computing Frontiers*. CF '15. Ischia, Italy: Association for Computing Machinery, 2015. ISBN: 9781450333580.

[143] Daniel CM De Oliveira, Ji Liu, and Esther Pacitti. "Data-intensive workflow management: for clouds and data-intensive and scalable computing environments". In: *Synthesis Lectures on Data Management* 14.4 (2019), pp. 1–179.

[144] A. Verma, A. H. Mansuri, and N. Jain. "Big data management processing with Hadoop MapReduce and spark technology: A comparison". In: *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*. 2016, pp. 1–4.

[145] Matei Zaharia et al. "Resilient distributed datasets: A {Fault-Tolerant} abstraction for {In-Memory} cluster computing". In: *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. 2012, pp. 15–28.

[146] Yassir Samadi, Mostapha Zbakh, and Claude Tadonki. "Performance comparison between Hadoop and Spark frameworks using HiBench benchmarks". In: *Concurrency and Computation: Practice and Experience* 30.12 (2018), e4367.

[147] Christina Delimitrou and Christos Kozyrakis. “Quasar: Resource-Efficient and QoS-Aware Cluster Management”. In: *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’14. Salt Lake City, Utah, USA: Association for Computing Machinery, 2014, pp. 127–144.

[148] Q. Llull et al. “Cooper: Task Colocation with Cooperative Games”. In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017, pp. 421–432.

[149] Vicent Sanz Marco et al. “Improving Spark Application Throughput via Memory Aware Task Co-Location: A Mixture of Experts Approach”. In: *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. Middleware ’17. Las Vegas, Nevada: Association for Computing Machinery, 2017, pp. 95–108. ISBN: 9781450347204.

[150] L. F. Bittencourt, E. R. M. Madeira, and N. L. S. Da Fonseca. “Scheduling in hybrid clouds”. In: *IEEE Communications Magazine* 50.9 (2012), pp. 42–47.

[151] A. Maros et al. “Machine Learning for Performance Prediction of Spark Cloud Applications”. In: *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*. 2019, pp. 99–106.

[152] Y. Zhao, F. Hu, and H. Chen. “An adaptive tuning strategy on spark based on in-memory computation characteristics”. In: *2016 18th International Conference on Advanced Communication Technology*. 2016, pp. 484–488.

[153] Di Chen et al. “An adaptive memory tuning strategy with high performance for Spark”. In: *International Journal of Big Data Intelligence* 4.4 (2017), pp. 276–286.

[154] P. Xuan et al. “Dynamic Management of In-Memory Storage for Efficiently Integrating Compute-and Data-Intensive Computing on HPC Systems”. In: *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. 2017, pp. 549–558.

[155] Zhuo Tang et al. “Dynamic memory-aware scheduling in spark computing environment”. In: *Journal of Parallel and Distributed Computing* 141 (2020), pp. 10–22. ISSN: 0743-7315.

[156] J. Bae et al. “Jointly optimizing task granularity and concurrency for in-memory mapreduce frameworks”. In: *2017 IEEE International Conference on Big Data (Big Data)*. 2017, pp. 130–140.

[157] David H Wolpert. “Stacked generalization”. In: *Neural networks* 5.2 (1992), pp. 241–259.

[158] Ji Liu et al. “A Survey of Data-Intensive Scientific Workflow Management”. In: *Journal of Grid Computing* 13.4 (Dec. 2015), pp. 457–493.

[159] P Herbert Raj, P Ravi Kumar, and P Jelciana. “Load Balancing in Mobile Cloud Computing using Bin Packing’s First Fit Decreasing Method”. In: *International Conference on Computational Intelligence in Information System*. Springer. 2018, pp. 97–106.

[160] Thar Baker et al. “Cloud-SEnergy: A bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications”. In: *Sustainable Computing: informatics and systems* 19 (2018), pp. 242–252.

[161] Georgios L Stavrinides and Helen D Karatza. “Scheduling real-time DAGs in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes”. In: *Future Generation Computer Systems* 28.7 (2012), pp. 977–988.

[162] Edward G Coffman Jr, Michael R Garey, and David S Johnson. “An application of bin-packing to multiprocessor scheduling”. In: *SIAM Journal on Computing* 7.1 (1978), pp. 1–17.

[163] Yoga Jaideep Darapuneni. “A Survey of Classical and Recent Results in Bin Packing Problem”. In: *UNLV Theses, Dissertations, Professional Papers, and Capstones* (2012).

[164] Fabrizio Marozzo et al. “A data-aware scheduling strategy for workflow execution in clouds”. In: *Concurrency and Computation: Practice and Experience* 29.24 (2017), e4229.

[165] A. O. Aseeri, Y. Zhuang, and M. S. Alkatheiri. “A Machine Learning-Based Security Vulnerability Study on XOR PUFs for Resource-Constraint Internet of Things”. In: *2018 IEEE International Congress on Internet of Things (ICIOT)*. 2018, pp. 49–56.

[166] Benjamin Carver et al. “Wukong: A scalable and locality-enhanced framework for serverless parallel computing”. In: *Proceedings of the 11th ACM Symposium on Cloud Computing*. 2020, pp. 1–15.

[167] Fabrizio Marozzo et al. “A Data-aware Scheduling Strategy for Workflow Execution in Clouds”. In: *Concurrency and Computation: Practice and Experience* 29.24 (2017).

[168] Salvatore Giampà et al. “A data-aware scheduling strategy for executing large-scale distributed workflows”. In: *IEEE Access* 9 (2021), pp. 47354–47364.

[169] Barcelona Supercomputing Center (BSC). *MareNostrum IV Technical Information*. 2018. URL: `https://www.bsc.es/marenostrum/marenostrum/technical-information`.

[170] Mohammad Sultan Mahmud et al. “A survey of data partitioning and sampling methods to support big data analysis”. In: *Big Data Mining and Analytics* 3.2 (2020), pp. 85–101.

[171] *Apache Hadoop*. URL: `https://hadoop.apache.org/`.

[172] *Apache Spark*. URL: `https://spark.apache.org/`.

[173] Sergio Ramirez-Gallego et al. “Fast-mRMR: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data”. In: *International Journal of Intelligent Systems* 32.2 (2017), pp. 134–152.

[174] Raul-Jose Palma-Mendoza et al. “Distributed correlation-based feature selection in spark”. In: *Information Sciences* 496 (2019), pp. 287–299.

[175] Mohammed Al-Kateb et al. “Hybrid Row-Column Partitioning in Teradata”. In: *Proc. VLDB Endow.* 9.13 (Sept. 2016), pp. 1353–1364. ISSN: 2150-8097.

[176] Geomar A. Schreiner et al. “A Hybrid Partitioning Strategy for NewSQL Databases: The VoltDB Case”. In: iiWAS2019. Munich, Germany: Association for Computing Machinery, 2019, pp. 353–360.

[177] Salman Salloum, Joshua Zhexue Huang, and Yulin He. “Random sample partition: a distributed data model for big data analysis”. In: *IEEE Transactions on Industrial Informatics* 15.11 (2019), pp. 5846–5854.

[178] Chenghao Wei et al. “A two-stage data processing algorithm to generate random sample partitions for big data analysis”. In: *International Conference on Cloud Computing*. Springer. 2018, pp. 347–364.

[179] Sara Migliorini et al. “CoPart: a context-based partitioning technique for big data”. In: *Journal of Big Data* 8.1 (2021), pp. 1–28.

[180] Massimiliano Bertolucci et al. “Static and dynamic big data partitioning on apache spark”. In: *Parallel Computing: On the Road to Exascale*. IOS Press, 2016, pp. 489–498.

[181] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. “Scalable script-based data analysis workflows on clouds”. In: *Proceedings of the 8th workshop on workflows in support of large-scale science*. 2013, pp. 124–133.

[182] Enric Tejedor et al. “PyCOMPSs”. In: *International Journal of High Performance Computing Applications* 31.1 (Jan. 2017), pp. 66–82.

[183] Javier Álvarez Cid-Fuentes et al. “dislib: Large Scale High Performance Machine Learning in Python”. In: *Proceedings of the 15th International Conference on eScience*. 2019, pp. 96–105.

[184] Francesc Lordan and et al. “ServiceSs: An Interoperable Programming Framework for the Cloud”. In: *Journal of Grid Computing* 12.1 (2014), pp. 67–91.

[185] Pierre Baldi et al. “Parameterized machine learning for high-energy physics”. In: *arXiv preprint arXiv:1601.07913* (2016).

[186] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[187] Frank Schmuck and Roger Haskin. “{GPFS}: A {Shared-Disk} File System for Large Computing Clusters”. In: *Conference on File and Storage Technologies (FAST 02)*. 2002.

[188] Kam Hamidieh. “A data-driven statistical model for predicting the critical temperature of a superconductor”. In: *Computational Materials Science* 154 (2018), pp. 346–354.

[189] Gustavo Scalabrini Sampaio et al. “Prediction of Motor Failure Time Using An Artificial Neural Network”. In: *Sensors* 19.19 (Oct. 2019), p. 4342.

[190] Mark James Abraham et al. “GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers”. In: *SoftwareX* 1-2 (2015), pp. 19–25. ISSN: 2352-7110.