ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (30^{th}cycle)

Classification algorithms for Big Data

With applications in the urban security domain

By

Luca Venturini

******

Supervisor:
Prof. Elena Maria Baralis, Supervisor

Doctoral Examination Committee:
Prof. Amr El Abbadi, Referee, University of California Santa Barbara
Prof. Paolo Ciaccia, Referee, Università degli Studi di Bologna
Prof. Paolo Garza, Politecnico di Torino
Prof. Marco Mellia, Politecnico di Torino
Prof. Ruggero G. Pensa, Università degli Studi di Torino

Politecnico di Torino
2018

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Luca Venturini
2018

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Dedico questa tesi ai miei genitori,

che mi hanno tirato su per l'uomo che sono

e l'hanno poi sopportato.

Acknowledgements

I would like to acknowledge the guidance of my supervisor, Prof. Elena Baralis, and her support during the years of the PhD.

A big part of this thesis would not be here if not for the help and the experience of Dr. Paolo Garza, to whom I'm deeply thankful. I also owe much to Dr. Fabio Pulvirenti, that in this journey has been a guide and a dear friend. I hope I am not too much in debt with you still.

I would like to express my gratitude to Prof. Vania Ceccato, that has invited me to spend three months at KTH. Her passion for teaching is admirable and her PhD course is undoubtly the most intense I have followed in my career.

I thank all the wonderful persons I met during my visiting period in Sweden, in Stockholm and in Norrtälje. The blank in this page is too little for all the love you gratuitously gave me.

I thank all the colleagues and friends at Lab5, whenever they lived there or only were welcome guests. You have made these years somewhat enjoyable. I owe you a thousand smiles and some really good laughs.

In our lives we meet several persons that deserve our gratitude and estimation, and I have the luck of having met many of them. Usually, we call them friends. I thank all the persons that fall under this definition, hoping that I do enough to make them recognize in it.

I thank my family. A life without love would not be worth living.

Abstract

A classification algorithm is a versatile tool, that can serve as a predictor for the future or as an analytical tool to understand the past. Several obstacles prevent classification from scaling to a large Volume, Velocity, Variety or Value. The aim of this thesis is to scale distributed classification algorithms beyond current limits, assess the state-of-practice of Big Data machine learning frameworks and validate the effectiveness of a data science process in improving urban safety.

We found in massive datasets with a number of large-domain categorical features a difficult challenge for existing classification algorithms. We propose associative classification as a possible answer, and develop several novel techniques to distribute the training of an associative classifier among parallel workers and improve the final quality of the model. The experiments, run on a real large-scale dataset with more than 4 billion records, confirmed the quality of the approach.

To assess the state-of-practice of Big Data machine learning frameworks and streamline the process of integration and fine-tuning of the building blocks, we developed a generic, self-tuning tool to extract knowledge from network traffic measurements. The result is a system that offers human-readable models of the data with minimal user intervention, validated by experiments on large collections of real-world passive network measurements.

A good portion of this dissertation is dedicated to the study of a data science process to improve urban safety. First, we shed some light on the feasibility of a system to monitor social messages from a city for emergency relief. We then propose a methodology to mine temporal patterns in social issues, like crimes. Finally, we propose a system to integrate the findings of Data Science on the citizenry's perception of safety and communicate its results to decision makers in a timely manner. We applied and tested the system in a real Smart City scenario, set in Turin, Italy.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

In the recent years, Big Data have received much attention by both the academic and the industrial world, with the aim of fully leveraging the power of the information they hide. The term "Big" spans over several dimensions, that are usually remembered as "Vs". The three traditional Vs of Big Data are Volume, Variety and Velocity, although several Vs have been added to this triangle.

Linked with the Vs of Big Data are several problems, that prevent traditional techniques from scaling in that dimension. Most often, it is a *technological problem*. For example, many traditional solutions fail to scale to large Volumes because of the hardware constraints of the single machine, or to high throughput (Velocity) because of limits of the bandwidth or of the processor unit. Sometimes, solutions exist for each of the subproblems we face, but the complexity relies in the cooperation of the single solutions. We have thus a *problem of integration*, and it is a common obstacle when growing in Variety. Finally, we have an *analytical problem* when choosing the direction that our investigation must follow, and eventually a *communication problem* in presenting our results. These two are the crucial issues in extracting Value, the fourth of the three Vs, as only actionable results given to informed decision makers can really make data valuable. Analysis and communication are usually the two primary tasks of Data Science, which can be thus seen as the final step in the pipeline of the Big Data.

The focus of this dissertation is on classification algorithms. Classification is a supervised learning task, that aims at labeling new records of data. A classification model is a versatile tool, that can serve as a predictor for the future or as an analytical

tool to understand the past. The training of the model can meet serious obstacles when scaling on large Volumes. The update, or the application of the model on new records can limit its scalability on high Velocity. The lack of a known, ground truth limits its applicability on a Variety of datasets. The use of a complex model limits or impedes its readability and thus the extraction of Valuable insights. In this dissertation, we will humbly try to propose solutions to all these issues.

The Value of a data science process is higher as greater the impact that can have on the world. And the entire human society can greatly benefit from data-driven decisions on themes of public or global interest, like social issues, relief to natural disasters, water scarcity, etc. A good portion of this dissertation is dedicated to the study of a data science process to improve urban safety, as an application that would have a potential impact on the lives of nearly 4 billion people.

Thesis statement: *The aim of this thesis is to scale distributed classification algorithms beyond current limits, assess the state-of-practice of Big Data machine learning frameworks and validate the effectiveness of a data science process in improving urban safety.*

In the next section, we will review the dissertation plan.

1.1 Dissertation plan and research contribution

This dissertation is divided into three parts, which reflect the three folds of the thesis statement:

1. *Scale distributed classification algorithms beyond current limits,* which focuses on a special kind of classification algorithms, associative classifiers, trying to cope with some limitations faced by many classification techniques on Big Data.

2. *Assess the state-of-practice of Big Data machine learning frameworks.* Nowadays, machine learning libraries offer many off-the-shelf solutions for Big Data. In this part, we evaluate the validity of these solutions in integrating and adapting to a complex Big Data scenario, like that of computer networks measurements.

3. *Validate the effectiveness of a data science process in improving urban safety,* where we see if and how we can leverage on multiple sources of information to provide useful insights for policing and policy making.

In the following, we briefly introduce each chapter.

1.1.1 Scaling associative classification to very large datasets

Supervised learning algorithms are nowadays successfully scaling up to datasets that are very large in volume, leveraging the potential of in-memory cluster-computing Big Data frameworks, like Apache Spark. Still, massive datasets with a number of large-domain categorical features are a difficult challenge for any classifier. Most off-the-shelf solutions cannot cope with this problem. Chapter 2 proposes Associative Classification as a possible answer.

After a brief introduction to the background theory, we examine how the task of Frequent Itemset Mining is distributed in MapReduce frameworks. This gives us a sound ground to build a scalable solution for the training of an associative classifier, which has many points in common with the frequent itemsets mining task. We then propose two solutions: BAC, a Bagged Associative Classifier, and DAC, a Distributed Associative Classifier. Both exploit ensemble learning to distribute the training of an associative classifier among parallel workers and improve the final quality of the model. BAC tests this approach on a number of medium-sized datasets, proving that we can reach the quality that associative classifiers have on single machines. DAC improves on these results adopting several novel techniques to reach high scalability, without sacrificing quality.

The proposed techniques contribute to the state of the art in scaling supervised learning algorithms to large Volume and Velocity. To prove this, we run the final experiments of DAC on a real large-scale dataset, with more than 4 billion records and 800 million distinct categories. The training time of the model was in some cases 25 times smaller than the one of Random Forests, keeping at the same time the prediction times of DAC lower than the competitor. The model generated by DAC, made of association rules, was still readable at the end of the pipeline, while the Random Forest lost its readability for reasons we will explain in detail in the chapter. This adds Value to the classification model as an instrument to understand the data and make decisions.

1.1.2 Building a Big Data machine learning pipeline

Understanding the behavior of a network from a large scale traffic dataset is a challenging problem. Big data frameworks offer scalable algorithms to extract information from raw data, but often require a sophisticated fine-tuning and a detailed knowledge of machine learning algorithms. To streamline this process, in Chapter 3 we propose SeLINA (Self-Learning Insightful Network Analyzer), a generic, self-tuning tool to extract knowledge from network traffic measurements. SeLINA includes different data analytics techniques providing self-learning capabilities to state-of-the-art scalable approaches, jointly with parameter auto-selection to off-load the network expert from parameter tuning. We combine both unsupervised and supervised approaches to mine data with a scalable approach. SeLINA embeds mechanisms to check if the new data fits the model, to detect possible changes in the traffic, and to, possibly automatically, trigger model rebuilding.

The result is a system that offers human-readable models of the data with minimal user intervention, supporting domain experts in extracting actionable knowledge and thus Value. Self-learning features help dealing with a Variety of datasets, where the ground truth of the observations is potentially unknown. SeLINA's current implementation runs on Apache Spark. We tested it on large collections of real-world passive network measurements from a nationwide ISP, investigating YouTube and P2P traffic. The experimental results confirmed the ability of SeLINA to provide insights and detect changes in the data that suggest further analyses.

1.1.3 Data Science for urban security

The global population is nowadays increasingly urban, with more than half living in a city [1]. Moreover, in the last twelve years our planet has added rooms for more than a billion new guests, totaling more than 7 and a half billion people on the globe [2]. Growing in size, cities have become more complex to manage and face exploding issues, like pollution, traffic, social unrest and injustice, and crime. It is only a matter of necessity if institutions have recurred to Information Technology to manage this complex system, aiming at fully implementing the paradigm of a Smart City.

Chapter 4 focuses on how Data Science can help a Smart City improving urban security. We start the chapter investigating on the quality of the information hidden in social media, in the special context of a mass emergency. From this study, we gain precious knowledge that sheds some light on the feasibility and validity of a system that monitors social messages from a city for emergency relief, highlighting the potential limitations. We then propose a methodology to mine temporal patterns in social issues, and particularly crimes. Finally, we propose a system to integrate the findings of Data Science on the citizenry's perception of safety and communicate its results to decision makers in a timely manner, facilitating the production of Value, like punctual interventions and corrective policies. We apply and test the system in a real Smart City scenario, set in Turin, Italy.

1.1.4 Dissertation plan

This dissertation is organized as follows. Chapter 2 focuses on scaling associative classification to very large datasets. Chapter 3 proposes a Big Data machine learning pipeline for network traffic analysis. Chapter 4 reports our approach towards integrating Data Science in the urban security process. Finally, Chapter 5 draws the final conclusions.

Chapter 2

Scaling associative classification to very large datasets

Associative classification is a powerful supervised learning technique, which is well-known to produce accurate models. Among its several advantages, we count the extreme readability of the generated models, that are made of association rules, and the native support for categorical features. Unfortunately, state-of-art associative classifiers do not scale effectively on a distributed framework like Apache Spark, as they need multiple readings of the data and make an extensive usage of off-load computation.

In this chapter, we will see different solutions to scale the training of an associative classifier to very large datasets. First, we will review existing scalable solutions for a very similar problem, frequent itemset mining. From the findings of this section, we will build two different proposals for scaling associative classification, namely a Bagged Associative Classifier (BAC) and a Distributed Associative Classifier (DAC).

Part of the contents of this chapter were originally published in [3–5]. The chapter is organized as follows. In Section 2.1 we review the theoretical background. Section 2.2 surveys existing solutions for the frequent itemset mining task and scalable approaches. Section 2.3 introduces BAC, a Bagged Associative Classifier. Section 2.4 introduces an advanced approach, DAC, a Distributed Associative Classifier. In Section 2.5, we review related works. Finally, Section 2.6 concludes the chapter and sums up.

2.1 Theoretical background

In this section, we review some of the theory of the frequent itemset mining problem and associative classification, which will serve as a foundation for the rest of the chapter.

2.1.1 Frequent itemset mining

tid	Transaction	Class
1	$\{A,B,D,E\}$	+
2	$\{B,C,E\}$	-
3	$\{A,B,D,E\}$	+
4	$\{A,B,C,E\}$	-
5	$\{A,B,C,D,E\}$	+
6	$\{B,C,D\}$	-

Table 2.1 An example transactional dataset, binary-labeled.

A frequent itemset represents frequently co-occurring items in a transactional dataset. More formally, let $\mathscr{I}$ be a set of items. A transactional dataset $\mathscr{D}$ consists of a set of transactions $\{t_1,\ldots,t_n\}$. Each transaction $t_i \in \mathscr{D}$ is a collection of items (i.e., $t_i \subseteq \mathscr{I}$) and is identified by a transaction identifier (tid_i). Figure 2.1 reports an example of a transactional dataset with 6 transactions.

An itemset I is defined as a set of items (i.e., $I \subseteq \mathscr{I}$) and is characterized by a support value, which is denoted by $sup(I)$ and defined as the ratio between the number of transactions in $\mathscr{D}$ containing I and the total number of transactions in $\mathscr{D}$. In the example dataset in Table 2.1, for example, the support of the itemset *{A,B,D}* is 50% (3/6). This value represents the frequency of occurrence of the itemset in the dataset. An itemset I is considered frequent if its support is greater than a user-provided minimum support threshold *minsup*.

Given a transactional dataset $\mathscr{D}$ and a minimum support threshold *minsup*, the Frequent Itemset Mining [6] problem consists in extracting the complete set of frequent itemsets from $\mathscr{D}$.

The dimension of the search space can be represented as a lattice, whose top is an empty set. Its size increases exponentially with the number of items [7, 8].

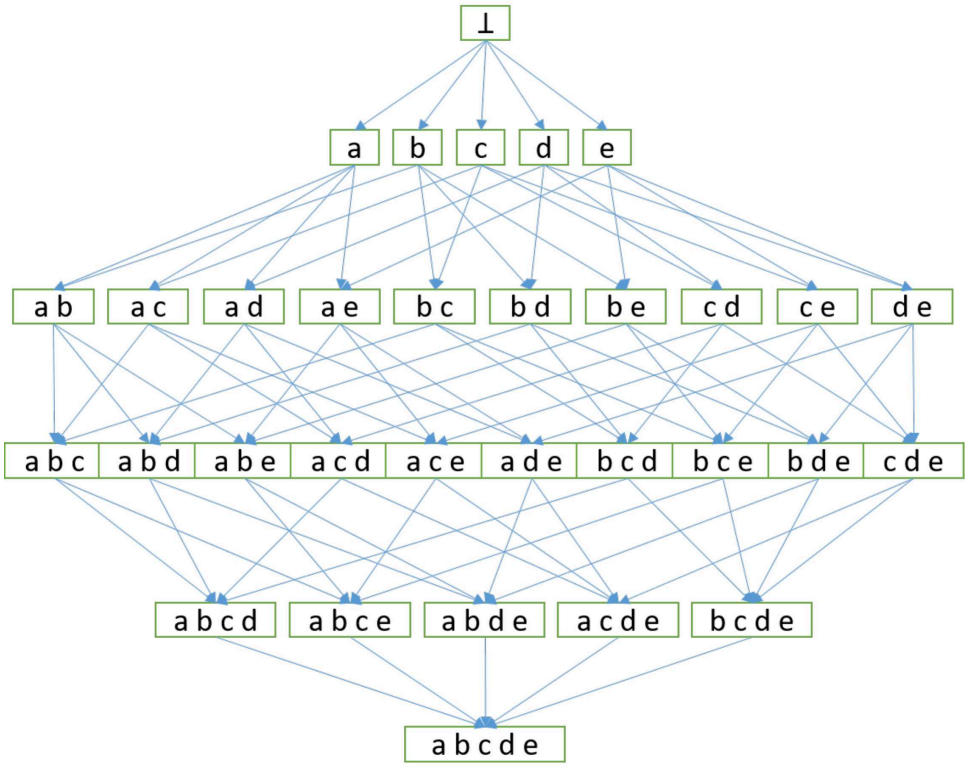


Fig. 2.1 Lattice representing the search space based on the items appearing in the example dataset $\mathscr{D}$

Due to the exponential growth of the lattice, data mining techniques, like associative classification itself, make often use of an approximate representation or a subset of the complete lattice, which is also difficult to store. In Figure 2.1, the lattice related to our example is shown.

2.1.2 Associative classification

In the classification problem, the dataset is represented as a structured table of records and features. Each feature is identified by a *feature_id*, that is set to some value v for each record, or to a null value for not available information. More formally, the input dataset $\mathscr{D}$ is represented as a relation $\mathscr{R}$, whose schema is given by d distinct features $A_1 \ldots A_d$ and a class attribute $C \in \mathscr{C}$, where $\mathscr{C}$ is the set of distinct classes. Each transaction in $\mathscr{R}$ can be described as a collection of pairs *(feature, value)*, plus

a class label (a value belonging to the domain of class attribute C). Each pair *(feature, value)* can be seen as an *item* and we can use the notation in Section 2.1.1.

Not available data are represented with a null value, or not represented at all, as transactional datasets do not have a fixed structure. If a transaction has a length of exactly d items, or equivalently has no missing values, it's a *point* in a space of d dimensions. The common practice in classification is to define a training set, i.e. a part of the labeled dataset that is used to train the algorithm, and a test set, from which the labels are removed. The two sets are used together with other techniques, like cross-validation, to simulate the behavior of the algorithm towards unlabeled, new data and validate its performance.

Association rules [9] are made of an *antecedent* itemset A, and a *consequent* itemset B, and are read as A yields B, or $A \Rightarrow B$. When the consequent is made of a single item, and specifically an item belonging to the set of class labels, the association rule can be used to label the record. We inherit the naming in [10] and call these rules Class Association Rules, or CARs.

Both association rules and CARs share a number of metrics that measure their strength and statistical significance [11, 12]. The support count (*supCount*), or absolute support, of an itemset is the number of transactions in the dataset $\mathscr{D}$ that contain the whole itemset. The support (*sup*), or relative support, of a rule $A \Rightarrow B$ is defined as $supCount(A \cup B)/|\mathscr{D}|$, where $|\mathscr{D}|$ is the cardinality of $\mathscr{D}$. The confidence (*conf*) of the rule is defined as $supCount(A \cup B)/supCount(A)$, and in CARs it measures how precise the rule is at labeling a record. The lift of a rule (*lift*) is a measure of the (symmetric) correlation between the antecedent and consequent of the extracted rules and it is defined as $conf(A \Rightarrow B)/sup(B)$. Lift values significantly above 1 indicate a positive correlation between rule antecedent and consequent, meaning that the implication between A and B holds more than expected in the source dataset. The χ^2 of a CAR is the value of the χ^2 statistics computed against the distribution of the classes in $\mathscr{D}$, which states whether the assumption of correlation between the antecedent and the consequent is statistically significant. Other measures used to sort rules and CARs are their length (*len*), i.e. the total number of items of A and B, and the lexicographical order (*lex*).

Another measure that is widely used in classification algorithms is the Gini impurity [13]. The Gini impurity measures how often a record would be wrongly labeled, if labeled randomly with the distribution of the classes in the dataset. It is

used for example in decision trees, to evaluate the quality of the splits at each node. The Gini impurity of a dataset, or portion of it, is computed as

$$Gini = \sum_{i \in \mathscr{C}} f_i(1 - f_i)$$

where f_i is the frequency of class i in the dataset, or portion of it, for which we are computing the impurity. A portion of dataset is considered pure if its Gini is equal to 0, that happens when only a single label appears. We will refer to the Gini Impurity of an itemset, as the impurity of the portion of the dataset that contains the itemset.

In association rule mining and associative classification, the user is usually able to set some minimum threshold for the above-mentioned quality measures, like a minimum support *minsup*, a minimum confidence *minconf*, a minimum positive lift $min^{+}lift$, etc. The model generation phase of associative classifiers is usually based on two steps: (i) Extraction of all the CARs with a support higher than a minimum support threshold *minsup* and a minimum confidence threshold *minconf* and (ii) Rule selection by means of the database coverage technique, firstly introduced in [10]. The database coverage technique works as follows. First, given an ordered list of CARs extracted from a training set, it considers one CAR r at a time in the sort order and selects the transactions in the training set matched by r. For each matched transaction t, it checks also if r classifies properly t. If r classifies properly at least one training record, then r is kept. Differently, if all the training transactions matched by r have a class label different from the one of r, then r is discarded. If r does not match any training data then r is discarded as well. Once r has been analysed, all the transactions matched by r are removed from the training set and the next CAR is analysed by considering only the remaining transactions.

2.2 Frequent Itemset Mining and distributed frameworks

2.2.1 Centralized algorithms

The search space exploration strategies of the distributed approaches are often inspired by the solutions adopted by the centralized approaches. Hence, this section

shortly introduces the main strategies of the centralized itemset mining algorithms. This introduction is useful to better understand the algorithmic choices behind the distributed algorithms.

The frequent itemset mining task is challenging in terms of execution time and memory consumption because the size of the search space is exponential with the number of items of the input dataset [7]. Two main search space exploration strategies have been proposed: (i) level-wise or breadth-first exploration of the candidate itemsets in the lattice and (ii) depth-first exploration of the lattice.

The most popular representative of the breadth-first strategy is Apriori [14]. Starting from single items, it iteratively generates and counts the support of the candidate itemsets of size $k+1$ from the frequent itemsets of size k. At each iteration k, the supports of the candidate itemsets of length k are counted by performing a new scan of the input dataset. The search space is pruned by exploiting the downward-closure property, which guarantees that all the supersets of an infrequent itemset are infrequent too. Specifically, the downward-closure property allows pruning the set of candidate itemsets of length $k+1$ by considering the frequent itemsets of length k. The Apriori algorithm is significantly affected by the density of the dataset. The higher the density of the dataset, the higher the number of frequent itemsets and hence the amount of candidate itemset stored in main memory. The problem becomes unfeasible when the number of candidate itemsets exceeds the size of the main memory.

More efficient and scalable solutions exploit the depth-first visit of the search space. FP-Growth [15], which uses a prefix-tree-based main memory compressed representation of the input dataset, is the most popular depth-first based approach. The algorithm is based on a recursive visit of the tree-based representation of the dataset with a "divide and conquer" approach. In the first phase the support of each single item is counted and only the frequent items are stored in the "frequent items table" (F-list). This information allows pruning the search space by avoiding the analysis of the itemsets extending infrequent items. Then, the FP-tree, that is a compact representation of the dataset, is built exploiting the F-list and the input dataset (together they compose the "header table") . Specifically, each transaction is included in the FP-tree by adding or extending a path on the tree, exploiting common prefixes. Once the FP-tree associated with the input dataset is built, FP-growth recursively splits the itemset mining problem by generating conditional

FP-trees and visiting them. Given an arbitrary prefix p, where p is a set of items, the conditional FP-tree with respect to p, also called projected dataset with respect to p, is substantially the compact representation of the transactions containing p. Each conditional FP-tree contains all the knowledge needed to extract all the frequent itemsets extending its prefix p. FP-growth decomposes the initial problem by generating one conditional FP-tree for each item and invoking the itemset mining procedure on each of them, in a recursive depth-first fashion.

FP-growth suits well dense datasets, because they can be effectively and compactly represented by means of the FP-tree data structure. Differently, with sparse datasets, the compressions benefits of the FP-tree are reduced because there will be a higher number of branches [6] (i.e., a large number of subproblems to generate and results to merge).

Another very popular depth-first approach is the Eclat algorithm [16]. It performs the mining from a vertical transposition of the dataset. In the vertical format, each row includes an item and the transaction identifiers (tid) in which it appears ($tidlist$). After the initial dataset transposition, the search space is explored in a depth-first manner similar to FP-growth. The algorithm is based on equivalence classes (groups of candidate itemsets sharing a common prefix), which allows smartly merging tidlists to select frequent itemsets. Prefix-based equivalence classes are mined independently, in a "divide and conquer" strategy, still taking advantage of the downward closure property. Eclat is relatively robust to dense datasets. It is less effective with sparse distributions, because the depth-first search strategy may require generating and testing more (infrequent) candidate itemsets with respect to Apriori-like algorithms [17].

2.2.2 Itemset mining parallelization strategies

Two main algorithmic approaches are proposed to address the parallel execution of the itemset mining algorithms by means of the MapReduce paradigm [18]. They are significantly different because (i) they use different solutions to split the original problem in subproblems and (ii) make different assumptions about the data that can be stored in the main memory of each independent task.

Data split approach. It splits the problem in "similar" subproblems, executing the same function on different data chunks. Specifically, each subproblem

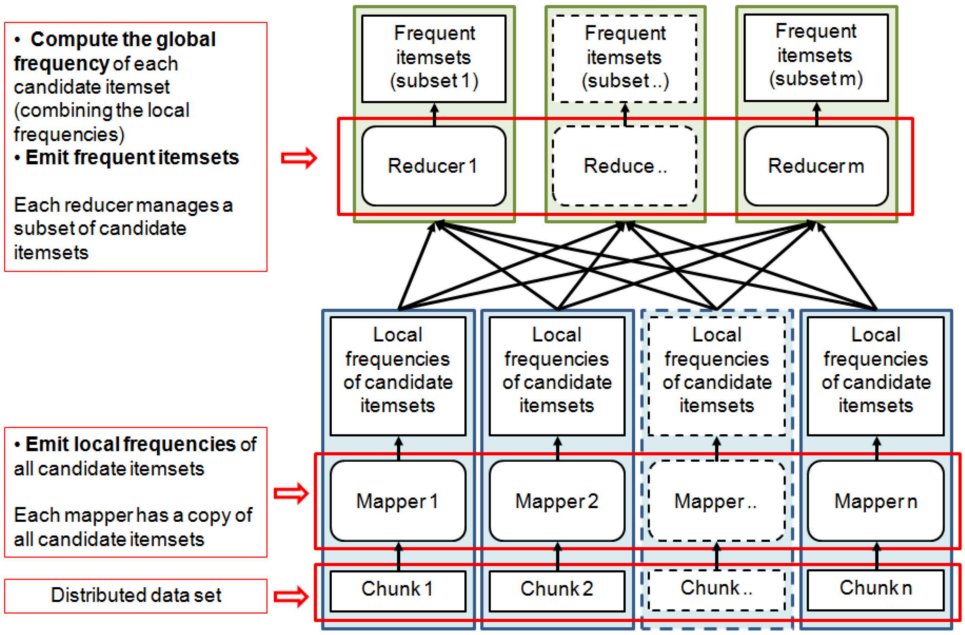


Fig. 2.2 Itemset mining parallelization: Data split approach

computes the local supports of all candidate itemsets on one chunk on the input dataset (i.e., each subproblem works on the complete search space but on a subset of the input data). Finally, the local results (i.e., the local supports of the candidate itemsets) emitted by each subproblem/task are merged to compute the global final result (global support of each itemset). The main assumptions of this approach are that (i) the problem can be split in "similar' subproblems working on different chunks of the input data and (ii) the set of candidate itemsets is small enough that it can be stored in the main memory of each task.

Search space split approach. It splits the problem by assigning to each subproblem the visit of a subset of the search space (i.e., each subproblem visits a part of the lattice). Specifically, this approach generates, from the input distributed dataset, a set of projected datasets, each one small enough to be stored in the main memory of a single task. Each projected dataset contains all the information that is needed to extract a subset of itemsets (i.e., each dataset contains all the information that is needed to explore a part of the lattice) without needing the contribution of the results of the other tasks. The final result is the union of the itemset subsets mined from each projected dataset.

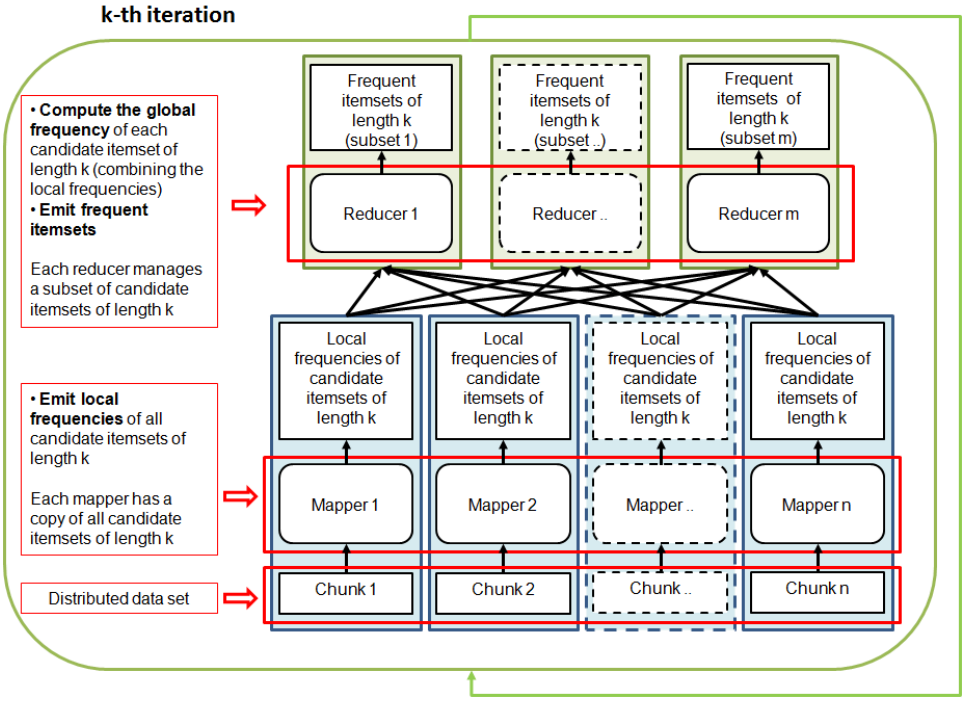


Fig. 2.3 Itemset mining parallelization: Iterative Data split approach

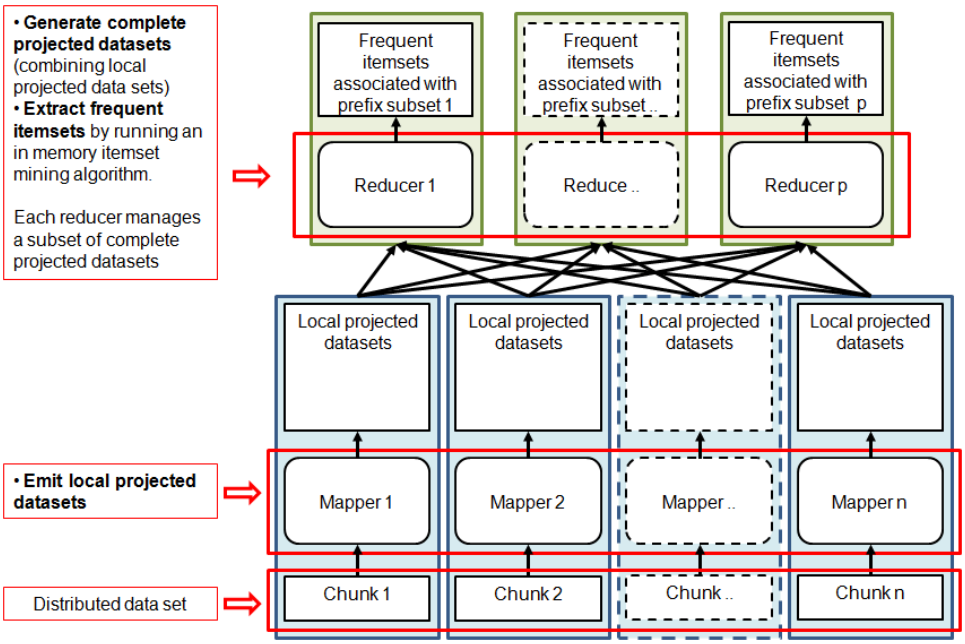


Fig. 2.4 Itemset mining parallelization: Search space split approach

Figures 2.2 and 2.4 depict the first and the second parallelization strategies, respectively. In the data split approach (Figure 2.2), the map phase computes the

local supports of the candidate itemsets in its data chunk (i.e., each mapper runs a "local itemset mining extraction" on its data chunk). Then, the reduce phase merges the local supports of each candidate itemset to compute its global support. This solution requires each mapper to store a copy of the complete set of candidate itemsets (i.e., a copy of the lattice). This set must fit in the main memory of each mapper. Since the complete set of candidate itemsets is usually too large to be stored in the main memory of a single mapper, an iterative solution, inspired by the level-wise centralized itemset mining algorithms, is used. Figure 2.3 reports the iterative solution. At each iteration k only the subset of candidates of length k are considered and hence stored in the main memory of each mapper. This approach, thanks also to the exploitation of the apriori-principle to reduce the size of the candidate sets, allows obtaining subsets of candidate itemsets that can be loaded in the main memory of every mapper.

In the search space split approach (Figure 2.4), the map phase generates a set of local projected datasets. Specifically each mapper generates a set of local projected datasets based on its data chunk. Each local projected dataset is the projection of the input chunk with respect to a prefix p.[1] Then, the reduce phase merges the local projected datasets to generate the complete projected datasets. Each complete projected dataset is provided as input to a standard centralized itemset mining algorithm running in the main memory of the reducer and the set of frequent itemsets associated to it are mined. Each reducer is in charge of analyzing a subset of complete projected datasets by running the itemset mining phase on one complete projected dataset at a time. Hence, the main assumption, in this approach, is that each complete projected dataset must fit in the main memory of a single reducer.

Table 2.2 summarizes the main characteristics of the two parallelization approaches with respect to the following criteria: type of split of the problem, usage of main memory, communication costs, load balancing, and maximum parallelization (i.e. maximum number of mappers and reducers).

Type of split/Split of the search space. The main difference between the two parallelization approaches is the strategy adopted to split the problem in subproblems. This choice has a significant impact on the other criteria.

[1] Note that the projected datasets can overlap because the transactions associated with two distinct prefixes p_1 and p_2 can be overlapped.

Criterion	**Iterative data split approach (Figure 2.3)**	**Search space split approach (Figure 2.4)**
Type of split/Split of the search space	Each subproblem analyzes a different subset of the input data and computes the local supports of all the candidate itemsets of length k on its chunks of data. The final result is given by the merge of the local results.	Each subproblem analyzes a different subset of itemsets/a different part of the search space. The final result is the union of the local results.
Usage of main memory	The candidate set of length k is stored in the main memory of a single task.	The complete projected dataset is stored in the main memory of a single task.
Communication cost	Number of candidate itemsets $\times$ number of mappers $\times$ number of iterations.	Sum of the sizes of the local projected datasets.
Load balancing	Load balancing is achieved by associating the same number of itemsets to each reducer.	The tasks could be significantly unbalanced depending on the characteristics of the projected datasets assigned to each node.
Maximum number of mappers	Number of chunks	Number of chunks
Maximum number of reducers	Number of candidate itemsets	Number of items

Table 2.2 Comparison of the parallelization approaches.

Usage of main memory. The different usage of the main memory of the tasks impact on the reliability of the two approaches. The data split approach assumes that the candidate itemsets of length k can be stored in the main memory of each mapper. Hence, it is not able to scale on dense datasets characterized by large candidate sets. Differently, the search space split approach assumes that each complete projected dataset can be stored in the main memory of a single task. Hence, this approach runs out of memory when large complete projected datasets are generated.

Communication costs. In a parallel MapReduce algorithm, communication costs are important, because the network can easily become the bottleneck if large amounts of data are sent on it. The communication costs are mainly related to the outputs of the mappers which are sent to the reducers on the network. For the data split approach the data that is sent on the network is linear with respect to the number of candidate itemsets, the number of mappers, and the number of iterations. Differently, for the search space approach, the amount of data emitted by the mappers is equal to the size of the projected datasets.

Load balancing. The different split of the problem in subproblems significantly impacts on load balancing. For the data split approach, the execution time of each mapper is linear with respect to the number of input transactions and the execution time of each reducer is linear with respect to the number of assigned itemsets. Hence, the data split approach can easily achieve a good load balancing by assigning the same number of data chunks to each mapper and the same number of candidate itemsets to each reducer. Differently, the search space split approach is potentially unbalanced. In fact, each subproblem is associated with a different subset of the lattice, related to a specific projected dataset and prefix, and, depending on the data distribution, the complexity of the subproblems can significantly vary. A smart assignment of a set of subproblems to each node would mitigate the unbalance. However, the complexity of the subproblems is hardly inferable during the initial assignment phase.

Maximum number of mappers and reducers. The two approaches are significantly different in terms of "maximum parallelization degree", at least in terms of number of maximum exploitable reducers. The maximum parallelization of the map phase is equal to the number of data chunks for both approaches. Differently, the maximum parallelization of the reduce phase is equal to the number of candidate itemsets for the data split approach, because potentially each reducer could compute

the global frequency of a single itemset, whereas it is equal to the number of global projected datasets for the second approach, which can be at most equal to the number of items. Since the number of candidate itemsets is greater than the number of items, the data split approach can potentially reach a higher degree of parallelization with respect to the search space split approach.

The two parallelization approaches are used to design efficient parallel implementations of well-known centralized itemset mining algorithms. Specifically, the data split approach is used to implement the parallel versions of level-wise algorithms (like Apriori [14]), whereas the search space split approach is used to implement parallel versions of depth-first recursive approaches (like FP-growth [15] and Eclat [16]).

2.2.3 Distributed itemset mining algorithms

This section describes the algorithms, and available implementations, representing the state-of-the-art solutions in the parallel frequent itemset mining context. We considered the following algorithms: YAFIM [19], PFP [20], BigFIM [21], and DistEclat [21]. The only algorithm which is lacking a publicly available implementation is YAFIM. Among the considered algorithms, YAFIM belongs to the ones based on the data split approach, while PFP and DistEclat are based on the search space split approach. Finally, BigFIM mixes the two strategies, aiming at exploiting the pros of them. For PFP we selected two popular implementations: Mahout PFP and MLlib PFP, which are based on Hadoop and Spark, respectively. The description of the four selected algorithms and their implementations are reported in the following subsections.

YAFIM

YAFIM [19] is an Apriori distributed implementation developed in Spark. The iterative nature of the algorithm has always represented a challenge for its application in MapReduce-based Big Data frameworks. The reasons are the overhead caused by the launch of new MapReduce jobs and the requirement to read the input dataset from disk at each iteration. YAFIM exploits Spark RDDs to cope with these issues. Precisely, it assumes that all the dataset can be loaded into an RDD to speed up the counting operations. Hence, after the first phase in which all the transactions are loaded in an RDD, the algorithm starts the iterative Apriori algorithm organizing the

candidates in a hash tree to speed up the search. Being strongly Apriori-based, it inherits the breadth-first strategy to explore and partition the search space and the preference towards sparse data distributions. YAFIM exploits the Spark "broadcast variables abstraction" feature, which allows programmers to send subsets of shared data to each slave only once, rather than with every job that uses those subset of data. This implementation mitigates communication costs (reducing the inter job communication), while load balancing is not addressed.

Parallel FP-growth (PFP)

Parallel FP-growth [20], called PFP, is a distributed implementation of FP-growth that exploits the MapReduce paradigm to extract the k most frequent closed itemsets. It is included in the Mahout machine learning Library (version 0.9) and it is developed on Apache Hadoop. PFP is based on the search space split parallelization strategy reported in Section 2.2.2. Specifically, the distributed algorithm is based on building independent FP-trees (i.e., projected datasets) that can be processed separately over different nodes.

The algorithm consists of 3 MapReduce [18] jobs.
First job. It builds the F-list, that is used to select frequent items, in a MapReduce "Word Count" manner.
Second job. In the second job, the mappers project with respect to group of items (prefixes) all the transactions of the input dataset to generate the local projected contributions to the projected datasets. Then, the reducers aggregate the projections associated with the items of the same group and build independent complete FP-trees from them. Each complete FP-tree is managed by one reducer, which runs a local main memory FP-growth algorithm on it and extracts the frequent itemsets associated with it.
Third job. Finally, the last MapReduce job selects the top k frequent closed itemsets.

The independent complete FP-trees can have different characteristics and this factor has a significant impact on the execution time of the mining tasks. As discussed in Section 2.2.2, this factor significantly impacts on load balancing. Specifically, when the independent complete FP-trees have different sizes and characteristics, the tasks are unbalanced because they addresses subproblems with different complexities. This problem could be potentially solved by splitting complex trees in sub-trees, each one associated with an independent subproblem of the initial one. However,

defining a metric to split a tree in such a way to obtain sub-mining problems that are equivalent in terms of execution time is not easy. In fact, the execution time of the itemset mining process on an FP-Tree is not only related to its size (number of nodes) but also to other characteristics (e.g., number of branches and frequency of each node). Depending on the dataset characteristics, the communication costs can be very high, especially when the projected datasets overlap significantly because in that case the overlapping part of the data is sent multiple times on the network.

Spark PFP [22] represents a pure transposition of PFP to Spark. It is included in MLlib, the Spark machine learning library. The algorithm implementation in Spark is very close to the Hadoop sibling. The main difference, in terms of addressed problem, is that MLlib PFP mines all the frequent itemsets, whereas Mahout PFP mines only the top k closed itemsets.

Both implementations, being strongly inspired by FP-growth, keep from the underlying centralized algorithm the features related to the search space exploration (depth-first) and the ability to efficiently mine itemsets from dense datasets.

DistEclat and BigFIM

DistEclat [21] is a Hadoop-based frequent itemset mining algorithms inspired by the Eclat algorithm, whereas BigFIM [21] is a mixed two-phase algorithm that combines an Apriori-based approach with an Eclat-based one.

DistEclat is a frequent itemset miner developed on Apache Hadoop. It exploits a parallel version of the Eclat algorithm to extract a superset of closed itemsets

The algorithm mainly consists of two steps. The first step extracts k-sized prefixes (i.e., frequent itemsets of length k) with respect to which, in the second step, the algorithm builds independent projected subtrees, each one associated with an independent subproblem. Even in this case, the main idea is to mine these independent trees in different nodes, exploiting the search split parallelization approach discussed in Section 2.2.2.

The algorithm is organized in 3 MapReduce jobs.
First job. In the initial job, a MapReduce job transposes the dataset into a vertical representation.
Second job. In this MapReduce job, each mapper extracts a subset of the k-sized prefixes (k-sized itemsets) by running Eclat on the frequent items, and the related

tidlists, assigned to it. The k-sized prefixes and the associated tidlists are then split in groups and assigned to the mappers of the last job.
Third job. Each mapper of the last mapReduce job runs the in main memory version of Eclat on its set of independent prefixes. The final set of frequent itemsets is obtained by merging the outputs of the last job.

The mining of the frequent itemsets in two different steps (i.e., mining of the itemsets of length k in the second job and mining of the other frequent itemsets in the last job) aims at improving the load balancing of the algorithm. Specifically, the split in two steps allows obtaining simpler sub-problems, which are potentially characterized by similar execution times. Hence, the application is overall well-balanced.

DistEclat is designed to be very fast but it assumes that all the tidlists of the frequent items should be stored in main memory. In the worst case, each mapper needs the complete dataset, in vertical format, to build all the 2-prefixes [21]. This impacts negatively on the scalability of DistEclat with respect to the dataset size. The algorithm inherits from the centralized version the depth-first strategy to explore the search space and the preference for dense datasets.

BigFIM is an Hadoop-based solution very similar to DistEclat. Analogously to DistEclat, BigFIM is organized in two steps: (i) extraction of the frequent itemsets of length less than or equal to the input parameter k and (ii) execution of Eclat on the sub-problems obtained splitting the search space with respect to the k-itemsets. The difference lies in the first step, where BigFIM exploits an Apriori-based algorithm to extract frequent k-itemsets, i.e., it adopts the data split parallelization approach (Section 2.2.2). Even if BigFIM is slower than DistEclat, BigFIM is designed to run on larger datasets. The reason is related to the first step in which, exploiting an Apriori-based approach, the k-prefixes are extracted in a breadth-first fashion. Consequently, the nodes do not have to keep large tidlists in main memory but only the set of candidate itemsets to be counted. However, this is also the most critical issue in the application of the data split parallelization approach, because, depending on the dataset density, the set of candidate itemsets may not be stored in main memory.

Because of the two different techniques used by BigFIM in its two main steps (data split and then search space split), in the first step BigFIM achieves the best

performance with sparse datasets, while in the second phase it better fits dense data distributions.

DistEclat and BigFIM are the only algorithms specifically designed for addressing load balancing and communication cost by means of the prefix length parameter k. In particular, the choice of the length of the prefixes generated during the first step affects both load balancing and communication cost.

2.2.4 Experimental evaluation

In this section, the results of the experimental comparison are presented. The behaviors of the algorithm reference implementations are compared by considering different data distributions and use cases. The experimental evaluation aims at understanding the relations between the algorithm performance and its parallelization strategies. Algorithm performance is evaluated in terms of execution time and scalability under different datasets and conditions.

Experimental setup

The experimental evaluation includes the following four algorithms, which are described in Section 2.2.3:

- the Parallel FP-Growth implementation provided in Mahout 0.9 (named Mahout PFP in the following) [23],
- the Parallel FP-Growth implementation provided in MLlib for Spark 1.3.0 (named MLlib PFP in the following) [22],
- the June 2015 implementation of BigFIM [24],
- the version of DistEclat downloaded from [24] on September 2015.

We recall that Mahout PFP extracts the top k frequent closed itemsets, BigFIM and DistEclat extract a superset of the frequent closed itemsets, while MLlib PFP extracts all the frequent itemsets. To perform a fair comparison, Mahout PFP is forced to output all the closed itemsets. Additionally, in our experiments, the numbers of frequent itemsets and closed itemsets are in the same order of magnitude.

Therefore, even if the extraction of the complete set of frequent itemsets is usually more resource-intensive than dealing with only the set of frequent closed itemsets, the disadvantages related to the more intensive task performed by MLlib are mitigated[2]

We defined a common set of default parameter values for all experiments. Specific experiments with different settings are explicitly indicated. The default setting of each algorithm was chosen by taking into account the physical characteristics of the Hadoop cluster, to allow each approach to exploit the hardware and software configuration at its best.

- For Mahout PFP, the default value of k is set to the lowest value forcing Mahout PFP to mine all frequent closed itemsets.
- For MLlib PFP the number of partitions is set to 6,000. This value has shown to be the best tradeoff among performance and the capacity to complete the task without memory issues. In particular, with lower values of the the number of partitions MLlib PFP cannot scale to very long transactions or very low *minsup*. Higher values, instead, do not lead to better scalability, while affecting performance.
- The default value of the prefix length parameter of both BigFIM and DistEclat is set to 2, which achieves a good tradeoff among efficiency and scalability of the two approaches.
- We did not define a default value of *minsup*, which is a common parameter of all algorithms, because it is highly related to the data distribution and the use case, so this parameter value is specifically discussed in each set of experiments.

For the evaluation, synthetic datasets have been generated by means of the IBM dataset generator [25], commonly used for performance benchmarking in the itemset mining context. We tuned the following parameters of the IBM dataset generator to analyze the impact of different data distributions on the performance of the mining algorithms: T = average length of transactions, P = average length of maximal patterns, I = number of different items, C = correlation grade among patterns, and

[2]We recall that the complete set of frequent itemsets can be obtained expanding and combining the closed itemsets by means of a post-processing step. Hence, to obtain the same output, the execution times of Mahout PFP, BigFIM and DistEclat may increase with respect to MLlib PFP

D = number of transactions. The full list of synthetic datasets is reported in Table 2.3, where the name of each dataset consists of pairs <parameter,value>.

All the experiments, except the speedup analysis, were performed on a cluster of 5 nodes running the Cloudera Distribution of Apache Hadoop (CDH5.3.1) [26]. Each cluster node is a 2.67 GHz six-core Intel(R) Xeon(R) X5650 machine with 32 Gigabytes of main memory and SATA 7200-rpm hard disks. The dimension of Yarn containers is set to 6 GB. This value leads to a full exploitation of the resources of our hardware, representing a good trade-off between the amount of memory assigned to each task and the level of parallelism. Lower values would have increased the level of parallelism (i.e. the number of concurrent parallel tasks) at the expense of the tasks available memory and, therefore, their ability to complete the frequent itemset mining. Higher values, instead, would have decreased the maximum level of parallelism.

For the speedup experiments we used a larger cluster of 30 nodes[3] with 2.5 TB of total RAM and 324 processing cores provided by Intel CPUs E5-2620 at 2.6GHz, running the same Cloudera Distribution of Apache Hadoop (CDH5.3.1) [26].

From a practical point of view, all the implementations revealed to be quite easy to deploy and use. Actually, the only requirement for all the implementations to be run was the Hadoop/Spark installation (from a single machine scenario to a large cluster). Only the MLlib PFP implementation requires few additional steps and some coding skills, since it is delivered as a library: users must develop their own class and compile it.

Impact of the minimum support threshold

The minimum support threshold (*minsup*) has a high impact on the complexity of the itemset mining task. Specifically, the lower the minsup, the higher the complexity of the mining task [8]. For this reason, this set of experiments uses very low minsup values. Specifically, we have tried to lower as much as possible the minsup values to understand the behavior of the algorithms dealing with such challenging tasks. Moreover, the selected minsup values strongly affect the amount of mined knowledge (i.e., the number of mined itemsets).

[3] http://bigdata.polito.it

ID	Name/IBM Generator parameter setting	Num. of different items	Avg. # items per transaction	Size (GB)
1	**T10**-P5-I100k-C0.25-**D10M**	18001	10.2	0.5
2	**T20**-P5-I100k-C0.25-D10M	18011	19.9	1.2
3	**T30**-P5-I100k-C0.25-D10M	18011	29.9	1.8
4	**T40**-P5-I100k-C0.25-D10M	18010	39.9	2.4
5	**T50**-P5-I100k-C0.25-D10M	18014	49.9	3.0
6	**T60**-P5-I100k-C0.25-D10M	18010	59.9	3.5
7	**T70**-P5-I100k-C0.25-D10M	18016	69.9	4.1
8	**T80**-P5-I100k-C0.25-D10M	18012	79.9	4.7
9	**T90**-P5-I100k-C0.25-D10M	18014	89.9	5.3
10	**T100**-P5-I100k-C0.25-D10M	18015	99.9	5.9
11	T10-P5-I100k-C0.25-**D50M**	18015	10.2	3.0
12	T10-P5-I100k-C0.25-**D100M**	18016	10.2	6.0
13	T10-P5-I100k-C0.25-**D500M**	18017	10.2	30.4
14	T10-P5-I100k-C0.25-**D1000M**	18017	10.2	60.9

Table 2.3 Synthetic datasets

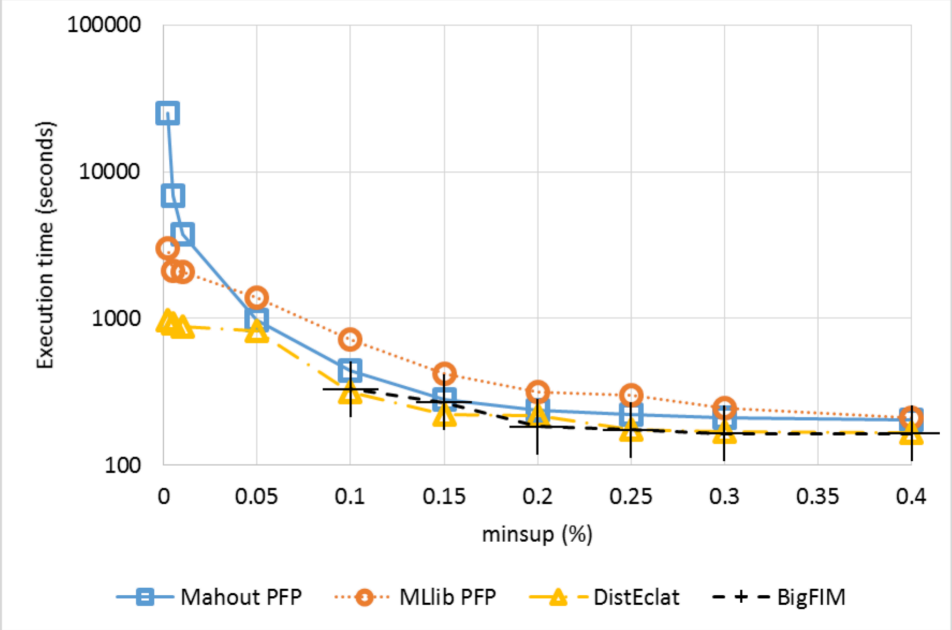


Fig. 2.5 Execution time for different *minsup* values (Experiment 1)

To avoid the bias due to a specific single data distribution, two different datasets have been considered: Dataset #1 for Experiment 1 and Dataset #3 for Experiment 2

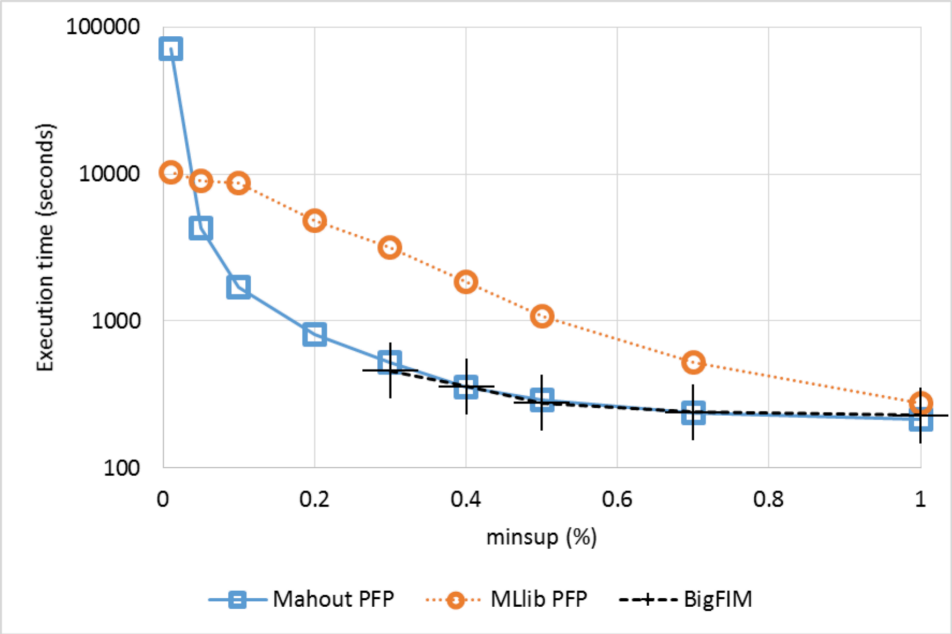


Fig. 2.6 Execution time for different *minsup* values (Experiment 2)

(Table 2.3). They share the same average maximal pattern length (5), the number of different items (100 thousands), the correlation grade among patterns (0.25), and the number of transactions (10 millions). The difference is in the average transaction length: 10 items for Dataset #1 and 30 items for Dataset #3. Being the other characteristics constant, longer transactions lead to a higher dataset density, which results into a larger number of frequent itemsets.

Experiment 1. Figure 2.5 reports the execution time of the algorithms when varying the *minsup* threshold from 0.002% to 0.4% and considering Dataset #1. DistEclat is the fastest algorithm for all the considered *minsup* values. However, the improvement with respect to the other algorithms depends on the value of *minsup*. When *minsup* is greater than or equal to 0.2%, all the implementations show similar performances. The performance gap largely increases with *minsup* values lower than 0.05%. BigFIM is as fast as DistEclat when *minsup* is higher than 0.1%, but below this threshold BigFIM runs out of memory during the extraction of 2-itemsets.

Experiment 2. In the second set of experiments, we analyzed the execution time of the algorithms for different minimum support values on Dataset #3, which is characterized by a higher average transaction length (3 times longer than Dataset #1), and a larger data size on disk, with the same number of transactions (10 millions).

Since the mining task is more computationally intensive, *minsup* values lower than 0.01% were not considered in this set of experiments, as this has proven to be a limit for most algorithms due to memory exhaustion or too long experimental duration (days). Results are reported in Figure 2.6. MLlib PFP is much slower than Mahout PFP for most *minsup* values (0.7% and below), and BigFIM, as in the previous experiment, achieves top-level performance, but cannot scale to low *minsup* values (the lowest is 0.3%), due to memory constraints during the k-itemset generation phase. Finally, DistEclat was not able to run because the size of the initial tidlists was already too big. Using the data-split approach, instead, BigFIM generates the set of candidates to be tested in independent chunks of the dataset. With a low minsup value, the set of candidates of the first phases is already too large to be stored and tested in each independent task. Overall, as expected, DistEclat is the fastest approach when it does not run out of memory. Mahout PFP is the most reliable implementation across almost all *minsup* values, even if it is not always the fastest, sometimes with large gaps behind the top performers. MLlib is a reasonable tradeoff choice, as it is constantly able to complete all the tasks in a reasonable time. Finally, BigFIM does not present advantages over the other approaches, being unable to reach low *minsup* values and to provide fast executions.

Impact of the average transaction length

We analyzed the effect of different average transaction lengths (*len*), from 10 to 100 items per transaction. We fixed the number of transactions to 10 millions. To this aim, Datasets #1–10 were used (see Table 2.3). Longer transactions often lead to more dense datasets and a larger number of long frequent itemsets. This generally corresponds to more computationally intensive tasks. We executed two sets of experiments, with a respective *minsup* value of 1% (Experiment 3) and 0.1% (Experiment 4). The execution times obtained are reported in Figure 2.7 and Figure 2.8.

Experiment 3. In Figure 2.7, BigFIM and DistEclat execution times for transaction length of 10 and 20 are not reported because, for these configurations, no 3-itemsets are extracted and hence the two algorithms could not complete the mining.[4] For higher transaction lengths, DistEclat is not included since it runs out of

[4]Due to the absence of a specific test, BigFIM and DistEclat present some issues if no itemsets longer than the value of the prefix length parameter are mined.

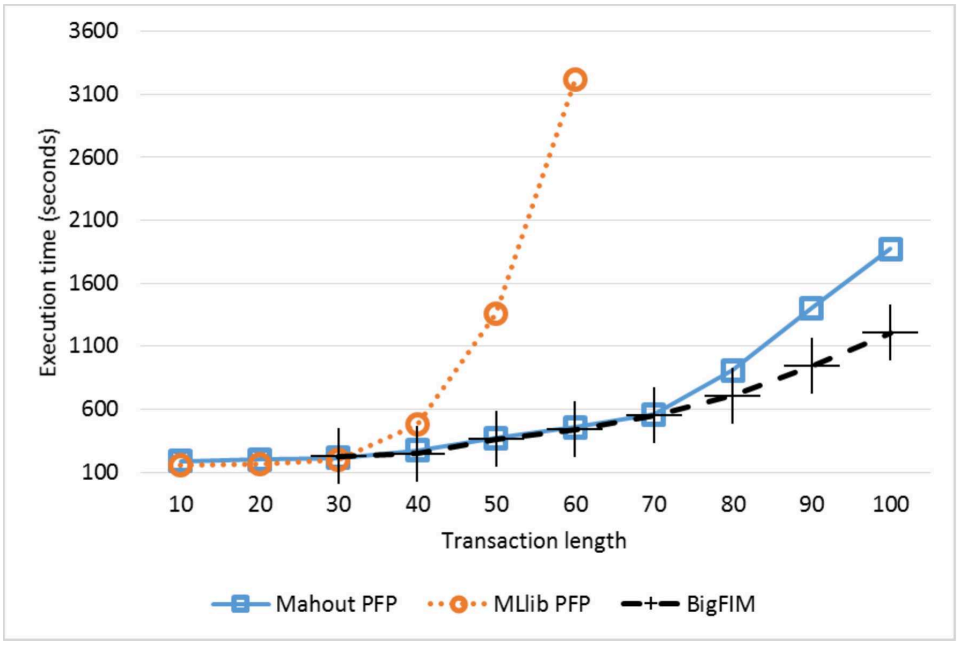


Fig. 2.7 Execution time with different average transaction lengths (Experiment 3).

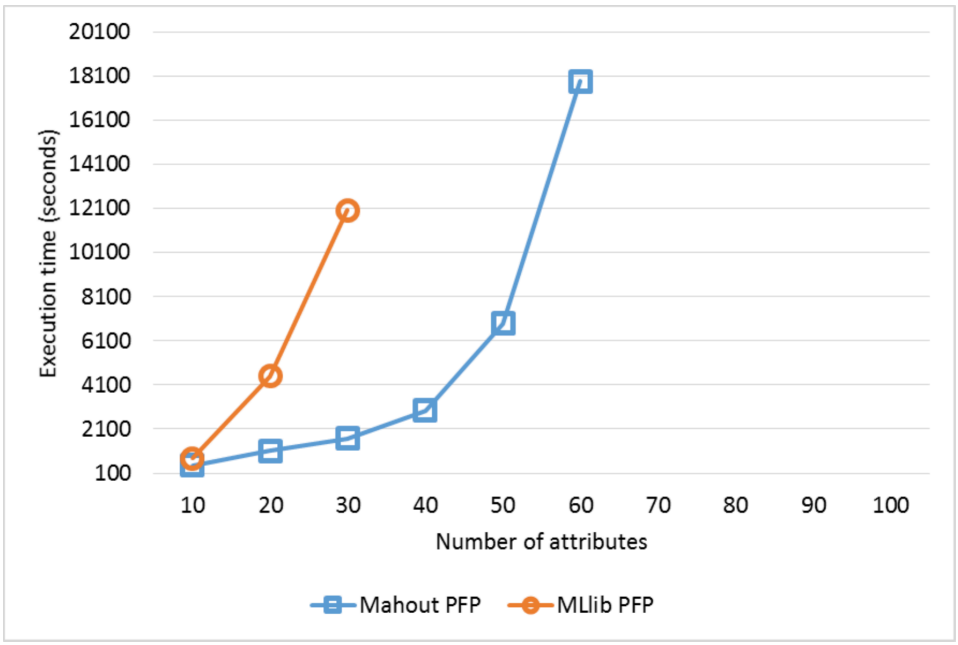


Fig. 2.8 Execution time with different average transaction lengths (Experiment 4).

memory for values beyond 20 items per transaction. The other algorithms have similar execution times for short transactions, up to 30 items. For longer transactions, a clear trend is shown: (i) MLlib PFP is much slower than the others and it is not

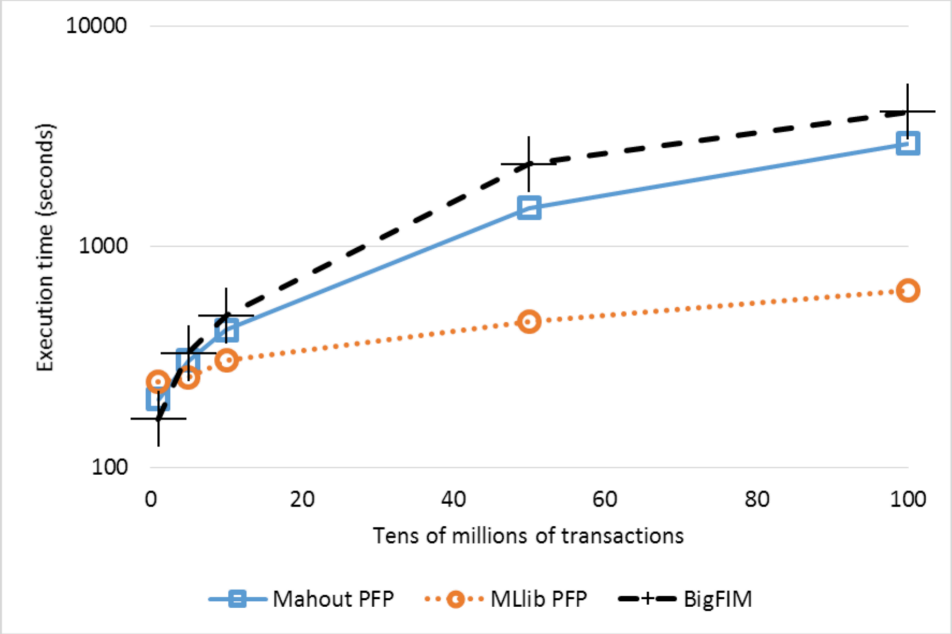


Fig. 2.9 Execution time with different number of transactions (Experiment 5).

able to scale for longer transactions, as its execution times abruptly increase until it runs out of memory; (ii) Mahout PFP and BigFIM have a similar trend until 70 items per transactions, when Mahout PFP becomes slower than BigFIM.

Experiment 4. The results of Figure 2.8 show a very similar trend, with exception that also BigFIM is not able to run. With this *minsup*, both Mahout PFP and MLlib PFP reach their limit way before the previous experiment.

Overall, despite the Apriori-based initial phase, BigFIM proved to be the best scaling approach for very long transactions and a relatively high minsup. When the *minsup* is decreased, BigFIM is penalized by the data-split approach which assumes to store all the candidates in each task memory, and only Mahout PFP is able to cope with the complexity of the task.

Impact of the number of transactions

Experiment 5. We evaluated the effect of varying the number of transactions, i.e., $|\mathscr{D}|$, without changing intrinsic data characteristics (e.g., transaction length or data distribution). The experiments have been performed on Datasets #1, #11–14 have

been used (see Table 2.3), which have a number of transactions ranging from 10 millions to 1 billion. The *minsup* is set to 0.4%, which is the highest value for which the mining leverages both phases of BigFIM, and it corresponds to the highest value used in the experiments of Section 2.2.4. Since in the experiment the relative minsup threshold is fixed, from the mining point of view, the search space exploration is similar and not particularly challenging, as shown in Section 2.2.4. What really affects this experiment is the algorithms reliability dealing with such amounts of data.

As shown in Figure 2.9, all the considered algorithms scale almost linearly with respect to the dataset cardinality, with BigFIM being the slowest, closely followed by Mahout PFP, and with MLlib PFP being by far the fastest approach, with execution times reduced by almost an order of magnitude. PFP implementations are faster than BigFIM because they read from the disk the input dataset only twice. BigFIM pays the iterative disk reading activities during its initial Apriori phase when the number of records of the input dataset increases. Finally, DistEclat fails under its assumption that the tidlists of the entire dataset should be stored in each node, and it is not able to complete the extraction beyond 10 million transactions.

Discussion

The experiments confirm that the performance of the data-split-based algorithms (i.e., BigFIM in its first phase) is highly affected by the number of candidate itemsets, which must be stored in the temporary main memory of each task. Specifically, BigFIM crashes during its Apriori-based phase when low *minsup* values or dense datasets are considered, due to the large number of generated candidate itemsets. This issue does not affect the approaches based on the search split strategy (Mahout PFP and MLlib PFP), since they do not need to store candidate itemsets as an intermediate result. Hence, Mahout PFP and MLlib PFP proved to be more suitable than BigFIM to process large dataset sizes, high-density datasets, and low *minsup* thresholds. DistEclat deserves a separate consideration: even if it is based on the search space approach, it often runs out of memory, because in its initial job it needs to store the *tidlists* of all frequent items in main memory and this operation becomes easily unfeasible when large or dense datasets are considered.

Experiments also highlight the predominant importance of load balancing in the itemset mining problem, in particular when comparing BigFIM to Mahout PFP. Since

the initial mining phase of BigFIM is based on the data split parallelization approach, it reads many times the input dataset (differently than Mahout PFP). Moreover, BigFIM is also characterized by greater communication costs than Mahout PFP. These two factors should impact significantly on the execution time of BigFIM. Instead, not only the execution time of BigFIM is comparable with that of Mahout PFP with 1000-million record datasets (Figure 2.9), but BigFIM is also even faster than Mahout PFP in specific cases, e.g., with datasets with an average number of items per transaction greater than 70 (Figure 2.7). The rationale of such results is the better load balancing of BigFIM with respect to Mahout PFP.

2.2.5 Digest of the experimental session

To analyze in detail the impact of the choice of an approach over the other, we run an extensive set of experiments, as shown in Section 2.2.4, aimed at finding the strengths and the limitations of each available algorithm. These experiments provided a wide view of the different behaviours of the algorithms in various experimental settings. With this digest, we aim at supporting the reader in a conscious choice of the most suitable approach, depending on the use case at hand. Pursuing this target, we measured the real-life performance of the openly-available frequent-pattern mining implementations for the most popular distributed platforms (i.e., Hadoop and Spark). They have been tested on many different datasets characterized by different values of minimum support (*minsup*), transaction length (dimensionality), number of transactions (cardinality), and dataset density, besides two real-life use cases. Performance in terms of execution time, load balancing, and communication cost have been evaluated: a one-table summary of the results is reported in Table 2.4. As a result of the described experience, the following general suggestions emerge:

- **High reliability.** Without prior knowledge of dataset density, dimensionality (average transaction length), and cardinality (number of transactions), **Mahout PFP** is the algorithm that best guarantees the mining task completion, at the expense of longer execution times. Mahout PFP is the only algorithm able to always reach the experimental limits.
- **High cardinality and low-dimensional data.** On most real-world use cases, with limited dimensionality (up to 60 items per transaction on average), **MLlib**

	Exp. 1 *minsup*	Exp. 2 *minsup*	Exp. 3 *len*	Exp. 4 *len*	Exp. 5 $\|\mathscr{D}\|$
Mahout PFP	0.002%	0.01%	**100**	**60**	100M
MLlib PFP	0.002%	**0.01%**	60	30	**100M**
BigFIM	0.1%	0.3%	100	-	100M
DistEclat	**0.002%**	-	-	-	1M

Table 2.4 Summary of the limits identified by the experimental evaluation of the algorithms (lowest *minsup*, maximum transaction length *len*, largest dataset cardinality $|\mathscr{D}|$) in Section 2.2.4. The fastest algorithm for each experiment is marked in bold.

PFP has proven to be the most reasonable tradeoff choice, with fast execution times and optimal scalability to very large datasets.

- **High-dimensional data.** For high-dimensional datasets, **BigFIM** resulted the fastest approach, but it cannot cope with *minsup* values as low as the others. In those cases, **Mahout PFP** represents the only option.
- **Limited dataset size.** When the dataset size is small with respect to the available memory, **DistEclat** has proven to be among the fastest approaches, and also to be able to reach the lowest experimental *minsup* values. DistEclat experiments showed that it cannot scale for large or high-dimensional datasets, but when it can complete the itemset extraction, it is very fast.

2.2.6 Choosing an approach for scaling associative classification

In this section, we have reviewed the two main approaches used by distributed frequent itemset mining algorithms to divide the workload and reach high scalability. The training phase of an associative classifier often requires the generation of all association rules that satisfy some minimal constraints, namely a minimum support and a minimum confidence, and thus relies on the generation of frequent itemsets, with their support, in the first step. Therefore, associative classification shares with the frequent itemset mining problem many of its scalability issues, like the usage of main memory, the communication costs and the load balancing. The design of a scalable associative classifier needs to cope with these issues.

In Section 2.2.2 we have seen as the main advantage of the data split approach is its good load balancing. This characteristic is fundamental for a distributed

algorithm, as the slowest worker in the process will be the bottleneck of the entire chain of operations. On the other hand, the search space split approach shows a very good scalability. One of its implementation in particular, FP-Growth, and its parallel version PFP, deals in a very effective way with the usage of the main memory. Unfortunately, search space split approaches do not manage load balancing as well as the data split ones, and their performance can highly vary depending on the distribution of the frequent itemsets in the dataset.

The usage of main memory is a crucial issue in the frequent itemset problem as much as in associative classification. In the frequent itemset mining task, both approaches to distribution are limited by the amount of available memory. The limit of the data split approach is the amount of candidate itemsets that can be stored, and thus it fails to scale on dense datasets. The search space split approach, instead, runs out of memory when large projected datasets are generated. This issue, though, is mitigated by the usage of efficient data structures, like the FP-tree in the FP-growth implementation of this approach, for example. In associative classification, the amount of memory available is not only used in the itemset generation phase, but also to store candidate rules before a final pruning phase that selects the best rules for classification. The memory issue becomes therefore, even more than in frequent itemset mining, the bottleneck of scalability. In the next sections, we will see two approaches to distributed associative classification that exploit heavily the data split approach to foster load balancing, with two different paradigms of memory management.

2.3 BAC: a Bagged Associative Classifier

In this section, we aim at distributing the training phase of an associative classifier. The distribution of the work on a cluster of machines could have several advantages, among which a performance boost, or an increase of the dataset size which can be analyzed. Unfortunately, as we will see in Section 2.3.1, the training of an associative classifier is highly sequential for its nature. Our approach exploits bagging, a well-known supervised learning technique that splits the input dataset and the works in different partitions. In this way, it applies the data split paradigm that we have seen very effective for frequent itemset mining (Section 2.2.2). Preliminary experimental results exploiting a distributed approach on Apache Spark applied to real-world and

synthetic datasets showed promising results: ensembles of multiple models trained on small subsets of the original dataset (as small as 10%) lead to accuracies comparable with the single model trained on the whole dataset. Our method thus proves to be a viable solution to distribute the workload of this task, without compromising on the quality of the results.

This section is organized as follows. Section 2.3.1 describes the data mining background, and Section 2.3.2 presents the proposed approach.

2.3.1 Background

In this section, we describe the centralized L^3 associative classifier [27], since the distributed BAC algorithm proposed in this chapter is based on L^3. We also provide an overview on the bagging technique.

The L^3 classifier

The proposed distributed algorithm (Section 2.3.2) is based on the L^3 associative classifier. Hence, we describe the centralized version of L^3. The model generation phase of L^3 algorithm is based on two main steps: (i) frequent CAR mining and (ii) rule selection by means of a lazy database coverage technique. To address the mining step efficiently, L^3 exploits an FP-growth like association rule mining algorithm. The exploited mining algorithm is optimized to directly extract CARs.

Once the potentially large set of frequent CARs is available, a lazy pruning step, based on the database coverage approach is applied. Before performing the pruning phase, a global order is imposed on the extracted frequent CARs. Let r_1 and r_2 be two CARs. Then r_1 precedes r_2, denoted as $r_1 > r_2$ if

1. $conf(r_1) > conf(r_2)$, or
2. $conf(r_1) = conf(r_2)$ and $sup(r_1) > sup(r_2)$, or
3. $conf(r_1) = conf(r_2)$ and $sup(r_1) = sup(r_2)$ and $len(r_1) > len(r_2)$, or
4. $conf(r_1) = conf(r_2)$ and $sup(r_1) = sup(r_2)$ and $len(r_1) = len(r_2)$ and $lex(r_1) > lex(r_2)$.

After the rule sorting operation, only the rules satisfying the χ^2 test, at a significance level of 95% are selected (i.e., the rules with a correlation between the antecedent and the consequent of the rule).

Finally, the lazy pruning approach is applied. The idea behind the lazy pruning is to discard only the rules that do not correctly classify any training record. To achieve this goal, the lazy pruning approach splits the frequent CAR set in three subsets:

- **First level rules**. This rule set contains the mining set of CARs that is needed to properly "cover" the training data. This set of rules represents the main characteristics of the majority of the training transactions.
- **Second level rules**. These rules are not included in the first level, but are potentially useful to represent the characteristics of "special" records that are slightly different with respect to the most common ones appearing in the training set. These rules are also called "spare" rules and are useful to properly label new unlabelled data not represented by the rules of the first level.
- **Harmful rules**. Some rules, even when applied on the training set, always perform wrong predictions. These rules must be removed since they contribute only negatively to the quality of the generated model.

To identify the three described subsets, L^3 applies the database coverage technique described in Section 2.1.2, with the difference that rules that do not match any training transaction are not discarded, and saved in the second level of the classifier. The rules that classify properly at least one training transaction are stored in the first level of the classifier. Harmful rules, as in the original technique, are discarded.

To predict the class label of an unlabelled transaction t, the rules of the first level are initially considered. If no rule in the first level matches t, the second level is used.

Bagging

The bagging technique is frequently used to build accurate models by combining a set of "weak" classifiers [28]. The basic idea is that a set of classifiers can provide better predictions than a single model. It was firstly introduced in [29], and later adopted in the definition of Random Forests [30].

The bagging technique works as follows.

1. Generate N datasets by applying random sampling with replacement on the training dataset.
2. Build N classification models (one for each dataset generated during Step 1).
3. Predict the class label of the new unlabelled records by using a majority voting approach combining the predictions of the N classifiers that have been built during Step 2.

2.3.2 The proposed approach

This section describes the proposed approach to scale on different machines the training of an associative classifier, namely L^3, which is presented in Section 2.3.1. The Big Data framework we exploited is the well-known Apache Spark, which poses some specific technological issues to the design of an associative classifier.

We recall, as mentioned in Section 2.3.1, that an important phase of the generation of such a classifier is represented by the database coverage (lazy pruning in L^3). Unfortunately, the database coverage algorithm does not fit a MapReduce approach, as it is sequential in its nature: the database must be covered in the strict order of the rules for the algorithm to be effective. Such requirement prevents most of the work of this phase to be executed in parallel.

To cope with the scalability of the process, in order to fully exploit the distributed framework, we adopt bagging. The solution consists in generating several models, each one from a portion of the original full dataset. Each model can then be trained independently, thus also in parallel on multiple machines. The model trained locally on each portion of the dataset is a variant of L^3, where we discard the second level rules after the database coverage phase. The FP-growth algorithm used is the one implemented in Apache Spark, slightly modified to run locally. The ensemble of the single models eventually generate a single prediction by majority voting.

In the following we provide details about the split of the dataset among the different machines (i.e., Apache Spark workers). The ensemble of models is eventually collected and can be used on any number of machines to classify new records.

Dataset distribution

Each model, as said, is trained on a different portion of the original dataset. Each portion is drawn by sampling the original records with replacement, as this method is well-proven in literature, as mentioned in Section 2.3.1. Moreover, whereas this method allows for obtaining any sort of combination for number of models and partitions size, sampling without replacement would limit the total number of records to the original dataset size. In other words, sampling with replacement can produce, for example, multiple models trained on a dataset as large as the original dataset (and still different), or even larger; or we might have a dozen models, each trained on half of the original dataset, while without replacement we would have just two halves available.

The core APIs of Apache Spark provide a method for performing sampling with replacement. The method is characterized by two parameters: the input dataset $\mathscr{D}$ and a real number λ that is used to specify the size of the sample as a fraction of the input dataset. Specifically, a sample of size $|\mathscr{D}| \times \lambda$ is generated. The provided method performs a sampling with replacement, generating k copies for each input record, where k is drawn from a Poisson distribution.

In our case, we are interested in N samples, each one with a size equal to $|\mathscr{D}| \cdot \alpha$, since we want to build N models for N different samples. α is a real parameter of the algorithm and it is used to specify the size of each sample. We can generate the N samples invoking, sequentially, the Spark sampling API N times setting λ to α. However, this approach will reduce the parallelism of our algorithm. Hence, we decided to use another approach that allows us generating simultaneously N samples, each of size $|\mathscr{D}| \cdot \alpha$. Specifically, we proceed in the following way. First, we sample the dataset $\mathscr{D}$ with $\lambda = N \cdot \alpha$, as to have an expected single sample with a total size equal to $|\mathscr{D}| \cdot N \cdot \alpha$. Now we need to split this sample in N subsamples. We can perform this operation by using the parallel Spark API `repartition` that splits the input data in a user-specified number of partitions (in our case we set it to N). Hence, each partition contains a sample of size $|\mathscr{D}| \cdot \alpha$ of the input dataset.

Once obtained N samples with the desired fraction of data, each on a separate partition, we can simply call `mapPartitions` in Spark to apply the same function on each, in our case the training function of the model.

2.3.3 Experimental evaluation

To validate the proposed approach, we implemented BAC in Scala on top of Apache Spark. Experiments aim at assessing the usefulness of a distributed approach against two alternatives: (i) working on the whole dataset on a single machine, which is not always feasible, and (ii) working on a sample of the original dataset, that is always feasible for small portions, but at the cost of a lower quality of the model (e.g., accuracy). Experiments also evaluate the effect of the number of estimators (models) of the ensemble on the final quality of the whole ensemble model.

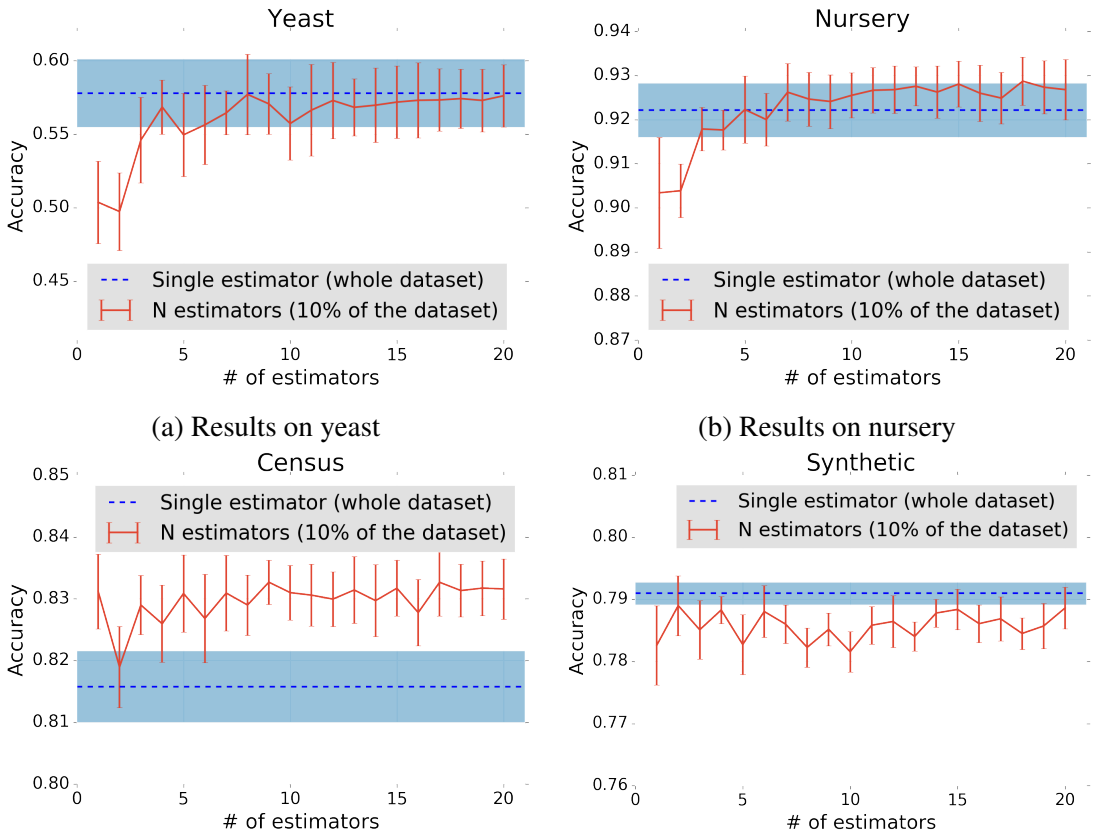
As evaluation criterion, we focus primarily on the accuracy, computed on a 10-fold cross-validation of the selected datasets. For each mean accuracy, we computed a 95% confidence interval based on the standard error of the mean, using a t-student statistics. We also computed the average time for training.

Three very popular datasets have been used for the experiments: `yeast`, `nursery`, and `census`, from the UCI repository [31]. We have chosen these datasets as they are heterogeneous in dimensions, shape and distribution, and state-of-art associative classifiers do not perform well, so that there is still margin for improvements. The continuous attributes have been discretized applying the entropy-based discretization technique [28]. Since the bigger of the three datasets counts for 30162 records only, we generated a fourth synthetic dataset containing 1 million tuples and 9 attributes with different distributions, using the IBM data generator. Continuous attributes have then been discretized to 10 bins each.

All experiments share the same minimum support threshold (1%) and the same minimum confidence value (50%), as in previous works of associative classifiers [27] they proved to generate a good amount of significant rules. Experiments were performed on a cluster with 30 worker nodes running Cloudera Distribution of Apache Hadoop (CDH5.5.1), which comes with Spark 1.5.0. The cluster has 2.5TB of RAM, 324 cores, and 773TB of secondary memory. The size of each container has been set to 2GB.

Results

Focusing on real datasets, that are `yeast`, `nursery`, and `census`, Figures 2.10a, 2.10b, and 2.10c show the average accuracy. The dashed line represents the accuracy



(a) Results on yeast

(b) Results on nursery

(c) Results on census

(d) Results on the synthetic dataset

Fig. 2.10 Accuracy results

obtained by the classifier trained over the whole dataset on a single machine. The range of its confidence interval, highlighted in the figures, serves as reference for our evaluation. On the x axes we have the number of models trained, each on a 10%-portion of the original dataset. For $x = 1$, we have the simple sampling. For `yeast` (Fig. 2.10a), we see how the simple sampling can reduce the total accuracy of more than 8% with respect to the level of the classifier trained on the whole dataset. In `nursery` (Fig. 2.10b) we see the same behaviour. Here the drop is less marked, less than 2%, but still significant if we compare the confidence intervals. Curiously, `census` (Fig. 2.10c) shows a completely different behaviour. In this dataset, representing a US census of the population, sampling obtains an accuracy even higher than our reference. This surprising result can be due to a simpler, even if weaker, model less prone to overfitting the training data. The 10-fold cross-validation indeed penalizes overfitting models.

Let us now inspect the results of BAC, with an increasing number of models, up to 20. For all datasets, the accuracy level stabilizes way before 10 models. We see that `yeast` (Fig. 2.10a), for example, enters the confidence interval of the reference model from 3 models on, becoming statistically indifferent. Large confidence intervals are a peculiarity of the k-fold cross-validation, and highly depend on the dataset. It is as well a measure of the robustness of the model towards different training sets. `Nursery` (Fig. 2.10b) and `census` (Fig. 2.10c) confirm the same trend as `yeast`, with BAC entering the confidence interval of the centralized classifier from 3 models onward. These results are interesting, since 3 models cover less than a third of the original dataset, and they are enough to reach similar accuracies to our reference model. This means also that distributing the dataset only once, i.e., 10 models of 10% each, we can be confident enough of having entered already the steady region, without the need of more replicas/models. Furthermore, having a number of models equal to one corresponds to the simple sampling, whereas two models would affect the contribution of the majority voting among models. Thus 3 is also the minimum sensible setting of this parameter.

Results for the synthetic dataset are shown in Figure 2.10d. The dataset counts for a million records, and tries to emulate a possible use case of BAC, to show its potential on large dataset. Firstly, the accuracy of the reference classifier is characterized by a very narrow confidence interval. This, as mentioned above, comes from the shape and distribution of the dataset: since it is synthetic, the distribution of the attributes and of the labels is uniform among the folds, thus the standard error of the mean accuracy is very little compared to the real datasets, feature that is further sharpened by the greater size of the folds. Sampling the 10% of the dataset results in an almost negligible detriment to the quality, less than a point of accuracy. This outcome is not surprising, as the distribution of the attributes in the sample can not vary too much from the whole, being the attributes generated from a given distribution. The accuracy of BAC increasing the number of models steadily stays in the range of the single sample, often overlapped with the results of the reference model, showing that adding data from the remainder of the dataset does not add information to our model. Though in this case we cannot conclude for absolute better performances on either side (namely, BAC or single-machine), we see an interesting fact in the way these results were obtained. On the cluster described at the beginning of the section, indeed, the generation of the reference classifier, the single model trained on the whole dataset, failed, running out of the memory available to

its (single) container. To obtain the value of reference plotted in Figure 2.10d we executed the training on a standalone Spark machine, where we set the amount of memory for the Java VM to the whole RAM available on the machine itself (32 GB). All experiments for BAC, instead, completed successfully, proving that distributing workload can be a way to overcome the limits of the single machine and expand the size of the explorable datasets.

Finally, Table 2.5 shows the average training time for the 10% sample, for BAC with 10 models on a tenth of the data each, and the reference model, that is the classifier trained on the whole data. As said, the last-mentioned classifier failed its execution for the synthetic dataset, so its timing is not fairly comparable with the others. On the three real datasets, we notice that, unsurprisingly, simple sampling outperforms the more accurate models. BAC is the slowest, as the overhead of the distributed framework and the communication costs are very high for such small datasets. On the larger synthetic dataset, these overheads are absorbed by the real computational costs, resulting in little difference between one or 10 machines working on models of the same size/complexity. We need further investigations to show the real impact of parallelization on a real dataset of this size or larger.

To sum up, from these results we can conclude that:

1. the mere sampling is not always sufficient to reach a good accuracy,
2. training an associative classifier over the whole dataset is not always feasible,
3. bagging is a viable solution to reach the quality of a single classifier trained on the whole dataset, and offers an easy way to distribute the work among multiple workers.

Dataset	Records	Avg training time for a fold [ms]		
		1 model, 10%	10 model, 10%	1 model, 100%
Census	30162	57692	340020	302147
Nursery	12960	983	3457	2181
Yeast	1484	642	1799	850
Synthetic	1000000	14235	15150	n.a.

Table 2.5 Average training time of the different approaches.

2.4 DAC: a Distributed Associative Classifier

This section introduces DAC, a Distributed Associative Classifier, whose training phase is designed to be distributed in an in-memory cluster computing framework like Apache Spark.

In Section 2.3, we have seen an effective approach to split the training of an associative classifier without losing predictive quality, which is bagging. This strategy alone has a limit, though, as the amount of memory needed grows with the number of rules extracted, and thus does not scale. We build upon the findings of Section 2.2, where we found the most scalable frequent itemset miners the ones relying on the search space split approach and an effective usage of memory, and we design an approach that aims at limiting the amount of rules stored in memory and making an effective usage of memory with ad-hoc data structures.

The section is organized as follows. Section 2.4.1 explains how DAC works, and Section 2.4.2 describes the experimental evaluation of DAC.

2.4.1 The proposed approach

Traditionally, the training phase of an associative classifier is a memory-intensive process, often executed out-of-core. The vast majority of the techniques has at least an instant of time where a very large set of itemsets or rules has been extracted and not yet pruned. This model cannot leverage the advantages of our reference architecture, an in-memory cluster computing framework like Apache Spark. In building a scalable associative classifier, we have been guided by the two following design principles: i) anticipating pruning before the actual extraction of the rules, and ii) moving from a large model that predicts with only the first matching rule toward a lightweight model, that compensates the loss in size by applying all the rules that match. These two principles aim at reducing the amount of rules contemporarily present in the main memory at any given instant of time, allowing for an effective exploitation of the in-memory computing platform.

The baseline framework on which we build for the training of our Distributed Associative Classifier, namely DAC, is as follows.

1. The dataset is split into N partitions, each one sampled from the original dataset with a ratio α;
2. Within each partition, a rule extraction phase occurs, that produces a model as a set of CARs. The CARs found are filtered by minimum support, minimum confidence and minimum χ^2 and optionally further pruned with a database coverage phase;
3. The generated N models are collected in an ensemble.

Following our first design principle, we aimed at devising an extraction phase that made the work of the posterior pruning extremely reduced or null, in the best case. We have therefore adopted a greedy approach based on the Gini impurity of an item, keeping in mind the second design principle presented before, that we finally want a smaller model where several rules can collaborate for the prediction, instead of a single first-match. This calls for shorter rules, that can more easily match new records and avoid over-fitting. In order to follow such a route without sacrificing predictive quality we designed several solutions that will be presented in the next sections, namely: i) an FP-growth-like CAR extractor that produces only useful classification rules, in a greedy fashion, by exploiting the Gini impurity; ii) an added model consolidation phase for the generation of the ensemble that reduces further the size of the final model; iii) new voting strategies for the ensemble that exploit the before-mentioned novelties.

CAP-growth

The FP-tree is an effective solution for frequent itemsets extraction, and is often adapted to the extraction of CARs [11]. Moreover, it adapts well to in-memory computing, as its construction needs only two scans of the dataset and, once built, the FP-tree stores in the main memory all the necessary information for frequent itemsets or CARs extraction.

However, there is a twofold motivation behind designing an alternative to the FP-tree, like [32, 33], as method of storage for the patterns that will build the final CARs. First, the FP-tree is designed to build all frequent patterns, that are a superset of what we look for when we build CARs. Second, being frequent does not always coincide with being useful, and using the standard FP-growth algorithm would yield

the growth of an overwhelming number of rules that would impede the descent to lower supports, where more useful information may dwell. Guided by these considerations, and keeping in mind the design principles outlined in the beginning of the section, we designed an FP-growth-like algorithm called CAP-growth, for Class Association Patterns growth.

CAP-growth stores the information that is useful for extracting CARs in a CAP-tree. Similarly to an FP-tree, this structure allows to compactly store all the information needed to extract association rules reading the dataset only twice. Differently from the FP-tree, a CAP-tree stores in each node extra information useful to extract only CARs, as it is usually done in single-machine approaches[32, 33]. Moreover, the first phase of the CAP-tree's construction sorts the frequent items by their Gini impurity, which will help the extraction of more useful rules in the CAP-growth phase.

The algorithm that builds a CAP-tree is detailed in Algorithm 2.1.

Algorithm 2.1: CAP-tree building

Input : A transaction DB labeled with classes - $\mathscr{D}$
Input : A minimum support threshold - *minsup*
Output : A CAP-tree

```
1  Scan 𝒟 once. Collect L, the list of frequent items (support >= minsup).
     Sort L by decreasing IG and filter out items with IG ≤ 0.
2  Create the root of a CAP-tree T and label it as null.
3  for each labeled transaction t do
4  |   select only the items in t that appear in L and sort them according to the
   |     order in L, obtaining t'
5  |   call insert(t', T)
6  end
7  Function insert (transaction t, node T)
8  |   h = first item of t
9  |   if T has a child T' s.t. T'.id = h.id then
10 |   |   T'.freqs[t.class]+=1
11 |   else
12 |   |   create a new node T'
13 |   |   init T'.id = h.id and T'.freqs to an array of zeros
14 |   |   T'.freqs[t.class]+=1
15 |   |   T'.parent = T
16 |   |   update the header table
17 |   end
18 |   t' = t\h
19 |   if t' is not empty then
20 |   |   insert(t', T')
21 |   end
```

Given a minimum support threshold, which is used to recognize frequent itemsets, the algorithm scans the dataset twice. In the first pass (line 1), it builds a list L of frequent items, with decreasing and strictly positive Information Gain. Since we are considering the item alone, we assume that the $(1-w_i)$-th part of the dataset not covered by the item has a distribution of the labels identical to the global distribution

tid	Transaction	Class
1	$\{A,B,D,E\}$	+
2	$\{B,C,E\}$	-
3	$\{A,B,D,E\}$	+
4	$\{A,B,C,E\}$	-
5	$\{A,B,C,D,E\}$	+
6	$\{B,C,D\}$	-

Table 2.6 An example transactional dataset, binary-labeled.

(same Gini). Hence, the Information Gain is computed as follows:

$$IG_i = Gini_{\mathscr{D}} - [w_i Gini_i + (1 - w_i) Gini_{\mathscr{D}}] \tag{2.1}$$

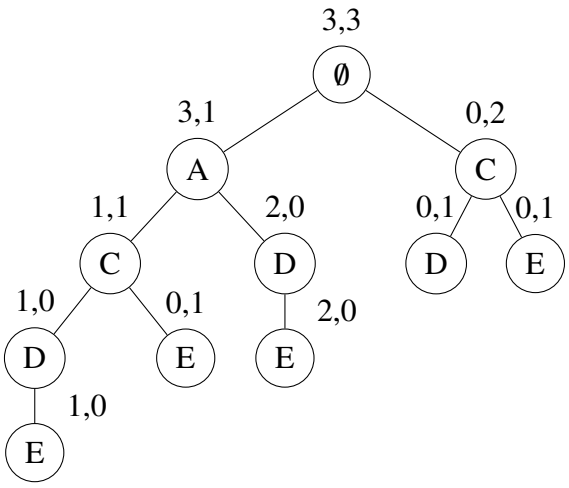
in which $Gini_{\mathscr{D}}$ is the impurity of the global dataset, $Gini_i$ is the impurity of item i, and w_i is the ratio of dataset $\mathscr{D}$ containing item i.

Equation 2.1 simplifies as

$$IG_i = w_i(Gini_{\mathscr{D}} - Gini_i) \tag{2.2}$$

In this first passage, we can also obtain the frequency of the classes in the entire dataset, which is used in the CAP-growth's extraction phase. In the second pass (lines 2-6), we insert each read transaction in the CAP-tree (line 5), maintaining a header table that keeps track of the pointers to the nodes in the tree that store the frequent items, like in the original FP-tree (line 16). Before being inserted, the transaction is cleaned from the infrequent items and reordered according to the order of L (decreasing IG) (line 4). The insertion updates the structure of the CAP-tree to keep track of the label of the transaction in an array of frequencies (lines 9-17). This allows the direct extraction of CARs and the computation of the IG and the confidence of the rules in the CAP-growth.

Figure 2.11 shows the CAP-tree built on the toy dataset in Table 2.6, with the minimum support threshold set to 0.3, that is 2 records. Each node of the tree is labeled with the array of the frequencies of the classes, positive and negative respectively. In this tree we see how item B has been pruned, since its IG is 0, and how the remaining items are sorted and inserted by their IG, with item A being the



Header Table		
Item	Freqs	
	+	-
A	3	1
C	1	3
D	3	1
E	3	2

Fig. 2.11 A CAP-tree example built over the toy dataset with minimum support equal to 0.3

Item	w_i	$Gini_i$	IG_i
A	4/6	0.375	8.3%
B	1	0.5	0
C	4/6	0.375	8.3%
D	4/6	0.375	8.3%
E	5/6	0.48	1.7%

Table 2.7 *IG*, weight and Gini for the items in the toy dataset

first and the most useful for classification. The *IG* of all items of the toy dataset are shown for reference in Table 2.7, together with their weight *w* and Gini.

CAP-growth extracts a set of CARs from the CAP-tree descending the tree greedily. This yields that, since the frequent items are sorted by decreasing *IG*, we evaluate the rules made of high-*IG* items first. The rationale that guided the design of the algorithm is to avoid redundant rules, where possible, while keeping the length of the rules minimal.

The following example illustrates some ways in which redundancy affects CARs. In other approaches, this redundancy is often reduced after the extraction of CARs, as shown in Section 2.5. We provide this example so that the reader may later gain an intuition of where CAP-growth helps reducing redundancy before the extraction itself. In Figure 2.12 we see all the CARs in the set of association rules extracted

$$
\begin{array}{cc}
EDA \Rightarrow + & BEDA \Rightarrow + \\
ED \Rightarrow + & BED \Rightarrow + \\
EC \Rightarrow - & BEC \Rightarrow - \\
EA \Rightarrow + & BEA \Rightarrow + \\
DA \Rightarrow + & BDA \Rightarrow + \\
A \Rightarrow + & BA \Rightarrow + \\
E \Rightarrow + & BE \Rightarrow + \\
C \Rightarrow - & BC \Rightarrow - \\
D \Rightarrow + & BD \Rightarrow +
\end{array}
$$

Fig. 2.12 An example model with CARs for the dataset in Table 2.6

with the standard FP-Growth, with minimum support set to 0.3 (2 rows or more) and minimum confidence 0.51 on the toy dataset in Table 2.6. 18 CARs for a dataset of 6 records are clearly redundant. A first, evident source of this redundancy is item B, which is present in all the records in the dataset. This results in having, for any rule generated, an identical rule with B appended, that does not contribute to the classification and lengthens the model. A similar situation happens with item E. Likewise, item C appears in many rules, all of which agree in classifying a record as negative: item C itself would be sufficient as antecedent of the rule. The same holds for other rules as well.

As previously stated, CAP-growth aims to avoid the redundancy of the example above. Algorithm 2.2 shows the pseudocode for CAP-growth. Similarly to Equation 2.2, we define the Information Gain for a node as

$$IG_T = w_T(Gini_{T.parent} - Gini_T) \tag{2.3}$$

where w_T is the ratio of transactions represented in node T with regards to its parent node, and Gini is computed on the frequencies of the labels stored in the node.

The algorithm is a recursive call to the function extract (line 6), which visits in a depth-first fashion the CAP-tree. The stopping criteria of this visit are:

1. a negative Information Gain for the current node. In this case, we do not generate any rule (line 9).

Algorithm 2.2: CAP-growth

Input : a CAP-tree
Input : A minimum support threshold - minsup
Input : A minimum confidence threshold - minconf
Input : A minimum chi2 threshold - minchi2
Output : A list of CARs

```
1  rules = ∅
2  for each child T of CAP-tree.root do
3  |  rules += extract(T)
4  end
5  return rules
6  Function extract(node T)
7  |  rules = ∅
8  |  if IG(T) <= 0 then //negative Information Gain: do not generate any
   |   rule
9  |  |  return ∅
10 |  end
11 |  if Gini(T) == 0 then //pure node: try to generate a rule
12 |  |  return generateRule(T)
13 |  end
14 |  for each child T' of T do
15 |  |  rules += extract(T')
16 |  end
17 |  if rules is ∅ then //none of the children has produced a rule: try to
   |   generate a rule
18 |  |  return generateRule(T)
19 |  end
20 |  return rules
21 Function generateRule(node T)
22 |  consequent = class with highest value in T.freqs[]
23 |  antecedent = set of items in the path from T to CAP-tree.root
24 |  tree = CAP-tree conditioned by the items in antecedent
25 |  freqs = tree.root.freqs
26 |  sup = freqs[consequent] / totCount
27 |  supAntecedent = freqs.sum / totCount
28 |  from sup, supAntecedent and the global frequencies of the classes
   |   computed in the first pass of Algorithm 2.1 compute support,
   |   confidence and χ^2 for the generated rule: antecedent ⇒ consequent
29 |  if sup < minsup or conf < minconf or χ^2 < minchi2 then
30 |  |  return ∅
31 |  end
32 |  return rule
```

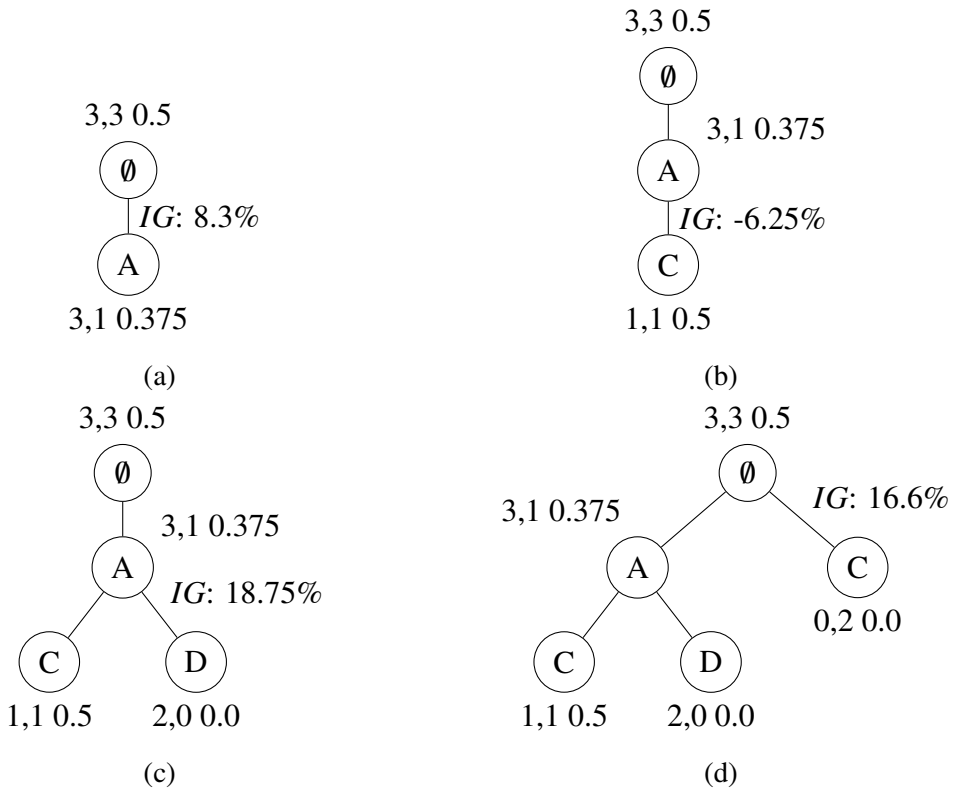


Fig. 2.13 Example visit of the CAP-tree in Figure 2.11

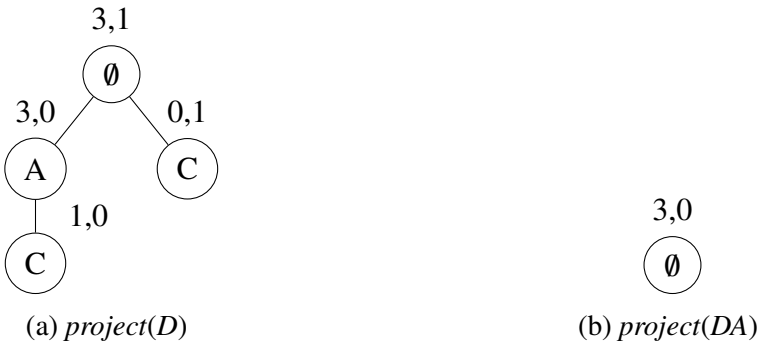


Fig. 2.15 Example projection of the CAP-tree in Figure 2.11 to reconstruct the support of itemset {A,D}

2. a Gini impurity for the current node equal to 0. Being the Gini impurity always strictly decreasing, this makes the current node the first pure node in the path from the root to this node, i.e. we see only one label for it. We try to generate a rule(line 12).

Whenever none of the children of a node does generate a rule, the node itself tries to generate a new rule (lines 14-19). This can occur when the children nodes do not see enough samples to satisfy the minimum support threshold, for example, or if the current node is a leaf.

The function that generates a new rule (lines 21-32) needs first to recollect the frequencies of the labels from all the nodes where the current pattern appears. Like in the original FP-growth, this is done by projecting the CAP-tree recursively on all the items of the pattern, that is all the nodes in the path to the root (line 24). At the end of the projection, the root node contains the array of classes' frequencies for the pattern (line 25). With it, we can compute the support, the confidence and the χ^2 of the rule we are trying to generate (lines 26-28). If any of the measures does not satisfy the minimum constraints, the rule is not generated (line 29).

In Figure 2.13 we see an example of the CAP-growth algorithm, run on the CAP-tree of Figure 2.11, with minimum support, confidence and χ^2 set respectively to 0.3, 0.51 and 0. In the figure, each node is labeled by the array of frequencies of the classes and the resulting Gini impurity. The root of the tree has a Gini impurity of 0.5. Its first child to be explored stores item A with a Gini of 0.375 (Figure 2.14a). Having a positive IG and a non-null Gini, we continue the descent to its children. The first to be explored describes the pattern A,C (Figure 2.14b). This node has a Gini index of 0.5, thus a negative IG. This means that the addition of item C to the pattern only worsens the ability of A in predicting a label. We therefore do not explore anymore this pattern and its offsprings. The other sibling (Figure 2.14c), storing item D, is pure for the positive class: continuing the descent further would only lengthen the rule without any improvement. We reconstruct the real frequencies of itemset $\{A,D\}$ to see if the rule $A,D \Rightarrow +$ is really worth to generate and compute its support, confidence, and χ^2. First, we need to project the CAP-tree for item D. The header table stores the pointers to the three nodes that store this item. Only the parts of the tree that end to these three nodes are kept, and all the surviving nodes and the header table are updated in their frequency arrays to reflect this change (Figure 2.16a). Now we have a CAP-tree storing only the transactions that contain item D.

We project again for item A. The header table points to a single node that stores this item, and its frequency array, updated in the step before, is [3,0]. By projecting, the CAP-tree reduces to the root node alone, whose frequencies also are updated to [3,0] (Figure 2.16b). This is the frequency array for itemset $\{A,D\}$. Thus, the rule $A,D \Rightarrow +$ has confidence 1 and support 0.5, and satisfies the minimum thresholds[5]. Rule $A \Rightarrow +$ is not generated, as one of the subpatterns of A has already produced one rule. Finally, we move to the second child of the root, storing item C (Figure 2.14d). This is again a pure node. We recollect the frequencies of item C by projection as seen before and get the array [1,3], which produces the rule $C \Rightarrow -$ with support 0.5 and confidence 0.75. The final model is made of only two rules.

It is worth paralleling the strategy in CAP-growth with the one in the database coverage pruning [10]. The database coverage scans the rules extracted and already sorted by prediction quality, and keeps on adding them to the model if they predict correctly at least a transaction not yet covered, and until all the transactions have been covered at least once. Similarly, CAP-growth keeps on adding rules that cover transactions not yet covered, since they are extracted in different branches of the CAP-tree, and does so without extracting the entire set of CARs that satisfy the minimum thresholds. The main difference between the two strategies is in the moment when the pruning is performed: the database coverage acts at the end of the extraction, when all the rules have been already extracted, whereas CAP-growth anticipates the pruning in the extraction phase. The aim of both strategies is the same, that is generating the least, shortest rules, avoiding redundancy in the model.

Model consolidation

CAP-growth generates a single model, in each partition of the dataset, that is at the same time compact and useful. Still, with massively large datasets, it may happen that the number of partitions to have a sufficient division of the workload is in the order of thousands, or more. Consequently, the number of single models in the ensemble explodes. This results in a larger model to store, more complex to be read and examined by a human, and with longer execution times when applied to predict new records.

[5]The minimum χ^2 is set to 0 in this example.

To cope with these issues, we shrink the ensemble of the models to a unique model. This is done by merging the models, combining rules with identical antecedent and consequent into a single, new rule. The new rule will need to have an approximation for its support, confidence and χ^2, as it is too expensive to reconstruct the exact ones in this phase. In other words, we anticipate part of the voting that eventually classifies new records to this phase: establishing how two identical rules collapse to a single one is establishing how they would eventually vote in the classification, a priori. Algorithm 2.3 shows how the consolidation is done.

Algorithm 2.3: Model consolidation

Input : A list of models - *models*
Output : A single model, as a list of CARs

```
1  model = ∅
2  for each m in models do
3  |  model = merge(model, m)
4  end
5  return model
6  Function merge(model m1, model m2)
7  |  m = new model
8  |  rules = m1.rules ∪ m2.rules
9  |  gr = group rules by same antecedent and consequent
10 |  for each group of rules i in gr do
11 |  |  m = m ∪ aggregate(i)
12 |  end
13 |  return m
14 Function aggregate(rules)
15 |  rule = new rule
16 |  r = rules.first
17 |  (rule.antecedent, rule.consequent) = (r.antecedent, r.consequent)
18 |  supports = ⋃_{r∈rules} r.support
19 |  confs = ⋃_{r∈rules} r.confidence
20 |  chis = ⋃_{r∈rules} r.chi2
21 |  (rule.support, rule.confidence, rule.chi2) = g(supports, confs, chis)
22 |  return rule
23 Function g(supports, confs, chis)
24 |  return (max(supports), max(confs), max(chis))
```

We recall that DAC 's training has split the dataset in N partitions and runs a CAP-growth over each partition, thus generating an ensemble of N models. These

models are the input for the model consolidation algorithm (Algorithm 2.3). We reduce the models by applying, recursively two by two, a function *merge* (line 3). This function simply makes the union of the rules in the two models (line 8) and, for each set of identical (in the antecedent and consequent) rules found, applies function *aggregate* (line 11).

Function *aggregate* returns a new rule by choosing the new support, confidence and χ^2 with *g()* (line 21), which actually sets the strategy for the consolidation. Function *g()* must have the properties of associativity and commutativity. Associativity and commutativity in $g()$ make the consolidation runnable in parallel. The default behavior of $g()$ is returning the maximum of the supports, confidences and χ^2 in input, as an upper bound estimation (line 24). We have also experimented with other possibilities, namely the minimum and the product. In Section 2.4.2 we give details on these experiments.

Voting

In associative classifiers, the models usually label a record by applying the first matching rule based on a quality ranking. Differently from other families of classifiers, associative classifiers usually do not have a score or a vector of probabilities for the prediction, but only the predicted class. Introducing a score for the prediction of the associative classifier, the predictions can express their strength in a continuous domain and we can use measures different from the accuracy to compare the model with others, like the Area Under Curve. Moreover, we have a way to weigh the votes in the ensemble, whereas in its simplest implementation every model would have voted with an equal weight, independently of the confidence or support of the rules of each model. This last point is indeed partially covered by the consolidation, but we can still hold in the consolidated model rules, with different antecedents or consequents, that come from different models and contemporarily match a record. Defining a score would mean defining how these many rules contribute to strengthen our belief in predicting a class, when they all agree, or to mitigate our certainty, when they partially disagree.

Given an unlabeled record, for each label i, we define a score s_i as a function of some measure for all rules matching the transaction, i.e.

$$s_i = f(m(\vec{r}_i)), \qquad \forall i : \vec{r}_i \neq \emptyset$$

where $\vec{r}_i$ is the array of matching rules for label i, $m()$ is a measure, e.g. the support or the confidence, and $f()$ a function with domain in $[0,1]$. If there are no matching rules for a label and the transaction, s_i is defined as

$$s_i = s_X/|X|, \qquad \forall i \in X$$

where X is set of labels for which we do not have a matching rule, and s_X is defined under a naive assumption of independence as

$$s_X = \prod_{j:\vec{r}_j \neq \emptyset} (1 - s_j), \qquad X = \{i : \vec{r}_i = \emptyset\}$$

If there are no matching rules at all, s_i is default to the probability of each label i in the original dataset. The score vector $\vec{s}$, containing the scores s_i as above defined, is finally normalized to sum to one.

The default setting for $m()$ is the confidence, that is a common choice in associative classifiers for the rules' ranking. In preliminary experiments, we tried several alternative choices for $m()$, i.e. the support, its complement $(1 - sup)$ and the χ^2. We performed further experiments on the two most promising of these, that is the confidence and $1 - sup$, which we report in Section 2.4.2.

The default setting for $f()$ is the $max()$ function, which is an upper bound estimation of the quality of the rule, based on the measures from the models where it was found. Alternatives to this choice are, for example, the minimum or the mean, which are always valid scores whenever $m(\vec{r}_i)$ is defined between 0 and 1. We test and discuss these alternatives in Section 2.4.2.

2.4.2 Experimental evaluation

In our experimental evaluation, we want to compare DAC with state-of-art approaches in a realistic, large-scale scenario. Among publicly available datasets, we found only one dataset to be very large (i.e. over the Terabyte) and with the characteristics of our problem (i.e. many categorical features), and is described below. As competitors to DAC, we choose the algorithms implemented in the Apache Spark Mllib library [22], as it is a well-proven framework for machine learning on distributed computing [34, 35].

The experiments were performed on a cluster with 30 worker nodes running Cloudera Distribution of Apache Hadoop (CDH5.8.2), which comes with Spark 1.6.0. The cluster has 2TB of RAM, 324 cores, and 773TB of secondary memory. Unless differently specified, all the single experiments are run on 100 executors and a master node with one virtual core and 7GB of RAM each. We used version 1.6.0 of Apache Spark Mllib[6] and version 2.1 of DAC, which is released as open source[7].

The dataset used in the experiments is the Criteo dataset [36], which has already been used as a benchmark in classification tasks, although only on its continuous features, in [37]. The dataset counts more than 4 billion records, describing the behavior of users in 24 consecutive days towards web ads. The positive class is a click on the showed ad and the negative is a non-click. The records are described by 13 continuous features and 26 categorical features, whose semantics is not disclosed. For the experiments, we selected the categorical features only, as DAC does not handle continuous features without a discretization phase, which is outside the scope of this evaluation. The resulting dataset contains more than 800 million unique items, each appearing once or more, and is larger than 1.2 TB. The negative class appears 97% of the times.

The dataset is characterized by the presence of categorical features and the extreme imbalance of the classes. In the next paragraphs, we will describe our approach toward the two issues.

Managing categorical features. To deal with categorical features, we need either an algorithm that supports them natively, like DAC, or a proper encoding of the features into integer or binary values. A common solution, which would enable the exploitation of many widely-used classification algorithms, like SVMs or artificial neural networks, is to use the so-called "one-hot" encoding. With it, all the distinct items appearing in the dataset are transformed to a binary feature, which represents the presence or absence of the value in the record. With all the categorical features mapped to binary ones, we would be able to try many solutions for classification.

We tried one-hot encoding as implemented in Mllib. Unfortunately, with so many unique values (more than 800 million) the preprocessing quickly grows in memory and fails. A possible reason is the fact that the records are stored in a

[6]https://spark.apache.org/docs/1.6.0/mllib-guide.html
[7]https://gitlab.com/dbdmg/dac/tags/v2.1

dense vector. Since with this encoding only a few features would be non-zero, we tried to implement the encoding with a sparse matrix, but the dimensions involved (billions of records by billions of features) showed to be too large also for this kind of representation, and our attempts exhausted the memory available to our testbed.

A different approach is selecting an algorithm that supports natively categorical features without a special encoding, like decision trees or random forests. Again, the number of distinct values in each feature is an issue, due to the metadata that these algorithms need to collect and store to decide the binnings and the splits at each iteration. Not surprisingly, all preliminary experiments again failed for out-of-memory errors. We decided therefore to exploit a technique known as "hashing trick" [38]. With this method, all values are hashed to reduce dimensionality, with inevitable collisions. We therefore progressively reduced the domain of each feature down to 100000 categories, value that allowed the execution of the random forest algorithm without memory issues. After the reduction of dimensionality, the application of one-hot encoding was still impossible. This therefore excludes from our analysis the Mllib implementations of linear SVM, logistic regression and multilayer perceptron.

Dealing with class imbalance. Preliminary experiments showed that neither Random Forests nor DAC were able to handle the highly unbalanced distribution of classes in this dataset. Indeed, the resulting models were respectively trees with all the leaf nodes predicting the majority class and sets of CARs where the minority class was highly underrepresented, when not absent. To cope with this issue, we investigated several techniques, among which instance-based weighting, oversampling, and subsampling. Instance-based weighting assigns a given weight w to each sample, that while building the model is thus counted as if present w times. In the decision tree and the random forest, this weight affects the sample counts of each node and the split decisions. When the weight w is equal to the inverse of the frequencies of the sample's class, this technique can balance the dataset without a physical replication of the records. Although implemented in several popular random forest implementations [39, 40], instance-based weighting is not implemented in Mllib. Oversampling replicates some of the records belonging to the minority class or classes, so that the dataset gets balanced [41]. In our scenario, the application of this technique to the training set did not converge successfully due to memory constraints. Conversely, subsampling extracts a fraction of the majority class or classes, to reduce their volume to a size comparable to the minority class [41]. We

applied this technique to the negative class to have a cardinality roughly equal to the positive one, in the training set. The test set was not subsampled.

In these preliminary evaluations we have also tried several settings of the general architecture of DAC. In this phase, we found that sampling with replacement yields a better load balancing, as this operation triggers the shuffling of part of the training dataset and leads to equally-sized partitions, whereas the default partition can see blocks of very different sizes. We have set α, the sampling size for each one of the N models, to $1/N$, to have a final training dataset sized as the original one. We have also tried several N, finding in 100 for each partition a value that allowed the CAP-tree of each model to be stored in memory.

Summarizing, our competitor to DAC will be a Random Forest with categorical features, with the values of the categories hashed down to 100000 different values at most[8]. DAC will be instead evaluated without hashing trick, as it is not necessary. For both, we will subsample the majority class in the training set.

Experimental comparison of DAC and Random Forest

In this section, we evaluate the quality and the performance of DAC and a Random Forest. Our objective is to show how our proposed technique can manage a dataset characterized by a very large volume and domain, and compare the quality of the resulting model with the state of the art. We evaluate our results in the binary tasks with the AUROC, i.e. the area under the ROC curve [42]. DAC will be evaluated with its default settings, i.e. $f() = max$, $m() = conf$ and $g() = max$. The experiments are run with a 5-fold cross-validation on the whole dataset, with the K-fold function implemented in the MLUtils of Mllib. Each one had a variable duration on our testbed between 2 and 30 hours. All the confidence intervals shown in the plots were computed using a t-student distribution at 95% confidence. After subsampling, each training fold sees an average of 111 million distinct items[9].

Figure 2.17 shows the resulting AUROC for the candidate set of models, consisting of i) DAC with $f() = max$, $g() = max$ and $m() =$ confidence, varying minimum support thresholds from 0.01 to 0.0002; ii) Random Forests with a depth of 4, varying

[8]Hashing to larger values or not using hashing was not a viable option for the memory issues explained before.

[9]This statistics is computed in a separate experiment, and does not thus affect the execution times shown in the remainder of the section.

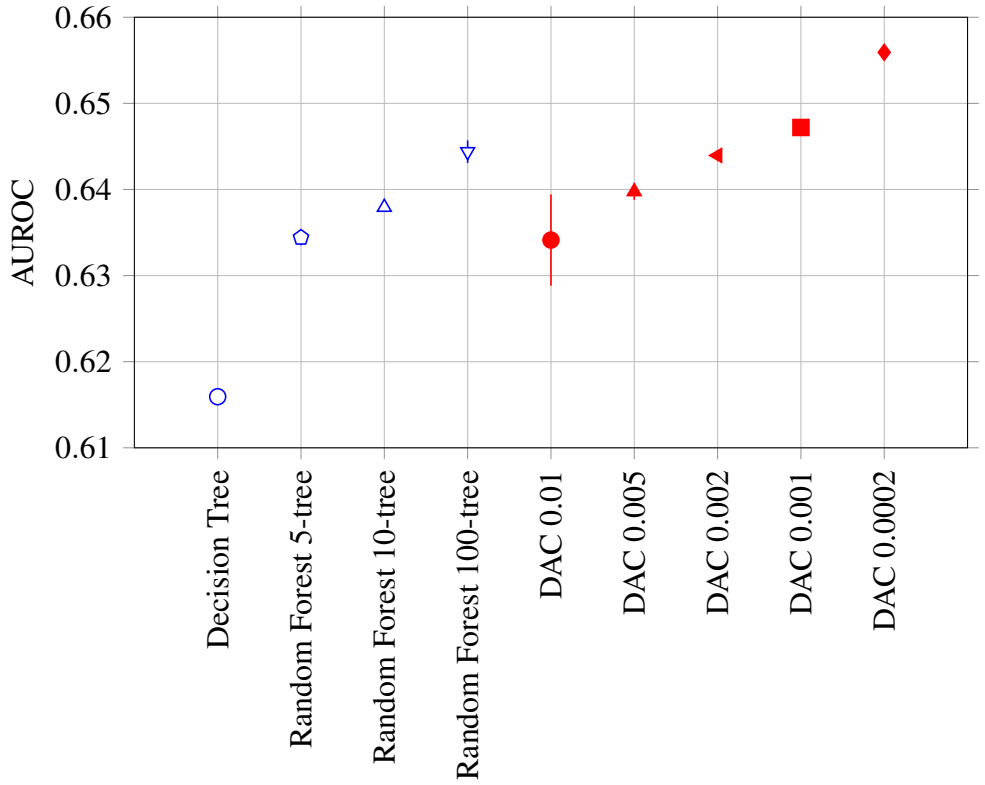
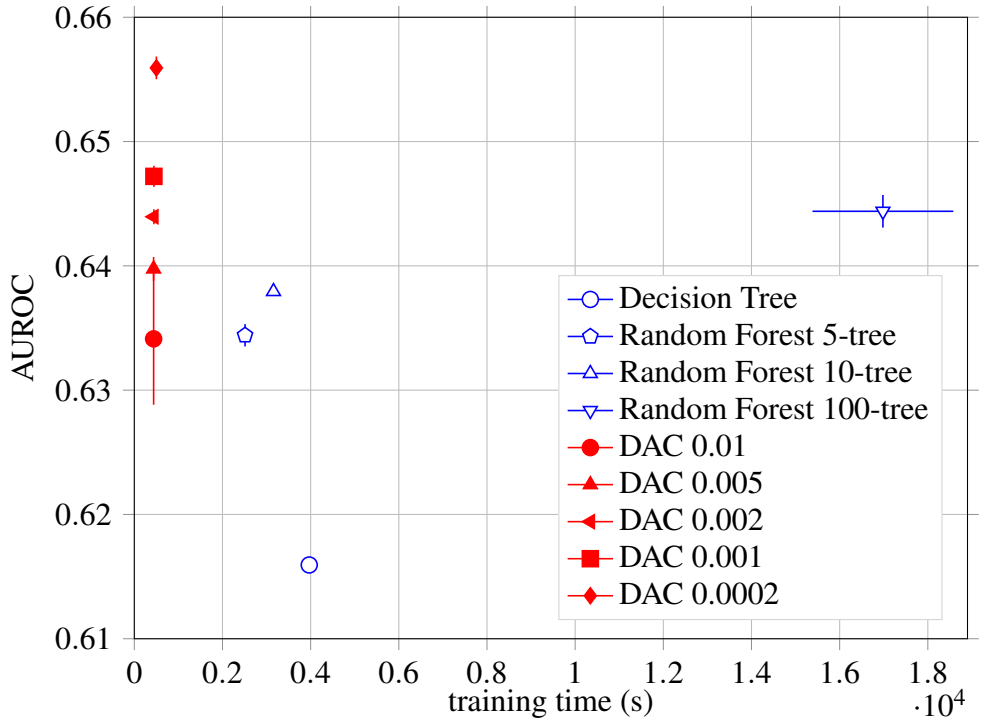


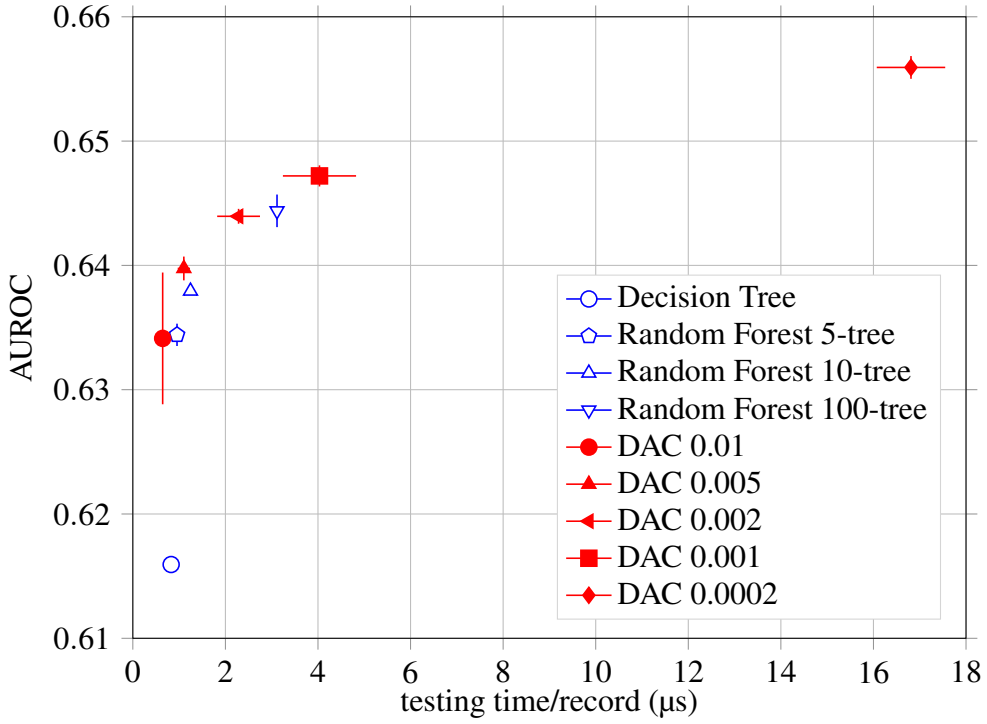
Fig. 2.17 DAC vs Random Forest. AUROC cross-validated with 5-fold on the entire dataset

the number of trees from 5 to 100; iii) a single Decision Tree, with depth 4. The baseline for the results is set by the Decision Tree, which is almost two points below the Random Forest and DAC. The quite large confidence interval for DAC with 0.01 as minimum support makes uncertain the comparison with the two smallest forests, with 5 and 10 models respectively. These last three models are all clearly below the results of the 100-tree forest and DAC with minimum support 0.002, that have a comparable AUROC of 0.644. Significantly better are the results of DAC with minimum supports of 0.001 and 0.0002, this last one scoring the highest AUROC of 0.655, a good point above the 100-tree forest.

Notably, the cross-validation experiments for the 100-tree forest lasted 30 hours in our testbed, against the 20 hours of the DAC with minimum support of 0.0002. These high computation costs would certainly be a heavy factor in the choice of a model, as the model with the highest score is not always a viable path. We therefore plot the same scores against the training and testing time of their models, in Figure 2.18. In Figure 2.19a we see how the training times of the Random Forest grow with



(a)



(b)

Fig. 2.18 DAC vs Random Forest. AUROC vs training and testing times, cross-validated with 5-fold on the entire dataset

the number of models. The Decision Tree shows times higher than both the 5-tree and 10-tree forests, as it does not perform any feature selection, whereas the forests randomly choose $\sqrt{d}$ features for each tree, where d is number of columns, i.e. 26 in our scenario. The half-point advantage in the AUROC of the 100-tree forest on the 10-tree one comes with a cost five times higher in terms of training times. This large gap could make the difference in a scenario where the model needs to be frequently updated, e.g. an application with nightly updates to the training dataset, and could lead to the choice of the shallower model. DAC here demonstrates a highly desirable behavior, as the best model trains in only 500 seconds, a time respectively 5 and 25 times smaller than the 5-tree and the 100-tree forests, which also have a worse AUROC. Moreover, the gap between the training times of the least and most accurate models for DAC is under the 15%, so the latter one is clearly preferable.

The testing times of DAC and the Random Forest have similar trends, depicted in Figure 2.19b. Both appear to grow exponentially with the AUROC reached, symptom of models that are more and more complex. For the Random Forest, this complexity is proportional to the total number of splits, or equivalently to the number of trees, since we have a fixed depth. This justifies the alignment of the Decision Tree to the trend of the Random Forests, as in this phase it is practically identical to a Random Forest with one tree. For DAC, the complexity depends on the number of rules extracted and, in this strategy that applies *max* for both $f()$ and $g()$, on the position of the first matching rules in the model for each class, for we need only these for the score. This explains the slightly larger confidence interval on the time axes, whereas the forests all have negligible intervals, due to the constant number of splits traversed by each record for a prediction. Despite this, we can still safely affirm that DAC reaches the same quality of a Random Forest within smaller testing times, as it happens with DAC with minimum support 0.002 and the 100-tree forest. At the same time, we can say that, given a comparable testing time, DAC performs better, as in the case with minimum support 0.005 and the 10-tree forest.

Evaluation of DAC parameters

We tested the effect of the choice of the algorithms' parameters on the quality of the model, to eventually select one or several candidates for more thorough tests.

For DAC, we evaluated different choices for:

- the use of the database coverage technique, [yes/no]
- the function used in the voting, $f()$, [*max*/*min*/*mean*]
- the measure used in $f()$, $m()$, [$conf/1-sup$]
- the function used in the model consolidation phase, $g()$, [*max*/*min*/*product*]
- the minimum support threshold, [9 values from 5% to 0.01%]

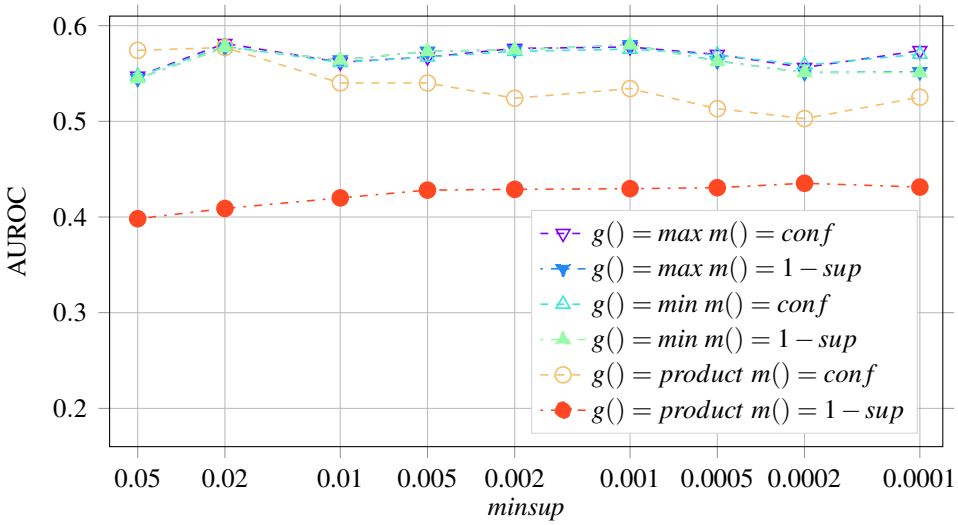
for a total of 324 runs, considering all the combinations of values. $f()$ was chosen among *max*, *min* and *mean*. We tested two values for measure $m()$, that is the confidence of the matching rules, which is a common choice in associative classifiers, and $1-sup$, following the intuition that a rule (a set of words) is the better in labeling the more is rare [43]. $g()$ was chosen among *min*, *max* and *product*, three functions that have the properties of associativity and commutativity, which are important for the distribution of the workload. We tried nine different values for the minimum support threshold, from 5% to 0.01%. The database coverage was either used or not. The minimum confidence has been set to 50%, for the rationale that any rule better than random guessing should positively contribute to the quality of the labeling. The minimum χ^2 was set to 3.841, corresponding to a p-value of 0.05 for the statistics.

We ran this session of experiments on the day 0 of the dataset, which is a 24th of the whole dataset, and without cross-validation, keeping 30% of the dataset out for testing. This reduced the execution time by more than two orders of magnitude, allowing us to test a larger selection of parameter values within some days of execution.

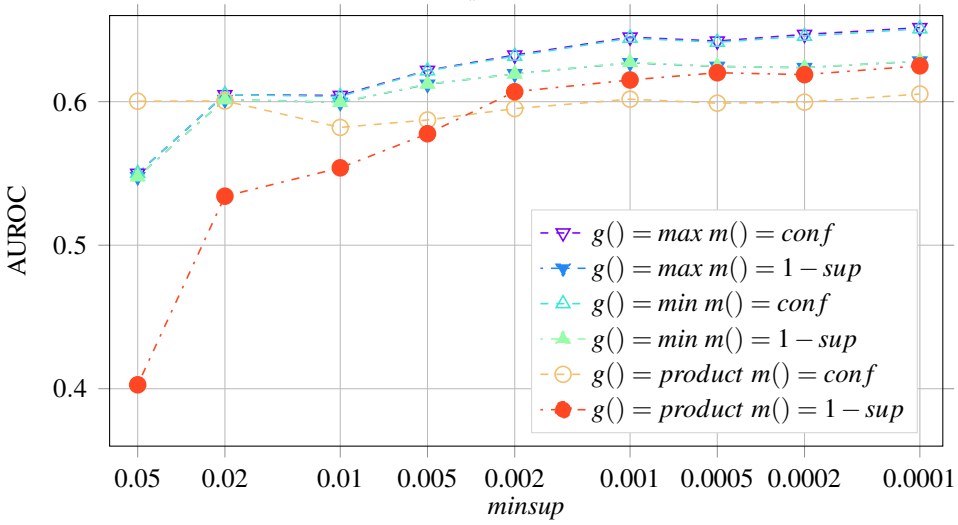
Database coverage. The first, immediate finding was on the use of the database coverage, which did not show effects on the quality of the model trained. The amount of rules pruned by this technique has been constantly below 5%. Thus, CAP-growth is effective in selecting useful rules with limited overlapping. For example, with a minimum support of 0.1%, the number of rules of the model produced by DAC was 339, reduced to 328 with the database coverage. The training time with this technique grew instead with the number of rules, motivating us not to use it in the following experiments.

Figure 2.20 shows the results of the runs without the database coverage.

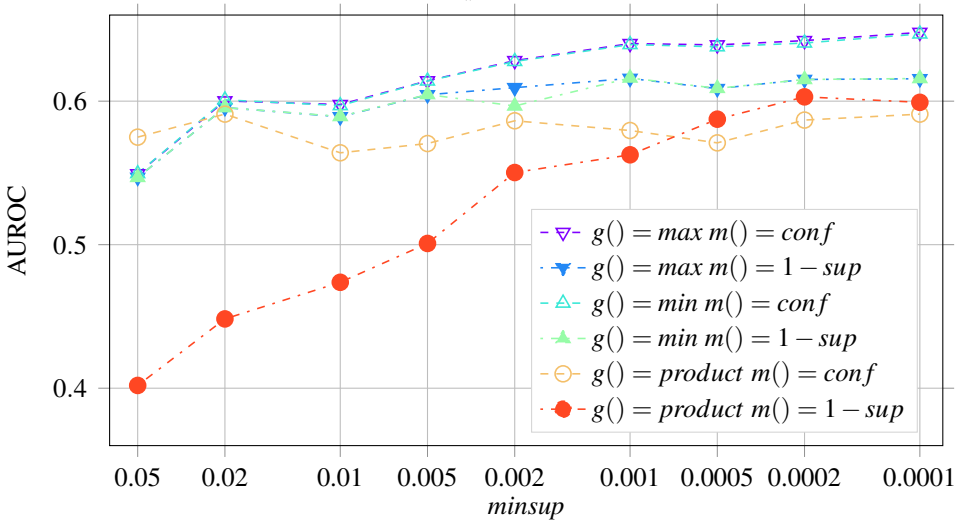
Function $f()$. Choosing *min* as $f()$ (Figure 2.21a) is comparable with other options only with shallow models (*minsup* 5%). With this support, the number



(a) $f() = min$



(b) $f() = max$



(c) $f() = mean$

Fig. 2.20 DAC tuning. Comparison of different choices for $f()$, $g()$, $m()$ and *minsup*

of rules extracted (6) is so little that $f()$ rarely affects the voting. Decreasing the support, *min* does not show improvements, as only more confident and rare rules are being added to the models. Thus, the minimum does not change. Both $f() = mean$ (Figure 2.21c) and $f() = max$ (Figure 2.21b), instead, improve their performance with a similar rate, with the top performers almost overlapping, and the top AUROC for *max* standing 0.4% above the one for *mean*.

Measure $m()$. Preliminary experiments already led us not to choose the support itself and the χ^2 for $m()$. Confidence proves to be the best choice. Against the trend is the case where $g()$, in the consolidation function, is set to be the product of the measures. In this case $1 - sup$ is the better choice for $m()$, reaching an AUROC of 0.625 with *max* as $f()$, ranking third among all experiments but still two points below the best scenario.

Function $g()$. As for what concerns $g()$, the function applied to two identical rules in the consolidation phase to choose the new confidence, support and χ^2, choosing either *min* or *max* is identical in this set of experiments. Choosing *product*, instead, shows contrasting outcomes. Together with the confidence as m, it never shows improvements with lower supports, reaching at most an AUROC of 60%. With $f()$ set to *min* (Figure 2.21a), it has the worst quality among all the combinations, often below the AUROC of a random choice (50%). With $f()$ set to *max* (Figure 2.21b) and $1 - sup$ as $m()$, instead, as said above, it reaches the first quartiles of the results and is able to equal the AUROC of the alternatives at the lowest support.

Minimum support. With varying minimum support thresholds, from 0.02 to 0.0001, the best performing solution is stably with $f() = max$ (Figure 2.21b) and $m =$ confidence, and indifferently *max* or *min* as $g()$. This, with an arbitrary choice of $g = max()$, is the solution we tested on the whole dataset and compared with the state of the art.

Model selection for Random Forest

With a Random Forest, the parameters that would affect the quality and the performance of the resulting model are mainly two, the number of trees and their depth. Similarly to the previous section, we run some preliminary tests to evaluate different choices for these parameters, on the same portion of the dataset, again without cross-validation.

We evaluated the AUROC for depths of 4, 8 and 16, starting with 10 trees and increasing their number until possible.

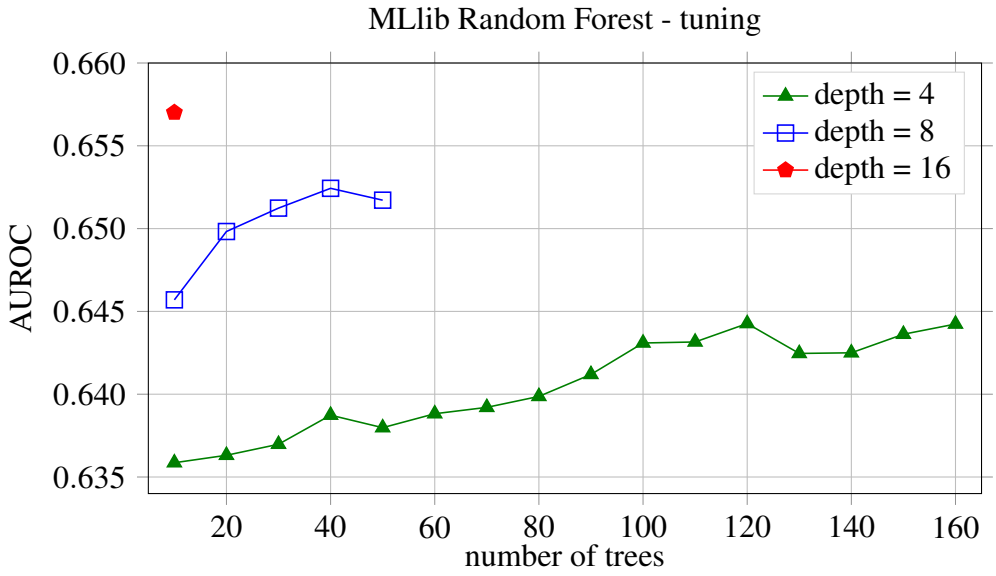


Fig. 2.22 Random Forest tuning. Performance (AUROC) with different parameter settings

Figure 2.22 shows the results. With depth 4, the quality of the classification improves steadily until reaching a plateau after 100 trees. The execution with 170 trees repeatedly failed, raising an `OutOfMemoryError` on our testbed. With depth 8 and 10 trees, the quality improves of a not negligible point over the shallower version, and the gap still augments with more trees. Unfortunately, the `OutOfMemoryError` appears even faster, with only 60 trees. Finally, the only execution attempted with depth 16 scored 65.7%. This result was obtained in an experiment lasting more than 17 hours, which would become, assuming linear scalability, more than 100 days for the tests with the complete dataset. Similarly, building any forest with depth 8 has an unfeasible expected duration. The solutions we tested on the whole dataset were thus focused on the forests with depth 4.

Experimental evalutation of DAC on small and medium sized datasets

In order to compare our work with previous works, we have evaluated a local, single-model version of DAC on a number of medium-size datasets from the UCI repository, on which results for other associative classifiers were available. For CBA [10], we used the results published in [27]. The experiments, performed with a 5-fold cross-

	CBA	DAC
austrad	0,8550	**0,8561**
breastd	0,9530	**0,9544**
cleved	0,7720	**0,8052**
crxd	**0,8590**	0,8393
diabetesd	0,7290	**0,7478**
germand	**0,7320**	0,7000
heartd	**0,8190**	0,7742
horsed	0,8210	**0,8223**
hypod	**0,9840**	0,9527
ionod	**0,9210**	0,8994
labord	0,8300	**0,8802**
pimad	0,7310	**0,7451**
sickd	**0,9730**	0,9392
sonard	**0,7830**	0,7440

Table 2.8 Single-instance DAC vs CBA, average accuracy on binary-labeled UCI datasets

	BAC	DAC
censusd	0.8286	**0.8315**
nursery	**0.9255**	0.9243
yeastd	0.5573	**0.5678**

Table 2.9 DAC vs BAC, average accuracy on selected UCI datasets

validation, showed that DAC performs similarly to CBA, reaching higher accuracies as often as not, as shown in Table 2.8. Moreover, DAC reaches these results with a significantly lower number of rules, without any posterior pruning. This sets the single-model DAC as a good choice for a base model in an ensemble, where usually shallow models are preferred as baseline models.

We also evaluated the quality of DAC on the three datasets on which we made our experimental evaluation of BAC, in Section 2.3.3. We set for both the algorithms a minimum support of 1%, a sample size of 10% and 10 models for the ensemble. The results, shown in Table 2.9, demonstrate that DAC reaches a good accuracy also on these datasets of limited size, with scores that are better or similar to the ones of BAC. Moreover, this good quality is reached with a significantly less sizeable model, as shown in Table 2.10, which depicts the average number of rules of the models generated in the cross-validation.

	BAC	DAC
censusd	3236	352
nursery	12057	1212
yeastd	33051	3351

Table 2.10 DAC vs BAC, average number of rules on selected UCI datasets

2.5 Related work

Associative classifiers exist in a number of fashions, and a precise taxonomy has been already made in [11]. Among all, we can distinguish classifiers exploiting CARs (Class Association Rules), as introduced in [10], and others exploiting EPs (Emerging Patterns), like [44]. Our approach falls in the first category, together with works like [32, 33, 45–52]. Since its introduction in [10], the database coverage technique has been exploited with success by many classifiers, e.g. [32, 33, 48, 49], and several others have also exploited similar techniques, e.g. [32, 44, 53]. One of these is the redundant rule pruning, which scans again the set of rules found to delete the extensions of a rule that follow the rule itself, and that therefore are never applied [32]. These techniques have proved to be very effective in the reduction of the model and the improvement of the quality of the classifier. However, the amount of rules that are first extracted and then reordered is often enormous, demanding proportionate resources both in terms of memory and CPU. We argue that, in order to scale to very large dimensions and effectively exploit the potentials of a MapReduce-like framework, an effective associative classifier should aim at reducing, if not eliminating, the contribution of these techniques to the reduction of the model size, and focus on the extraction of a small, good quality subset of the rules.

The increase of the overall accuracy of the predictions is also addressed by means of ensemble techniques [28]. The authors of [54] analyzed the impact of the boosting ensemble technique when a set of associative classifiers are used as building block. However, the impact of the bagging ensemble approach on associative classifiers has never been analyzed. Differently from [54], in this work we perform this analysis by proposing a bagging version of an associative classifier.

An attempt to bring the training of an associative classifier onto a framework for parallel computing and scale to large datasets has been done also in [50]. The authors of [50] proposed a MapReduce solution based on a parallel implementation of FP-growth [55], modified to extract CARs, followed by two pruning phases that

are slight variants of the database coverage and the above-mentioned rule pruning. This solution was implemented and tested in the Hadoop framework. The work follows the strategy of generating the complete set of CARs and prune in a second phase, similarly to BAC. We could not attempt a direct comparison with [50] as their code is not publicly available. Furthermore, most of the datasets used in their experiments are characterized by continuous features. Thus, the application domain is much different from the one of BAC and DAC, that are designed to work on large-scale and large-domain categorical datasets.

Several works have already explored the possibility of combining more than one rule for prediction, thus defining weights akin to a score for each class. The authors of [32] have proposed to use the top K rules that match and weigh their vote with a weighted χ^2-analysis. In [46], the top rule for each class is first determined, then the prediction is made on the one that maximizes the Laplace accuracy. [52] has proposed a weighted-voting based on some metrics, e.g. support, confidence and conviction of the rule. [47] sets as score the sum of the confidences for the matching rules. All these techniques have been used selecting as label the class that maximizes the defined score. The majority of associative classifiers, though, does not use a score and predicts the label with the first rule that matches the record [33, 45, 48–50].

2.6 Summary

In this chapter, we have proposed an approach to scale an associative classifier on very large datasets.

A survey of several techniques to distribute a very similar task, frequent itemset mining, showed the advantages and the limitations of the two main approaches, a data split approach and a search space split approach (Section 2.2). From this, we built our approach to scaling associative classification on the good load balancing of the data split approach, while aiming at replicating for associative classification the effective data structures proposed by a search space split approach, FP-growth.

A naive way to implement the data split approach in associative classification is by creating an ensemble of state-of-art classifiers, through bagging. In Section 2.3 we have seen as this approach can effectively reach the same quality of a single-machine implementation and distribute the workload on a cluster of machines.

For our Distributed Associative Classifier (DAC) we considered an in-memory cluster-computing architecture, Apache Spark (Section 2.4). In this architecture, the large availability of memory is heavily exploited to streamline the computation, avoiding disk access whenever possible, allowing an extremely faster sequential processing and caching of the intermediate results. This scheme, and the available memory, have of course their limits. In preliminary experiments, we identified the major issue of designing an associative classifier in this framework in the large number of extracted rules, which are only eventually pruned in the training phase. Therefore, we have anticipated all the pruning into a novel extraction algorithm, CAP-growth. DAC trains an ensemble model by means of bagging, which eases the distribution of the computation. Each model is generated by an instance of CAP-growth. A final consolidation phase for the models of the ensemble and a new voting strategy help further reduce the size of the model and improve the quality of the predictions.

To validate our approach, we have performed experiments in a real large-scale scenario, a binary-labeled dataset with more than 4 billion records, 800 million distinct values in its categorical features and larger than 1.2 TB in storage. The pruning done in CAP-growth has proved to be effective. When executing database coverage pruning as a final step, a negligible fraction of rules are pruned by this technique, always below 5%, without improvements in quality. DAC demonstrated better performance than a state-of-the-art technique, a Random Forest, both in terms of quality of the prediction and execution time. The best setting for DAC improves the AUROC upon the best for the Random Forest by 1% with a total training time that is 25 times smaller.

The DAC classifier, differently from a Random Forest, generates a readable model. The "hashing trick", which allows the Random Forest to deal with a large number of distinct values in the categoric fields, has the major drawback of making the model unintelligible by a human. This hampers the usability of the model for decision-making and makes also extremely difficult its debugging. DAC did not require hashing, though larger scales, i.e. billions or trillions of distinct values, might make it necessary. In this scenario, the model produced by DAC, without hashing, is made of rules containing the items exactly as they appear in the dataset, with all their semantics left intact. We believe this feature to be highly valuable for a classification model.

Future works will experiment different model generation strategies. For example, we will introduce a projection by column in the ensemble like the one implemented in Random Forests.

2.7 Relevant publications

[3] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Paolo Garza, Fabio Pulvirenti, and Luca Venturini. Frequent itemsets mining for big data: A comparative analysis. *Big Data Research*, 9:67 – 83, 2017

[4] Luca Venturini, Paolo Garza, and Daniele Apiletti. Bac: A bagged associative classifier for big data frameworks. In *East European Conference on Advances in Databases and Information Systems*, pages 137–146. Springer, 2016

[5] Luca Venturini, Elena Maria Baralis, and Paolo Garza. Scaling associative classification for very large datasets. *JOURNAL OF BIG DATA*, 4(1):1–24, 2017

Chapter 3

Building a Big Data machine learning pipeline

Inspecting the performance of a network from a massively big traffic logs' dataset is a difficult task. Big data frameworks provide scalable solutions to mine information from raw data, but frequently need a complicate fine-tuning and a thorough understanding of machine learning algorithms. To streamline this process, we propose SeLINA (Self-Learning Insightful Network Analyzer), a generic, self-tuning, simple tool to extract knowledge from network traffic measurements. SeLINA encompasses several data analytics routines adding self-learning abilities to state-of-the-art scalable approaches, in concert with parameter auto-selection to discharge the network analyst from the tuning of the parameters. We incorporate both unsupervised and supervised procedures to mine data and scale up to the size of the task. SeLINA includes instruments to automatically test if the new data fits the model, to discover changes in the traffic, and to trigger the rebuilding of the model.

The result is a methodology that provides human-readable models of the data with limited user mediation, aiding domain experts in pointing up interesting insights and retrieving actionable knowledge. SeLINAis currently implemented on Apache Spark. We tested it on large datasets consisting of real-world passive network measurements from a nationwide ISP, investigating YouTube and P2P traffic. The experimental results confirm the ability of SeLINA to provide insights and detect changes in the data that suggest further analyses.

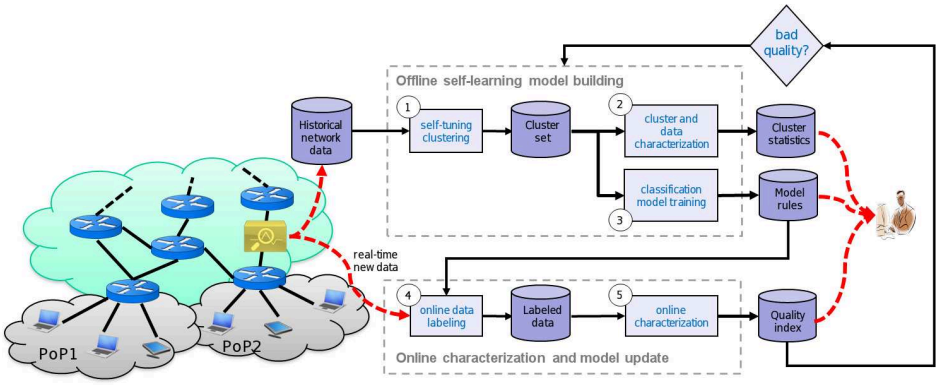


Fig. 3.1 SeLINA building blocks.

The contents of this chapter were originally published in [56, 57]. This chapter is organized as follows. Section 3.1 introduces the problem we aim to solve. Section 3.2 provides an overview of the proposed methodology, while Sections 3.3-3.4 describe its main building blocks. Section 3.5 provides an overview of the experimental evaluation campaign, while Sections 3.6-3.7 thoroughly discuss the experiments performed on two real use cases based on real traffic datasets. Finally, Section 3.8 compares our approach with previous work, while Section 3.9 draws conclusions and presents future developments of this work.

3.1 Introduction

Internet monitoring is paramount in network management, in order to understand how the network behaves, how the nodes access contents, and how to correctly administer and supervise the infrastructure. Network managers and analysts are challenged every day by the growth of traffic, users, services and applications, and need to deal with and understand the growing system complexity. Big data and machine learning methodologies have given birth to approaches that aim at the mechanical extraction of knowledge from the raw data that the monitoring infrastructures collect, and their application to network traffic analysis has been devoted a significant effort. Most of the proposed systems target a specific problem, e.g., monitoring of a CDN [58–60], detecting anomalies [61, 62], or simply offering scalable platforms [63].

However, few works have targeted the general-purpose extraction of useful information from the raw data exposed by the system, i.e., the application of the data

mining approach to information discovery, a classic application of unsupervised machine learning approaches. While methodologies exist, to the best of our knowledge, they require non-trivial skills and the domain expert needs to be able to fine tune the underlying algorithms. In this work, we target the design of an unsupervised machine learning tool that allows the network administrator to discover properties of the traffic, without requiring her to be a machine learning expert. We identified the following requirements.

- Scalability, as the ability to (i) process very large datasets, but (ii) provide compact representations of the traffic, independently of the data size.
- Auto-configuration, as the capability to (i) self-adapt to different data distributions (e.g., data densities, cluster shapes), and to (ii) self-tune the algorithm parameters to avoid human intervention.
- Human-readability of both results and underlying models, to make the knowledge better exploitable and more actionable.
- Self-assessment and self-evolution, to autonomously evaluate the model quality and trigger a rebuilding when the model fitting to new data degrades.

The above-mentioned design guidelines led to the design of SeLINA (Self-Learning Insightful Network Analyzer), which exploits both supervised and unsupervised data-mining techniques by combining their strengths. Specifically, effective unsupervised approaches are used to autonomously identify clusters of homogeneous traffic flows, thus reducing the granularity of objects to observe from millions of single flows to few tens of clusters, and generating a model of traffic. Human-readable and fast supervised approaches are used then to classify flows on the fly and assign them to clusters, and to offer valuable information about the main characteristics of each class. The system computes internal quality indices to check whether the new data does not fit anymore the historical model, suggesting to the analyst changes in the new network traffic, and, possibly automatically, triggering a new clustering phase to update the model.

SeLINA has been implemented in a state-of-the-art Big Data framework, Apache Spark, and has been applied to two real-world large use cases: a YouTube video streaming dataset and a peer-to-peer traffic dataset. Experimental results showed that SeLINA is able to provide insightful network traffic models, e.g., pinpoint different

groups of YouTube servers with different properties, and suggest the presence of changes in the infrastructure that have caused well-known issues to end-users [60].

3.2 Methodology overview

In this section, we provide an overview on the proposed methodology. The methology assumes that a network flow is described and stored as a *point* with d features, for example network measurement that describe the flow. Figure 3.1 depicts its main components.

Offline self-learning model building. This component, which analyzes historical network traffic flows, aims at building a self-learning data characterization model, and consists of three phases: (1) a self-tuning clustering phase, (2) a cluster and data characterization phase, and (3) a classification model training phase.

Online characterization and model update. This component analyzes new network data in real-time by applying the model built in the previous block to detect changes in the network traffic characterization. It consists of two phases: (4) a real-time data labeling phase, and (5) an online characterization phase.

In details, step (1) of the proposed methodology consists of a self-tuning clustering algorithm, which is run over historical data to discover homogeneous groups of traffic flows without prior knowledge, in a fully autonomous and unsupervised fashion. Effectively applying cluster analysis on real datasets requires the non-trivial choice of algorithm-specific parameters, a typically difficult task for domain experts exploiting data mining techniques. To this aim, SeLINA includes ad-hoc strategies to automatically tune the clustering parameter values. In step (2), the resulting cluster set is then enriched by both general-purpose and domain-specific statistics, whose aim is to support network analysts in understanding the semantics of the identified clusters.

The cluster set is also the input to the model training phase of step (3), where a classification model is built by exploiting clusters as classification labels. The model is able to self-learn how to assign each network flow to the proper cluster. Different classification techniques could be exploited, depending on the preference towards pure performance (e.g., accuracy) or human-readability of the model. A good candidate could be the technique proposed in Section 2.4, as the associative

classifier produces a human readable model with good predictive quality. However, the simultaneous presence of categorical and continuous features lead us to choose another solution, able to manage effectively both. SeLINA choice is a decision tree algorithm, which is among the most popular classification techniques and provides an easily readable model in the form of classification rules. The latter feature supports network analysts in getting more meaningful insights on the reasons for the classifier underlying choices.

The classification model is exploited in the real-time data labeling phase at step (4), where each new network flow is assigned a label. Then, at step (5) the quality index computation is executed, by exploiting different quality indicators to self-assess the model fitting and its results over time. When the quality index falls below a given threshold, the offline model building can be automatically triggered to rebuild a model better fitting the new data, thus providing self-evolutionary features.

SeLINA is a general-purpose methodology which can be easily exploited to analyze large collections of network data (e.g., network traffic headers, network flow characteristics, statistical measurements of traffic flows). As a case study, in the chapter we apply SeLINA to analyze network measurements collected through TSTAT [64].

3.3 Offline self-learning model building

The core of the SeLINA approach is the offline self-learning model building component, which consists of (1) a self-tuning clustering phase, (2) a cluster and data characterization phase, and (3) a classification model training phase. Details on each phase are provided in the following.

3.3.1 Self-tuning clustering phase

SeLINA exploits clustering to autonomously identify homogeneous groups of network traffic flows without prior knowledge. This phase performs a preliminary data normalization step, by means of the standard z-score technique [28], and then a clustering algorithm is applied to the normalized data. Among the many clustering algorithms available to this aim, SeLINA provides an advanced DBScan-based [65]

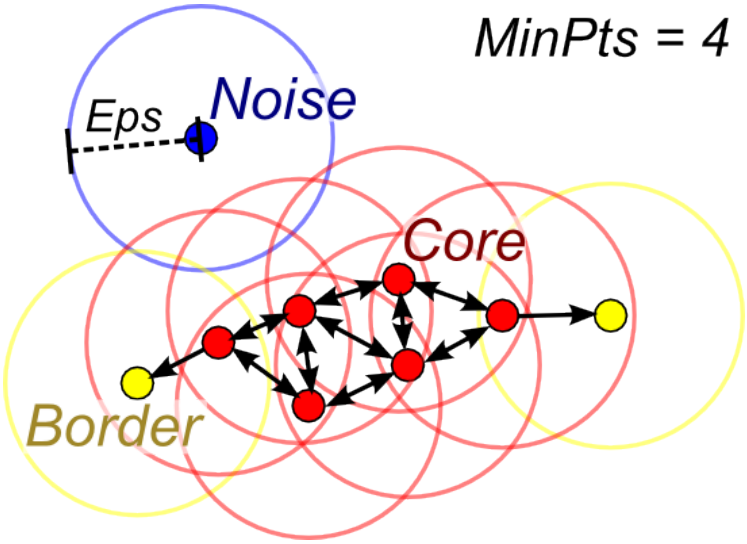


Fig. 3.2 Toy example for DBScan, with *MinPoints* set to 4. The picture highligths the area of radius *Epsilon* around each point, which can divide in *core points*, in red, *border points*, in yellow, and *noise*, in blue. Adapted from [66].

algorithm providing high-quality clusters on very-large real data collections. Since the clustering phase is at the core of the SeLINA self-learning feature, in the following subsections its building blocks are presented.

Basic DBScan

DBScan is a density-based approach that identifies clusters as dense areas of data points surrounded by lower density spaces, whose points are marked as noise. The identification of the dense areas is driven by two parameters: *epsilon* and *MinPoints*. Given an arbitrary point p, the density of the area of radius *epsilon* centered in p is considered, and the points in this area are counted. If the number of points in the area is at least *MinPoints* then p is called core point, the area is considered dense, and it is merged with adjacent dense areas to form a cluster. A point in the area of a core point that is not a core point itself is called border point. The remainder of the points are considered noise. Figure 3.2 shows a toy example where all the three kinds of point are present.

DBScan is a well-known clustering algorithm, fruitfully exploited in a variety of application contexts. Its strength is the ability to identify arbitrary-shaped clusters, and isolate noise and outliers. The results provided by DBScan are usually better

than those provided by other popular clustering algorithms. However DBScan requires longer execution times, due to its quadratic complexity. To scale to very large datasets, we exploit the Spark-based distributed implementation of DBScan[1] proposed by Aliaksei Litouka, whose main difference with respect to the original centralized version is an additional partitioning step, performed at the beginning.

Self-tuning Multi-level DBScan

SeLINA improves the basic DBScan approach by addressing two main issues: (i) parameter setting, and (ii) diverse data densities within the same dataset. To offload domain experts from the critical task of configuring DBScan-specific parameters, SeLINA includes a self-tuning strategy to automatically set proper values. Furthermore, very-large real datasets are often characterized by diverse data distributions in different regions, and this is an issue hardly handled by the standard DBScan version. To address both issues, the SMDBScan algorithm in SeLINA builds upon an advanced version of DBScan successfully proposed in [67]. SMDBScan features a multi-level iterative approach and a smart automatic parameter-setting procedure.

Multi-level iterative approach. At each iteration, SMDBScan considers the data points which have not been assigned to a cluster yet (at the first iteration, the whole dataset is considered), then (i) partitions them to allow parallel computation, (ii) automatically selects the most appropriate values of $epsilon$ and $MinPoints$, and (iii) executes the standard DBScan with such parameter settings. At the end of each iteration, the newly found clusters are included in the global set of clustering results, while the noise points become the dataset for the next iteration.

Automatic parameter setting: $epsilon$**.** To automatically compute the values of $epsilon$ and $MinPoints$ at each iteration, SMDBScan introduces a self-tuning procedure, consisting of two heuristics. The two heuristics are very intertwined, the second depending on the $epsilon$ set by the first, and they are designed to fit the scope of a multi-level strategy.

To determine $epsilon$, SMDBScan exploits the density-based concept of cluster: "a dense area surrounded by a lower density zone". To this aim, a greedy approach is exploited, selecting the best potential $epsilon$ for each point separately. A final decision is then taken globally given all the local best $epsilons$.

[1] Downloaded from https://github.com/alitouka/spark_dbscan

Algorithm 3.1: Automatic setting of the *epsilon* parameter value.

Input : Dataset partitions - *dataPartitions*
Input : Epsilon step - *epsStep*
Output : Estimate of best epsilon - *bestEpsilon*

```
1  List<Double> potentialEpsilons = {};
2  for partition in dataPartitions do
3      for p in partition do
           /* Compute the density of the areas centred in p of
              a radius eps multiple of epsStep                    */
4          Map<Double,Int> densities = {};
5          eps=epsStep;
6          density_before = 0;
7          found = False;
8          while (not found and
            eps <=distanceFromFarthestPoint(p,partition)) do
9              numNeighbours = numNeighboursRadiusEps(p, eps, partition);
10             density = numNeighbours/eps^d ;    /* d is the number of
                attributes */
11             if (density < density_before) then
12                 found = True;
13             end
14             density_before = density
15         end
           /* The potential value of epsilon for p is the one
              before the first density decrease                   */
16         epsilonP = eps;
17         potentialEpsilons.add(epsilonP);
18     end
19 end
   /* Among the potential values of epsilon, select the one
      corresponding to the first quartile                         */
20 bestEpsilon=FirstQuartile(potentialEpsilons);
21 return bestEpsilon;
```

In particular, given an arbitrary point p, the algorithm tries to identify the boundary of the dense area around p. To this aim, it computes the density distribution in the hypersphere having radius R and center in p, with increasing values of R. The larger R, the higher the number of points inside the hypersphere will be. We define dense areas when the number-of-point increasing rate is higher than the growth in volume. At the border of a dense area, this growth rate will show an inversion of the trend, as soon as the volume starts growing faster than the number of points. The proposed heuristics chooses the first inversion point as the border. If many inversion points occur, greedily choosing the first one reduces the computational time and leaves margin for further exploration in the next levels of SMDBScan. The final value of $epsilon$, actually used for each run of DBScan, is selected by considering the first quartile of the set of border values generated by applying the border-detection procedure for all points p. The first quartile value produces a set of dense clusters covering a representative subset of our data: taking the first quartile leads to having at least a quarter of all the points set as core points with high probability, which will help covering a good portion of the dataset in few levels.

The border-detection procedure increases the R value at $epsStep$ increments. This is the only parameter, whose main impact is on the execution time: very small steps lead to many iterations to converge. In our experiments, we found that a value of 10^{-3} was reasonable for the hardware at our disposal.

The pseudo-code for the $epsilon$ self-tuning is reported in Algorithm 3.1. Since the data partitions are independent, the main loop (Algorithm 3.1, lines (2)-(19)) is executed in a distributed fashion by exploiting Spark (each data partition is associated with an independent task).

Automatic parameter setting: $MinPoints$**.** Once $epsilon$ has been set, the value of $MinPoints$ is automatically set by selecting the value of $MinPoints$ for which the product $MinPoints \times numberOfCorePoints(\mathscr{D}, MinPoints, epsilon)$ is maximum, where *numberOfCorePoints* returns the number of core points in $\mathscr{D}$ found by DBScan with the *MinPoints* and *epsilon* given. The approach stems from the following observations. $MinPoints$ represents the minimum size of the generated clusters. Since small clusters are not interesting, because they represent a negligible part of our data, while we are interested in the main groups and their characterization, we aim at setting high values of $MinPoints$: The higher the value of $MinPoints$, the higher the minimum cardinality of the generated clusters. However, the higher the

Algorithm 3.2: Automatic setting of the *MinPoints* parameter value.

Input : Dataset partitions - *dataPartitions*
Input : Epsilon - *epsilon*
Output : Estimate of best MinPoints - *bestMinPoints*

```
1  Map<Int,Int> histogramNeighbours = {};
2  for partition in dataPartitions do
3      for p in partition do
           /* p is a core point if MinPoints is lower than or
              equal to the number of its neighbours          */
4          numNeighbours = numNeighboursRadiusEps(p, epsilon, partition);
           /* Update the statistics about the number of points
              with numNeighbours neighbours                  */
5          histogramNeighbours[numNeighbours] =
            histogramNeighbours[numNeighbours] + 1;
6      end
7  end
8  Map<Int,Int> numOfCorePoints = {};
9  neighboursAfterMinPoints = 0;
   /* Given MinPoints, the number of core points is the number
      of points with more than MinPoints neighbours          */
10 for MinPoints in histogramNeighbours.keys().sort().reverse() do
11     numOfCorePoints[MinPoints] = histogramNeighbours[MinPoints] +
        neighboursAfterMinPoints; neighboursAfterMinPoints =
        numOfCorePoints[MinPoints]
12 end
13 max=0;
   /* Select the MinPoints value that maximizes
      MinPoints × number of core points                      */
14 for MinPoints in histogramNeighbours.keys() do
15     numCorePoints = numOfCorePoints[MinPoints];
       /* d is the number of attributes                      */
16     if (MinPoints > d and numCorePoints × MinPoints > max) then
17         max=numCorePoints × MinPoints;
18         bestMinPoints=MinPoints;
19     end
20 end
21 return bestMinPoints;
```

value of $MinPoints$, the lower the number of core points will be. With lower values of $numberOfCorePoints$, the amount of clustered data potentially decreases, while the amount of noise points increases. Since we are interested in clustering as many data points as possible, discarding only the minimum amount of objects in lower-density areas, we should consider high values of $numberOfCorePoints$. To balance the two discussed trends, we set the $MinPoints$ trade-off to the value that maximizes $MinPoints \times numberOfCorePoints(\mathscr{D}, MinPoints, epsilon)$. Algorithm 3.2 reports the pseudo-code of the algorithm that automatically sets $MinPoints$ given the value of $epsilon$. Also in this case, the procedure can be parallelized by assigning each data partition to a different Spark task.

3.3.2 Cluster and data characterization

Since clusters are anonymous groups of network traffic flows, but human-readable results are much more valuable to domain experts, the SeLINA methodology, as reported in block (2) of Figure 3.1, is designed to enrich clusters with (i) general attribute-based statistics, and (ii) domain-specific knowledge, for each cluster in the resulting set, as detailed in the following. The former does not require user intervention, whereas the latter can be guided by domain experts, by a-priori selecting specific attributes of interest.

- *Number of flows.* It provides insights into the data distribution, by identifying clusters covering most of the dataset and others identifying small "remote" groups of traffic flows. For instance, some network datasets present a predominant cluster with regular traffic and many smaller clusters identifying deviations. Other datasets may present similarly-sized clusters, (i.e., with the same number of flows) for different subnets or services.

- *Top characterizing attributes*. SeLINA provides the attributes with the highest Variance Reduction Ratio (VRR) for each cluster with respect to their variance over the whole normalized dataset. Given an estimator for the variance $\hat{\sigma}^2$, the Variance Reduction Ratio (VRR) for the j-th cluster and i-th feature x^i is defined as follows.

$$VRR_j(x^i) = \frac{\hat{\sigma}^2_{\mathscr{D}}(x^i) - \hat{\sigma}^2_j(x^i)}{\hat{\sigma}^2_{\mathscr{D}}(x^i)} \tag{3.1}$$

where $\hat{\sigma}^2_{\mathscr{D}}$ is the variance over the whole dataset and $\hat{\sigma}^2_j$ is the variance over the j-th cluster. The rationale behind the variance reduction is to quantify the information gain, for a given attribute, obtained by isolating some of the flows in a cluster; it is inherited from decision trees [68], where the order of the attributes in the tree influences performance and results. Together with the variance itself, VRR is a strong indicator of the features that characterize a cluster the most and their relative importance.

- *Network domain statistics.* Network-oriented features of interest provided by SeLINA are the number of different source IP addresses, ports, and service types per cluster. Furthermore, the current implementation of SeLINA computes and plots the Cumulative Density Function (CDF) of selected dataset attributes (see Table 3.2 and Sec. 3.5 for details). For instance, per-cluster statistics of server IP addresses, server L4-ports, L7-application protocols, etc., are provided. Such attributes, despite being discarded during the clustering, are often crucial to allow domain experts to correctly extract meaning from the results.

3.3.3 Classification model training

All flows processed by the clustering algorithm (excluding the final iteration noise points) are labelled according to their cluster (e.g., cluster 1, 2, 3), and form a labeled dataset (i.e., a training set), which can be exploited for supervised learning. Thus, the goal of this phase, depicted in block (3) of Figure 3.1, is to build a classifier to efficiently label new unseen flows as they are captured.

Even if the cluster set could be directly exploited for labeling unseen data, a new ad-hoc classifier is trained separately to reach two design goals: (i) to provide a real-time high-performance classifier, and (ii) to build a human-readable model that can harness the knowledge inside the data.

To this aim, SeLINA exploits decision trees [69] to build the classification model. They are a well-known popular and mature technique able to reach both good accuracy and easy model interpretability, with the latter being a highly-valued feature for domain experts. To provide the intuition of how a decision tree works, we describe a toy example in the following.

tid	$RTT[ms]$	*DataByte*	Class
1	3	2M	Cl. 1
2	20	900k	Cl. 2
3	12	1.5M	Cl. 1
4	15	500k	Cl. 2
5	12	3M	Cl. 1

Table 3.1 A toy dataset

Tree example. Table 3.1 shows a simple training set with 5 records, each characterized by two attributes. Two clusters/classes are present (Cl. 1 and Cl. 2). A possible decision tree is reported in Figure 3.3. The node labels represent a feature (e.g., the size of the flow in bytes), while each branch is labelled with a possible value, or a range of values, for the feature within the node. In our example, the split from the root node is done on a range of values of the minimum round trip time. Each path from the root node to a leaf node represents a rule characterizing a class (a cluster in our case). The path within the dotted box models the simple rule $RTT < 5ms \rightarrow cluster1$, thus this leaf can be interpreted by the analyst as a set of flows served by a cache in the nearby, with cluster 1 partly served by this cache, which serves uniquely this cluster. This kind of information is human-readable and provides a good characterization of how the traffic labeling is performed.

Knowledge model. The output tree provides an easy-to-read overview of the features that best split the dataset according to the labels: for each node of the tree the split criterion can be written as an if/else condition over a single feature and a splitting value, and few levels of the tree are usually sufficient to show the most significant splits for the purpose of the classification.

Split criterion. In the current work, the impurity-based criterion used to grow the tree is the *Gini index*, defined in Section 2.1.2. The Gini index is among the most popular choices and typically yields high-quality results. We exploited the Spark decision-tree implementation, which provides both the Gini and the entropy criteria. We performed some experiments, not reported here, to compare the accuracy of the classification models based on the Gini and the entropy indices and their results are very similar. We refer the reader to [70] for details about the entropy index.

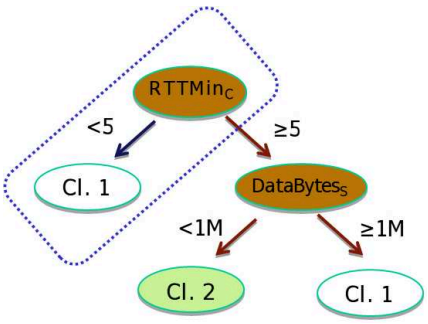


Fig. 3.3 A toy example of a decision tree.

3.4 Online characterization and model update

This component analyzes new network data in real-time by applying the model built in the previous block. As depicted in Figure 3.1, it consists of two phases: a real-time data labeling phase (4), and an online characterization phase (5).

The classification model is exploited in the real-time data labeling phase of step (4), where each new network flow is assigned a label.

Then, at step (5) the quality index computation is executed, to self-assess the model fitting and its results over time. When the quality index falls below a given threshold, the offline model building can be automatically triggered to rebuild a model better fitting the new data, thus providing self-evolutionary features. While the online data labeling phase (4) is straightforward, as it consists of a classification model application, in the following we provide details on the quality index computation in step (5) and the self-evolution policy stemming from such quality evaluation.

3.4.1 Quality index

When no external information is provided (e.g., ground-truth class labels), the clustering results are evaluated on the shape of the clusters themselves. To this purpose, SeLINA exploits a well-known quality index, named *Silhouette* [71]. This index measures both intra-cluster cohesion and inter-cluster separation to evaluate the appropriateness of the assignment of a data object to a cluster rather than to another one.

Let $\mathbb{C} = \{C_1, \ldots, C_n\}$ be a set of clusters. The Silhouette value for a given data point p_i in a cluster $C_k \in \mathbb{C}$, given a distance measure $dist$, is computed as

$$\mathscr{S}(p_i) = \frac{b(p_i) - a(p_i)}{\max\{a(p_i), b(p_i)\}}, \tag{3.2}$$

where $a(p_i)$ is the average distance of point p_i from all other points in cluster C_k, i.e.

$$a(p_i) = \frac{1}{|C_k|} \sum_{p_j \in C_k} dist(p_j, p_i) \tag{3.3}$$

and $b(p_i)$ is the lowest average distance from all other clusters, i.e.

$$b(p_i) = \min_{C_l \in \mathbb{C}} (\frac{1}{|C_l|} \sum_{p_j \in C_l} dist(p_j, p_i)), \forall C_l \neq C_k. \tag{3.4}$$

The Silhouette value for an arbitrary cluster $C_k \in \mathbb{C}$ is the average Silhouette value on all points in C_k. It is computed as

$$\mathscr{S}(C_k) = \frac{1}{|C_k|} \sum_{p_i \in C_k} \mathscr{S}(p_i) \tag{3.5}$$

Lastly, the average $\mathscr{S}(p_i)$ over all data of the entire dataset is a measure of how appropriately the data has been clustered. The distance measure *dist* must be the same used for clustering, thus the Euclidean distance in our case.

The Silhouette coefficients take values in $[-1, 1]$. Negative and positive Silhouette values represent wrong and good object placements, respectively. Hence, the ideal clustering algorithm splits the data in a set of clusters $\mathbb{C}$ such that all clusters in $\mathbb{C}$ have a Silhouette value equal to 1. However, Silhouette values around 0.5 are already considered very high values representing a strong clustering result [71].

3.4.2 Characterization and self-evolution policy

As the quality of the network traffic model is subject to ageing, SeLINA continuously evaluates the quality degradation of the model itself, with a two-fold objective: (i) highlighting substantial changes in the network and (ii) triggering the regeneration of the model as soon as the quality index falls below a threshold.

Since SeLINA computes the Silhouette for each new flow against the ones seen during the training phase, this quality index indicates how well the new flow fits the old clusters. A Silhouette close to 1 would indeed mean that the intra-cluster distances are negligible compared to the inter-cluster ones. The Silhouette values for the clusters ($\mathscr{S}(C_j)$) are recomputed every M new records, where M is set by the user. The Silhouette index for the clusters significantly changes as soon as new kinds of data (not seen during the training) are added to the input (see Section 3.6.2 for an example). Assignments to the wrong cluster should get a negative Silhouette value, while outliers, i.e. points that are distant from all clusters, will have a Silhouette close to 0.

Besides the Silhouette indicator, SeLINA also tracks the number (percentage) of new flows assigned to each cluster over time. This helps in detecting changes in the traffic characterization due to (i) degradation in the clustering quality and (ii) shift in the distribution of the traffic flows among different clusters, as discussed in the experiments.

The final goals of the real time evaluation are to keep track of the state of the network, to identify changes and react. The reaction strongly depends on the use case and on the type of change. When SeLINA is trained on a standard behaviour, e.g. a usual working day without interruptions of service or congestion, a change is a strong hint of an anomaly, a strong congestion or an attack. The identification can be performed by looking at the current clusters and their cardinality in recent time frames. If the Silhouette value is unchanged, the current clusters do still model well the traffic, and the anomaly occurs only in the distribution of the flows among the clusters. If the Silhouette value of one or more clusters decreases, instead, the change is way more significant: the current model cannot describe the traffic anymore. The inspection in this case needs a new clustering, which can also be automatically triggered by the system. The new clustering can be executed on the whole historical dataset or on the most recent flows only. The latter option generates a more specific up-to-date model, that could be less general due to fewer training data.

3.5 Experiments and datasets

We experimentally evaluated SeLINA on two real network traffic datasets, associated with two different use cases. Our goal is to show how SeLINA (i) effectively

Metric	Description	Intuition
$L7-Data$	Amount of application payload transferred	Identifies possible different type of flows, e.g., data vs signaling
$Duration$	Time since the first SYN to the last segment	Related to performance issues, and type of flow, e.g., bulk transfer or persistent connections
$RTTMin$	Per-flow minimum RTT	Estimate of the "distance" between client and server, and of possible congestion
P_{reord}	Per-flow reordering probability	Identifies possible packet losses occurred before the probe
P_{dup}	Per-flow duplicate probability	Identifies possible packet losses occurred after the probe

Table 3.2 Features used by SeLINA as input.

characterizes network traffic, and (ii) supports the analyst in understanding changes of the traffic mix. We focused on two real-world use cases. The first one consists of a dataset of YouTube flows in which we know the CDN had changed over time, causing possible issues to both end-users and ISPs [60]. The second case deals with the understanding of P2P traffic, for which, instead, little knowledge is available. In both cases, SeLINA autonomously extracts information from the automatic analysis of the traffic summaries, and presents results to domain experts in an interpretable format.

We collected network traffic data through a passive probe located on the access link (vantage point) connecting an ISP Point of Presence (PoP) to the Internet. The passive probe sniffs all packets flowing on the link. The probe runs Tstat [72], a passive monitoring tool that extracts flow level logs. Tstat rebuilds each TCP (and UDP) stream by matching incoming and outgoing segments (and messages). A flow-level analysis is performed, and for each flow a set of metrics is logged [64]. Tstat offers advanced classification mechanisms that we leveraged to split traffic according to the application that generated it.

In this work, we focus on two datasets, collected during two different time periods. The first one consists of flows carrying YouTube videos. The second one collects all TCP flows excluding web traffic, i.e., it consists of mostly P2P traffic. We refer to each dataset as "YouTube" and "P2P" in the following.

The YouTube dataset consists of TCP flows collected during May 2013 by a probe placed on a PoP of a nation-wide ISP in Italy where the traffic aggregate from more than 10,000 customers is monitored. We use data from May 1st, 2013 to let SeLINA build the offline model. Datasets from May 2nd to May 31st are used instead to run the online model update phase and highlight traffic changes possibly suggesting the automatic model rebuilding. For this dataset, we know that during the second part of May 2013 the YouTube CDN had relevant changes affecting end-user quality of experience [60, 58]. Hence, we consider this as ground-truth information that allows us to verify if SeLINA correctly identifies interesting events.

The P2P dataset refers to April 17, 2012. From it we extract all TCP flows whose application protocol is neither HTTP nor HTTPS, i.e., where the majority of the traffic is due to P2P applications [73]. Traffic comes again from a backbone link of a nation-wide ISP in Italy.

Among the measurements exposed by Tstat, we consider the metrics reported in Table 3.2. We selected them since they are correlated to both system configuration and possible performance issues. For instance, the measure of the Round Trip Time (RTT) is related to both the distance from the server, and possible congestion on the path. Similarly, both reordering and duplicate probabilities increase during periods of congestion. Finally, duration and amount of carried data are possibly linked to the type of service the flow carries, e.g., short-lived signaling flows carrying little data rather than long lived data flows carrying a large amount of data. Since SeLINA model building is based on unsupervised clustering, we expect the system to automatically leverage information offered by these features to identify proper classes of flows.

The metrics are computed by observing the TCP headers, and correlating them with information in the correspoding TCP ACKs. For instance, P_{reord} and P_{dup} are computed by keeping track of TCP sequence number evolution over time, while $RTTMin$ is the minimum delay observed between a data segment and the corresponding acknowledgement. Since TCP offers a bidirectional service, we consider measurements for each half-flow, i.e., segments from the client to the server, and vice versa. We denote them in the following by adding a subscript C or S for client or server side, respectively. For instance $RTTMin_S$ is the minimum delay observed at the probe between segments sent by the server and ACKs sent by the client, i.e., it is the delay between the probe and the customer client – the access network delay.

Conversely, $RTTMin_C$ measures the time since the probe observes the client segment and the server ACK, i.e., it is the minimum RTT between the probe and the server – the backbone network delay.

Additional attributes and measures are included in the final results flows aggregated in the same cluster. Clusters are annotated by SeLINA before being presented to the domain experts. The additional features are not considered during the model building phase. For instance, once the cluster is built, the system computes Cumulative Density Functions (CDF), average, percentiles, etc. of the per-metric distribution of information extracted directly from features.

The datasets have been stored in a cluster at our University running Cloudera Distribution of Apache Hadoop (CDH5.3.1). All experiments have been performed on our cluster, which has 30 worker nodes, and runs Spark 1.2.0, HDFS 2.5.0, and Yarn 2.5.0. The cluster has a total of 2.5TB of RAM, 324 cores, and 773TB of secondary memory. The current implementation of SeLINA is a project developed in Scala exploiting the Apache Spark framework.

3.6 YouTube use case

In this section we discuss the network traffic characterization of the YouTube dataset first, as a result of the offline SeLINA component, and then we present an evaluation of the online part. The default values of $EpsStep$=0.001 and 3 clustering levels led to meaningful results for this experiment. Increasing the number of levels brings no improvement. After the third iteration, new clusters become very small and have very low Silhouette values, a clear sign that the system is artificially aggregating data that are actually very fragmented.

3.6.1 Offline cluster and model characterization

Clustering results provide meaningful insights into network traffic when enriched by means of relevant statistics and features. As such, we present traffic analyses provided by both the cluster statistics and the classification model.

Cluster statistics

Table 3.3 reports the clusters obtained by running SMDBScan on the YouTube dataset of May 1st, 2013. For each cluster, SeLINA returns the top-3 attributes according to VRR, i.e., it presents to the network analyst those features that best represent the data inside the cluster itself. For instance, consider the cluster number 1. It is the biggest one, collecting approximately 60,000 (36%) flows. It is primarily characterized by a rather low P_{dup} value (0.65%$\pm$0.71%), and clients requesting 4kB of data on average ($L7-Data_C$=3992$\pm$2422.4 bytes), a rather sizable HTTP request size. $RTTMin_S$ is 33.7$\pm$16.1 ms, which suggests quite standard and not congested DSL lines. The cluster thus collects the most common flows. This is the only cluster identified during the first iteration of the multi-level clustering. During iteration 2 and 3, more clusters emerge (one in step 2, and two in step 3), each with several thousands of flows. This confirms the ability of SMDBScan to identify large clusters, despite different densities, thanks to the multi-level approach. At the end of the whole clustering process, the noise cluster aggregates all remaining points. There are 40,000 of them (23%), which are very sparse, as proven by the high variance in their characterizing attributes.

Clusters 2 and 3 represent a sizable part of the traffic, with 16% and 22% of the flows, respectively. Interestingly, those are characterized by two very different $RTTMin_C$ values. Recall that $RTTMin_C$ represents the distance of the YouTube CDN server to the probe. Servers in Cluster 2 are 25.3$\pm$1.4 ms far, while servers in Cluster 3 are much closer (5.4$\pm$4 ms). P_{dup} is significantly different too, with Cluster 2 P_{dup} being one order of magnitude smaller than cluster 3. This probably reflects higher congestion in the path from the probe to the client in cluster 2.

Cluster 4 collects fewer points (5,500, 4%). P_{dup} and $RTTMin_C$ are similar to Cluster 2, but here duration (51$\pm$32 s) is very large. This possibly hints for TCP flows of long lived video sessions.

All of this reflects the typical scenario of the YouTube CDN [60], and proves the ability of SeLINA to provide insights on the traffic mix. The analyst is offered few and consistent clusters, instead of thousands of single measurements.

Lev.	Cl. id	Num. flows	Top-3 representative attributes ranked by highest variance reduction ratio		
			Attribute	Avg. value	Std. dev
1	1	59846	P_{dup}	0.65%	7.12E-03
			$L7-Data_C$	3992.1	2422.4
			$RTTMin_S$	33.7	16.1
2	2	27158	$RTTMin_C$	25.3	1.4
			P_{dup}	0.55%	6.42E-03
			$L7-Data_C$	5357.8	2916.9
3	3	37964	P_{dup}	2.97%	2.31E-02
			$RTTMin_C$	5.4	4.0
			$RTTMin_S$	52.6	42.2
	4	5569	$RTTMin_C$	25.5	1.2
			P_{dup}	4.11%	1.28E-02
			$Duration$	51464.0	31969.4
noise		40318	P_{reord}	0.000002%	3.12E-06
			$L7-Data_S$	14465449.3	30919388.4
			$RTTMin_S$	78.7	160.6

Table 3.3 YouTube dataset. Cluster characterization.

Cluster id	Precision	Recall
1	96.28%	80.17%
2	97.73%	93.02%
3	97.91%	99.25%
4	88.93%	98.22%

Table 3.4 Quality of the classification algorithm. 3-fold cross-validation

Classification rules

The decision-tree described in Section 3.3.3 and trained with a maximal depth of 4 levels has been evaluated with a 3-fold cross-validation scheme on the training data. The average accuracy over the three cross-validation runs is 93%, and results for precision and recall for each class are shown in Table 3.4. All clusters are extremely well represented by the model for both precision and recall (93% to 99%), apart from a lower value in Cluster 1 recall (80%).

Being the model so accurate, rules that form the decision tree can be used to understand how the different clusters split the network traffic. Each path from the

root to one leaf of the decision tree is translated into a rule for the class of that leaf. Each rule characterizes the data of its class (i.e., a cluster in our case). Rule-based modeling provides further insights into correlations among attributes. Analying the rules of such classifier, we observe the following:

- $\{(RTTMin_C > 15.7ms)\ and\ (P_{dup} > 2.5\%)\} \rightarrow Cluster4$. This is the only rule associated with cluster 4. It states that all flows of cluster 4 are simultaneously characterized by a high $RTTMin_C$ and a high P_{dup}.
- $\{(RTTMin_C > 21.5ms)\ AND\ (P_{dup} \leq 2.5\%)\} \rightarrow Cluster2$. This rule provides a characterization of cluster 2, where flows have an even higher $RTTMin_C$ but a lower P_{dup} with respect to cluster 4.

Insights provided by such rules are relevant since we would not have been able to easily distinguish the differences between clusters 2 and 4 by considering only the available statistics. The rules, which simultaneously consider more than one measure, allow supporting domain experts to more easily characterize the content of the clusters and also perform comparisons among them by considering at the same time many facets.

3.6.2 Online data characterization and model update

The decision tree model is exploited to assign new flows, in real time, to the most appropriate class (which is one of the clusters). Every N assignements, SeLINA evaluates the quality of the current cluster set, by means of the Silhouette quality index and the distribution in number of flows assigned to each cluster. These two indicators provide the self-evolution feature to SeLINA, which is able to trigger a model rebuilding phase. The analysis of the Silhouette index indicates whether the classifier model does not fit the current data anymore, and the distribution of the flows among the clusters indicates whether the traffic patterns are changing. This information is also valuable for the analyst since it reflects changes in the traffic mix.

The upper part of Figure 3.4 reports the value of the Silhouette index for three different days (May 2nd, May 3rd, and May 29th from left to right). The lower part reports the percentage of flows assigned to each cluster over time. The values are computed every $N = 10,000$ flows, which corresponds to 2-3 hours at night, and

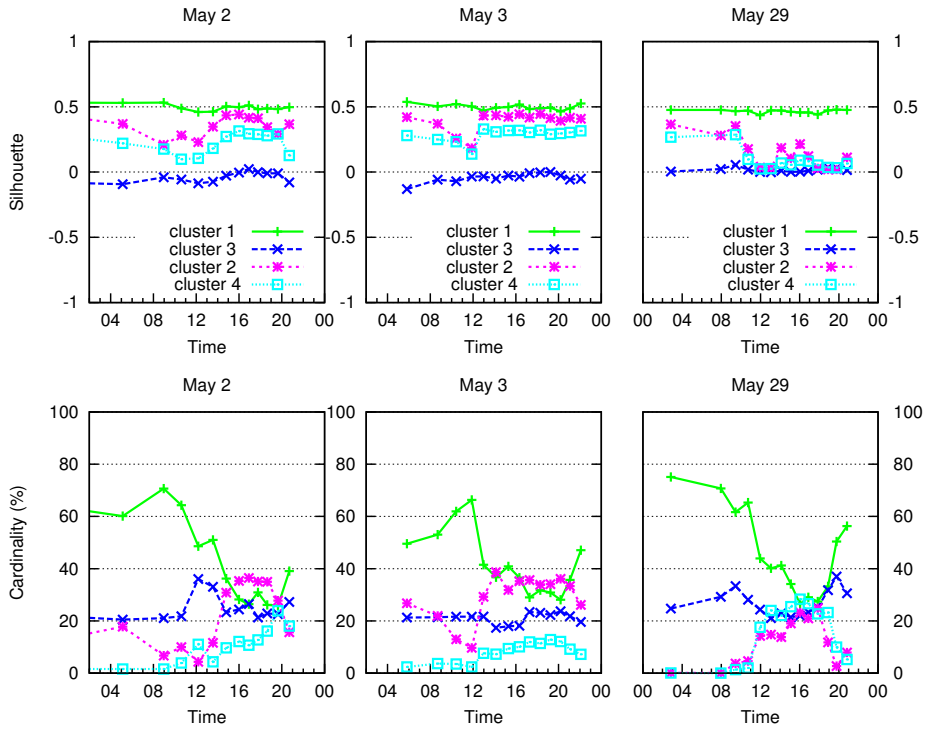


Fig. 3.4 YouTube dataset. Real-time data labeling: Silhouette and percentage of new flows assigned to each cluster.

approximately 1 hour during peak traffic hours. Recall that the model had been trained on the May 1st dataset. Traffic from following days is assigned to clusters based on the classifier, but without re-running the clustering itself. The starting point for the plots of May 2nd represents the Silhouette and cardinality value for that cluster for the training data (May 1st).

As discussed by domain experts in [60], network traffic in the first part of May is very similar to May 1st. On the contrary, in the second part of May, a change in the YouTube CDN occurred. As such, we would expect the Silhouette to reflect this situation, especially during peak time when the traffic is more significant. This is indeed the case. The Silhouette value is rather stable for all clusters during the first days of May, of whom we reported here May 2nd and May 3rd, meaning that there are no important changes in the traffic with respect to the May 1st model. It still fits the new data. Only cluster 2 and 4 show temporary and limited drops in Silhouette values from 10am to 12am, but at that time, they only account for very low percentages of the traffic (less than 10% each). The Silhouette of Cluster 3, although always very close to 0, is costantly similar to the Silhouette of the training

data, May 1st. A possible explanation for the low Silhouette value is that the other clusters surround Cluster 3, and compete to "steal" points assigned to this cluster, and this situation does not vary over time.

The Silhouette values of clusters 2 and 4 during May 29th, when the significant change in the YouTube CDN already occurred, drop suddenly from 12pm to 8pm of May 29th. At that time, a sizable amount of traffic is assigned to these two clusters ($\approx 20\%$ each), but the clusters 2 and 4 do not fit the data anymore, so that new flows present un-modeled characteristics, and they fall into the low-Silhouette clusters. Interestingly, cluster 1 still has a high Silhouette value (and counts for 30-40% of the traffic), reflecting that not all traffic is affected by the change, like Cluster 3.

A detailed analysis of May 29th highlights a significant increase of the $RTTMin_C$ values for cluster 2 and cluster 4 flows. While in May 1st and May 2nd, almost all flows have an $RTTMin_C$ lower than 25ms, in May 29th there are many flows with $RTTMin_C$ from 80ms to 100ms. The increase of the $RTTMin_C$ values is associated with changes in the YouTube's CDN previously identified in [60]. In the model built by SeLINA on May 1st data, there are no clusters representing this traffic pattern. Thus, the sudden drop of the Silhouette values automatically highlights the changes and can be used to raise an alarm.

To better highlight the difference in SeLINA results, we ran a set of experiments considering the first and last five days of May. We aim at identifying groups of similar traffic days, by means of the Silhouette pattern. Fig. 3.5 shows the correlation matrix of the per-day Silhouette indexes, i.e., for each pair of days, we measure how similar the Silhouette trends are. We use the Pearson correlation coefficient [70] among the Silhouette values during two days: when close to 1, the Pearson correlation depicts a strong positive linear correlation; when the coefficient is close to -1, it highlights a negative correlation, and when it is close to 0, negligible correlation is found. Left plot of Fig. 3.5 clearly highlights that Cluster 1 Silhouette is always very similar among those days, and stable over time. Cluster 1 consistently represents the part of traffic not affected by the CDN change, and the model always fits the new data. Right plot of Fig. 3.5, instead, clearly shows that Cluster 4 exhibits two patterns over time: during the beginning of May, it is consistent with the model. But during the last part of May (after the CDN change), its Silhouette daily pattern becomes very different from before. These results prove the strong link between the change in the

Silhouette trends and the change in the traffic patterns, and the validity of using the drop of the Silhouette as a trigger for the generation of a new model of the network.

Focusing on the percentage of new flows assigned to each cluster (bottom plots in Fig. 3.4), we can detect changes related to the CDN allocation policy. For all days, the distribution of the flows changes from the late morning till evening. In particular, Cluster 1 traffic, which is characterized by low $RTTMin_c$ values, decreases from 60-70% (at night) to 30-40% in the 10am-9pm period. On the contrary, the number of flows assigned to Clusters 2 and 4 increases from less than 5% to 20-30% each. Clusters 2 and 4 are characterized by higher $RTTMin_c$ values, i.e., traffic is now being served by far-away servers. The difference between the two days is that in May 2nd and May 3rd the Silhouette values are high and stable, hence the changes in the distribution of the flows among the clusters are meaningful. On the contrary, on May 29th the drop in Silhouette values means that clusters cannot be trusted, and thus a model rebuilding is required.

We let then SeLINA rebuild the whole model (clustering and classifier) on the flows of May 29th from 10am to 9pm. This leads to a new characterization of the network traffic, shown in Table 3.5. 10 clusters (instead of 4) are identified, with very different characterizing features, both in terms of number of flows and statistical distribution of values. The new model includes some clusters with very high $RTTMin_C$ values, which means that the presence of new CDN servers that were not properly represented by the previous clusters are now covered. For example, cluster 2, which contains about 12,000 flows, is characterized by an average $RTTMin_C$ value of 99ms$\pm$11ms, a significantly higher value than those of the former clusters (see Table 3.3). These results are consistent with those in [60], and confirm the ability of SeLINA to automatically identify changes in traffic pattern. Moreover, SeLINA extracts clusters which fit the new data and provide insightful analyses of the network traffic evolution.

3.7 P2P use case

In this section we show how SeLINA can help characterize the flows of the P2P dataset. We recall that this dataset contains all the TCP flows captured by Tstat, except the HTTP and HTTPS ones. Also in this case, we executed the offline self-learning phase by using the default values of $EpsStep$=0.001 and the first 3 levels of

Lev.	Cl. id	Num. flows	Top-3 representative attributes Ranking by highest variance reduction ratio		
			Attribute	Avg. value	Std. dev
1	1	36006	P_{dup}	0.97%	9.6E-03
			$L7-Data_C$	4738.4	2869.7
			$Duration$	36037.7	22179.2
2	2	11965	P_{dup}	2.8%	1.5E-02
			$RTTMin_C$	99.4	11.2
			$Duration$	52103.9	32688.7
	3	11279	P_{dup}	5.2%	1.7E-02
			$RTTMin_C$	4.5	4.8
			$L7-Data_C$	5337.7	3638.9
	4	4624	$RTTMin_C$	38.2	9.0
			P_{dup}	1.2%	8.9E-03
			$L7-Data_C$	6023.9	3013.9
	5	1413	$RTTMin_C$	3.3	2.1
			P_{dup}	0.77%	6.6E-03
			$Duration$	88390.6	22657.7
3	6	9063	$RTTMin_C$	4.5	5.1
			P_{dup}	1.5%	1.5E-02
			$L7-Data_C$	14291.3	9252.3
	7	7270	P_{dup}	10%	0.04
			$RTTMin_C$	101.6	9.0
			$Duration$	1.1E+05	6.1E+04
	8	2259	$RTTMin_C$	55.7	16.2
			P_{dup}	3.1%	2.1E-02
			$Duration$	60754.7	35240.9
	9	1473	$RTTMin_C$	4.8	5.4
			P_{dup}	12%	1.8E-02
			$Duration$	42006.0	33815.0
	10	349	$RTTMin_C$	122.4	4.5
			P_{dup}	0.36%	3.7E-03
			$Duration$	50917.6	31206.7
noise		14299	P_{reord}	0.000017%	1.1E-05
			$RTTMin_S$	85.3	543.3
			$L7-Data_S$	1.6E+07	2.7E+07

Table 3.5 YouTube dataset. New clusters' characterization. Clusters obtained by using SMDBScan and setting $EpsStep$=0.001

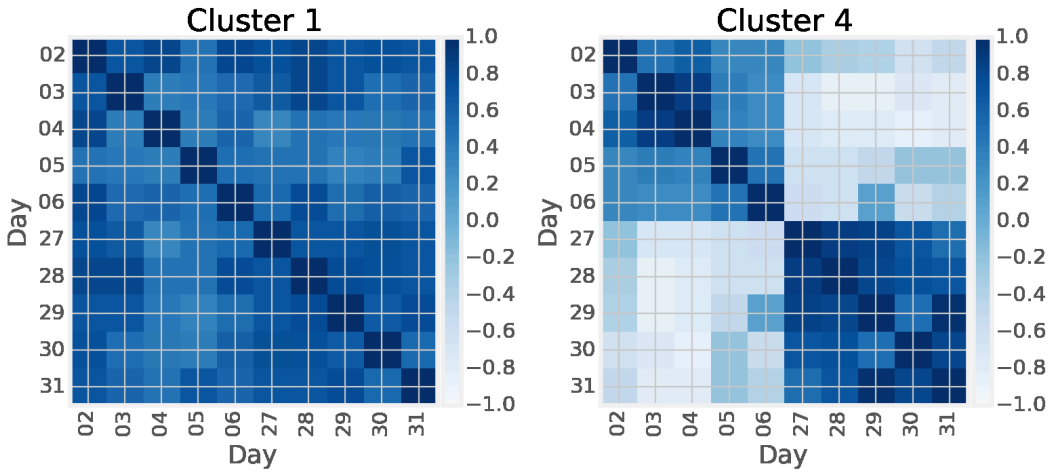


Fig. 3.5 YouTube dataset. Per-day correlation matrices of silhouettes for cluster 1 and 4.

clustering. Differently from the YouTube use case, here we have no ground truth at our disposal, and thus SeLINA is used as a data exploration tool.

3.7.1 Offline cluster and model characterization

Table 3.6 reports the main characteristics of the extracted clusters and their top-3 characterizing attributes. We immediately notice a cluster with approximately 64% of the flows (Cluster 1, 98186 flows) that is significantly larger than any other cluster. Cluster 1 represents "standard" flows which are characterized by a similar average value of $RTTMIN_C$ and $RTTMIN_S$ (i.e. the communication time is similar in both directions of the flow). This balancing between $RTTMIN_C$ and $RTTMIN_S$ values is normal when no congestions are present. Indeed, P2P traffic is exchanged between residential clients, therefore, we can expect the distance between the probe and the peers to be somewhat equal, and so the RTT.

Other clusters provide interesting knowledge to domain experts as well. For instance, Cluster 2 has an average $RTTMin_S$ (235.5ms) that is two orders of magnitude higher than $RTTMin_C$ (9.4ms). These values highlight a significant asymmetry between the server side and the client side. The $RTTMin_S$ is very high (the average $RTTMin_S$ value of the "standard" flows in Cluster 1 is 33.5ms). This situation describes a possible congestion in one direction of the communication flow. We recall that we are analyzing P2P flows among ISP customers where many users are

Lev.	Cl. id	Num. flows	Top-3 representative attributes Ranking by highest variance reduction ratio		
			Attribute	Avg. value	Std. dev
1	1	98186	$RTTMin_S$	33.5	38.6
			$RTTMin_C$	35.0	45.7
			$Duration$	33154.1	43104.8
2	2	15090	P_{dup}	3.30E-06	2.04E-04
			$RTTMin_S$	235.5	74.8
			$RTTMin_C$	9.4	22.2
	3	12152	P_{dup}	2.37E-05	5.44E-04
			$RTTMin_S$	14.6	22.9
			$RTTMin_C$	295.3	72.9
	4	4530	P_{dup}	4.44E-01	1.34E-03
			$RTTMin_S$	11.2	16.2
			$RTTMin_C$	18.5	12.9
3	5	3302	P_{dup}	1.31E-05	5.07E-04
			$RTTMin_S$	542.0	103.6
			$RTTMin_C$	5.2	13.6
	6	2524	P_{dup}	2.80E-01	2.49E-02
			$RTTMin_S$	6.9	15.0
			$RTTMin_C$	32.5	37.9
	7	1993	P_{dup}	1.05E-05	3.65E-04
			$RTTMin_S$	16.9	28.9
			$RTTMin_C$	608.1	101.5
	8	1892	P_{dup}	1.60E-01	1.88E-02
			$RTTMin_S$	14.7	29.6
			$RTTMin_C$	35.6	42.1
noise		13647	$Duration$	560334.5	4671692.5
			$L7-Data_S$	3585728.2	27938244.0
			P_{reord}	5.05E-03	2.90E-02

Table 3.6 P2P dataset. Cluster characterization. Clusters obtained by using SMDBScan and setting $EpsStep$=0.001

connected through an ADSL connection, with uplink capacity limited to 1Mbps. When a remote peer downloads a large amount of data from a local peer, the uplink of the latter may saturate, causing congestion (i.e., the average $RTTMin_S$ increases). On the contrary, the small $RTTMin_C$ suggests that the remote peer is connected via high speed FTTH technology (where access delay is much smaller being the

upling capacity >10Mbps). A similar situation is valid also for the flows of Cluster 7. However, in Cluster 7 the $RTTMin_C$ is very high (608ms) and the $RTTMin_S$ is low (16.9ms). This second case reflects a simmetric scenario: high-speed local client downloading a lot of data from ADSL remote peers, whose uplink results congested.

3.7.2 Online data characterization and model update

For the P2P dataset we applied the evolving part of the framework to analyze how new flows are assigned to the clusters and identify possible changes in the type of traffic. Results present no significant changes in the Silhouette. This trend, that is confirmed by further statistics computed on the flows, highlights that there are no anomalies or changes in the traffic for the P2P dataset. The clusters identified by the clustering phase are still representative of the network flows, also of the future traffic. Since the day we analyzed is a "normal" one, SeLINA correctly identifies no changes and hence the re-execution of the clustering phase is not triggered.

3.8 Related work

A significant effort has been devoted to the application of data mining techniques and statistical methods to network traffic analysis. The application domains include studying correlations among network data (e.g., association rule extraction for network traffic characterization [74, 75]; for router misconfiguration detection [76]; interesting correlations from web-based e-business system [77]), extracting information for prediction (e.g., multilevel traffic classification [78], Naive Bayes classification [79], throughput prediction [80], analytics and statistical models for LTE Network Performance [81], one-class SVM [82] for intrusion detection), grouping network data with similar properties (e.g., clustering algorithms for intrusion detection [83–85, 61, 62], for deriving node topological information [86], for automatically identifying classes of traffic [87–90], for unveiling YouTube CDN changes [60]), and context specific applications (e.g., multi-level association rules in spatial databases [91]).

However, in most cases no approach offloads the user from arbitrary parameter choices, and can be easily adapted to domain-specific requirements and semantics as the methodology proposed in this chapter. Differently from analytics approaches tailored to a specific network application [60, 83, 84, 86, 85, 61, 62], SeLINA is a

general purpose methodology that can be easily exploited to analyze different and transversal network data (e.g., network traffic headers, network flow characteristics, statistical measurements of traffic flows). In the experiments of Section 3.6 we considered a scenario firstly addressed by Youlighter [60]. Youlighter is a system that detects very specific macro-changes in the YouTube traffic pattern involving the CDN spatial distribution. It is not distributed nor scalable. SeLINA, instead, introduces a general-purpose, distributed, and fully autonomous engine exploiting a completely different methodology and addressing a more general research issue in the network traffic analysis than the Youlighter system.

The performance of most state-of-the-art general purpose approaches [74, 75, 87–90] depends on the choice of different parameters, and the optimal trade-off between execution time and accuracy must be handpicked for a given application. On the contrary, focusing itself on self-learning capabilities of state-of-the-art scalable approaches, SeLINA is able to build a model of the data with minimal user intervention by offloading the user from the non trivial task of configuring the miner system and highlighting possibly meaningful interpretations to domain experts.

Some research effort has been devoted to automatic setting of data mining algorithm parameters (e.g., clustering algorithms [92, 93], itemset mining [94]). Authors in [92, 93] proposed a hierarchical strategy to aggregate lower density regions discovered through DBSCAN. Differently from [92, 93], SMDBScan automatically sets DBScan parameters at each iteration level when DBScan is exploited in a multiple-level fashion. Furthermore, the SeLINA clustering results include clusters with a diverse degree of density, because each subset of clusters with a similar density is discovered at a given iteration level. The method in [92, 93] instead gets a flat partition composed of clusters extracted from local cuts through the cluster tree.

An intensive research activity has been devoted to designing innovative algorithms and methodologies to support large scale analytics based on MapReduce, such as [95–97]. A step further has been proposed in [98]. Apache Spark with its Resilient Distributed Datasets and its smart APIs, outperforms Hadoop MapReduce in terms of performance and overcomes its limitations, with particular focus on iterative in-memory computation, which is a common characteristic of many data mining algorithms. Its machine learning library MLlib [22] provides a broad range of analytics algorithms. SeLINA exploits the computational advantages of distributed computing frameworks, as the current implementation runs on Spark. Applications

of this techniques to network traffic analysis becomes natural, given the volume of traffic [63, 99–101]. These works adopt Hadoop or Spark, and apply either standard machine learning algorithms, or design specific solutions to their problem.

The idea of defining a generic framework and of tightly integrating self-learning capability in a scalable data mining engine tailored to traffic data was first introduced by ourselves in [57]. However, SeLINA significantly enhances the methodology proposed in [57]. The SeLINA data mining engine (named SaFe-Nec in [57]) provides an innovative and more accurate explorative approach coupled with self-configuring strategies (i.e., the SMDBScan algorithm). Thus, SeLINA allows exploiting cluster analysis on real datasets in a fully autonomous fashion. The SMDBScan algorithm is characterized by configuration parameters whose setting is rather difficult. In [57] the less effective, but easier to configure, K-means algorithm was used. SeLINA also includes ad-hoc strategies to automatically tune the clustering parameter values, which is a typically difficult task also for domain experts. The exploitation of a multilevel DBScan-like algorithm jointly with self-configuring strategies allowed for better clustering results than the ones produced by the K-means based approach proposed in [57]. Moreover, SeLINA integrates innovative self-assessment features and a new set of network domain statistics that are often vital to let the domain expert interpret the results. Finally, with respect to [57], in this chapter we added a new interesting case study on YouTube traffic analysis and a thorough analysis of the results from a networking point of view.

3.9 Summary

This chapter presents a self-learning data analytics system that effectively mines network traffic data. The proposed methodology is based on a two-phase approach that

1. builds a self-evolving human-readable traffic model by autonomously splitting traffic data into homogeneous groups;
2. classifies new data in real-time and identifies the presence of changes in the traffic mix.

The SeLINA methodology features a distributed implementation in Apache Spark. It is a general purpose approach, which can be easily exploited to analyze network traffic data in different conditions. The approach has been tested in two real-world use cases. The performed experiments highlighted its ability to autonomously identify evolutions in the network and support the analyst by selecting characterizing features.

3.10 Relevant publications

[56] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Paolo Garza, Danilo Giordano, Marco Mellia, and Luca Venturini. Selina: a self-learning insightful network analyzer. *IEEE Transactions on Network and Service Management*, 13(3):696–710, 2016

[57] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, and L. Venturini. Safe-nec: A scalable and flexible system for network data characterization. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 812–816, April 2016

Chapter 4

Data Science for urban security

In the last decades, municipalities and public bodies have used and produced large, digital corpora of data, releasing part of it to the citizenry as Open Data, as part of a broader set of policies usually reunited under the hat of a Smart City. Only recently Smart Cities have understood the value of Data Science to extract precious insights from these large corpora and other unstructured sources of information like social media. One such scope of application of Data Science is that of urban security, to help the police, the emergency responders, and the decision makers design solutions that fit the problems that emerge from the data.

This chapter aims at answering the following research questions:

1. is it possible to retrieve social media messages and exploit the spatial information hidden in them to help the understanding of an emergency?
2. are there temporal patterns in social issues like crimes, and is it possible to mine the data to see them?
3. how do we integrate the discoveries of Data Science with the work of practitioners and decision makers, and how do we present the insights on data in the most timely and effective way?

The contents of this chapter were originally published in [102–104]. The chapter is organized as follows. In Section 4.1 we analyze the spatial information contained in tweets during mass emergencies, trying to answer our first question. In Section 4.2 we propose a methodology to find seasonal patterns in societal data and answer the

second question. In Section 4.3 we propose an integrated data mining and Business Intelligence architecture for the analysis of non-emergency data acquired in a Smart City context, that is our attempt to answer the third question. Finally, 4.4 draws the conclusions.

4.1 Analyzing spatial data from Twitter during a disaster

In the last years, social media have met an unprecedented success and become a widespread, fast, and economical tool to access and share information. As such, they are an invaluable help in a mass emergency situation like that of a natural disaster, and are already actively used to communicate to the population involved in the preparation or in the aftermath of a disaster.

However, exploiting social media to lead decisions in a mass emergency presents multiple challenges, including parsing information, handling the information overload, and prioritizing different types of information, as discussed in [105]. One of these challenges is handling geographical information, which translates in identifying the content produced or related to a specific area, and placing this content on a map. Coping with these two issues would mean, in a scenario hit by a natural disaster, being able to have instantaneous and immediate feedback from the population in the area, possibly with reports of the damages, requests for support or availability of food, shelters, or help.

This work is an exploratory study on the quantity and the quality of the geographical data officially provided by a social media service like Twitter, in contexts of mass emergencies and natural disasters. The aim is to assess how many tweets contain geographical information and of what kind, and whether these tweets contain useful information for disaster relief and management.

4.1.1 Related work

In recent years, social media emerged as a potential resource to improve the management of crisis situations (e.g., earthquakes, tsunamis, floods). Authors of [105] have extensively investigated the subject, surveying the methods available in literature

and their shortcomings, among which important issues of privacy, reliability, and accuracy of information. We refer the reader to this survey for a detailed review of the literature on the topic.

The spatiality of social messages has been addressed in previous works and raised concerns. The authors of [106] identify general spatial patterns in the occurrence of tweets through statistical analysis. The results show that messages near (up to 10 km) to severe crisis areas have a much higher probability of being related to the crisis. Although, in a review of the use of SMS and social media in the Haiti earthquake [107], the authors note that the value of such information at a detailed level was mainly useless on the field, while the aggregate information from various sources proved very helpful to focus work to areas where relief was most needed. Moreover, tweet datasets depict a specific period in time, typically defined by the use of particular hashtags. Thus, the analysis of social media during and after a disaster can resemble traditional media coverage, which has been often accused of paying attention to only the most sensational stories in a truncated timeframe [108].

Several works outside the scope of mass emergencies have already showed that social media contain very limited spatial information. In [109], for example, only 2% of the tweets in the study contained GPS location. The authors of [110] reported that only 0.42% of all tweets in their study had GPS-provided coordinates, and thus proposed a system to infer city-level location from the content of the tweet. In some contexts, these percentages do not impede a thorough spatial analysis. In the large dataset of tweets related to 2014 FIFA World Cup, for example, authors of [111] found more than 300 thousand out of 23,5 million tweets to be geo-located, which allowed a very large-scale analysis of the event.

4.1.2 Data collection and preprocessing

Twitter APIs provide access, with some restrictions, to the 140-character texts and the rich source of metadata associated with it. Among the optional fields in the metadata of a tweet we find geographical coordinates and a place id. The user, or rather the application posting on his or her behalf, can choose to add the precise location given by the GPS sensors, or instead associate the tweet to a nearby point of interest, which translates to a place id. A place is defined as an area with predefined geographical bounds, and can range from a venue to an entire region or country.

The Twitter developers' documentation states that roughly 1% of all tweets are geo-located. In addition to these, the documentation hints that natural language processing is used to enrich the results of a geo-spatial search. Thus, a search of tweets around the coordinates of Rome would return tweets with coordinates in the area and possibly tweets mentioning Rome in the text, or tweets by users who set Rome as location in their profile. No filter by country code or state is possible with the public APIs.

The two datasets used in this study were collected as follows.

Ischia The dataset scope is to represent all the tweets in Italy on 21/08/2017[1]. On that date, an earthquake with magnitude 4.2 hit the island of Ischia, causing 42 injuries and 2 deaths and extended damage to the buildings [112]. The tweets were searched with two different queries. The first query searches for tweets in a radius of 600 km from the city of Rome, which covers approximately all the country. The second query searches for all tweets in Italian, which is a good proxy of all tweets in Italy, as Italian is spoken by the majority in Italy and little spoken abroad. All tweets belong to the day of 21/08/2017. The tweets coming from the first query were labeled as geo-referenced, with the broad meaning of having either geographical coordinates or NLP-enriched geographical references.

Texas The dataset aims to represent tweets in an area largely affected by hurricane Harvey, an Atlantic hurricane formed on 17/08/2017 that has caused the death of 78 people and the evacuation of more than 30,000 [113]. The tweets were downloaded within a circle of 300 km centered in Rockport, Texas, the city where the hurricane made the first landfall. The radius was set as to cover all the Texas coastline. The date of the tweets is 27/08/2017, the day after the initial landfall, when hurricane Harvey reclassified to storm and the heavy raining caused widespread floods in the whole state.

4.1.3 Discussion

The Ischia dataset contains 409392 tweets, of which 3566 (0.87%) are geo-referenced. This percentage is similar to the one stated by Twitter. The earthquake in Ischia hit the island at 20:57. The results for a search on a given date go until 2 in the

[1] note that Twitter developers' guide states that some tweets and users may be missing from search results

morning, which means the dataset contains 19 hours of tweets written before the earthquake, and 5 after. 67813 tweets in the dataset contain one or more hashtags, which thus offer a good sample of the topics discussed on the platform. The word clouds in Figure 4.1 show the most frequent hashtags, where the size of the word is proportional to its frequency. Figure 4.1(a) depicts the most frequent hashtags in non-geo-referenced tweets before the earthquake. TV programs and football teams take most of the social interest, with a little attention to exceptional events like the solar eclipse, happening at 20:26 local time and not visible from this timezone. The first tweet on the earthquake appears at 20:59. Figure 4.1(b) shows the frequent hashtags from then to 1:59. Ischia is indeed the most frequent hashtag, and we can spot other related terms like *terremoto* (earthquake) and Casamicciola, the location of the epicenter. Most of the word cloud, though, is yet crowded with mentions of TV shows broadcasted in that evening. The number of tweets containing the words Ischia, Casamicciola, or synonyms of earthquake is 9668, 5.2% of the tweets after 20:57. Care must be taken in taking this number as a measure of the interest to the event, as most tweets are made of stopwords and might follow up a conversation, and are therefore less likely to contain keywords or refer directly to the fact.

Figures 4.1(c) and 4.1(d) show frequent hashtags for geo-referenced tweets, respectively before and after the earthquake. These two clouds appear to speak of the same topic, and the abundance of English terms and the vocabulary used suggests that these are mainly promotional tweets that sponsor touristic areas, in which the earthquake is mostly ignored. Indeed, the percentage of geo-referenced tweets referring to the earthquake after 20:57 is only 2.8%.

Tables 4.1 and 4.2 show, respectively, the top 10 domain names of links in geo-referenced and non-geo-referenced tweets in the Ischia dataset. Links can be a hint on the kind of content posted, e.g. a video in the case of a link to Youtube, on the app used to post the tweet, e.g. Swarm in the case of *www.swarmapp.com*, or on an interaction on the Twitter platform, e.g. a reply to a tweet contains a link to the original tweet. The top 10 websites linked in non-geo-referenced tweets (Table 4.2) depict the behavior we would expect from a user of Twitter: an high interaction with other users (*twitter.com* is the most linked domain), videos (Youtube is the third most linked domain) and links to other popular social platform like Facebook and Instagram, that are respectively the second most and the fifth most linked domains. The top domains in geo-referenced tweets (Table 4.1) are very different, and in the top 10 list we do not see any of the top 3 websites linked in non-geo-referenced

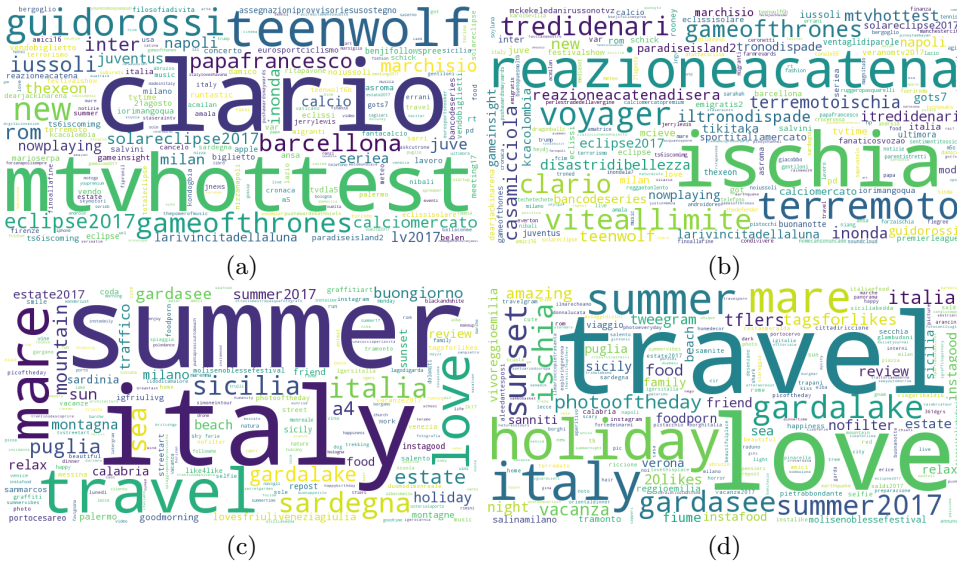


Fig. 4.1 Frequent hashtags in the Ischia dataset in non-geo-referenced tweets, (a) before the earthquake and (b) after the earthquake, and in geo-referenced tweets, (c) before the earthquake and (d) after the earthquake.

tweets, i.e. Twitter itself (with retweets), Youtube and Facebook. The top two domain names in geo-referenced tweets come from apps that post tweets from third parties, like Instagram or Foursquare (Swarm). These differences are a second clue of the different nature of geo-referenced tweets, and suggest that they are not a representative sample of the whole stream of tweets.

Texas dataset is made only of geo-referenced tweets, but with different granularities of geographical information. Its study can lead to a better assessment of the quality of the geographical information inside a tweet corpus. Of 6938 tweets resulting from the geographical query, 1275 have geographical coordinates, of which 1240 have also a place id. The remainder of the tweets is supposedly assigned to a location through named entity recognition, though no metadata gives details on this. Places divide further in three types (i.e. city, admin, and neighborhood), of which city is the most numerous, with 1039 records. Figure 4.5 lists the names of top 10 locations with their frequency. Not surprisingly, 9 out of 10 are cities, although Texas is the second most frequent place. Already the fifth most frequent city, Conroe, has less than 25 geo-tagged tweets, and the tenth, Mission, has only 11 of them. Figure 4.2 shows a frequency map of tweets with coordinates, binned in hexagonal cells of equal size. A large part of the map does not see any tweet, and most of the cells have less than 50 tweets. The most of them are located in the bigger cities, probably

URLs	#
https://www.instagram.com/	845
https://www.swarmapp.com/	51
https://goo.gl/	25
https://www.trendsmap.com/	16
http://dlvr.it/	8
http://www.olevanometeo.it/	5
http://n.mynews.ly/	5
https://www.gpone.com/	4
http://www.montedarena.com/	3
http://crwd.fr/	3

Table 4.1 Top 10 of domain names linked in geo-referenced tweets in Ischia dataset

URLs	#
https://twitter.com/	30890
http://fb.me/	21139
http://youtu.be/	7011
http://ift.tt/	5484
https://www.instagram.com/	4848
https://goo.gl/	4516
http://bit.ly/	3785
http://dlvr.it/	3024
http://l.ask.fm/	2271
https://curiouscat.me/	1890

Table 4.2 Top 10 of domain names linked in non-geo-referenced tweets in Ischia dataset

URLs	#
https://www.instagram.com/	4000
http://waterdata.usgs.gov/	871
http://bit.ly/	360
https://www.swarmapp.com/	240
http://bubly.us/	165
https://mesonet.agron.iastate.edu/	106
http://untp.beer/	94
https://twitter.com/	70
http://www.allaboutbirds.org/	42
http://tour.circlepix.com/	13

Table 4.3 Top 10 of domain names linked in tweets in Texas dataset

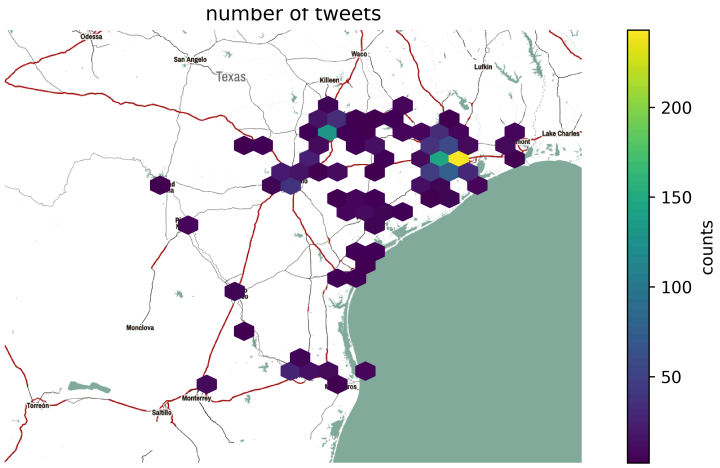


Fig. 4.2 Frequency of tweets by area in Texas dataset

following the distribution of the population, and only the cell over Houston sees more than 200 tweets. In the city, we see again a similar behavior, with more than 120 tweets located in the city center and the rest scattered around (Figure 4.3). If, as it is likely, people tweeted from the blank spots of the map, their tweet was ignored by this geographical search and there might be no information to link it to this area of the world.

Table 4.3 lists the top 10 domain names in links in this dataset. Similarly to Table 4.1, many of them link to Instagram and Swarm, and Facebook and Youtube are totally absent. Among them we can also notice domains of public services, that post warnings and tweets of public interest. Figure 4.4 shows the most frequent hashtags. The vocabulary used, differently from Ischia, seems to be largely related to the event. This can be due to the predictability of the hurricane, to the fact it is weather-related, or to the large extent of the area and population affected. The hashtags seem to belong mostly to weather warnings and automatic reports, with tags that refer directly to the city or area involved or in several cases are specific to the USGS service [114]. This, in line with the findings in Ischia dataset, shows how geo-referenced tweets belong to a special subset of users, and are not apt to describe the whole population, at least in a special situation like that of a natural disaster.

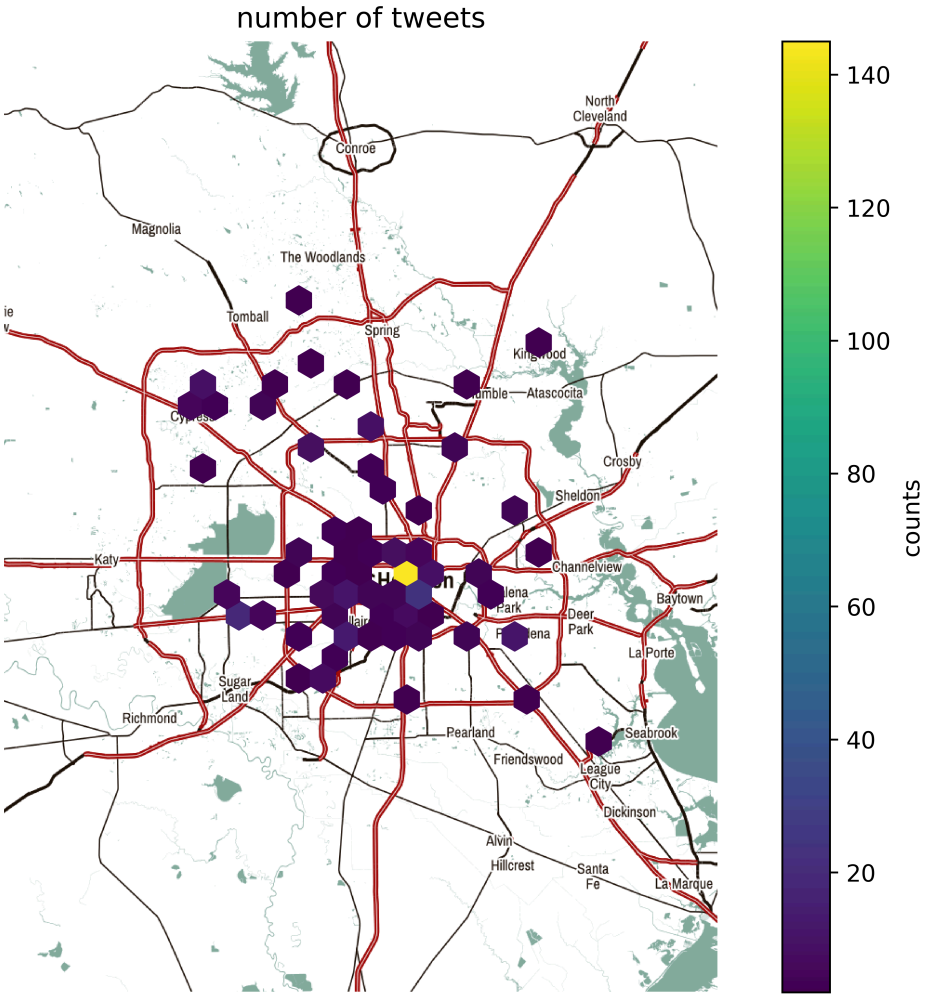


Fig. 4.3 Frequency of tweets by area in Texas dataset (place_id=Houston, TX)

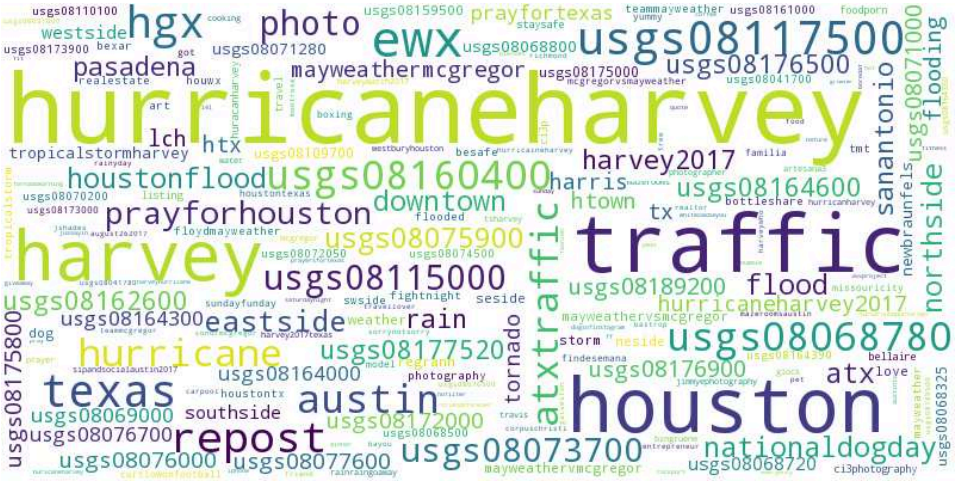


Fig. 4.4 Frequent hashtags in Texas dataset

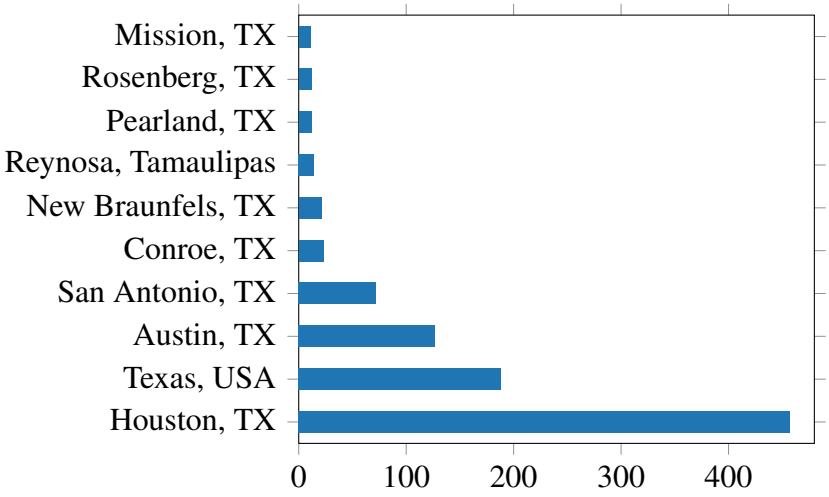


Fig. 4.5 Frequency of top 10 locations for Texas dataset

4.1.4 Suggestions for practitioners

The findings of this explorative study may outline some suggestions to the researcher, or practitioner, whose aim is to set up an analysis of a mass emergency in a social medium. These suggestions could be even useful for a social study of any kind, not limited to the scope of emergencies. The experimental evaluation we have performed is clearly limited, as it studied only two different events, and more analyses are needed to generalize the results.

We can divide a study of the social medium in three main challenges: message retrieval, message mapping, and the analysis itself. For each, we list some considerations in the following.

Message retrieval. Twitter is the only social network to offer public APIs for the retrieval of the posts on its platform. For the Streaming API, the limit is to formulate a query that returns at most 1% of the messages, otherwise the result will be subsampled. A spatially-bound query, like the one we used to retrieve the Texas dataset, is not suggested, as as we have seen it returns a small fraction of all actual tweets. One of the best options is to formulate a language-based query, and rely on the automatic language recognition of the Twitter platform, which has hopefully a small percentage of false negatives that will not be retrieved. This option, however, is only viable when the language(s) spoken in the country is rarely spoken abroad, and the volume of total messages does not exceed the 1%, as with Italian or Finnish. It is clearly not suitable for English, French or Spanish. The third method, the most used, is a keyword-based query. This method, in most cases, will ensure that the volume of messages queried will be under 1%, and also on topic. The number of false negatives, however, is potentially big, and care must be taken in the choice of keywords.

Message mapping. Once the messages are retrieved, we may want to locate them on a map. As said, only a few message will have spatial coordinates in the metadata. However, spatial information might be hidden elsewhere. One possible solution is to analyze the history of messages of the user to search for the last mentioned location, which could be as well placed in his or her biography. In the context of mass emergencies, though, this datum could be too old or imprecise to be of a practical use. Another possibilty, often used, is geocoding [105]. Geocoding is the retrieval of spatial coordinates from a string of text, which could be a toponym

or an address. It is usually coupled with Named Entity Recognition (NER) [105], to extract and identify geographical entities in the message. Though it augments the number of messages that can be placed on a map, its resolution is often limited to the city level, as users rarely write a complete street address in a message.

Message analysis. This is the most delicate part. As said above, there are many potential source of bias in a such collected dataset, of which the analyst should be aware. When studying tweets with spatial coordinates, for example, we are studying a special subset of Twitter users, who want for some reason share their location with the public, which is an option that is not active by default. The reasons they have could vary: they could be incentivated by a third-party app that promotes the location sharing through gamification, or they might want to improve the visibility of their tweet for a marketing campaign. Whatever the reason, they are not the average user. Moreover, this only adds to the fact that we are only studying one of many ways people share messages, of which most are private, and that there is always a good share of population that is not present on online social networks at all.

4.2 Spectral analysis of crimes

In the last decades, smart cities and administrations have released large amounts of data as Open Data. First pulled by a part of the citizenry advocating for transparency, Open Data have been lately pushed and supported by the same administrations, willing to sustain new studies and foster innovative applications of the data, as integrated visualizations or predictive models. In the years, wide, comprehensive and structured datasets of heterogeneous categories have been growing to dimensions that now allow significative insights and powerful applications.

Urban data, being strictly linked with human activities, usually hide repetitive patterns, which occur over time and space. The exploration of these patterns can follow the intuition of the scientist, inspired by a shared knowledge of the topic or by the literature, or be the fruit of a systematic approach of data mining. Highlighting reoccurring patterns gives valuable insights into the data and thus increases the knowledge on the subject, laying also the basis for predictive or explanatory models.

In this work, we aim at finding spatio-temporal patterns in a large dataset of urban crimes, through spectral analysis. Our research question is twofold: firstly,

we want to find out whether seasonal patterns exist in the data and can be found, and secondly, whether such patterns are global or if they vary by the category of crime. The answer to these questions can improve the current understanding of these phenomena and drive future research on intelligence-led policing. To this aim, we propose a methodology for mining seasonal components in a time series that we borrow from signal processing theory. We couple this technique with heatmap analysis to lay down an exploratory process able to highlight patterns in time and space.

4.2.1 Time-series analysis

Description of the dataset and motivations

The dataset used in our analysis describes all the crime events in the city of San Francisco from January 2003 to August 2016, as published by the San Francisco Open Data portal [115]. The dataset counts 1952810 records, split in 39 major categories of incidents, e.g. Assault, Vehicle Theft, Drug/Narcotic, and each record can be assigned to one of these or to Other Offenses. Some incidents can consist of several records, as in the case of multiple, contemporary arrests or of multiple charges on the same person. Each record reports also a short description of the incident and its resolution, namely if it led to an arrest. The metadata available are the timestamp and the location, with the address, the district and the spatial coordinates of the event.

The richness of this dataset spans multiple dimensions and allows a plethora of interesting insights and analyses. Here we focus primarily on the temporal aspect, and we show in Figure 4.6 a sample of the width of the data available, already narrowed down to a single category, i.e. Burglary. Figure 4.6a shows a glimpse on the more of 13 years of incidents recorded, with different scales of aggregation. The plot of the number of records per month, with its respective trend, computed as a moving average with a 12-month window, already shows how such events can hugely vary within months, seasons or years. The hundreds of records per month and the tens per day allow thorough, statistically significant analyses, on different scales and resolutions.

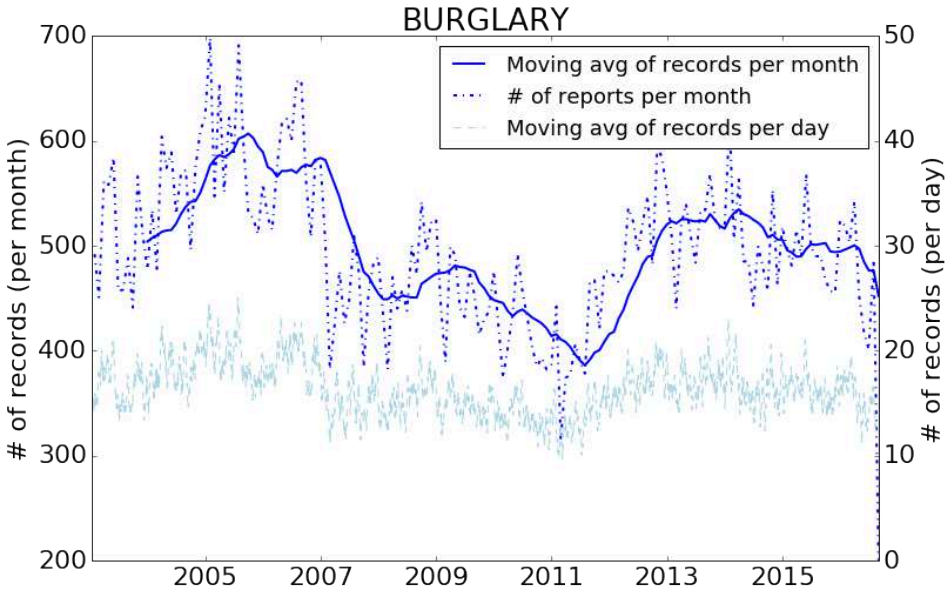
In Figure 4.6b we show a detail over the daily distribution, to highlight how the distribution of the events greatly varies not only from one month to the other, but also within consequent days and weeks. Thus, a model aiming at predicting or explaining the evolution of the incidents in the city would need to cope with a great complexity, result of several hidden seasonal components, each one potentially peculiar of only some categories of crime, added to the natural variation of the datum.

Many social studies tackle the problem of the temporal distribution of crimes with exploratory data analysis techniques, like histograms [116], radar charts [116, 117], or boxplots [118]. Authors of [118], for example, investigate temporal variations of crime in Campinas, Brasil with the use of boxplots. Among their findings, they see a drop in the total number of burglaries during weekends. We reproduce the same kind of plot in Figure 4.7, which shows the distribution of the daily count of burglaries in San Francisco by weekday. The plot clearly shows a peak in Fridays, while the weekend, contrarily to Campinas, does not see a significant drop. This kind of analysis is clearly useful to hint patterns in the data. However, it does not answer the question of whether the patterns repeat constantly through the dataset, or, in other words, whether there are significant periodicities in the data.

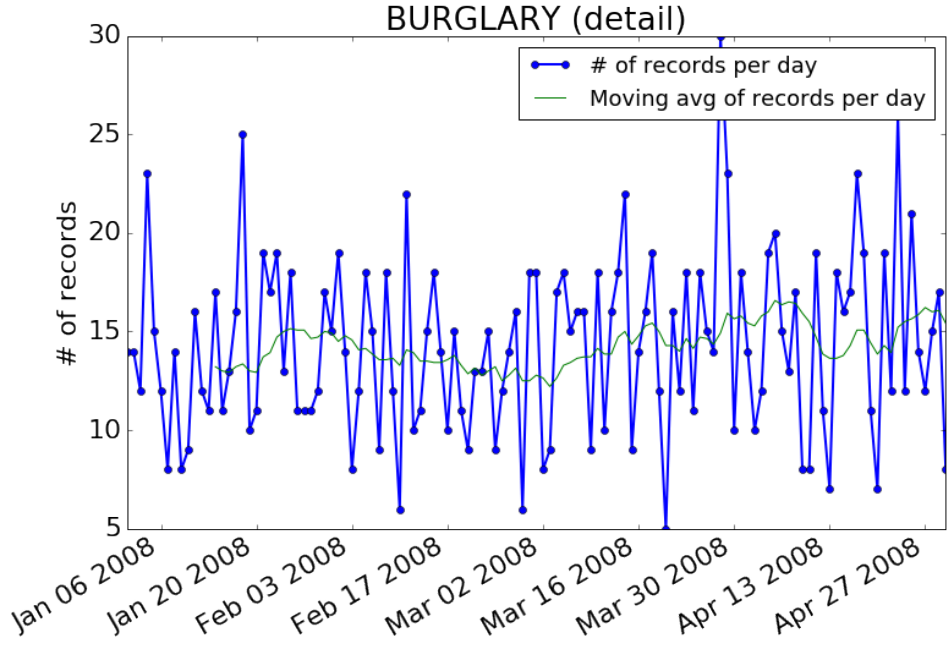
Spectral analysis

In order to highlight the seasonal components of the dataset we resort to the Lomb-Scargle periodogram [119, 120]. Periodograms, like the Lomb-Scargle, show the most likely periods in a time series and are thus a kind of spectral analysis. The Lomb-Scargle periodogram is usually preferred to the more known Fourier transform when dealing with missing samples or uneven sampling steps, and it is therefore of much wider application.

This technique is also known as least-squares spectral analysis, as it is a least-square fit of sinusoidal functions over the data. The peaks found in the resulting periodogram are thus the periods which best fit the dataset, with the y-axis showing the gap in the Bayesian Information Criterion, a measure of how likely one period fits better than other values.



(a) Temporal evolution of burglary from 2003 to 2016



(b) Daily evolution of burglary in 2008 (detail)

Fig. 4.6 Evolution of burglaries in San Francisco

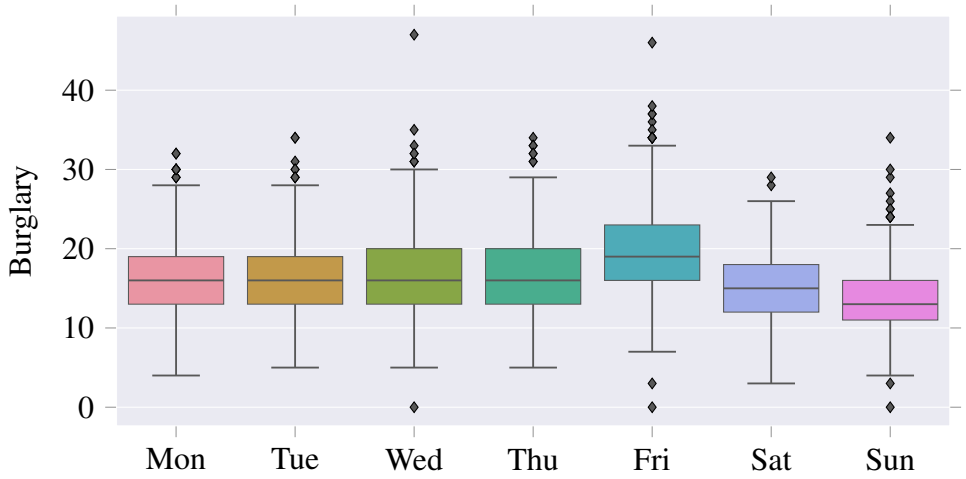


Fig. 4.7 Daily count of burglary by weekday

The Lomb-Scargle periodograms in our experiments were computed using the Python package AstroML [121]. The code and data to reproduce the analysis of this section are available online[2].

4.2.2 Results

Before applying the Lomb-Scargle analysis, we isolate the seasonal components of the series. We do this by subtracting the trend, computed with a simple 14-day moving average of the data itself. Figure 4.6a shows this moving average for burglaries. The result of the decomposition is shown in the upper part of Figure 4.8, a scatter plot of the detrended signal, whose central value is zero, as wanted. Below this signal, we see its periodogram, computed on a set of candidate values ranging from 1.1 to 50 days, to find short and medium-term periods. The periodogram evidences three significant periods for burglaries: one at seven days, one at 3.5 days, a sibling of the main peak, and one at a value between 30 and 31 days, which corresponds most probably to the average length of a solar month. The prominent period for burglaries is by far the one at 7 days.

We applied the Lomb-Scargle analysis to all the 39 categories of records, to find temporal patterns in the dataset and to state whether a unique predictive model can fit any category of crime, and thus all records, or if instead each category needs a different model. Several categories do not have more than some hundred samples,

[2] https://github.com/lucaventurini/timecrime

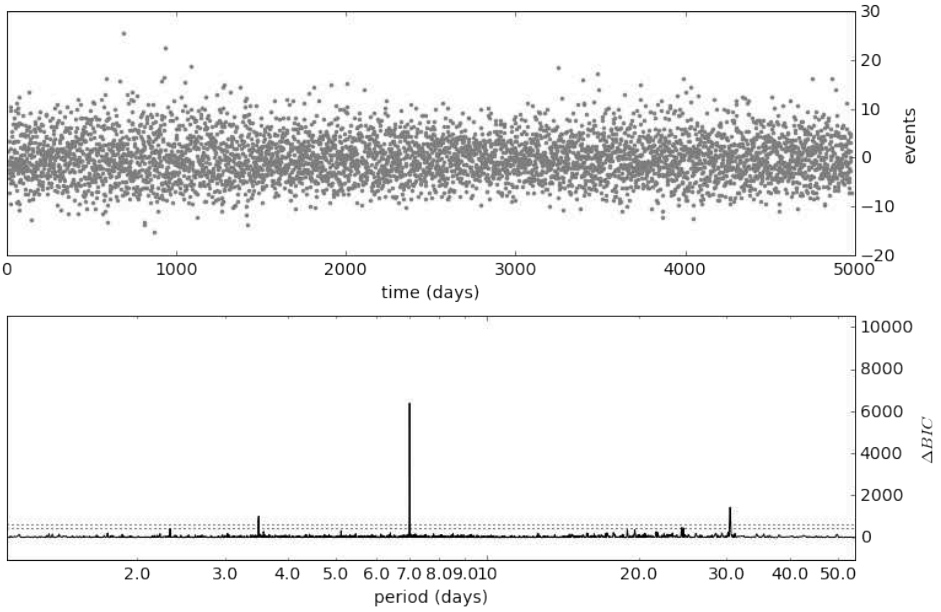


Fig. 4.8 Periodogram of burglaries with detrending

though, and therefore do not produce significant periodograms. In Figure 4.9 we show the periodograms for the 14 most-frequent categories of crime. Among them, we see very specific categories like Vehicle Theft, and broader categories as Other Offenses, Non Criminal and Secondary Codes. The latter did not show significant periods, most probably because of the variety of events that belong to it. On the other hand, the other very assorted category, that is Other Offenses, is depicted by a very neat peak at 7 days, with no sign of other possible periodicities. Also almost all the other categories show this clear weekly period. The second most represented period is the monthly one, which is discernible in at least half of these 14 categories. As seen with burglaries, this period is always slightly more than 30 days. This result excludes the hypothesis of a 4-week pattern, and suggests the idea that these crimes tend to happen at the same day of the month. Robberies, interestingly, as opposite to all other kind of crimes, seem to have such monthly patterns way more likely than the weekly ones. Also interesting is the fact that we never have periods of 14 days, and only 3 periodograms show peaks at 15 days. These results encourage the investigation of seasonal modelings of the temporal evolution of crimes and suggest to focus the attentions on weekly and monthly cycles primarily.

We now consider the spatial distribution of the events for one of the major categories, Vehicle Theft. The periodogram drives us towards an inspection of

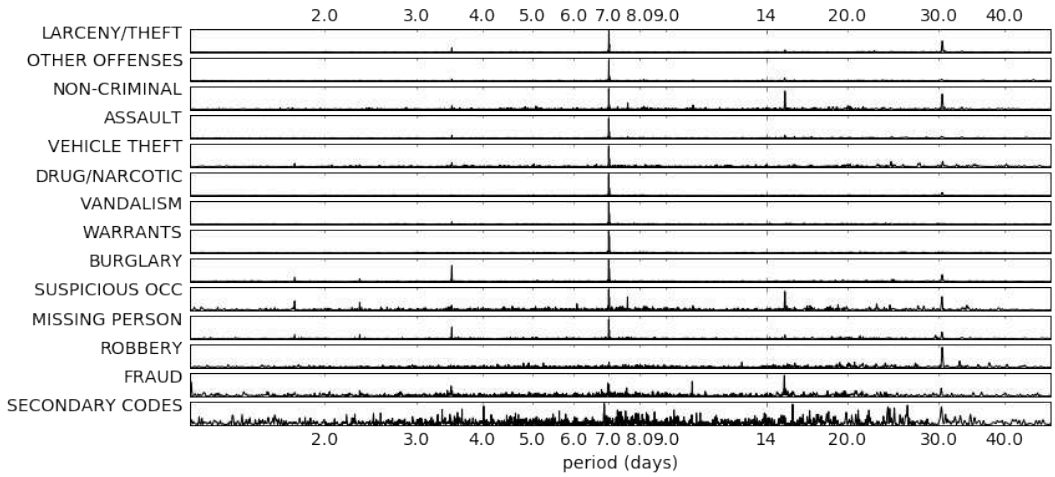


Fig. 4.9 Periodogram of most frequent categories of crime

weekly patterns, but still does not say if the patterns are homogeneous in space, or peculiar to some parts of the city. Figure 4.10 shows a heatmap of all the vehicle thefts recorded in 2015, aggregated by day of the week. The two central weekdays are thus where the hotspots of events are most extended, with a slight relaxation in Mondays and Saturdays. Like in the time series, we can glimpse here a trend, in the form of some core spots common to all days, and some components which are peculiar to each day. Focusing on the hottest zones (in red in the plot) of Wednesdays and Sundays, for example, we see in both two neat clusters, which are though different in shape, position and extension. These trends might suggest that analyses similar to what we have tried on time, are possible also on spatial dimensions.

4.2.3 Related Work

Seasonality in criminal incidents has been widely studied. Works like [122] and, more recently, [116–118, 123–126] have already attempted a reasoning over the reproduction of crimes on regular cycles. [117] considers the seasonal component of climatic seasons, aiming primarily at finding differences correlated with weather conditions and climate. Similarly, [116, 124, 125] deal with the link between some categories of crime, like assaults or property crimes, and times of the year, searching for peaks and correlations with external factors, using a number of statistical techniques, among which histograms, ANOVA and regression analysis. Our approach instead is agnostic towards any initial hypothesis of seasonality or correlations and

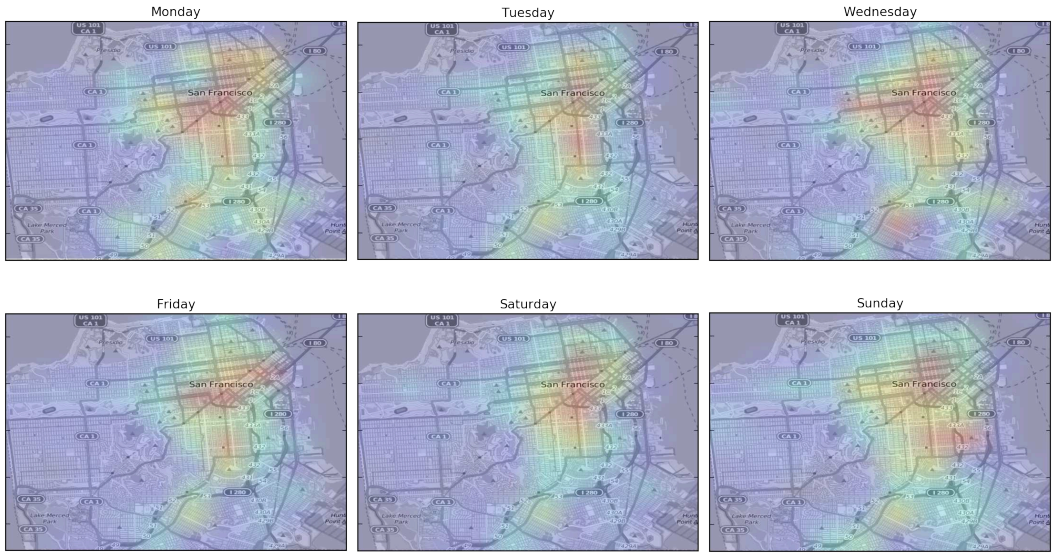


Fig. 4.10 Heatmap of vehicle thefts in different days of the week, in 2015

comprehensive of all categories of crimes. Closer to this effort is [123], which studies the seasonality of different kinds of crimes, but always with a focus on month periods. In our work we instead focused on short and medium-term cycles of few days or weeks, that could be helpful for short-term predictions. The availability of full timestamps for crimes is a must for a finer resolution of the analysis, e.g. at the day scale; works like [123, 125], for example, rely on the Vancouver open dataset, which exposes only the year and month of the happenings. In [118], the authors adopt the routine activity perspective to study spatial and temporal variations in crimes in Campinas, Brasil, on a number of crime categories. They do not find seasonalities in rape, robbery, burglary and theft, while finding a significant seasonality in homicide, confirming the results of [117]. Authors of [126] recently applied Fourier analysis to the study of violent crimes in South Africa. Their findings include a significant periodicity between 7 and 10 days, which is very alike the weekly period of assaults we found in San Francisco.

The usage of Lomb-Scargle periodograms in a scope of social interest is, to our knowledge, a novelty. Since its introduction in 1976 [119], it has been applied in its birth domain of astronomical studies [120], seismology [127], and, more recently, biology [128, 129], even though spectral analysis is a wide-spread technique in many scientific domains.

4.3 Monitoring the citizens' perception on urban security in Smart Cities

Smart cities are urban environments in which the municipality fosters the use of Information and Communication Technologies (ICTs) to engage citizens in city management and development [130]. A key aspect in Smart City governance is the participation of citizens in decision-making. The capability of ICTs to keep in touch citizens with municipality actors (e.g., assessors, area operators) is crucial for improving the effectiveness and efficiency of urban services.

Nowadays, *non-emergency data* on urban security issues are acquired by various Smart Cities all over the world. They consist of citizen warning reports that do not require an emergency response. Smart Cities allow citizens to signal these potential warnings to the local administrators through Web portals, apps, emails, and contact centers. To improve the citizens' quality of life, non-emergency data are worth analyzing because they represent the citizens' perception on urban security from different viewpoints. Based on the signaled warnings and their level of severity, the municipality can plan targeted actions on the urban area, which may vary according to the temporal evolution of the citizens' perception on non-emergency issues.

The main research contributions to this scenario can be classified as follows: (i) definition of collaborative models and open standards (e.g., Open311 [131]), to facilitate the development of smart applications [132, 133] and the cooperation between cities, (ii) design and development of smart cities platforms by private vendors [134], researchers [135, 136] and public administrators [137], and (iii) characterization and analysis of the perception of urban security sensed by users [138–141]. An overview of the key challenges in urban computing from the point of view of computer scientists is given in [142]. However, state-of the-art approaches are challenged by the increasing volume of analyzed data, which prompts the need for integrating data mining and Business Intelligence tools in non-emergency data analyses.

This section presents Non-Emergency Data Analyzer (NED), a new integrated data mining and Business intelligence environment targeted to the analysis of open non-emergency data on urban security issues acquired in a Smart City context. The NED system relies on an ensemble of open components, which are easily portable in different Smart City contexts. To profitably analyze the citizens' perception on urban security, non-emergency data are acquired, enriched with additional information

about the context of the warning reports (e.g., the related city area), and stored into a unique data repository. Next, two complementary analyses are performed: (i) Key Performance Indicator (KPI)-based analysis, and (ii) association rule discovery. In our scenario, KPIs are quantitative indicators measuring the level of warning of the citizens in a specific context, while association rules [143] are significant associations between warnings and contextual features (e.g., between a specific warning category and a city area). The aim of KPI generation is twofold: (i) to provide useful feedback to municipality users by generating informative dashboards on KPIs, and (ii) to automatically generate and notify alerting signals on critical situations. The aim of association rule extraction is to evaluate the strength of the correlations between warnings and contextual features and to automatically trigger alerts in case the extracted patterns highlight potentially critical situations [144]. For example, the city areas with maximal incidence of a specific warning category (e.g., *urban blight and renewal*) are those that may need maximal surveillance. On the other hand, a percentage decrease in the number of non-emergency warning signals from one year to the subsequent one may reveal a positive trend in the citizen perception on that specific urban security issue. The municipality can exploit such information, for example, to validate the effectiveness of the latest actions.

The effectiveness and usability of NED system have been evaluated on real non-emergency data acquired in a real Smart City context. We considered as analysis scenario the study of the non-emergency calls and emails received by the contact center of the Local Police Department of Turin, an important city located in the north-west of Italy, and we reported the results of a information and awareness campaign launched in Turin in September 2014 during a Europe-wide public event, namely the European Researchers' night.

This section is organized as follows. Section 4.3.1 describes the NED system, while Section 4.3.2 summarizes the experiments and the dissemination activities.

4.3.1 The NED system

Non-Emergency Data Analyzer (NED) is a new data mining and Business intelligence environment aimed at analyzing non-emergency open data acquired in a Smart City context. The main environment blocks are briefly introduced below and

schematized in Figure 4.11. A more detailed description is given in the following sections.

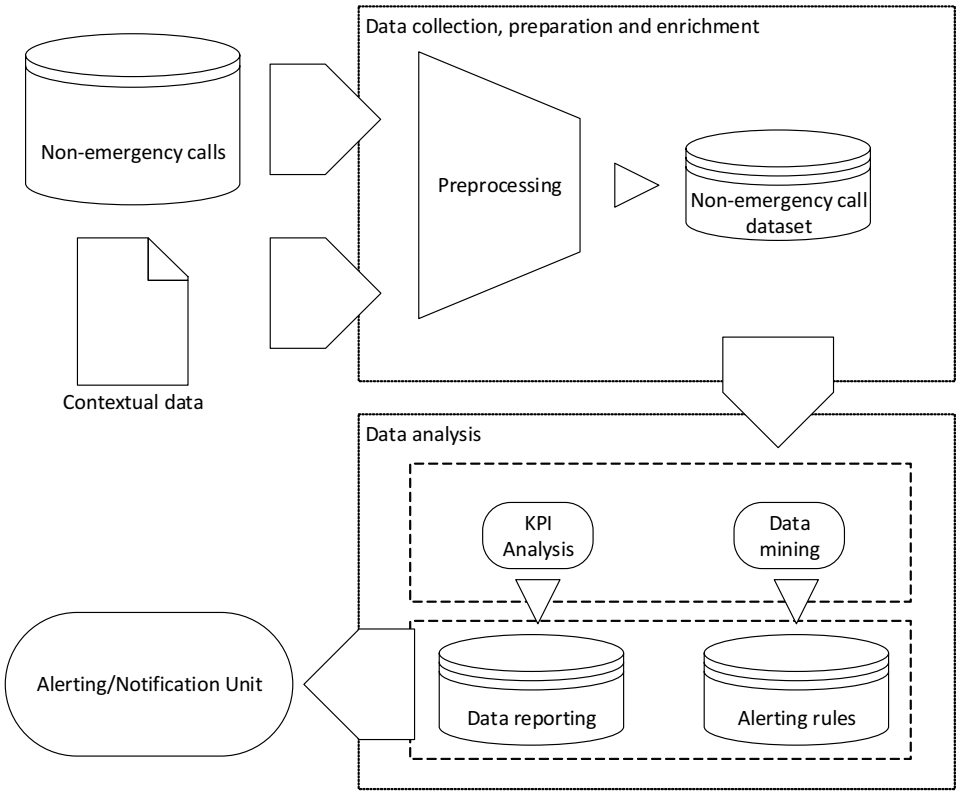


Fig. 4.11 Main architecture of NED system

Data preparation. To prepare non-emergency data to offline analyses, data is acquired, enriched through external open data sources with contextual data related to the warning reports, and then stored into a unique data repository (see Section 4.3.1).
Data analysis. From the prepared data, informative dashboards are generated based on a selection of Key Performance Indicators (KPIs), which are quantitative indicators reflecting peculiar data features. In parallel, association rules are extracted using an established data mining algorithm [145]. Dashboards and association rules provide complementary information on the perception of citizens on urban security (see Section 4.3.1).
Notification. Based on KPIs and association rules, alerts on critical situations are automatically generated and sent to the main municipality actors (e.g., city mayor, assessors, area operators). Notifications are selectively forwarded based on the role and authority of the municipality actors (see Section 4.3.1).

Category	Sub-category
Social tension	Vandalism
	Other
Civil tension	Youth gathering
	Disturbing behaviors
	Disturbance from dogs
	Disturbance from public venues
	Disturbance from other animals
	Noise nuisance
	Improper use of common areas
	Other
Urban quality	Urban blight and renewal
	Abandoned vehicles
	Other

Table 4.4 Categorization of non-emergency reports

The NED system implementation relies on the open FIWARE technologies[3] for Business Intelligence[4] and notification[5], while it uses the opensource RapidMiner suite v.5.0 [6] for data mining analyses.

Although the NED system is general and it can be applied to data acquired in different Smart City environments, hereafter we will consider as use-case scenario the analysis of the warnings perceived by the citizens of Turin.

Data preparation

The contact center of the Local Police department of the Turin Smart City acquires non-emergency data and periodically integrates them into a unique data repository to allow offline data analyses. Warning categories and sub-categories are assigned according to the classification reported in Table 4.4[7]. For example, citizens may signal noise coming from the street adjacent to their house due to the presence of a group of young persons raising their voice night-time. This warning may be classified as *Youth gathering*, which belongs to the more general category *Civil tension*.

[3] www.fi-ware.org
[4] SpagoBI v. 4.2, available at www.spagobi.org
[5] CAP Context Broker, available at http://forge.fi-ware.org/plugins/mediawiki/wiki/fiware/index.php/
[6] https://rapidminer.com
[7] http://aperto.comune.torino.it

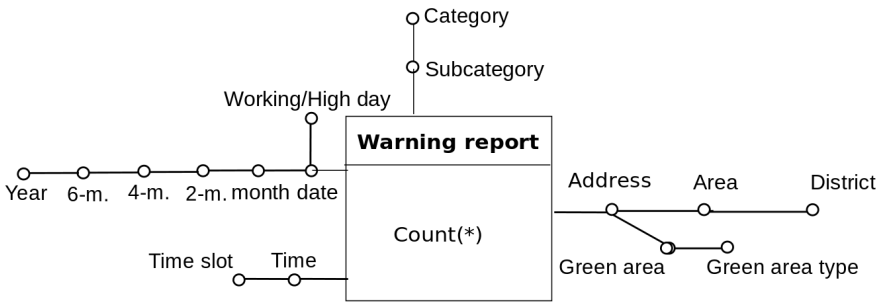


Fig. 4.12 Data warehouse dimensional fact model.

For each non-emergency report, the time and the address to which it refers to are also stored. In addition, the NED system enriches the initial data schema with extra contextual information acquired from external open data sources. Non-emergency data enriched with contextual information is stored in a data warehouse whose dimensional fact model [146] is depicted in Figure 4.12. More specifically,
(i) to analyze the *temporal distribution* of non-emergency data, the following time granularities are considered: *day*, *month*, *2-month*, *4-month*, *6-month*, and *yearly* time periods. Moreover, the day is classified as *working or high day*, and the warning report *time* is aggregated into the corresponding *daily time slot* (morning, afternoon, evening, or night).
(ii) to analyze the *spatial distribution* of non-emergency data, higher-level space granularities are also considered beyond the location addresses from which the warning report refers to. Specifically, each *address* is mapped to the corresponding *city area* and to the *city district* including that area. While the address and district are recorded by the contact center employees, the city area corresponding to the address is added as additional contextual feature to the final repository to aggregate data at an intermediate granularity level. For warning reports referring to a green area, the *green area name* and *type* (e.g., park, garden) are stored.

Furthermore, topological and demographic information about city areas, districts, and green areas are integrated in the repository as well. Topologies are used to graphically analyze the most significant spatial trends in non-emergency data, while demographic information is exploited to characterize non-emergency data according to the distribution of citizens per area, district, and gender (e.g., the number of male citizens per city area). Demographic statistics and topologies about the city areas and districts of Turin were acquired from the official GeoPortal of the Turin

municipality[8]. Topologies were encoded in GeoJSON, which is a standard format for encoding a variety of geographic data structures.

Data analysis

The prepared non-emergency data are analyzed to gain insights into the perceived citizen's warnings. The NED system performs two complementary (offline) analyses: (i) KPI analysis and (ii) data mining based on association rule analysis. The aim of data analysis is twofold: (i) to produce useful feedback to the municipality users by generating informative dashboards (using KPIs) and (ii) to automatically generate and notify alerting messages on critical situations (using KPIs and association rules). In the following we present KPI and association rule analyses.

KPI analysis. In Business Intelligence, the analysis of Key Performance Indicators (KPIs) is an established methodology [147]. KPIs help organizations define and measure progress toward organizational goals by monitoring most significant achievements. In our context, KPIs are quantitative indicators of the perception of citizens on urban security. To apply KPI analyses to data coming from a real scenario, we defined a set of KPIs related to the non-emergency data categories and sub-categories reported in Table 4.4. These KPIs analyze non-emergency data from two main viewpoints: (i) the temporal dimension and (ii) the spatial dimension of warning reports. More specifically, to reveal potentially critical situations in a specific urban area, we analyzed the incidence per city area/district/green area of the perceived warnings related to a given category/sub-category. Furthermore, to keep track of the temporal evolution of the citizen perception on urban security, we also computed the percentage variation between the number of calls received from a given city area/district/green area in a given time period (e.g., in year 2013) and those received in the same area in a preceding time period (e.g., in year 2012). To avoid bias due to the unbalanced distribution of citizens per city area/district, we normalized both the aforesaid KPIs by the total number of citizens per city area/district. For example, a positive yearly differential between the number of calls related to sub-category *Abandoned vehicles* coming from a given city area may reveal a negative trend, which may be due to an insufficient surveillance of the area. Thus, for this area, the municipality may plan targeted actions.

[8] https://www.comune.torino.it/geoportale

Association rule analysis. This step aims at discovering interesting associations between warning categories/sub-categories and context features in the form of association rules. Association rules may represent underlying correlations among data items which are hardly inferable by performing KPI analyses. In our context, they represent sets of warning categories/sub-categories and contextual features (city areas, time periods) that are strongly correlated with each other.

Many quality measures can be exploited to select the most interesting association rules. In our context of analysis, co-occurrences between warnings and contextual features are deemed to be reliable if they are frequent and the corresponding items are strongly correlated with each other. Hence, in the NED system we selected all the association rules whose support value is equal to or above a minimum support threshold *minsup*, and the lift value is above or equal to the minimum positive correlation threshold $min^{+}lift$ ($min^{+}lift > 1$) . The association rule extraction task is accomplished by using the Java-based RapidMiner implementation of the FP-Growth algorithm (Section 2.2.1). However, different association rule mining algorithms can be easily handled by the NED system as well.

Notification

To allow the municipality to constantly monitor non-emergency data, the NED system automatically generates periodic notification messages when KPIs and rules may indicate potentially critical situations.

To receive notifications, users are asked to subscribe to the NED notification service. Since municipality actors have different roles and authorities, the system grants users to receive notification messages based on their area of expertise. Specifically, NED first defines a set of roles and then assigns a geographical scope (e.g., at the level of the municipality, district, or area) to each role based on its authority. During the subscription phase, users have to indicate their role in the Municipality (e.g., city mayor, area operator). According to the role and authority the NED system assigns them a scope (i.e., at level of Municipality, District, or Area).

For example, authoritative actors at the level of the municipality (e.g., the city mayor) receive all the notifications at the levels of city and district, but not those at the level of city area, because the latter provide information at a too fine granularity

level. Conversely, each area operator is granted to receive only the notifications of the corresponding area.

To generate notification messages, a set of alerting rules is generated from the KPIs and association rules mined at the previous step (see Section 4.3.1). Each alerting rule has a scope and may be characterized by a severity level of the warning. Each user receives by email the subset of alerting rules pertinent to his role scope, enriched with dashboards on the corresponding KPIs.

For example, alerting rules related to vandalism acts to street furnitures in specific city areas are sent to all users whose scope is at level of city area and whose area of expertise comprises street furnitures (e.g., the area operators).

Alerting rules from KPIs. To generate alerting rules from KPIs, the NED system considers the percentage differential between consecutive time periods per city area, district, and green area. The aim is to monitor the variation of the citizens' perception on urban security over time in specific urban areas, and notify it to the municipality actors. Based on KPI values, the severity level of the warnings is classified as *stable*, *substantial* increase/decrease or *critical* increase/decrease, respectively. *Stable* KPI variations indicate a relatively stable trend in the citizen perception on urban security. Conversely, KPI variations falling in the other levels indicate moderately/significantly decreasing/increasing trends. Usually, stable trends do not require triggering alerting signals. Instead, levels *substantial* and *critical* trigger pre-alerting and alerting signals, respectively. Notifications are selectively forwarded to the municipality actors in charge of the specific issue, who may decide to perform targeted actions (e.g., periodic surveillance of specific areas, redevelopment of green areas).

For each severity level of the warnings (stable, substantial, critical), the corresponding KPI value ranges are analyst-provided. In our experiments, we classified the KPI variations in the range [-2%; 2%] as *stable*, those in the ranges [-5%;-2%] and in [2%;5%] as *substantial* decrease and increase, respectively, and those below -5% and above 5% as *critical* decrease and increase, respectively.

Alerting rules from Association Rules. Association rules represent potentially critical situations arising from non-emergency data when a specific city area/district/green area appears to be strongly correlated with a given combination of categories/subcategories. To define alerting rules, the NED system first extracts the association rules from non-emergency data by enforcing *minsup* equal to 1% (i.e., the implications must hold for at most 1% of the source data) and $min^{+}lift$ equal to 10 (i.e., the

rule antecedent and consequent must be strongly correlated with each other). The top-K ranked rules in order of decreasing lift value (where K is an analyst-provided parameter) are exploited to automatically generate notifications to the municipality actors.

Rules represent implications between green areas/city areas/districts (one or more) and a specific warning category/sub-category. They can be exploited to trigger alerts related to a specific urban area. To trigger more detailed alerts, more complex rules can be extracted by combining the spatial information about the green area/city area/district with the day or the time slot at which the critical situation occurs.

For example, rule {(Green Area, Pellerina Park)} $\Rightarrow$ {(Sub-category, Vandalism)} may indicate a alerting situation related to a specific green area of Turin. Instead, rule {(Green Area, Pellerina Park), (Time Slot, night)} $\Rightarrow$ {(Sub-category, Vandalism)} specializes rule {(Green Area, Pellerina Park)} $\Rightarrow$ {(Sub-category, Vandalism)} by providing also the information about the time slot at which most of non-emergency data were received.

4.3.2 Analysis scenario

The NED system was validated on real non-emergency data acquired in the Turin Smart City environment. More specifically, we analyzed an open non-emergency call dataset consisting of 4,672 calls received by the contact center of the Local Police Department of Turin in years 2012 and 2013, which is available in open municipality portal AperTo[9]. The main dataset attributes are described in Section 4.3.1. The experiments were performed on a quad-core 3.30 GHz Intel Xeon workstation with 16 GB of RAM, running Ubuntu Linux 12.04 LTS.

Two representative examples of dashboards generated from the KPIs defined in the NED system are reported in Figures 4.13 and 4.14. They show the incidence of calls for disturbance from public venues per district in years 2012 and 2013, respectively. Districts are colored with a 4-level scale ranging from blue (low percentage of calls) to red (high percentage of calls). District 1 (*Centro Crocetta*) corresponds to the city center and it is characterized by an averagely high number of calls in both years. Since the level of warning perceived by citizens in this district remains critical over the two years, the municipality would need to undertake further

[9] http://aperto.comune.torino.it

actions. Oppositely, in district 8 (*Borgata Lesna*) the number of calls decreased from year 2012 to year 2013 thus the issue appears to be overcome.

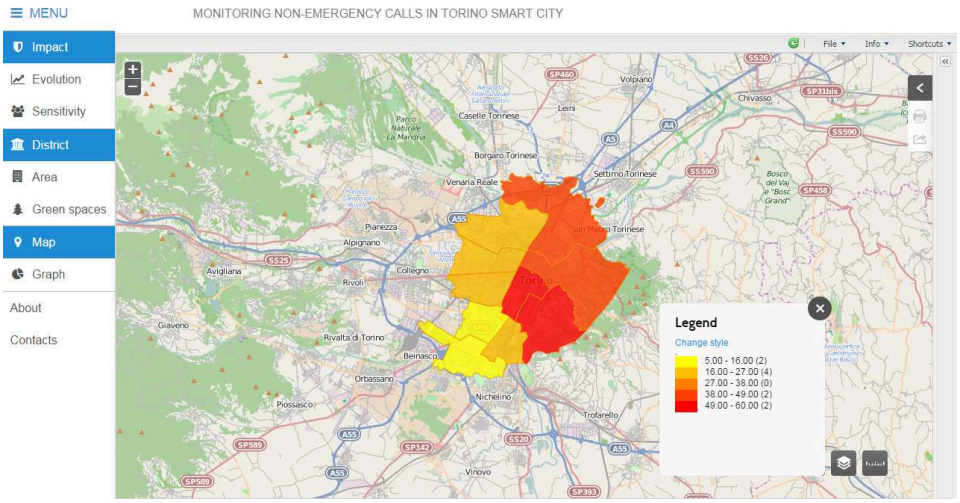


Fig. 4.13 Incidence of disturbance from public venues per district in year 2012.

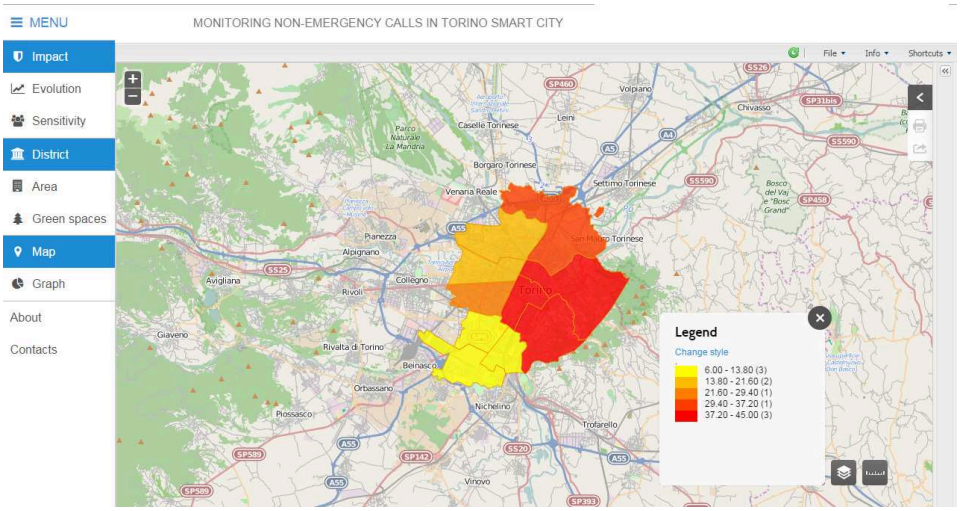


Fig. 4.14 Incidence of disturbance from public venues per district in year 2013.

Concerning district 1, the NED system extracted the following rules by enforcing *minsup*=1% and $min^{+}lift$=10:

{(District, 1)} $\Rightarrow$ {(Category, Civil tension)} (sup=9%, conf=60%, lift$>10^5$)

{(District, 1)} $\Rightarrow$ {(Sub-category, Disturbance from public venues)} (sup=3%, conf=19%, lift$>10^5$)

These rules indicate that 60% of the warnings raised the citizens of district 1 belong to category *Civil tension*. Among these rules, *Disturbance from public venues* appear to be most correlated subcategory (i.e., the rule lift is maximal w.r.t. all the rules in the form {(District, 1)} $\Rightarrow$ {(Subcategory, *)}). Hence, the local police may increase night-time surveillance close to restaurants, pubs, and discos located within the city center.

4.4 Summary

In the exploratory study of Section 4.1, we shed light on a number of features of the social medium during a mass emergency. First, we found that only a small fraction, under 1%, of tweets is georeferenced, which is in line with the numbers in [109, 110]. This implies that a spatially bounded search, like the one that has produced our Texas dataset, excludes from the results most of the messages and is therefore not recommendable in the handling of an emergency. Furthermore, less than a fifth of these tweets had precise GPS coordinates associated with them. Moreover, we have shown as the kind of results that return from a geographical search belong to special categories of users or services, e.g. tweets from third party apps like Instagram or official weather warnings. In the case of the Ischia earthquake, these tweets did not resemble, in their contents and in the vocabulary used, the entirety of the community. Researchers and practitioners should therefore be aware of the bias introduced in making a search of this sort.

In Section 4.2, we have analyzed the open dataset of San Francisco crimes and its temporal evolution, proposing a methodology for seeking for seasonal patterns. The spectral analysis has brought to evidence a number of interesting insights that were not immediately clear from a simple look at the curves, hidden by the complexity of the data. For example, we discovered the tendency of some categories of crime to repeat on a monthly basis, like robberies, and how almost all show a weekly period, like vehicle thefts.

In Section 4.3, we have proposed NED (Non-Emergency Data Analyzer). NED is a new data mining and Business Intelligence environment aimed at supporting the analysis of non-emergency data acquired in a Smart City context. Specifically, the proposed system aims at supporting Smart City municipalities in studying the perception of citizens on urban security. To this aim, it performs both KPI-based and

data mining analyses. The analytical results are selectively notified to municipality actors based on their role, authority, and area of expertise. As case study, we evaluated the applicability of the proposed system in a real Smart City context, i.e., the analysis of the non-emergency reports received by the contact center of the local police department of Turin (Italy). The results demonstrate the effectiveness of the proposed systems in monitoring citizens' perception on different warnings and thus its ability to alert the municipality as soon as a potentially critical situation emerges.

These studies highlight the crucial role played by voluntary reports made by the citizens, but do not exclude the integration of new forms of interaction. The monitoring of a social medium is subordinated to the ability of retrieving messages from the area of interest. Future research should aim at searching for tweets related to the emergency in ways that do not rely on spatial information, like for example [148]. As evidenced in the Ischia scenario, these methods should be able to identify a stream of tweets that may not surge in the trends and make not exclusively use of hashtags, as already suggested in [108]. Another interesting approach is that of improving citizenry's participation through gamification, like in [149], even though the use in a mass emergency would be very limited.

The methodology of Section 4.2 can uncover valuable information, but its results need to be digested in a way that can be easily understood by decision-makers, before being integrated in a framework like NED. As an example, the spectral analysis can be the first component of an algorithm aimed at predicting near-future crime events, like the one in [150]. The discovered temporal seasonalities and the stationarity of the time series support the design of predictive models based on weekly and monthly patterns.

4.5 Relevant publications

[102] Luca Venturini and Evelina Di Corso. Analyzing spatial data from twitter during a disaster. In *Proceedings of 2017 IEEE International Conference on Big Data*, pages 3779–3783. IEEE, 2017

[103] Luca Venturini and Elena Baralis. A spectral analysis of crimes in san francisco. In *Proceedings of the 2nd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, page 4. ACM, 2016

[104] Luca Cagliero, Tania Cerquitelli, Silvia Chiusano, Pierangelo Garino, Marco Nardone, Barbara Pralio, and Luca Venturini. Monitoring the citizens' perception on urban security in smart city environments. In *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pages 112–116. IEEE, 2015

Chapter 5

Conclusion

This dissertation contributes to the state of art in Big Data developments in several ways.

To scale distributed classification algorithms beyond current limits, we needed to know what these limits are. In Chapter 2, we found that a large domain of features, namely a very large number of distinct categories, is a tough challenge for current distributed implementations. The proposal of a Distributed Associative Classifier tackles these limits, raising the bar for the training of classification algorithms for a large Volume of data. Moreover, it also improves the scaling on another important dimension, Velocity, as the training of the model of DAC requires significantly less time than state-of-the-art solutions, while keeping the prediction time the same or even lower.

To assess the state-of-practice of a Big Data machine learning framework, we field-tested Apache Spark MLlib algorithms on a real Big Data scenario, like the one of computer networks measurements. This scenario is very challenging especially on the Variety of the data, as the behaviour, the size, or the expected measurements of a network are not known in advance, change from network to network and do not hold for long periods of time. Therefore, a Big Data solution in such a scenario must cope with the absence of a ground truth, adapt to the domain expert and the application that is calling it, and react to changes. We successfully tested our proposed solution, SeLINA, together with domain experts, building a tool that answers as ad-hoc solutions tuned by experts, but learns all the needed setup from the data.

To validate the effectiveness of a data science process in improving urban safety, we first investigated on the potential use of social messages and on the amount and quality of spatial information they contain, in the context of a mass emergency. A spatial footprint is essential in querying a social medium like Twitter for messages coming from a delimited area, e.g. a city. Unfortunately, a very small percentage of tweets provide this information, and of this another small portion contains precise coordinates. Moreover, from our study, presented in Chapter 4, it appears that this subset of tweets is dominated by digital marketing and automated bots, which are not interesting when sensing the opinion of the citizenry or searching for requests for intervention. However, "traditional" sources can still provide important insights on the data. In the following of the chapter, we presented a methodology to mine temporal patterns in social phenomena, and applied it to crimes in San Francisco. We also proposed to mine patterns and track the evolution of a phenomenon with association rules and KPIs, respectively, and integrated these techniques in an automated system, NED.

Through all the chapters, we stressed the major importance of the "fourth" of the three Vs of Big Data, that is Value. In Chapter 2, Value is preserved maintaining a readable model, whereas the only scalable alternative needed the features to be irreversibly hashed. Moreover, the Value of the model is augmented, with an higher quality of predictions. In Chapter 3, the Value relies in the insights provided to the domain expert, than can help him or her detecting anomalies in the network or even attacks, or understand how the network is legitimately used. All this, without his or her intervention on the algorithm or its parameters, in a self-learning fashion that can also automatically trigger model updates, if desired. In Chapter 4, Value is generated by giving actionable insights to decision makers, timely and automatically. This can lead to the definition of policies to fight the issues found, as for example an augment of vandalism in green areas.

Future work can improve our findings in several ways. The work on a scalable associative classifier can be extended with a mixed strategy, that combines both the data and the search split approach of frequent itemsets miners. The integration of multiple machine learning algorithms in a Big Data pipeline should be improved with the addition of a self-learning feature selection technique. The potential of such a framework make it a promising tool for other fields of application as well, and it should thus be tested in different scenarios. The potential applications of Data Science to urban safety are numerous. The methodology proposed to find temporal

patterns needs to be further developed and integrated with tools that can make its results comprehensible to the final users of the analysis. A very promising field of application for the periodogram would be as a first step in a machine learning pipeline, as an automated tool to produce meta-features for time series.

Appendix A

Further investigations on the performance of distributed FIM algorithms

This Appendix contains a set of experiments and results that investigate further the performance of the algorithms surveyed in Section 2.2. In Section 2.2.4, we have already analyzed the empirical limitations of each solution, summarized in Section 2.2.5. With this Appendix, we inspect in detail the reasons of the above-mentioned limitations, by measuring several internal metrics of the algorithms' performance like load balancing, communication cost or resource utilization. The results confirm and strengthen the findings already outlined in Section 2.2.4, and may be of possible use to the researcher interested in improving the algorithms that are subject of the study.

The contents of this Appendix were originally published in [3]. The Appendix is structured as follows. Section A.1 gives details on the horizontal scalability of the algorithms. Section A.2 experiments with different configurations of the underlying framework. Section A.3 analyzes the impact on the total execution time of the different phases of the algorithms. Section A.4 studies the performance of the algorithms in two real use case scenarios. Section A.5 analyzes the load balancing of the algorithms. Finally, Section A.6 studies the communication costs.

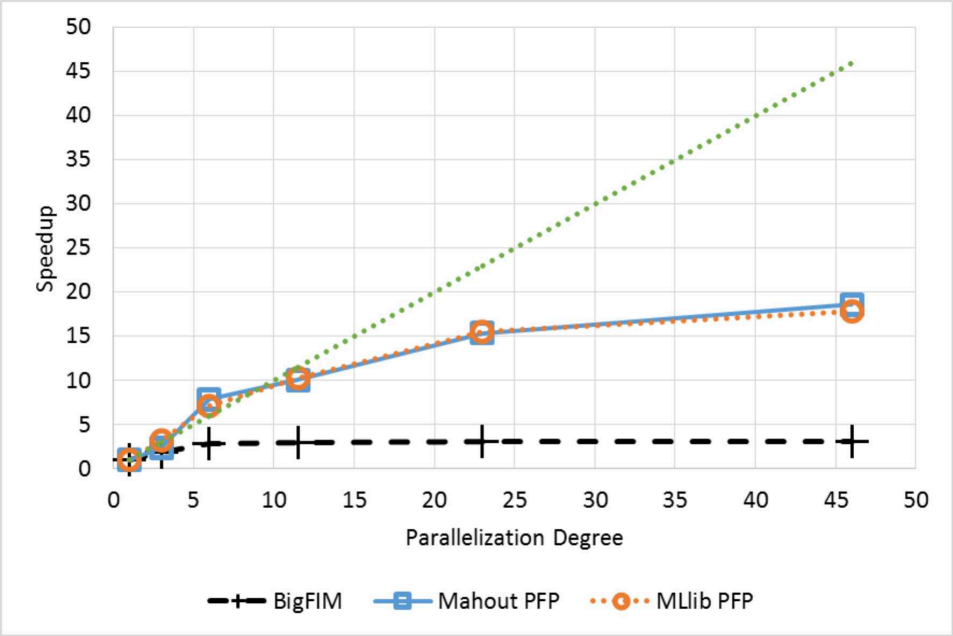


Fig. A.1 Speedup with different parallelization degrees (Dataset #14, *minsup* 0.4%, the green line represents the optimal behavior.)

A.1 Scalability in terms of parallelization degree

We analyzed the speedup by running the same mining problem with increasing numbers of parallel tasks. The dataset selection and the *minsup* parameter choice are difficult since we need to identify a mining problem satisfying two conditions: (i) allowing all the executions to complete with any number of parallel tasks, and, at the same time, (ii) being very demanding so that the distributed framework is actually exploited. We selected *minsup* 0.4% and Dataset #14 (see Table 2.3) to be light enough for condition (i) and demanding enough for condition (ii). The speedup of a configuration with a parallelization degree equal to p is computed as

$$speedup(paral\_degree = p) = \frac{Exec\_Time(paral\_degree = 1)}{Exec\_Time(paral\_degree = p)}$$

Ideally, the speedup should be equal to the parallelization degree p itself, i.e., increasing the number of resources (parallel tasks) of a factor p, should lead to a speedup equal to p.

Figure A.1 shows the speedup results. A parallelization degree equal to 1 corresponds to the minimal computational resource setting. In our case, it matches a configuration with only two parallel independent tasks. Its executiion time is used as reference to compute the speedup related to the other, more robust, configurations. For instance, the speedup related to a parallelization degree equal to five is measured through a configuration exploiting five times the amount of resources related to the basic configuration (i.e. ten parallel independent tasks).

In this experiment, it is clear that the FP-Growth-based implementations provide a better speedup. BigFIM, on the contrary, is not able to leverage a number of parallel tasks higher than 6. Because of the size of the dataset, DistEclat is not able to perform the mining.

A.2 Impact of framework and hardware configurations

We performed a set of experiments to test the behavior of the algorithms with different framework and hardware configurations to identify possible bottlenecks. We selected a set of configurations characterized by different combinations of (i) parallelization degree, (ii) computational power (cores per task) and (iii) memory (memory per task). The selected configurations are reported in Table A.1. Conf. 1 is considered the reference configuration. The differences of the other configurations with respect to Conf. 1 are reported in bold in Table A.1.

Conf. 1, Conf. 2, and Conf. 3 are used to evaluate the impact of the computational power (in terms of number of cores per task), Conf. 1 and Conf. 4 are used to evaluate the impact of the available memory, while Conf. 1, Conf. 5, and Conf. 6 are used to compare the impact of the previous features with respect to the parallelization degree. Experiments have been performed on dataset #1, with a fixed minsup set to 0.2%, and on dataset #5, with a minsup value set to 1.5%.[1] The main difference between the two datasets is the average transaction length (10 attributes per transaction in Dataset #1, 50 attributes per transaction in Dataset #5). In this way, it is possible to evaluate if the impact of hardware configuration is affected by data distribution.

[1]This support value is higher than that used in Section 2.2.4 to allow the execution of the experiments also for the BigFIM algorithm with all the selected hardware configurations.

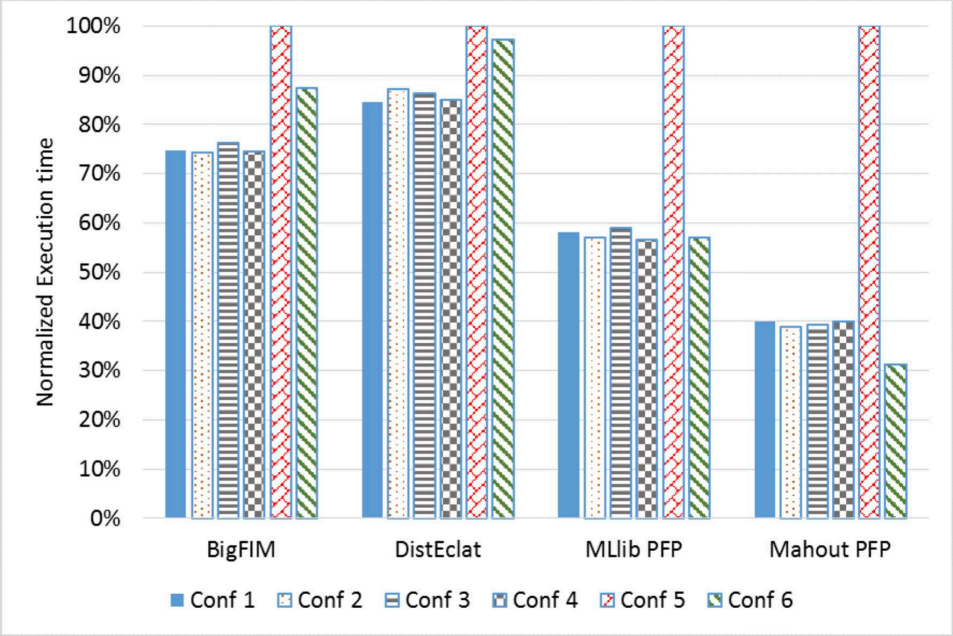


Fig. A.2 Performances with different hardware configurations (Dataset #1, *minsup* 0.2%)

For DistEclat, in the experiments with Dataset #1, we were forced to reduce the dataset size to 1/10. In this way we were able to complete its experiments in all configurations (please note that the intra-algorithm comparison is still possible in percentage). As evidenced in Section 2.2.4, DistEclat does not suit large transactions length and, for this reason, we were not able to run any experiment with Dataset #5.

Configuration name	Parallelization Degree	Number of cores per task	Memory per task (GB)
Conf. 1	5	1	1.5
Conf. 2	5	**2**	1.5
Conf. 3	5	**3**	1.5
Conf. 4	5	1	**3**
Conf. 5	**2**	1	1.5
Conf. 6	**10**	1	1.5

Table A.1 Framework and Hardware configurations

Figure A.2 and A.3 present the normalized execution time for each algorithm over different configurations on Dataset #1. For each algorithm, the normalized

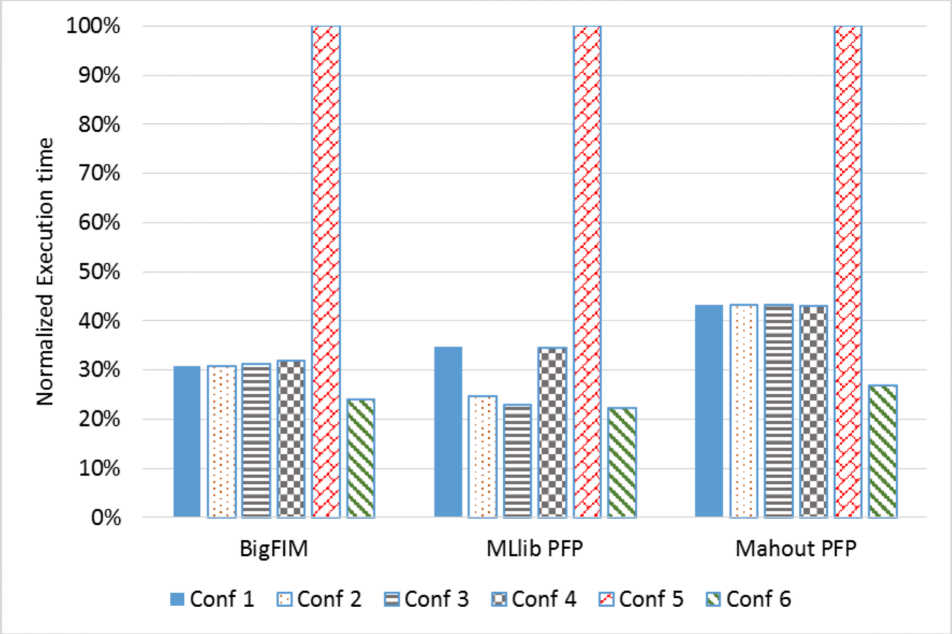


Fig. A.3 Performances with different hardware configurations (Dataset #5, *minsup* 1.5%)

execution time is computed by dividing the execution time of each configuration by the execution time of the slowest configuration. Hence, for each algorithm, 100% is associated with the slowest configuration.

The comparison of Conf. 1, 2, and 3 shows that the number of cores per task does not impact on the execution time of the algorithms. Only in the second experiment (Figure A.3), MLlib PFP seems to take advantage of the superior computational power. This means that the work assigned to each task, in the majority of the cases, can be performed by one single core. Hence, increasing the number of cores per task is not much effective.

Similarly, the main memory assigned to each task does not impact on the execution time of the algorithms (see Conf. 1 and 4). Specifically, the main memory per task impacts only on the size of the sub-problem that can be managed by each task, but not on its execution time. Hence, a proper setting of the main memory per task is required to be able to complete the execution and obtain the results, but not for its efficiency and performance. Finally, Configurations 1, 5, and 6 confirm that the parallelization degree is the most important factor affecting the execution time of the considered algorithms, as deeply investigated in Section A.1, and especially in the cases with a large amount of attributes per transactions Figure A.3.

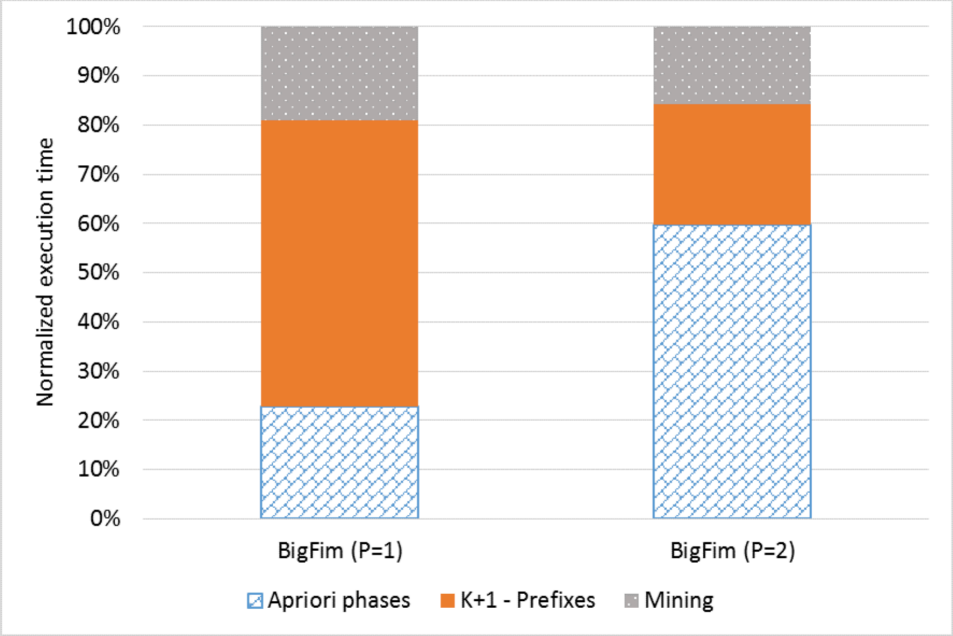


Fig. A.4 BigFIM: Execution time of its phases

A.3 Execution time breakdown into phases

To investigate possible bottlenecks inside multi-phase algorithms, we compared the execution times related to each phase. Specifically, for each algorithm, we computed the percentage of time associated with the execution of each phase with respect to the total execution time of the algorithm.

We selected Dataset #1 and we set minsup to 0.15%, which allowed us to complete the full set of experiments with all algorithms.[2]

As reported in Figure A.4, for BigFIM the length of the prefixes extracted in the first phase strongly affects the weight of that phase in the overall process. For DistEclat (Figure A.5), instead, the difference is not that heavy.

The last phase of both algorithms (i.e. the top dotted part on the graphs), that is associated with the mining of the itemsets with a length greater than the prefix-length threshold, has a lower impact on the execution time of the algorithms, especially when a higher prefix threshold is set. These data, and the failures reported in the

[2]In this set of experiments, we used a smaller configuration of our cluster to guarantee network isolation. For this reason, we had to use a reduced version of Dataset #1 (1/10) for DistEclat, very sensitive to memory issues.

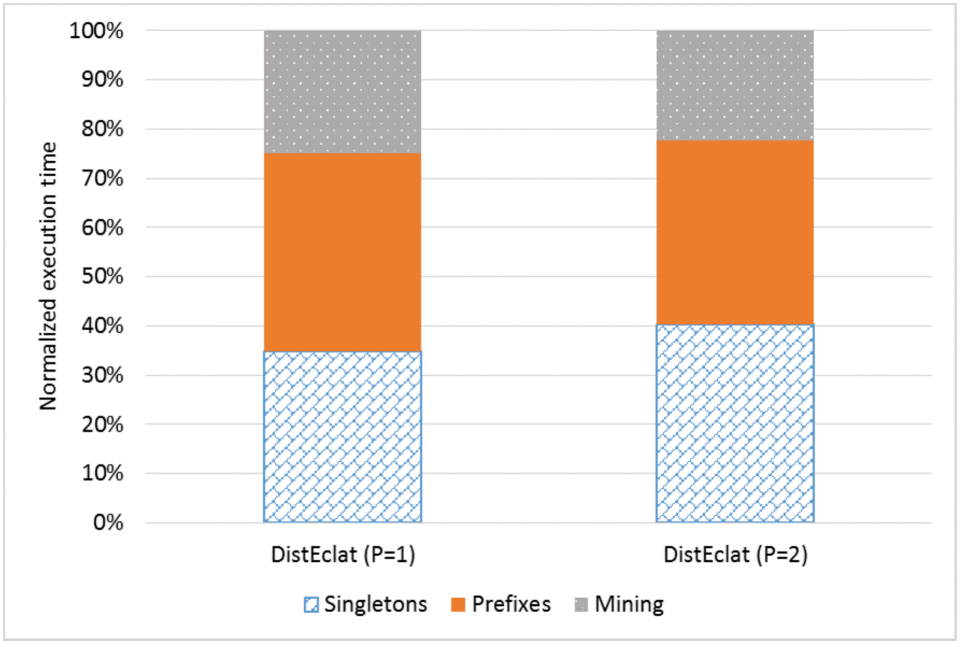


Fig. A.5 DistEclat: Execution time of its phases

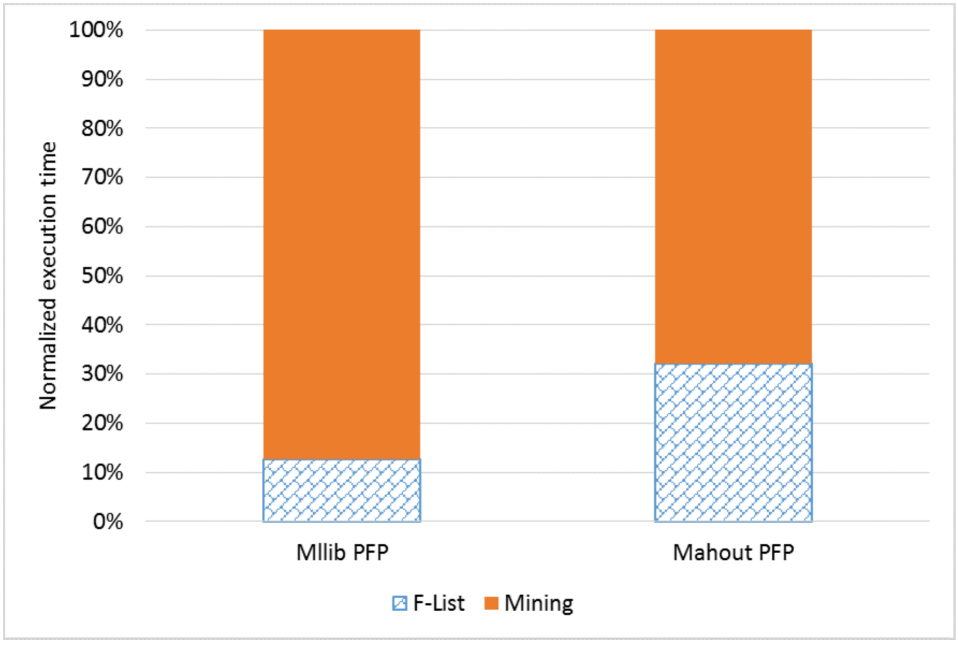


Fig. A.6 Mahout and MLlib PFP algorithms: Execution time of their phases

experiments of the previous subsections, indicate that the first two phases are the main bottlenecks for both algorithms. For BigFIM, each phase is strongly exposed to memory issues, as resumed in Table A.2. The experiments demonstrate that the Apriori phase is particularly challenging. For DistEclat, instead, the very first stage is dedicated to the mining of 1-itemsets and it is mostly affected by high reading and communication costs. However, we have experienced some memory issues, which are probably related to the handling of the tidlists. The other stages, instead, are more likely to be affected by memory constraints.

Figure A.6 reports the results for the PFP implementations. Mahout PFP spends 1/3 of the time in the first phase, in which the F-list is generated, while MLlib PFP is on the second phase for almost 90% of the time.[3] The difference between the two approaches is motivated by the less elastic handling of the different jobs by Hadoop with respect to the Spark framework. Even if, especially for the Mahout PFP, the F-list generation could take a good amount of time, it is not a possible bottleneck of the whole mining. Firstly, it is a very flat WordCount-like application, characterized by high reading and communication costs, and secondly, it has never shown to be a point of failure in any previous experiment. From Figure A.6, the bottleneck for the FP-growth-based algorithms is the itemset extraction phase (i.e., the second phase of both MLlib PFP and Mahout PFP), strongly constrained by memory.

All the algorithms and the majority of their phases are strongly bottlenecked by memory issues. Memory availability is the main factor affecting the ability of each algorithm to complete the itemset extraction. Interestingly, we have seen that it does not affect the execution time performances (Subsection A.2).

We have also tried to track and measure the resource utilization in terms of disk usage (read and write phases of HDFS), network communication, and CPU usage. Please note that the values are normalized with respect to the maximum resource utilization. Specifically, Figures A.7a and A.7b report the achieved results for BigFIM and DistEclat, while Figures A.8a and A.8b show the results for the PFP-based implementations.

Figures A.7a and A.7b highlight two main peaks in resources utilization for BigFIM and DistEclat.[4] For BigFIM the first peak is related to the Apriori phase

[3]Please note that we have forced the materialization of all the preliminary results with the Spark-based MLlib PFP.

[4]For the sake of clarity we have used a prefix length of 1 to enhance the effect of the last mining phase.

Algorithm	Phases	Bottleneck
FP-growth-based Algorithms	F-List	Reading and Communication Cost
	FP-Tree Mining	Memory
BigFIM	Apriori Phases	Memory
	K+1 Prefixes	Memory
	Eclat Mining	Memory
DistEclat	Singletons	Read. and Comm. Cost + Memory
	Prefixes	Memory
	Eclat Mining	Memory

Table A.2 Stage Bottlenecks

and the k+1-prefixes generation, while the second is related to the depth-first mining. Similarly, for DistEclat the first peak is related to the singleton and prefixes generation while the second to the depth-first mining.

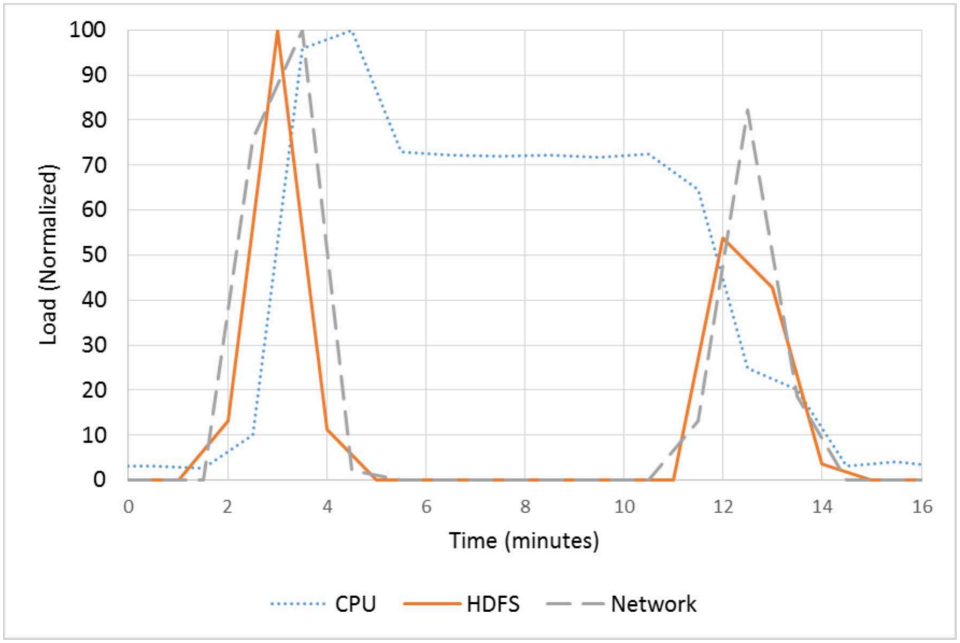
In Figure A.8a it is shown the behavior in terms of resource utilization of Mahout PFP. The first peak in terms of HDFS and Network communication is related to the initial F-list generation. After that, the tree exploration starts and the CPU is more exploited. The last peaks are related to the aggregation job used to extract the top-k frequent closed itemsets. Figure A.8b shows instead the MLlib PFP resource usage. Also the MLlib implementation of PFP is characterized by an initial peak in terms of HDFS operations followed by a peak in terms of CPU usage, associated with the intensive mining phase.

A.4 Real use cases

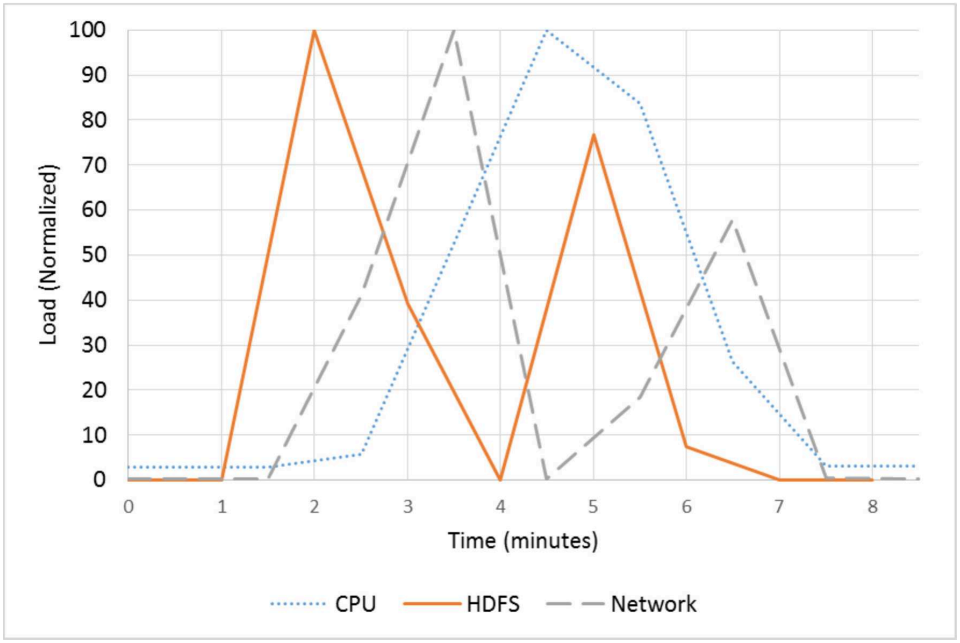
In the following, we analyze the performance of the mining algorithms in two real-life scenarios: (i) URL tagging of the Delicious dataset and (ii) network traffic flow analysis. The characteristics of the two datasets are reported in Table A.3.

ID	Name	Num. of different items	Avg. *len* per transaction	Transactions	Size (GB)
15	Delicious	57,372,977	4	41,949,956	44.5
16	Netlogs	160,941,600	15	10,729,440	0.61

Table A.3 Real-life use-cases dataset characteristics

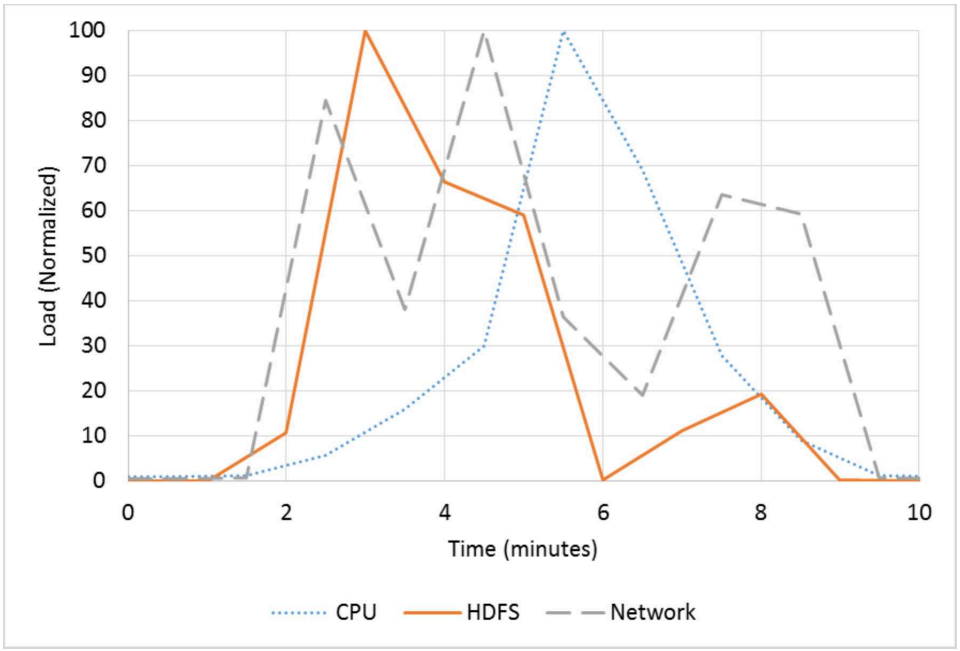


(a)

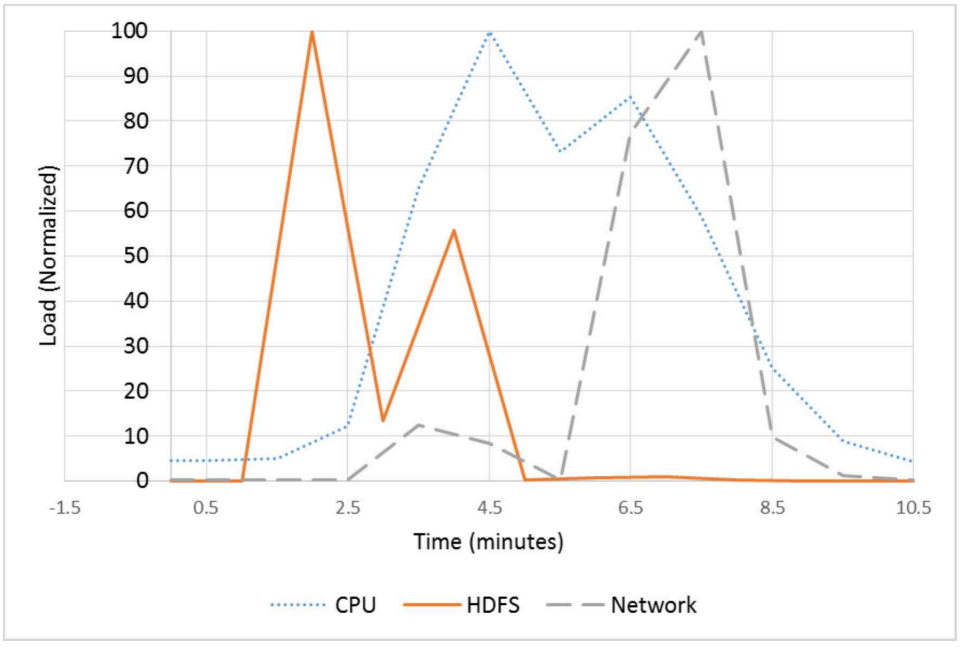


(b)

Fig. A.7 Resource utilization of (a) BigFIM (b) DistEclat



(a)



(b)

Fig. A.8 Resource utilization of (a) Mahout PFP (b) MLlib PFP

A.4.1 URL tagging

We evaluated the selected algorithms on the Delicious dataset [151], which is a collection of web tags. Each record represents the tag assigned by a user to a URL and it consists of 4 attributes: date, user id (anonymized), tagged URL, and tag value. The transactional representation of the Delicious dataset includes one transaction for each record, where each transaction is a set of four pairs (attribute, value), i.e., one pair for each attribute. The dataset stores more than 3 years of web tags. It is very sparse because of the huge number of different URLs and tags. Additional characteristics of the dataset are reported in Table A.4.

This experiment simulates the environment of a service provider that periodically analyzes the web tag data to extract frequent patterns: they represent the most frequent correlations among tags, URLs, users, and dates. Many different use cases can fit this description: tag prediction, topic classification, trend evolution, etc. Their evolution over time is also interesting. To this aim, the frequent itemset extraction has been executed cumulatively on temporally adjacent subsets of data, whose length is a quarter of year (i.e., first quarter, then first and second quarter, then first, second, and third quarter, and so on, as if the data were being colleted quarterly and analyzed as a whole at the end of each quarter). The setting of *minsup* in a realistic use-case proved to be a critical choice. Too low values lead to millions of itemsets, which become useless as they exceed the human capacity to understand the results. However, too high *minsup* values would discard longer itemsets, which are more meaningful as they better highlight more complex correlations among the different attributes and values. Because of the high sparsity of the dataset, we identified the setting *minsup*=0.01% as the best tradeoff.

Table A.4 reports the cumulative number of transactions for the different periods of time (i.e., the cardinality of the input dataset) and the number of frequent itemsets extracted with a fixed *minsup* of 0.01%, while the execution times of the different algorithms are shown in Figure A.9.

MLlib PFP consistently proves to be the fastest approach, with DistEclat following. However, while DistEclat is slightly faster than MLlib PFP only with the first, smallest dataset (up to Dec 2003, with 150 thousands transactions), when the dataset size increases, DistEclat execution time does not scale. DistEclat eventually fails for the final 40-million-transaction dataset of Dec 2005, due to memory exhaustion.

Up to year, month, quarter	Number of transactions	Number of frequent itemsets
2003 Dec, Q4	153,375	7197
2004 Mar, Q1	489,556	6013
2004 Jun, Q2	977,515	5268
2004 Sep, Q3	2,021,261	5084
2004 Dec, Q4	4,349,209	4714
2005 Mar, Q1	9,110,195	4099
2005 Jun, Q2	15,388,516	3766
2005 Sep, Q3	24,974,689	3402
2005 Dec, Q4	41,949,956	3090

Table A.4 Delicious dataset: cumulative number of transactions and frequent itemsets with *minsup* 0.01%.

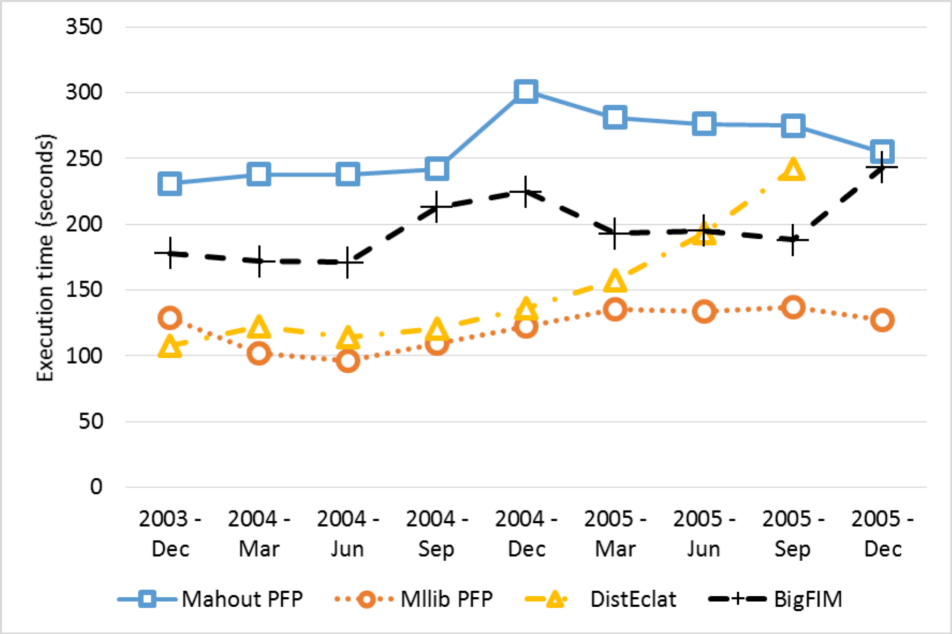


Fig. A.9 Execution time for different periods of time on the Delicious dataset (*minsup*=0.01%)

BigFIM and Mahout PFP consistently provide 2 to 3 times higher execution times. Apart from DistEclat, all algorithms complete the task with similar performance despite increasing the dataset cardinality from 150 thousand transactions to 41 millions, thanks to the constant relative *minsup* threshold which reduces the number of frequent itemsets for decreasing density of the dataset. Hence, MLlib PFP is the best

choice for this dataset characterized by short transactions (the transaction length is 4).

A.4.2 Network traffic flows

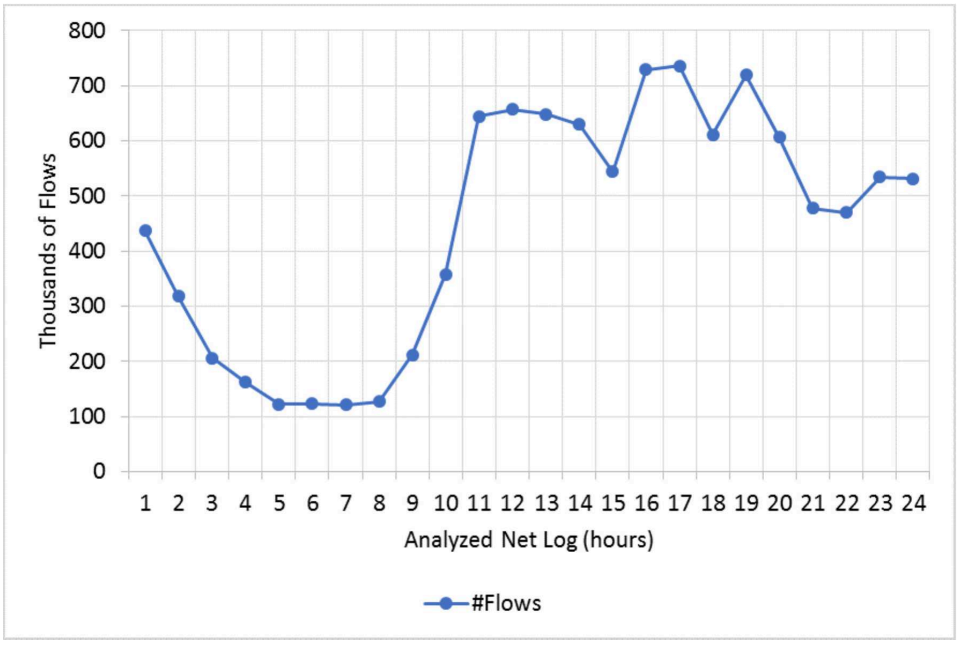


Fig. A.10 Number of flows for each hour of the day.

This use case entails the analysis of a network environment by using a network traffic log dataset, where each transaction represents a TCP flow. A network flow is a bidirectional communication between a client and a server. The dataset has been gathered through Tstat, in a way that is analogous to the one in Section 3.5. Each transaction of the dataset is associated with a flow and consists of pairs $(flow\ feature, value)$. These features can be categorical (e.g., TCP Port, Window Scale) or numerical (e.g., RTT, Number of packets, Number of bytes). Numerical attributes have been discretized by using the same approach adopted in [152]. Finally, we have divided the set of flows (i.e., the set of transactions) in 1-hour slots, generating 24 sub-datasets. The number of flows in each sub-dataset is reported in Figure A.10.

In this use case, the network administrator is interested in performing hourly analysis to shape the hourly network traffic. Hence, we evaluated the performance of the four algorithms, comparing their execution time, on the 24 hourly sub-datasets.

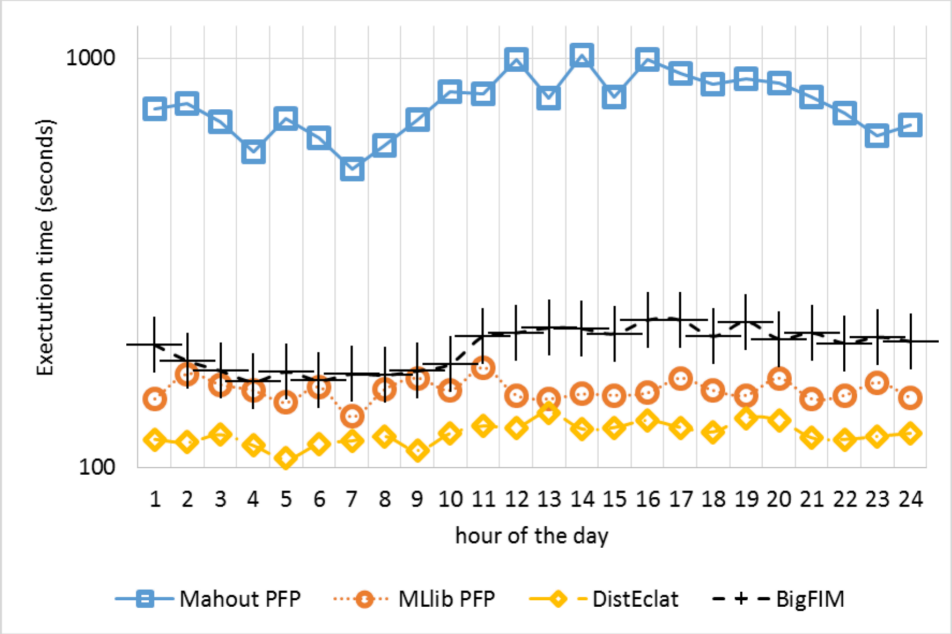


Fig. A.11 Execution time of different hours of the day. (dataset 16, *minsup*=1%)

For all the 24 experiments *minsup* was set to 1%, which was the tradeoff value allowing all the algorithms to complete the extraction.

The results are reported in Figure A.11, where the performance of the different approaches show a clear trend: DistEclat always achieves the lowest execution time, followed by MLlib PFP and BigFIM. Mahout PFP is the slowest. The execution time is almost independent of the dataset cardinality, as it slightly changes throughout the day. The low dataset size (less than 1 Gigabyte overall) and cardinality (less than 1 million transactions) make this the ideal use case for DistEclat, which strongly exploits in-memory computation.

A.5 Load balancing

We analyzed load balancing on a 1-hour-long subset of the network log dataset (Table A.3) with a fixed *minsup* of 1%. We consider the most unbalanced jobs of each algorithm and compare the execution times of the fastest and the slowest tasks. To this aim, we are not interested in the absolute execution time, but rather in the normalized execution times, where the slowest task is assigned a value of 100, and the fastest task is compared to such value, as reported in Figure A.12.

Hour of the day	Number of transactions	Number of frequent itemsets
0.00	437,417	166,217
1.00	318,289	173,960
2.00	205,930	163,266
3.00	162,593	166,344
4.00	122,102	157,069
5.00	123,683	164,493
6.00	121,346	170,129
7.00	127,056	159,921
8.00	211,641	169,751
9.00	357,838	187,912
10.00	644,408	191,867
11.00	656,965	183,021
12.00	648,206	184,279
13.00	630,434	180,384
14.00	544,572	175,252
15.00	729,518	192,992
16.00	735,850	189,160
17.00	611,582	177,808
18.00	719,537	179,228
19.00	607,043	174,783
20.00	477,760	161,153
21.00	470,291	159,065
22.00	534,103	144,212
23.00	531,276	164,516

Table A.5 Network traffic flows: number of transactions and frequent itemsets with *minsup* 0.1%.

MLlib PFP achieves the best load balancing, with comparable execution times for all tasks throughout all nodes, whose difference is in the order of 10%. Mahout PFP, instead, shows the worst load balancing issues, with differences as high as 90%. The difference between MLlib PFP and Mahout PFP can be correlated to the granularity of the subproblems. The smaller the subproblems, the better the load balancing because their execution times are more similar. MLlib PFP allows specifying the number of partitions, i.e., of subproblems, which obviously impacts on the granularity of each subproblem. Hence, setting opportunely this parameter, a good load balancing result is achieved. Differently, Mahout PFP automatically sets

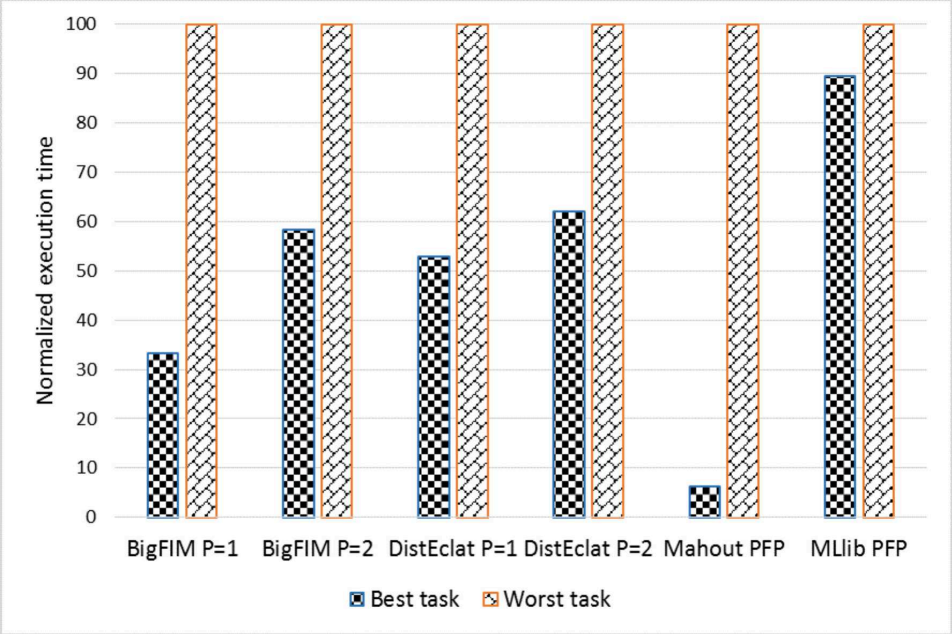


Fig. A.12 Normalized execution time of the most unbalanced tasks.

the number of subproblems and the current heuristic used to set it does not seem to work well on the considered datasets (unbalanced subproblems are generated).

We included BigFIM and DistEclat with 2 different first-phase prefix sizes. For these algorithms, the experiment confirms that a configuration with longer prefixes leads to a more balanced mining tasks than a configuration with short-sized prefixes, as mentioned in Subsection 2.2.3.

A.6 Communication costs

To evaluate the communication cost, we measure the amount of data transmitted and received through the nodes network interfaces. This information has been retrieved by means of the utilities provided by the Cloudera Manager tool.

The experiments have been performed on Dataset #1 with a fixed *minsup* value of 0.1%, which was the lowest value for which all algorithms completed the extraction. Figure A.13 reports, for each algorithm, the average value among transmitted and received traffic, compared to the total execution time. Firstly, the two measures do not seem to be correlated: higher communication costs are associated with low

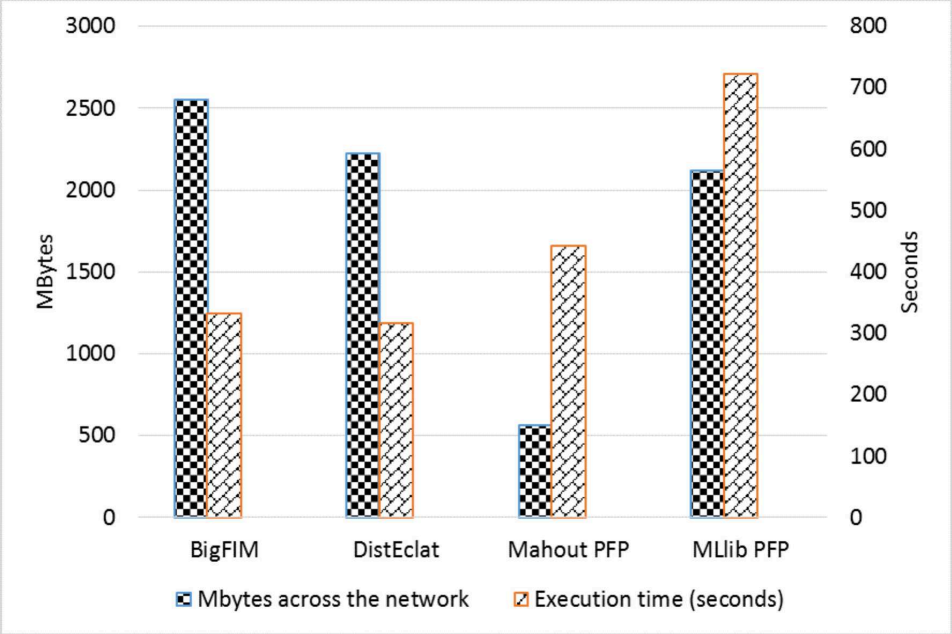


Fig. A.13 Communication costs and performance for each algorithm, Dataset #1, *minsup* 0.1%. The graph reports an average between transmitted and reveiced data.

execution times for BigFIM and DistEclat, whereas MLlib reports both measures with high values. Mahout PFP has a communication cost 4 to 5 times lower than all the others, which exchange an average of 2 Gigabytes of data. Mahout PFP average communication cost is around 0.5 Gigabytes, which is approximately the dataset size. The difference between DistEclat and BigFIM is not large because with only 2-length prefixes just an extra iteration is done by BigFIM. Even though Mahout PFP is the most communication-cost optimized implementation, the very low amount of data sent through the network is related to the adoption of compression techniques, which lead to higher execution times.

References

[1] United Nations Department of Economic and Social Affairs. World's population increasingly urban with more than half living in urban areas. *UN News Center*, 2014.

[2] United Nations Department of Economic and Social Affairs. World population prospects: The 2017 revision, key findings and advance tables. In *Working Paper No. ESA/P/WP. 248*, pages 1–53. 2017.

[3] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Paolo Garza, Fabio Pulvirenti, and Luca Venturini. Frequent itemsets mining for big data: A comparative analysis. *Big Data Research*, 9:67 – 83, 2017.

[4] Luca Venturini, Paolo Garza, and Daniele Apiletti. Bac: A bagged associative classifier for big data frameworks. In *East European Conference on Advances in Databases and Information Systems*, pages 137–146. Springer, 2016.

[5] Luca Venturini, Elena Maria Baralis, and Paolo Garza. Scaling associative classification for very large datasets. *JOURNAL OF BIG DATA*, 4(1):1–24, 2017.

[6] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: Current status and future directions. *Data Min. Knowl. Discov.*, 15(1):55–86, August 2007.

[7] Bart Goethals. Survey on frequent pattern mining. *Univ. of Helsinki*, 2003.

[8] Charu C Aggarwal and Jiawei Han. *Frequent pattern mining*. Springer, 2014.

[9] Rakesh Agrawal, Tomasz Imilienski, and Arum Swami. Mining association rules between sets of items in large databases. *In SIGMOD'93 , Washington DC*, May 1993.

[10] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 80–86. AAAI Press, 1998.

[11] Fadi Thabtah. A review of associative classification mining. *The Knowledge Engineering Review*, 22(01):37–65, 2007.

[12] P.-N. Tan and V. Kumar. Interestingness measures for association patterns: A perspective. *KDD 2000 Workshop on Postprocessing in Machine Learning and Data Mining*, 2000.

[13] Leo Breiman. Some properties of splitting criteria. *Machine Learning*, 24(1):41–47, 1996.

[14] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[15] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD '00*, pages 1–12, 2000.

[16] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *KDD'97*, pages 283–286. AAAI Press, 1997.

[17] Lan Vu and Gita Alaghband. Mining frequent patterns based on data characteristics. In *Proceedings of 2012 International Conference on Information and Knowledge Engineering*, pages 369–375, 2012.

[18] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[19] Hongjian Qiu, Rong Gu, Chunfeng Yuan, and Yihua Huang. YAFIM: A parallel frequent itemset mining algorithm with spark. In *IPDPSW'14*, pages 1664–1671, May 2014.

[20] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, RecSys '08, pages 107–114, New York, NY, USA, 2008. ACM.

[21] Sandy Moens, Emin Aksehirli, and Bart Goethals. Frequent itemset mining for big data. In *SML: BigData 2013 Workshop on Scalable Machine Learning*. IEEE, 2013.

[22] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Zadeh, Matei Zaharia, and Ameet Talwalkar. Mllib: Machine learning in apache spark. *J. Mach. Learn. Res.*, 17(1):1235–1241, January 2016.

[23] The Apache Mahout Project. The Apache Mahout machine learning library. Available: http://mahout.apache.org/ Last access on March 2013. 2013.

[24] Sandy Moens, Emin Aksehirli, , and Bart Goethals. Dist-eclat and bigfim. https://github.com/ua-adrem/bigfim, 2013.

[25] N. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *In IEEE TKDE*, 5(6), 1993.

[26] Cloudera, last Accessed: 16/10/2015.

[27] Elena Baralis and Paolo Garza. A lazy approach to pruning classification rules. In *Data Mining, 2002. Proceedings. IEEE International Conference on*, pages 35–42. IEEE, 2002.

[28] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[29] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[30] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[31] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[32] Wenmin Li, Jiawei Han, and Jian Pei. Cmar: Accurate and efficient classification based on multiple class-association rules. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 369–376. IEEE, 2001.

[33] Elena Baralis, Silvia Chiusano, and Paolo Garza. A lazy approach to associative classification. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):156–171, 2008.

[34] Sara Landset, Taghi M Khoshgoftaar, Aaron N Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):24, 2015.

[35] Dilpreet Singh and Chandan K Reddy. A survey on platforms for big data analytics. *Journal of Big Data*, 2(1):8, 2015.

[36] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61, 2015.

[37] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

[38] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

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

[40] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[41] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[42] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[43] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[44] Guozhu Dong, Xiuzhen Zhang, Limsoon Wong, and Jinyan Li. Caep: Classification by aggregating emerging patterns. In *International Conference on Discovery Science*, pages 30–42. Springer, 1999.

[45] Guoqing Chen, Hongyan Liu, Lan Yu, Qiang Wei, and Xing Zhang. A new approach to classification based on association rule mining. *Decision Support Systems*, 42(2):674–689, 2006.

[46] Xiaoxin Yin and Jiawei Han. Cpar: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining*, pages 331–335. SIAM, 2003.

[47] Jianyong Wang and George Karypis. Harmony: Efficiently mining the best rules for classification. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 205–216. SIAM, 2005.

[48] Fadi Thabtah, Peter Cowling, and Yonghong Peng. Mcar: multi-class classification based on association rule. In *Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference on*, page 33. IEEE, 2005.

[49] Fadi A Thabtah, Peter Cowling, and Yonghong Peng. Mmac: A new multi-class, multi-label associative classification approach. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 217–224. IEEE, 2004.

[50] Alessio Bechini, Francesco Marcelloni, and Armando Segatori. A mapreduce solution for associative classification of big data. *Information Sciences*, 332:33–55, 2016.

[51] Osmar R Zaïane and Maria-Luiza Antonie. Classifying text documents by associating terms with text categories. In *Australian computer Science communications*, volume 24, pages 215–222. Australian Computer Society, Inc., 2002.

[52] Alípio M Jorge and Paulo J Azevedo. An experiment with association rules and classification: Post-bagging and conviction. In *International Conference on Discovery Science*, pages 137–149. Springer, 2005.

[53] Xiaoyuan Xu, Guoqiang Han, and Huaqing Min. A novel algorithm for associative classification of image blocks. In *Computer and Information Technology, 2004. CIT'04. The Fourth International Conference on*, pages 46–51. IEEE, 2004.

[54] Y. Sun, Y. Wang, and A. K. C. Wong. Boosting an associative classifier. *IEEE Transactions on Knowledge and Data Engineering*, 18(7):988–992, July 2006.

[55] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 107–114. ACM, 2008.

[56] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Paolo Garza, Danilo Giordano, Marco Mellia, and Luca Venturini. Selina: a self-learning insightful network analyzer. *IEEE Transactions on Network and Service Management*, 13(3):696–710, 2016.

[57] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, and L. Venturini. Safe-nec: A scalable and flexible system for network data characterization. In *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 812–816, April 2016.

[58] Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Arian Bär, Alessandro Finamore, and Tanja Zseby. When youtube does not work - analysis of qoe-relevant degradation in google CDN traffic. *IEEE Transactions on Network and Service Management*, 11(4):441–457, 2014.

[59] Arian Bär, Alessandro Finamore, Pedro Casas, Lukasz Golab, and Marco Mellia. Large-scale network traffic monitoring with dbstream, a system for rolling big data analysis. In *2014 IEEE International Conference on Big Data, Big Data 2014, Washington, DC, USA, October 27-30, 2014*, pages 165–170, 2014.

[60] D. Giordano, S. Traverso, L. Grimaudo, M. Mellia, E. Baralis, A. Tongaonkar, and S. Saha. Youlighter: A cognitive approach to unveil youtube cdn and changes. *IEEE Transactions on Cognitive Communications and Networking*, 1(2):161–174, June 2015.

[61] Pedro Casas, Johan Mazel, and Philippe Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.

[62] Juliette Dromard, Gilles Roudiere, and Philippe Owezarski. Unsupervised network anomaly detection in real-time on big data. In *New Trends in Databases and Information Systems - ADBIS 2015 Short Papers and Workshops, BigDap,*

DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8-11, 2015. Proceedings, pages 197–206, 2015.

[63] Yeonhee Lee and Youngseok Lee. Toward scalable internet traffic measurement and analysis with hadoop. *ACM SIGCOMM Computer Communication Review*, 43(1):5–13, 2013.

[64] M. Mellia, M. Meo, L. Muscariello, and D. Rossi. Passive analysis of tcp anomalies. *Computer Networks*, 52(14):2663–2676, 2008.

[65] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231, 1996.

[66] DBScan. DBScan — Wikipedia, the free encyclopedia, 2011. [Online; accessed 15-June-2018].

[67] Dario Antonelli, Elena Baralis, Giulia Bruno, Tania Cerquitelli, Silvia Chiusano, and Naeem A. Mahoto. Analysis of diabetic patients through their examination history. *Expert Syst. Appl.*, 40(11):4672–4678, 2013.

[68] Lior Rokach and Oded Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2008.

[69] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[70] Pang-Ning T. and Steinbach M. and Kumar V. *Introduction to Data Mining*. Addison-Wesley, 2006.

[71] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987.

[72] A. Finamore, M. Mellia, M. Meo, M. Munafò, and D. Rossi. Experiences of internet traffic monitoring with tstat. *IEEE Network*, 25(3):8–14, 2011.

[73] J. L. Garcia-Dorado, A. Finamore, M. Mellia, M. Meo, and M. Munafo. Characterization of isp traffic: Trends, user habits, and access technology impact. *IEEE Transactions on Network and Service Management*, 9(2):142–155, June 2012.

[74] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, and Vincenzo D'Elia. Characterizing network traffic by means of the netmine framework. *Computer Networks*, 53(6):774–789, 2009.

[75] M. Hossain, SM Bridges, and RB Vaughn Jr. Adaptive intrusion detection with data mining. *IEEE Internation Conference on Systems, Man and Cybernetics*, 4, 2003.

[76] Franck Le, Sihyung Lee, Tina Wong, Hyong S. Kim, and Darrell Newcomb. Minerals: using data mining to detect router misconfigurations. In *MineNet '06*, pages 293–298, New York, NY, USA, 2006. ACM Press.

[77] Manoj K. Agarwal, Manish Gupta, Gautam Kar, Anindya Neogi, and Anca Sailer. Mining activity data for dynamic dependency discovery in e-business systems. *IEEE Transactions on Network and Service Management*, 1(2):49–58, 2004.

[78] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *SIGCOMM*, pages 229–240, 2005.

[79] Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05*, pages 50–60, New York, NY, USA, 2005. ACM Press.

[80] José Everardo Bessa Maia et al. Network traffic prediction using pca and k-means. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 938–941. IEEE, 2010.

[81] Ye Ouyang, M. Hosein Fallah, Sanqing Hu, Yong Ren Yong, Yirui Hu, Zhichang Lai, Mingxin Guan, and Wenyuan Lu. A novel methodology of data analytics and modeling to evaluate LTE network performance. In *2014 Wireless Telecommunications Symposium, WTS 2014, Washington, DC, USA, April 9-11, 2014*, pages 1–10, 2014.

[82] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, ODD '13, pages 8–15, 2013.

[83] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. *Proceedings of ACM CSS Workshop on Data Mining Applied to Security, PA,, November*, 2001.

[84] Q. Wang and V. Megalooikonomu. A clustering algorithm for intrusion detection. *Proc. SPIE*, 5812:31–38, 2005.

[85] Philippe Owezarski. Unsupervised classification and characterization of honeypot attacks. In *10th International Conference on Network and Service Management, CNSM 2014 and Workshop, Rio de Janeiro, Brazil, November 17-21, 2014*, pages 10–18, 2014.

[86] Elena Baralis, Andrea Bianco, Tania Cerquitelli, Luca Chiaraviglio, and Marco Mellia. Netcluster: A clustering-based framework to analyze internet passive measurements data. *Computer Networks*, 57(17):3300–3315, 2013.

[87] Luigi Grimaudo, Marco Mellia, Elena Baralis, and Ram Keralapura. Select: Self-learning classifier for internet traffic. *IEEE Transactions on Network and Service Management*, 11(2):144–157, 2014.

[88] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. Traffic classification using clustering algorithms. In *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 281–286. ACM, 2006.

[89] Jao Yoon Chung, Byungchul Park, Young J Won, John Strassner, and James W Hong. An effective similarity metric for application traffic classification. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 286–292. IEEE, 2010.

[90] Marcus Fabio Fontenelle do Carmo, Jose Everardo Bessa Maia, GP Siqueira, et al. An internet traffic classification methodology based on statistical discriminators. In *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pages 907–910. IEEE, 2008.

[91] F.A. Lisi and D. Malerba. Inducing multi-level association rules from multiple relations. *Machine Learning*, 55(2):175–210, 2004.

[92] Ricardo J. G. B. Campello, Davoud Moulavi, and Jörg Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining, 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*, pages 160–172, 2013.

[93] Ricardo J. G. B. Campello, Davoud Moulavi, Arthur Zimek, and Jörg Sander. Hierarchical density estimates for data clustering, visualization, and outlier detection. *TKDD*, 10(1):5, 2015.

[94] Gregory Buehrer, Roberto L. de Oliveira Jr., David Fuhry, and Srinivasan Parthasarathy. Towards a parameter-free and parallel itemset mining algorithm in linearithmic time. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1071–1082, 2015.

[95] Yaobin He, Haoyu Tan, Wuman Luo, Shengzhong Feng, and Jianping Fan. MR-DBSCAN: a scalable mapreduce-based DBSCAN algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

[96] Sandy Moens, Emin Aksehirli, and Bart Goethals. Frequent itemset mining for big data. In *Proceedings of the 2013 IEEE International Conference on Big Data, 6-9 October 2013, Santa Clara, CA, USA*, pages 111–118, 2013.

[97] Biswanath Panda, Joshua Herbach, Sugato Basu, and Roberto J. Bayardo. PLANET: massively parallel learning of tree ensembles with mapreduce. *PVLDB*, 2(2):1426–1437, 2009.

[98] Devendra Dahiphale, Rutvik Karve, Athanasios V. Vasilakos, Huan Liu, Zhiwei Yu, Amit Chhajer, Jianmin Wang, and Chaokun Wang. An advanced mapreduce: Cloud mapreduce, enhancements and applications. *IEEE Transactions on Network and Service Management*, 11(1):101–115, 2014.

[99] Vernon KC Bumgardner and Victor W Marek. Scalable hybrid stream and hadoop network analysis system. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 219–224. ACM, 2014.

[100] Yousun Jeong. Big Telco Real-Time Network Analytics Available: https://spark-summit.org/eu-2015/events/big-telco-real-time-network-analytics/. *Spark summit, Amsterdam, Netherland, October 27-29*, 2015.

[101] KV Swetha, Shiju Sathyadevan, and P Bilna. Network data analysis using spark. In *Software Engineering in Intelligent Systems*, pages 253–259. Springer, 2015.

[102] Luca Venturini and Evelina Di Corso. Analyzing spatial data from twitter during a disaster. In *Proceedings of 2017 IEEE International Conference on Big Data*, pages 3779–3783. IEEE, 2017.

[103] Luca Venturini and Elena Baralis. A spectral analysis of crimes in san francisco. In *Proceedings of the 2nd ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics*, page 4. ACM, 2016.

[104] Luca Cagliero, Tania Cerquitelli, Silvia Chiusano, Pierangelo Garino, Marco Nardone, Barbara Pralio, and Luca Venturini. Monitoring the citizens' perception on urban security in smart city environments. In *Data Engineering Workshops (ICDEW), 2015 31st IEEE International Conference on*, pages 112–116. IEEE, 2015.

[105] Muhammad Imran, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. Processing social media messages in mass emergency: A survey. *ACM Computing Surveys (CSUR)*, 47(4):67, 2015.

[106] Joao Porto De Albuquerque, Benjamin Herfort, Alexander Brenning, and Alexander Zipf. A geographic approach for combining social media and authoritative data towards identifying useful information for disaster management. *International Journal of Geographical Information Science*, 29(4):667–689, 2015.

[107] Julie Dugdale, Bartel Van de Walle, and Corinna Koeppinghoff. Social media and sms in the haiti earthquake. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12 Companion, pages 713–714, 2012.

[108] Kate Crawford and Megan Finn. The limits of crisis data: analytical and ethical challenges of using social and mobile data to understand disasters. *GeoJournal*, 80(4):491–502, 2015.

[109] Scott H Burton, Kesler W Tanner, Christophe G Giraud-Carrier, Joshua H West, and Michael D Barnes. "right time, right place" health communication on twitter: value and accuracy of location information. *Journal of medical Internet research*, 14(6), 2012.

[110] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 759–768. ACM, 2010.

[111] Xin Xiao, Antonio Attanasio, Silvia Chiusano, and Tania Cerquitelli. Twitter data laid almost bare: An insightful exploratory analyser. *Expert Systems with Applications*, 90:501–517, 2017.

[112] Elisabetta Povoledo. Deadly earthquake hits italian island of ischia, 8 2017.

[113] Eva Moravec. Storm deaths: Harvey claims lives of more than 75 in texas, 10 2017.

[114] USGS. U.s. geological survey. last access: October 2017.

[115] San francisco open data portal. https://data.sfgov.org/. Accessed: 2016-09-01.

[116] Vania Ceccato and Adriaan Cornelis Uittenbogaard. Space–time dynamics of crime in transport nodes. *Annals of the Association of American Geographers*, 104(1):131–150, 2014.

[117] Vânia Ceccato. Homicide in São Paulo, Brazil: Assessing spatial-temporal and weather variations. *Journal of Environmental Psychology*, 25(3):307–321, 2005.

[118] Silas Nogueira de Melo, Débora VS Pereira, Martin A Andresen, and Lindon Fonseca Matias. Spatial/temporal variations of crime: a routine activity theory perspective. *International journal of offender therapy and comparative criminology*, page 0306624X17703654, 2017.

[119] N. R. Lomb. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science*, 39(2):447–462, 2 1976.

[120] J. D. Scargle. Studies in astronomical time series analysis. II - Statistical aspects of spectral analysis of unevenly spaced data. *The Astrophysical Journal*, 263:835, 12 1982.

[121] Jacob VanderPlas, Andrew J. Connolly, Zeljko Ivezic, and Alex Gray. Introduction to astroML: Machine learning for astrophysics. In *2012 Conference on Intelligent Data Understanding*, pages 47–54. IEEE, 10 2012.

[122] Lee R. McPheters and William B. Stronge. Testing for seasonality in reported crime data. *Journal of Criminal Justice*, 1(2):125–134, 22 1973.

[123] Martin A. Andresen and Nicolas Malleson. Crime seasonality and its variations across space. *Applied Geography*, 43:25–35, 2013.

[124] G. D. Breetzke and E. G. Cohn. Seasonal Assault and Neighborhood Deprivation in South Africa: Some Preliminary Findings. *Environment and Behavior*, 44(5):641–667, 9 2012.

[125] Shannon J Linning. Crime seasonality and the micro-spatial patterns of property crime in Vancouver, BC and Ottawa, ON. *Journal of Criminal Justice*, 43:544–555, 2015.

[126] Gregory D. Breetzke. Examining the spatial periodicity of crime in South Africa using Fourier analysis. *South African Geographical Journal*, 98(2):275–288, 5 2016.

[127] Stefan Baisch and Götz H.R. Bokelmann. Spectral analysis with incomplete time series: an example from seismology. *Computers & Geosciences*, 25(7):739–750, 1999.

[128] H. P. A. Van Dongen, E. Olofsen, J. H. VanHartevelt, and E. W. Kruyt. Searching for Biological Rhythms: Peak Detection in the Periodogram of Unequally Spaced Data. *Journal of Biological Rhythms*, 14(6):617–620, 12 1999.

[129] E. F. Glynn, J. Chen, and A. R. Mushegian. Detecting periodic patterns in unevenly spaced gene expression time series using Lomb-Scargle periodograms. *Bioinformatics*, 22(3):310–316, 2 2006.

[130] S. Pellicer, G. Santa, A.L. Bleda, R. Maestre, A.J. Jara, and A. Gomez Skarmeta. A global perspective of smart cities: A survey. In *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2013 Seventh International Conference on*, pages 439–444, July 2013.

[131] Open311. A collaborative model and open standard for civic issue tracking. Available at http://www.open311.org/. 2014.

[132] Winnipeg311. Winnipeg 311 Mobile App. Available at http://www.winnipeg.ca/Interhom/contact/app.stm/. 2014.

[133] Bloomington. Open Source GeoReporter and uReport tools for Open311. Available at http://bloomington.in.gov/. 2014.

[134] IBM-IOC. IBM Intelligent Operations Center. Available: http://www-03.ibm.com/software/products/it/intelligent-operations-center. Last access on November 2014. 2014.

[135] M. Hamilton, F. Salim, E. Cheng, and S. L. Choy. Transafe: A crowdsourced mobile platform for crime and safety perception management. *SIGCAS Comput. Soc.*, 41(2):32–37, December 2011.

[136] M. Behrens, N. Valkanova, A. Fatah Schieck, and D. Brumby. Smart citizen sentiment dashboard: A case study into media architectural interfaces. In *Inter. Symp. on Pervasive Displays*, 2014.

[137] Smartdatanet. Smart data platform. Available at http://www.smartdatanet.it/. 2014.

[138] Mark Blythe, Peter C. Wright, and Andrew F. Monk. Little brother: could and should wearable computing technologies be applied to reducing older people's fear of crime? *Personal and Ubiquitous Computing*, 8(6):402–415, 2004.

[139] Morris Williams, Owain Jones, Constance Fleuriot, and Lucy Wood. Children and emerging wireless technologies: investigating the potential for spatial practice. In *Conf. on Human Factors in Computing Systems 2005*, pages 819–828.

[140] Farida Naceur. Impact of urban upgrading on perceptions of safety in informal settlements: Case study of bouakal, batna. *Frontiers of Architectural Research*, 2(4):400–408, December 2013.

[141] Cédric Bach, Regina Bernhaupt, Caio Stein D'Agostini, and Marco Winckler. Mobile applications for incident reporting systems in urban contexts: lessons learned from an empirical study. In *European Conference on Cognitive Ergonomics 2013, ECCE '13, Toulouse, France, August 26 - 28, 2013*, page 29, 2013.

[142] Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: Concepts, methodologies, and applications. *ACM Trans. Intell. Syst. Technol.*, 5(3):38:1–38:55, September 2014.

[143] R. Agrawal, T. Imielinski, and Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD 1993*, pages 207–216, 1993.

[144] Rakesh Agrawal and Giuseppe Psaila. Active data mining. In *KDD 1995*, pages 3–8.

[145] Jiawei Han, Jain Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *In SIGMOD'00, Dallas, TX*, May 2000.

[146] Matteo Golfarelli, Dario Maio, and Stefano Rizzi. The dimensional fact model: A conceptual model for data warehouses. *International Journal of Cooperative Information Systems*, 7:215–247, 1998.

[147] Ralph Kimball and Margy Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, NY, USA, 2nd edition, 2002.

[148] Alexandra Olteanu, Carlos Castillo, Fernando Diaz, and Sarah Vieweg. Crisislex: A lexicon for collecting and filtering microblogged communications in crises. In *International AAAI Conference on Web and Social Media*, 2014.

[149] A. Frisiello, Q. N. Nguyen, and C. Rossi. Gamified crowdsourcing for disaster risk management. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 3727–3733, Dec 2017.

[150] Alex Chohlas-Wood, Aliya Merali, Warren Reed, and Theodoros Damoulas. Mining 911 calls in new york city: Temporal patterns, detection, and forecasting. In *AAAI Workshop: AI for Cities*, 2015.

[151] Robert Wetzker, Carsten Zimmermann, and Christian Bauckhage. Analyzing social bookmarking systems: A del.icio.us cookbook. In *Mining Social Data (MSoDa) Workshop Proceedings*, pages 26–30. ECAI 2008, July 2008.

[152] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Silvia Chiusano, and Luigi Grimaudo. Searum: A cloud-based service for association rule mining. In *11th IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA-13*, pages 1283–1290, 2013.