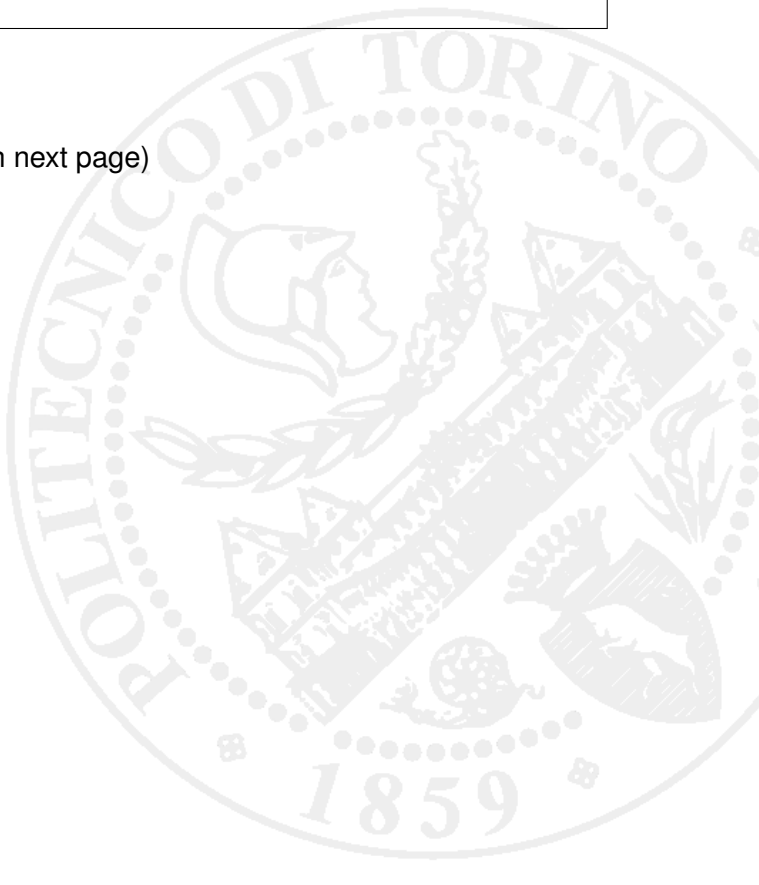


Politecnico di Torino

Doctoral Thesis

Data Mining Algorithms for Internet Data

from Transport to Application Layer

Author:
Luigi Grimaudo

Supervisor:
Prof. Elena Baralis
Co-supervisor:
Dr. Marco Mellia

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in

Information and System Engineering
XXVI cycle

March 2014

Politecnico di Torino

Doctoral Thesis

Data Mining Algorithms for Internet Data

from Transport to Application Layer

Author:
Luigi Grimaudo

Supervisor:
Prof. Elena Baralis
Co-supervisor:
Dr. Marco Mellia

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

in

Information and System Engineering
XXVI cycle

March 2014

"Never to limit yourself to one style, to keep an open mind."

Frank Dux

POLITECNICO DI TORINO

Abstract

Information and System Engineering
XXVI cycle

Doctor of Philosophy

Data Mining Algorithms for Internet Data
from Transport to Application Layer

by Luigi GRIMAUDO

Nowadays we live in a data-driven world. Advances in data generation, collection and storage technology have enabled organizations to gather data sets of massive size.

Data mining is a discipline that blends traditional data analysis methods with sophisticated algorithms to handle the challenges posed by these new types of data sets.

The Internet is a complex and dynamic system with new protocols and applications that arise at a constant pace. All these characteristics designate the Internet a valuable and challenging data source and application domain for a research activity, both looking at Transport layer, analyzing network traffic flows, and going up to Application layer, focussing on the ever-growing next generation web services: blogs, micro-blogs, on-line social networks, photo sharing services and many other applications (*e.g.*, Twitter, Facebook, Flickr, etc.).

In this thesis work we focus on the study, design and development of novel algorithms and frameworks to support large scale data mining activities over huge and heterogeneous data volumes, with a particular focus on Internet data as data source and targeting network traffic classification, on-line social network analysis, recommendation systems and cloud services and Big data.

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Prof. Elena Baralis, your advices on both research as well as on my career have been priceless. Your mentoring was precious and fundamental for the development of my PhD work.

I want also like to thank Dr. Marco Mellia, your qualified support and availability helped me throughout my research activity.

I would like to show my gratitude to Narus Inc. too, and in particular to all the members of the CTO office. I spent almost a third of my Phd as intern with you, giving me the opportunity to work on very interesting projects.

My PhD and my university career probably would not have been possible without the people that accompanied me in these years. I want to thank all my research group colleagues, Alberto, Alessandro, Tania, Daniele, Silvia, Paolo and Giulia for their constant support and suggestions. I would like also to thank all the people I meet during my PhD studies.

A special thank goes to my family for their constant belief and support. I will be always grateful to you.

Finally, a special acknowledgement to Piera, my future wife. Her love, madness and incessant will to flight abroad for a weekend support me through these years....

Contents

List of Figures

List of Tables

Dedicated to my family and my future wife Piera...

Chapter 1

Introduction

Nowadays we live in a data-driven world. Advances in data generation, collection and storage technology have enabled organizations to gather data sets of massive size. From the tweets or comments that users post about any moments of their life to the transaction data retailers gather from point-of-sale terminals, data is flooding organizations from every angle. Furthermore data has inherent value and cannot be discarded anymore. A new attitude towards data analysis arose: Gather whatever data you can, whenever and wherever possible. Despite that, extracting useful information is becoming more and more challenging.

Data mining is a discipline that blends traditional data analysis methods with sophisticated algorithms to handle the challenges posed by these new types of data sets. There are many data mining tasks. Among the most common ones we can mention classification (or supervised learning), clustering (or unsupervised learning) and association rule mining. Classification aims to learn a model from data that are labeled with pre-defined classes or categories, while clustering organizes data instances into groups or clusters according to their similarities (or differences). Finally, association rules finds out sets of data items that occur together frequently.

The Internet is a complex and dynamic system with new protocols and applications that arise at a constant pace. All these characteristics designate the Internet a valuable and challenging data source and application domain for a research activity, both looking at Transport layer, analyzing network traffic flows, and going up to Application layer, focussing on the ever-growing next generation web services: blogs, micro-blogs, online social networks, photo sharing services and many other applications (*e.g.*, Twitter, Facebook, Flickr, etc.).

On one side in the Internet traffic management and monitoring field, a critical task is the identification of application originating traffic flows, possibly in near real-time and in an automatic way, to support network operators on the adoption of ad-hoc countermeasures. On the other, on-line social networks and other Web 2.0 applications represent a powerful source of knowledge and a valuable matter to research on, but also one of the most common Big data source. Hence, design efficient data analysis approaches, able to scale horizontally with the data volumes, is becoming an interesting challenge.

In this thesis work we focus on the study, design and development of novel algorithms and frameworks to support large scale data mining activities over huge and heterogeneous data volumes, with a particular focus on Internet data as data source and targeting network traffic classification, on-line social network analysis, recommendation systems and cloud services and Big data.

1.1 Transport Layer

Traffic classification is still today a challenging problem given the ever evolving nature of the Internet in which new protocols and applications arise at a constant pace. In the past, so called behavioral approaches have been successfully proposed as valid alternatives to traditional Deep Packet Inspection (DPI) based tools to properly classify traffic into few and coarse classes. We push forward the adoption of behavioral classifiers by engineering a Hierarchical classifier [1] that allows proper classification of traffic into more than twenty fine grained classes. Thorough engineering has been followed which considers both proper feature selection and testing seven different classification algorithms. Results obtained over actual and large data sets show that the proposed Hierarchical classifier outperforms off-the-shelf non hierarchical classification algorithms by exhibiting average accuracy higher than 90%, with precision and recall that are higher than 95% for most popular classes of traffic.

Network visibility is a critical part of traffic engineering, network management, and security. The most popular current solutions, DPI and statistical classification, deeply rely on the availability of a training set. Besides the cumbersome need to regularly update the signatures, their visibility is limited to classes the classifier has been trained for. Unsupervised algorithms have been envisioned as a viable alternative to automatically identify classes of traffic. However, the accuracy achieved so far does not allow to use them for traffic classification in practical scenario. To address the above issues, we propose SeLeCT, a Self-Learning Classifier for Internet Traffic [2]. It uses unsupervised

algorithms along with an adaptive seeding approach to automatically let classes of traffic emerge, being identified and labeled. Unlike traditional classifiers, it requires neither a-priori knowledge of signatures nor a training set to extract the signatures. Instead, SeLeCT automatically groups flows into pure (or homogeneous) clusters using simple statistical features. SeLeCT simplifies label assignment (which is still based on some manual intervention) so that proper class labels can be easily discovered. Furthermore, SeLeCT uses an iterative seeding approach to boost its ability to cope with new protocols and applications. We evaluate the performance of SeLeCT using traffic traces collected in different years from various ISPs located in 3 different continents. Our experiments show that SeLeCT achieves excellent precision and recall, with overall accuracy close to 98%. Unlike state-of-the-art classifiers, the biggest advantage of SeLeCT is its ability to discover new protocols and applications in an almost automated fashion.

1.2 Application Layer

On-line social network websites such as Facebook, Twitter, and LinkedIn are quickly transitioned to global phenomena over the last few years. Among the plethora of such services, we put more focus on Twitter. Twitter is a micro-blog service that has attracted millions of users that generate a humongous flow of information at constant pace. The research community has thus started proposing tools to extract meaningful information from tweets. At the beginning, we target the generic stream of Twitter messages (*i.e.*, tweets) sent continuously by users. We propose a data analysis framework to discover groups of similar tweets posted on a given event [3]. By analyzing these groups, user emotions or thoughts that seem to be associated with specific events can be extracted, as well as aspects characterizing events according to user perception. To deal with the inherent sparseness of micro-messages, the proposed approach relies on a multiple-level strategy that allows clustering text data with a variable distribution. Clusters are then characterized through the most representative words appearing in their messages, and association rules are used to highlight correlations among these words. To measure the relevance of specific words for a given event, text data has been represented in the Vector Space Model using the TF-IDF weighting score. As a case study, two real Twitter datasets have been analysed.

Moreover, user-generated content (UGC) coming from social networks and online communities continuously grows and changes. By analyzing relevant patterns from the UGC, analysts may discover peculiar user behaviors and interests which can be used to personalize Web-oriented applications. In the last several years, the use of dynamic mining

techniques has captured the interest of the research community. They are focus on analyzing the temporal evolution of most significant correlations hidden in the analyzed data. However, keeping track of all temporal data correlations relevant for user behaviors, community interests, and topic trend analysts may become a challenging task due to the sparseness of the analyzed data. We present a novel data mining system [4] that performs dynamic itemset mining from both the content and the contextual features of the messages posted on Twitter. Dynamic itemsets represent the evolution of data correlations over time. The framework exploits a dynamic itemset mining algorithm, named HiGen Miner, to discover relevant temporal data correlations from a stream of tweet collections. In particular, it extracts compact patterns, namely the HiGens, that represent the evolution of the most relevant itemsets over consecutive time periods at different abstraction levels. A taxonomy is used to drive the mining process and prevent the discarding of knowledge that becomes infrequent in a certain time period. Experiments, performed on real Twitter posts, show the effectiveness and the usability of the proposed system in supporting Twitter user behavior and topic trend analysis.

Afterwards, we take a different angle from the mainstream of previous works: we explicitly target the analysis of the timeline of tweets from "single users". We define a framework - named TUCAN [5] - to compare information offered by the target users over time, and to pinpoint recurrent topics or topics of interest. First, tweets belonging to the same time window are aggregated into "bird songs". Several filtering procedures can be selected to remove stop-words and reduce noise. Then, each pair of bird songs is compared using a similarity score to automatically highlight the most common terms, thus highlighting recurrent or persistent topics. TUCAN can be naturally applied to compare bird song pairs generated from timelines of different users. By showing actual results for both public profiles and anonymous users, we show how TUCAN is useful to highlight meaningful information from a target user's Twitter timeline.

A common feature of Web 2.0 services, for instance *Delicious* and *Flickr*, is the ability to assign a label or metadata, namely a *tag*, to a Web resource (*i.e.*, photo, bookmark, etc.) to help in describing it. Tag recommendation is the task of predicting folksonomy tags for a given user and item, based on past user behavior (and possibly other information). Tag recommendation is focused on recommending useful tags to a user who is annotating a Web resource. A relevant research issue is the recommendation of additional tags to partially annotated resources, which may be based on either personalized or collective knowledge. However, since the annotation process is usually not driven by any controlled vocabulary, the collections of user-specific and collective annotations are often very sparse. Indeed, the discovery of the most significant associations among tags becomes a challenging task. We present a novel personalized tag recommendation system [6] that discovers and exploits generalized association rules, *i.e.*, tag correlations

holding at different abstraction levels, to identify additional pertinent tags to suggest. The use of generalized rules relevantly improves the effectiveness of traditional rule-based systems in coping with sparse tag collections, because (i) correlations hidden at the level of individual tags may be anyhow figured out at higher abstraction levels and (ii) low level tag associations discovered from collective data may be exploited to specialize high level associations discovered in the user-specific context. The effectiveness of the proposed system has been validated against other personalized approaches on real-life and benchmark collections retrieved from the popular photo-sharing system Flickr.

1.3 Association rule mining algorithms and Big data

We leverage association rule and itemset mining to design many of our analysis frameworks. Frequent generalized itemset mining is a data mining technique utilized to discover a high-level view of interesting knowledge hidden in the analyzed data. By exploiting a taxonomy, patterns are usually extracted at any level of abstraction. However, some misleading high-level patterns could be included in the mined set. We propose a novel generalized itemset type, namely the Misleading Generalized Itemset (MGI) [7]. Each MGI, denoted as $X \triangleright \mathcal{E}$, represents a frequent generalized itemset X and its set $\mathcal{E}$ of low-level frequent descendants for which the correlation type is in contrast to the one of X. To allow experts to analyze the misleading high-level data correlations separately and exploit such knowledge by making different decisions, MGIs are extracted only if the low-level descendant itemsets that represent contrasting correlations cover almost the same portion of data as the high-level (misleading) ancestor. An algorithm to mine MGIs at the top of traditional generalized itemsets is also proposed. The experiments performed on both real and synthetic datasets demonstrate the effectiveness and efficiency of the proposed approach.

As we have already seen, large volumes of data are being produced by various modern web applications at an ever increasing rate. The automatic analysis of such huge data volume is a challenging task since a large amount of interesting knowledge can be extracted. Association rule mining is an exploratory data analysis method able to discover interesting and hidden correlations among data. Since this data mining process is characterized by computationally intensive tasks, efficient distributed approaches are needed to increase its scalability. We propose a novel cloud-based service, named SeARuM [8], to efficiently mine association rules on a distributed computing model. SeARuM consists of a series of distributed MapReduce jobs run in the cloud. Each job performs a different step in the association rule mining process. As a case study, the proposed approach has been applied to the network data scenario. The experimental validation,

performed on two real network datasets, shows the effectiveness and the efficiency of SeARuM in mining association rules on a distributed computing model.

The thesis is organized as follows. A hierarchical classifier is described in Chapter 2, while Chapter 3 presents SeLeCT, a Self-Learning Classifier for Internet Traffic. A multi-level clustering approach and a dynamic itemset framework for Twitter data are described in Chapter 4 and 5, respectively. Chapter 6 presents TUCAN, a Twitter User Centric ANalyzer. Then, a tag recommendation system is illustrated in Chapter 7, while Chapter 8 presents a novel generalized itemset type. Moreover, SeARuM, a cloud-based SErvice for Association RUle Mining is described in Chapter 9. Experimental designs and results are reported in each chapter. Finally, Chapter 10 derives conclusions and presents future developments for the proposed approaches.

Chapter 2

Hierarchical Learning for Fine Grained Internet Traffic Classification

The identification and characterization of network traffic is at the base of network management activities for an operator. Through the continuous monitoring of the traffic, security policies can be deployed and tuned, anomalies can be detected, changes in the users behavior can be identified so that QoS and traffic engineering policies can be continuously improved.

In the last years, several traffic classification techniques have been proposed to overcome the limit of original port-based classifiers. Most popular approaches are coarsely based on *deep packet inspection* (DPI) or *behavioral* techniques. In the first case, the traffic is classified looking for specific tokens inside the packet payload. Behavioral techniques try to overcome the limitations of DPI by exploiting some description of the application behavior by means of statistical characteristics, such as the length of the first packets of a flow.

Both DPI and behavioral classifiers are supervised techniques. However, in case of DPI, the training is often cumbersome and complex, since it involves in most cases the manual identification of the tokens and regular expressions that define a class. In case of behavioral classifiers instead, the adoption of classification algorithms allows to automatically define the rules to label flows, provided a good training set is available. Behavioral approaches bring other advantages with respect to DPI: i) They do not inspect the packet payload, thus preserving privacy, and can then be used for lighter monitoring such as the one offered by, e.g., netflow; ii) They can be easily extended by going through a

quicker retraining phase; iii) The decision process can be computationally lightweight since feature computation is typically much simpler than regular expression parsing.

However, behavioral classifiers suffer from some drawback too [9]: i) A proper training set must be available, including a training set for the "unknown" class, i.e., flows that do not belong to any of the targeted classes; ii) Training must be customized to the monitored network, i.e., training is not portable; iii) And they are known to provide good accuracy when considering *few and coarse* traffic classes, like HTTP vs Peer-to-Peer (P2P) vs email. The last issue is particularly critical given the current trend to have a convergence of most applications going over the same protocol, namely HTTP. Therefore one natural question arises: is it possible to push further behavioral classifiers to correctly identify a large and granular set of classes? For instance, could it be possible to identify application specific traffic that runs over HTTP, like distinguishing Facebook, YouTube, or Google Maps traffic? How to handle the unknown class? In this work we address this latter problem by engineering and evaluating the performance of a novel Hierarchical behavioral classifier. The intuition is to split the classification process of flows into several stages. At the beginning, coarser classes are used, while in following stages finer grained classification is performed. Classifiers are organized in a tree-based structure, defined according to our domain knowledge. Each node is an independent classifier which operates on a subset of flow features specifically selected to maximize its accuracy, precision and recall. The root node simply separates flows into "unknown" or "known" protocols. The latter set is then classified into 7 classes, with P2P and HTTP appearing as generic classes to be further refined at the next step. For instance, 10 possible subclasses are possible for HTTP traffic.

We consider a benchmark in which 23 different classes are provided by an "oracle". We use Tstat [10], our DPI-based tool as ground truth generator. Extensive and thorough experiments are run considering 22 different data sets collected from a large ISP network and 3 additional data sets collected from our campus network. Results show that the proposed approach outperforms classical machine-learning based classification algorithms, which fail in handling flows of the "unknown" class, and when the number of samples in the training set is heavily unbalanced, as typical in real scenarios. The hierarchical classifier, instead, achieves better results thanks to splitting the decision process into several stages, each involving fewer classes.

2.1 Data set and classes

For the experiments carried out in this work we rely on the traffic monitoring and classification capabilities of Tstat [10], the passive sniffer developed at Politecnico di

TABLE 2.1: Set of protocols identified by Tstat that have more than 50 samples in one of the data sets used for training set.

ID	Class	Byte	Flow	Application protocol
1	Unknown/other	1.2G	355k	Unclassified or belonging to discarded classes
2	SMTP	394M	44k	Simple Mail Transfer Protocol - RFC 5321
3	POP3	182M	6k	Post Office Protocol - RFC 1939
4	IMAP4	55M	419	Internet Message Access Protocol - RFC 3501
5	SSL/TLS	968M	25k	Transport Layer Security protocol - RFC 5246
6	MSN	4M	137	Microsoft Messanger MSN Protocol
7	MSN HTTP	12M	162	Microsoft Messanger MSN Protocol tunneled over HTTP
8	Flickr	105M	2k	Flickr Photo download over HTTP
9	ADV	159M	11k	Advertisement content download over HTTP
10	MegaUpload	2.1G	225	Megaupload file download over HTTP
11	Gmaps	218M	2k	Google Maps images download over HTTP
12	Wiki	28M	661	Wikipedia content download over HTTP
13	Facebook	1.6G	40k	Facebook web page content download over HTTP
14	OpenSocial	6M	241	OpenSocial based social networks over HTTP
15	YouTube Video	4.9G	796	YouTube flash video streams over HTTP
16	YouTube Site	4G	4k	YouTube web page static content download over HTTP
17	Flash Video	848M	560	Generic flash video streams over HTTP
18	RTMP	72M	56	Generic flash video streams over Real Time Messaging Protocol
19	Other Video	200M	60	Generic video content over HTTP
20	ED2K Obf	23.6G	28k	Obfuscated Emule Protocol
21	ED2K	59G	17k	Plain Emule Protocol
22	BT	3G	14k	BitTorrent Peer Wire Protocol
23	BT MSE/PE	3G	16k	Encrypted BitTorrent Peer Wire Protocol

Torino since 2000 which is freely available from [11]. Tstat passively monitors network traffic carried on a link. It is capable to rebuild each TCP flow, computing a number of statistics. A complex DPI classifier is able to identify more than 50 different protocols. Its accuracy has proved to be very reliable in the past [12]. Among all possible traffic classes that Tstat is able to classify, we selected those for which at least 50 flows are present in each data set. Table 2.1 details the list of applications we target, showing also their predominance in one of the data sets used for training. Protocols and application are coarsely grouped to easy readability. As it is possible to see, we consider both simple and well known application protocols (SMPT/POP3/SSL/etc.), and finer grained classification. For example, we would like to distinguish among plain and obfuscated P2P protocols; for HTTP traffic, we would like to identify Facebook separately from social network platforms based on Google OpenSocial protocol. In total we consider 23 different classes.

To run performance evaluation on actual traffic, packet traces have been collected from two real networks: a nation-wide ISP in Italy that offers us three different vantage points, and our Campus network. ISP vantage points expose traffic of three different Points-of-Presence (POP) in different cities in Italy; each PoP aggregates traffic from more then 10,000 ISP customers, which range from home users to Small Office Home Office (SOHO) accessing the Internet via ADSL or Fiber-To-The-Home technology. It represents therefore a very heterogeneous and challenging scenario. We define a data

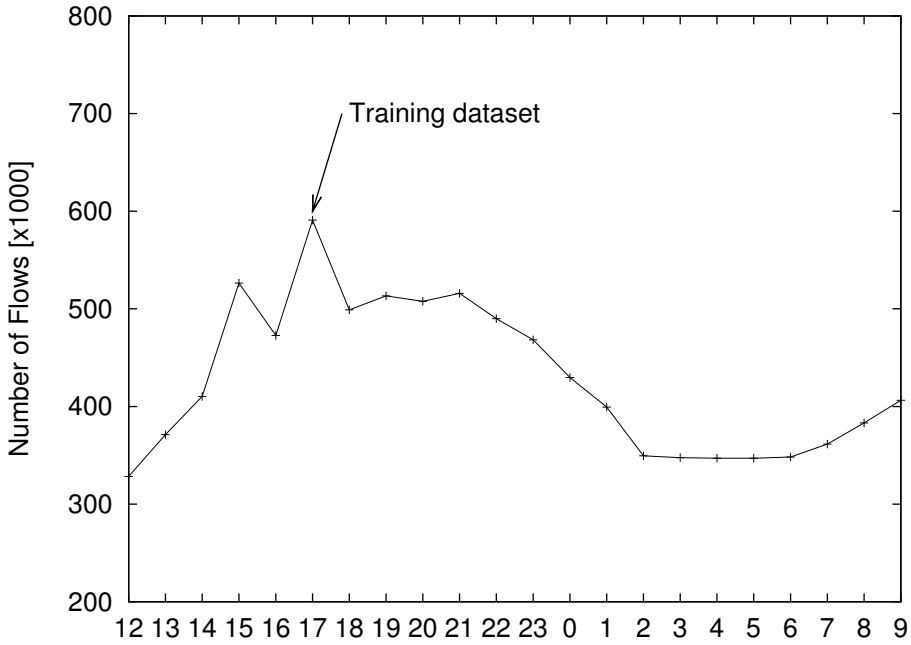


FIGURE 2.1: Number of flows in each ISP data set of 1 hour.

set as the set of all flows observed from a vantage point during a one-hour long time interval. In this work we focus our attention to one of the three ISP vantage points from which we have collected 22 different data sets, i.e., 22h long trace. Fig. 2.1 shows the number of flows that are present in each ISP data set. As expected, the number of flows grows during the day when web traffic is predominant. During the night, fewer flows are present, most of them due to P2P traffic. At 17:00, traffic reaches the peak. We consider this particular data set as "training data set" in the following. Other data sets are used for testing and validation purposes. Table 2.1 details the breakdown of the h.17 data set among different classes for Bytes and flows. Notice that some classes have several thousands of flows, while others count no more than few tens of flows.

Finally, 3 completely different data sets have been collected from our campus network and will complete our analysis. This represent a different scenario, in which traffic generated by more than 20,000 students, professors and staff members is present. In this scenario, there is little P2P traffic, since a firewall blocks plain P2P protocols.

2.1.1 Performance metrics

Performance of a classifier are typically assessed considering the *overall accuracy, recall, precision* and *F-measure* [13].

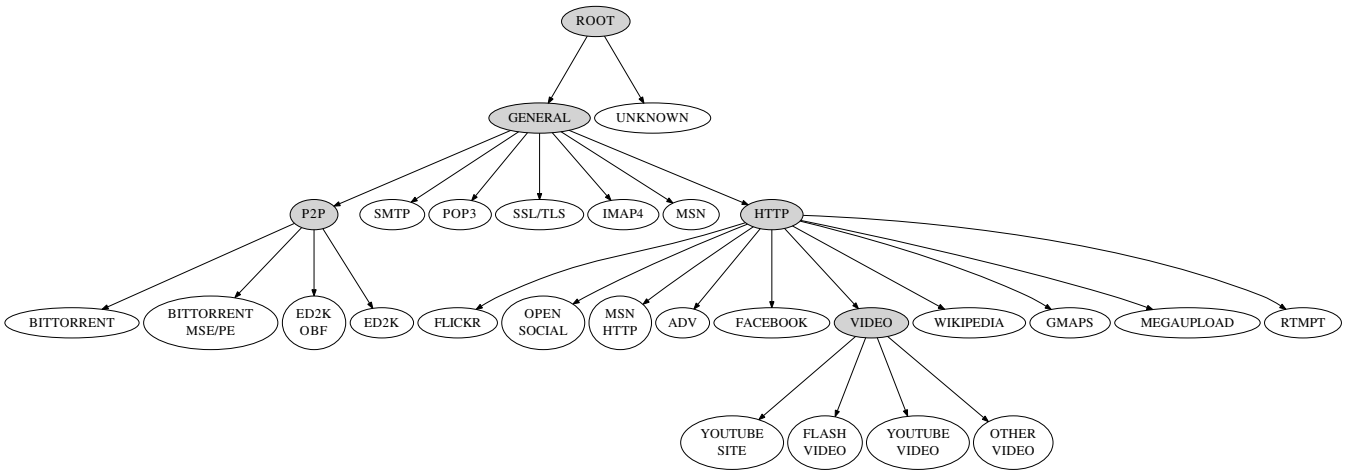


FIGURE 2.2: Tree structure for the Hierarchical classifier.

Accuracy, is the ratio of the sum of all True Positives (prediction and ground truth are in agreement) to the sum of all tests, for all classes. Accuracy however is biased toward the most predominant class in a data set.

Precision, for a given class, is the ratio of True Positives and the sum of True Positives and False Positive (a sample of another class that has been labeled as of this class). It determines the fraction of samples that actually turns out to be positive in the group the classifier has declared as a positive class. The higher the precision is, the lower the number of false positive errors committed by the classifier.

Recall, for a given class, is the ratio of the True Positives and the sum of True Positives and False Negatives (a sample of the class is labeled as not). It measures the fraction of positive samples correctly predicted by the classifier. Classifier with large recall have very few positive example misclassified as the negative class.

F-Measure, a widely used metric in classification, weights both precision and recall in a single metric by taking the harmonic mean: *2 × Recall × Precision / (Recall + Precision)*.

In this work we report Recall and F-Measure metrics to assess per-class performance, while Accuracy will be provided when comparing overall results. All experiments have been carried out using *RapidMiner* [14] on a 8-cores Intel Xeon E5450 based server PC equipped with 32GB of ram. Computational costs will be reported considering this setup.

2.2 Hierarchical Classification

All classification algorithms share the same idea: given a description of the object to classify in terms of "features", find the most likely class according to a model that has been derived from a set of objects properly labeled, i.e., the "training set". Which algorithm and which features to use are key points to address in the design of the classifier. Our proposal has been designed by performing a thorough selection among different alternatives. The key and novel idea we leverage is to build a classification scheme which is based on a *hierarchy of classifiers*. This allows each classifier to work on a limited subset of classes and on a specialized subset of features, i.e., the features that are most suitable to distinguish among the considered classes. In the following we describe the overall process.

2.2.1 Hierarchy Definition

All classification algorithms are known to suffer when the number of classes they have to choose among increases. For example, it can be easy to split P2P traffic from HTTP traffic. How to however correctly classify the single application running over HTTP may be trickier. Moreover, for example the features that allow to separate P2P traffic from HTTP traffic may be useless when trying to separate YouTube from Facebook flows.

The key idea we leverage in this work is to design a classification scheme based on a *hierarchy of classifiers*. At first, the flow will be classified into few coarse classes. At the following stages, finer and finer grained classification is achieved. To define the hierarchy, we rely on our domain knowledge. Fig. 2.2 shows the Hierarchical classifier we propose in this work. Gray nodes are sub-classifiers and white nodes represent the final classes. We use five classifiers. At the root, flows are split among the "known" and "unknown" classes. Then, a general classifier decides among protocols that we know it is easy to distinguish: P2P, HTTP, SMTP, etc., are well defined classes that have been already shown to be easily identified using behavioral algorithms [9]. At the next step, some classes can be further split into subclasses. For example, P2P traffic is split into BitTorrent versus eMule, while HTTP traffic is split into finer grained applications. Finally, video streams over HTTP will be further classified among YouTube streams, YouTube web site objects, generic Flash Video, or other Video streams.

In the following, for comparison purposes, we consider a classical classifier based on a single stage, in which the classification decision has to be taken at the root node directly. We refer to this case as "Flat" classifier.

TABLE 2.2: Selected feature on the server to client traffic.

Classifier	TCP port	RST sent	PURE ACK sent	unique bytes	data pkts	data bytes	RFC1323 ws	RFC1323 ts	WNDSCALE factor	Server SACK req.	MSS	max segment size	min segment size	RWND max	RWND min	CWND max	CWND min	initial CWND	stdev RTT	min TTL	max TTL	last segment time	msg 1 size	msg 2 size	msg 5 size	msg 7 size	msg 8 size	msg 10 size	# segments	seg 1 size	seg 2 size	seg 3 size	seg 4 size	seg 5 size	seg 6 size	seg 7 size	seg 8 size	seg 9 size	seg 10 size	seg 7 IPG	# features
Flat	x	x					x	x				x	x		x		x	x		x		x	x	x	x	x	x		x	x	x	x	x	x	x	x	x	x			26
ROOT							x		x			x								x	x								x	x	x	x	x								10
General	x						x					x	x		x		x	x												x	x	x									10
P2P										x			x				x				x			x					x		x		x								8
HTTP	x	x	x	x	x	x			x		x	x		x	x	x		x	x	x			x	x			x	x	x		x									x	22
Video								x		x	x			x	x		x						x	x						x			x						x		11

2.2.2 Feature Selection

For each classifier, the proper set of features must be selected. In the context of traffic classification, most of the proposals so far relies on a set of features that have been chosen based on authors' domain knowledge. For example, [15] uses a list of features that the authors think to be good to distinguish P2P traffic from client-server traffic. Similarly, [16] uses the size of the first packets as features given the focus on the "early traffic classification". While the choice of the features can be intuitive when dealing with few classes of traffic, it becomes suddenly difficult to properly select the most prominent features that allow to distinguish between a large list of applications. For example, how to distinguish YouTube video streams from other flash video streams?

In machine learning field, well-known algorithms have been proposed to solve the problem of feature selection, i.e., techniques for selecting a subset of relevant features for building robust learning models [17]. Among the different algorithms, the "minimum-Redundancy-Maximum-Relevance" (mRMR) algorithm is considered as the state-of-the-art [18]. mRMR is an approximation of the theoretically optimal maximum-dependency feature selection that maximizes the mutual information between the joint distribution of the selected features and the classification variable. The input of the feature selection algorithm is a "training" data set, in which *all possible* flow features are provided and flows are correctly labeled. The algorithm selects then the subset of most relevant features to properly assign the correct class. As initial set of features, we use all behavioral layer-4 features that are provided by Tstat. The overall list includes more than 200 features, most of which have been proposed in the past literature. For the sake of brevity we do not report the complete list.

Feature selection can be run independently for each classifier. This allows us to actually select a *different set of features* for each sub-classifier, a key and desirable property.

The results of the feature selection are reported in Table 2.2 which report the subset of features selected for each classifier considering server to client flow features. Three considerations hold: First, the list of selected features includes some intuitive choices, but also some unexpected selections. For example, the server RWND scale factor have been found to be useful by the ROOT and HTTP classifier only. Second, different classifiers use different features. Third, the Flat classifier has to consider 45 (26+19) features entailing a larger complexity; at most 35 (22+13) features have been selected for any hierarchical stage.

2.2.3 Classification Algorithm Selection

The proper classification algorithm has to be selected among the large number of approaches discussed in the literature: Naive Bayes, Bayesian Kernel Estimation, Rule Based, Decision Trees, Neural Networks, Support Vector Machine (SVM), K-Nearest Neighbor (K-NN) are popular techniques, each leveraging some different idea [17]. Most of these have also been used in the context of traffic classification [9, 19] with good results when dealing with *few* classes.

We run a preliminary set of experiments to see which is the classifier that would guarantee the best performance. For each algorithm, we consider the training data set. We apply the ten-fold cross-validation methodology to estimate the accuracy of each classifier.

Figure 2.3 reports the average among classes of the F-Measure and the Recall, on top and bottom plot, respectively. Performance of the Flat classifier (black) and the Hierarchical classifier (gray) are reported for each classification algorithm. First, notice that we were not able to complete the test of the SVM and the K-NN Flat classifiers, that were not able to complete the experiment after three days. As well known, dealing with a large number of classes and features poses computational issues for some algorithms. The Hierarchical solution scales better, since each classifier has to deal with a smaller number of classes and features. More details are provided in Section 2.3.3.

Second, the Hierarchical classifier outperforms the Flat classifier considering any classification algorithm. Average F-Measure and Recall are both smaller than 80% for the Flat classifier. On the contrary, the Hierarchical classifier achieves performance higher than 95% for both metrics when a Decision tree is used. This suggests that the problem in designing a Flat classifier is not in the choice of the classification algorithm; rather, any algorithm performs poorly with a large number of traffic classes. Therefore some ingenuity has to be used to improve performance, justifying the need for a hierarchical solution.

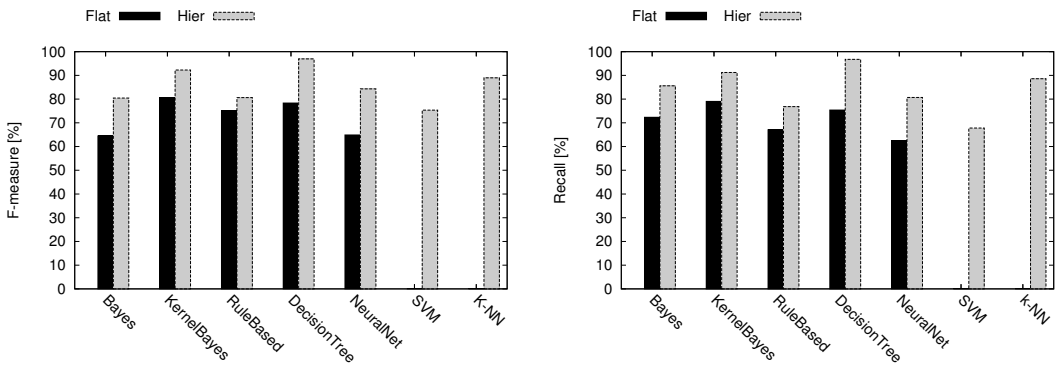


FIGURE 2.3: Comparison of different classification algorithms. Average F-measure and Recall considering ten-fold cross-validation test on a 1h long trace from ISP.

For the Hierarchical classifier, the Decision Tree is the only classifier that achieves excellent results for all sub-classifiers in the hierarchy. Other algorithms exhibit more variable results. For example, the SVM performs very well for P2P classification, but it performs poorly for Video classification. Note that the Hierarchical classifier allows also the selection of different classification algorithms for each internal sub-classifier. In the following we restrict our attention to the Decision Tree classifier only.

2.3 Experimental results

We now provide a more extensive and thorough performance evaluation. We start by considering the performance of the Hierarchical versus Flat classifier considering each subclass. We consider as training data set the ISP trace collected at h.17, and the h.18 trace for testing. Figure 2.4 details the results. Top plots compare the absolute F-Measure for each class; while plots on the bottom quantify the improvement guaranteed by the Hierarchical classifier for F-measure and Recall, respectively. Classes appear in the same order as in Table 2.1. Results allow to appreciate the benefit of the Hierarchical approach. F-Measure improves for all classes by 28% on average. Notably, some classes are basically ignored by the Flat classifier, e.g., MSN. On the contrary, the Hierarchical classifier deals with MSN flows at the Generic sub-level, where only 7 classes have to be identified. The F-Measure for MSN class then tops to 98%.
Recall improves by about 10% overall, since for some classes the Flat classifier is already achieving good results. In some cases, the Recall decreases by some percentage points. Notice that these are border cases in which the Flat classifier reaches good Recall, but bad F-Measure, i.e., bad Precision. For instance, consider the You-Tube Video class. In this case, the number of False Negatives is small, but the number of False Positives is very high. The Hierarchical classifier improves the F-Measure (thus lowering the False Positive) and overall it performs much better also in this case (F-measure grows by 80%).

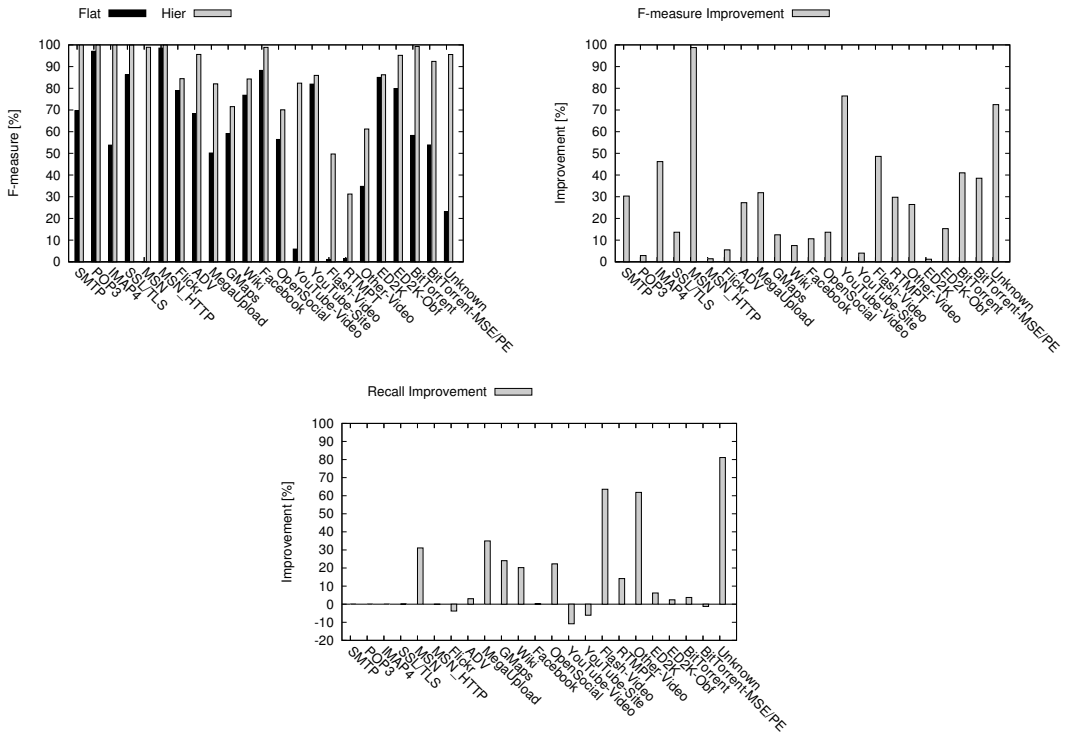


FIGURE 2.4: F-Measure and Recall for each class for the Hierarchical and Flat classifiers. Training on h.17 data set and testing on h.18 data set. ISP trace.

Notice that also popular classes are misclassified by the Flat classifier. For example, the Unknown class has very poor performance. Since the Recall is only 15%, the number of False Negative is very large. This is clearly critical, making it impractical to use the Flat classifier given that most of the unknown flows will be classified as one of the known classes. The Hierarchical classifier on the contrary is able to achieve excellent performance, with Recall and F-Measure higher than 95%.

2.3.1 Robustness versus time

One interesting question to answer is how the performance of a classifier change over time. Assume to train the classifier with a given data set collected at a given time. What happens if the classifier is used later? To answer this question we consider the whole ISP data set, which is 22h long. Training of the classifier is done considering the usual h.17 data set. Then performance is evaluated on the other 21 different data sets. To validate the statistical significance of the performance improvements, we used the paired t-test [20] at 95% of significance level for each data set. The overall Accuracy is reported in Figure 2.5. It shows that the Hierarchical classifier significantly outperforms the Flat classifier. The former guarantees an overall accuracy always higher than 88%, while the latter achieves reasonable performance only during night time when the traffic

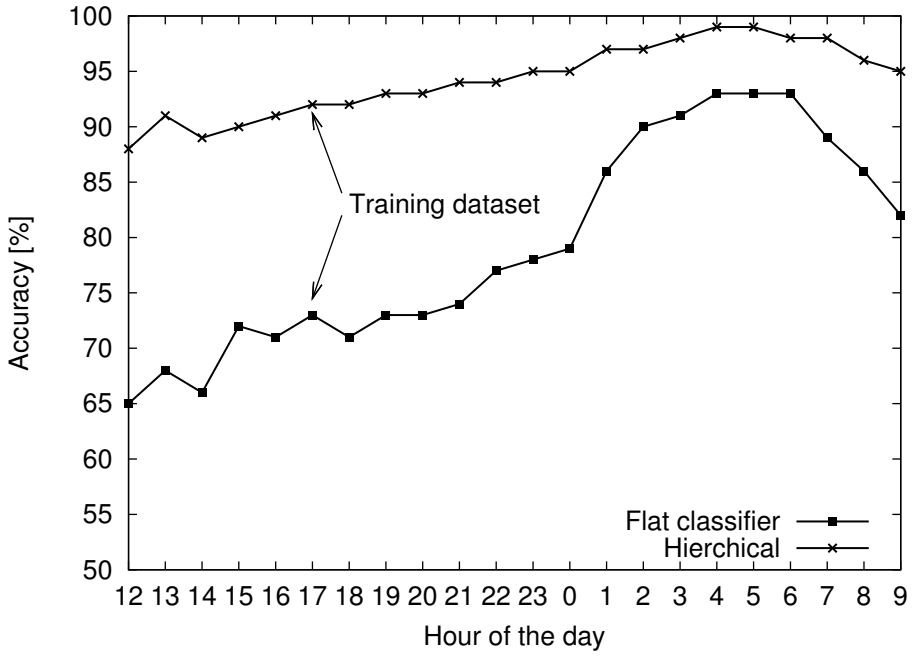


FIGURE 2.5: Accuracy of the Hierarchical classifier when used in real time. One day long data set from ISP.

is dominated by P2P traffic and thus few classes are "active". During the day it barely reaches 70% of overall Accuracy.

2.3.2 Experiment considering other data sets

We have repeated the experiment considering other data sets. For the sake of brevity, we report only one experiment considering two 1-hour long traces collected from our campus LAN at h.15 and h.19 on a normal working day. As previously, training has been done considering the h.15 trace and testing is done on the h.19 trace. The Recall improvement is reported in Figure 2.6. Also in this case the Flat classifier provides good results for some classes, while it completely misses others, while the hierarchical classifier improves results especially for less popular classes.

2.3.3 Computational Complexity

To gauge the overall computational costs of the classifiers, we were able to completely classify a 1h long data set in less than 1 second and using a very limited amount of memory; i.e., classification cost are very light. The Flat classifier can classify 89,750 flows per second, while the hierarchical classifiers tops to more than 368,400 decision per second. This results are mainly due to the adoption of a Decision Tree classifier at each node. Memory cost is also negligible. Notice that the Hierarchical classifier can be naturally implemented using parallel processes organized in a pipeline. These results show that it is possible to actually use the classifier in on-line system.

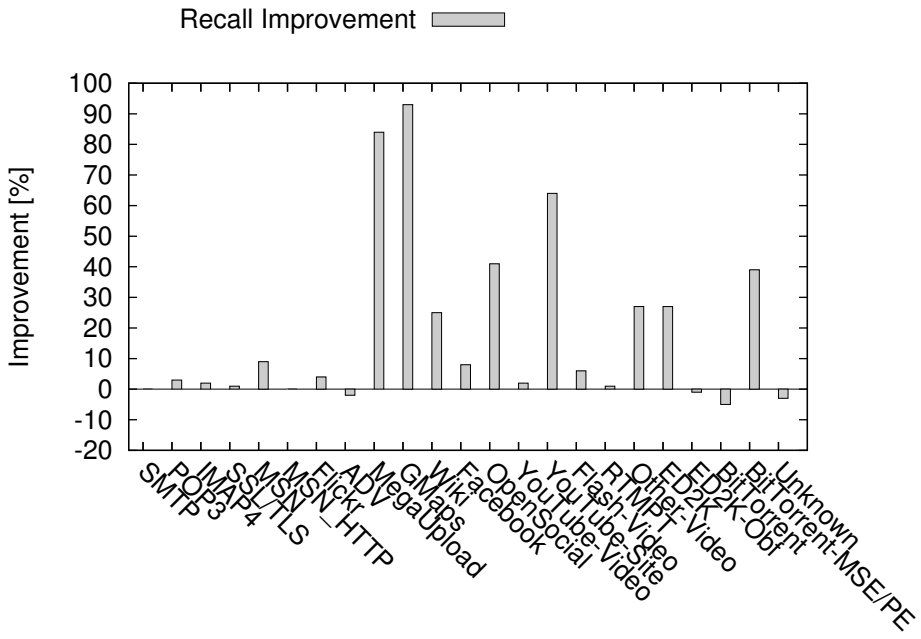


FIGURE 2.6: Improvement for each class for the Hierarchical and Flat classifiers. Testing on Campus data set.

TABLE 2.3: Computational and memory cost for different classifiers to execute a training phase on a 1h long campus data set.

	Flat	Root	General	HTTP	P2P	Video	Total
CPU time [s]	7849	1207	389	589	48	74	2307
Memory [GB]	29	17	11	13	3.4	2.5	46.9

Considering training cost, Table 2.3 reports the overall time need to perform a training on a 1h long trace. The campus network data set is considered, in which a total of 1.6M flows is present. Both total CPU execution time and total memory usage are reported considering the training phase. As it can be seen, the adoption of a hierarchy of classifiers allows to greatly reduce the computational cost and the maximum memory required at any given time. Each sub-classifier indeed benefits from the reduced number of classes and features. Moreover, fewer flows have to be considered to build the model and only those flows that belong to the subset of considered classes have to be taken into account. Note that the training phase cost is relatively important since it has to be seldomly performed off-line.

Chapter 3

SeLeCT: Self-Learning Classifier for Internet Traffic

As we have already discussed in Chapter 2, a critical part of network management and traffic engineering is the ability to identify applications and protocols originating traffic flows. To provide network visibility, in the last years several classification techniques have been proposed (see [9, 19] and references therein). Until a decade ago, *port-based* approaches were very popular. The effectiveness of pure port-based approach has diminished even if it has been shown that port numbers carry valuable information about the application and/or protocol [9]. Over the last few years *deep packet inspection* (DPI) has become popular [19], and *behavioral* techniques have been investigated since the seminal work of [15]. And in the previous Chapter, we proposed a novel approach to push further behavioral classification techniques.

However, both DPI and behavioral classifiers share some limitations. First, to achieve a high classification accuracy, either a cumbersome protocol reverse engineering to identify the signatures in DPI, or a tedious process to generate an accurate training set for behavioral classifiers is required. In other words, both approaches require *training*. Second, and most critical, the classifiers *can identify only the specific applications they have been trained for*. All other traffic is aggregated either in a generic class labeled as "unknown", or, even worse, it is mislabeled as one of the known applications. In other words, these classifiers cannot identify the introduction of a new application, or changes in the applications' protocol or behavior, unless a re-training phase is entered. Designing a classification engine capable of automatically identifying new emerging protocols is still an open and challenging research topic.

In this Chapter, we propose SeLeCT, a novel algorithm that overcomes the limitations highlighted above. Our goal is to provide a deeper network visibility for operators. In other words, we intend to offer the ability to semi-automatically identify prominent

classes of traffic, targeting network management and traffic engineering operations[1]. SeLeCT proves to be able to expose classes of traffic which are very specific and possibly are not already known to the operator. For example, SeLeCT has been able to separate Google Mail traffic from other mail services. It thus automatically allows to discover new classes of traffic, allowing arbitrary definition of labels.

In SeLeCT, we leverage unsupervised data mining algorithms to automatically split traffic into homogeneous subsets or clusters. We consider *flows* as the target of the classification. Each flow is characterized by using simple layer-4 metrics, like segment size and inter-arrival time. These features are known to carry valuable information about the protocol and/or application that generated the flow [9].

However, they perform not as good in the context of unsupervised (i.e. clustering) algorithms. Hence we have to adopt some ingenuity in order to improve cluster homogeneity. To overcome the limitation of off-the-shelf algorithms, we design an *iterative clustering* procedure in which a filtering phase follows each clustering phase to eliminate possible outliers. Filtering is based on the still valuable information provided by port numbers. Note that port number information is not embedded in a metric space, e.g., the distance between port 79 and 80 is not different from the one between port 80 and 8080. As such, it is hard to integrate port number as a simple feature into classical clustering algorithms.

Using traffic traces collected in different years from various ISPs located in 3 different continents, we show that the iterative clustering process leads to clusters with excellent properties. First, SeLeCT generated only a few cluster in each of these traces (typically less than 150). Second, clusters are very pure, i.e., the overall homogeneity of the clusters is close to 100%. This allows to easily inspect and label each cluster, thus assigning a proper label to all flows belonging to the same cluster.

As soon as some labels are assigned to flows, SeLeCT will automatically inherit them for classification of flows that arrive in the future.

We refer to this as *adaptive* or *progressive* seeding since flows labeled in the past are used to seed the subsequent datasets. Notably, this will minimize the bootstrapping effort required to label applications, and manual intervention is mainly required for the initial label assignment. This mechanism allows to naturally grow the intelligence of the system such that it is able to automatically adapt to the evolution of protocols and applications, as well as to discover new applications.

The idea of leveraging semi-supervised learning has been initially proposed in [21], where the authors leverage the standard k-means to construct clusters. Part of the flows to be clustered are assumed to be already labeled, and a simple voting scheme is used to extend the dominant label to the whole cluster.

[1] SeLeCT is *not* intended for security purposes where every single bit, packet, and/or flow must be carefully examined.

SeLeCT follows similar principles, extending the idea with i) iterative port filtering and ii) multi-batch seeding which, as we will see in Section 3.5, allow to significantly boost overall performance achieving 98% accuracy in practical cases. The iterative clustering algorithm and self-seeding approach provide several advantages: the number of clusters is reduced to less than 150, while at the same time homogeneity is significantly increased. This simplifies the labeling process so that manual inspection becomes almost trivial. Furthermore, the SeLeCT self-seeding process is more robust and results obtained from actual traffic traces show how SeLeCT helps in automatically identifying *fine grained classes of traffic* (e.g, IMAP vs POP3, XMPP vs Messenger), and even unveiling the presence of unknown/undesired classes (e.g., Apple push notification, Bot/Trojan, or Skype authentication traffic). In identifying standard protocols SeLeCT proves to be even more robust than professional DPI based tools which were fooled by non-English customizations of protocol error messages.

3.1 Related work

3.1.1 Clustering Algorithms

Data mining techniques may be grouped in two families: *supervised* and *unsupervised* techniques [13]. Supervised algorithms assume the availability of a training dataset in which each object is labeled, i.e., it is a-priori associated to a particular class. This information is used to create a suitable model describing groups of objects with the same label. Then, unlabeled objects can be classified, i.e., associated to a previously defined class, according to their features. For unsupervised algorithms, instead, grouping is performed without any a-priori knowledge of labels. Groups of objects are clustered based on a notion of distance evaluated among samples, so that objects with similar features are part of the same cluster.

Supervised algorithms achieve high classification accuracy, provided that the training set is representative of the objects. However, labeled data may be difficult, or time-consuming to obtain. Semi-supervised classification addresses this issue by exploiting the information available in unlabeled data to improve classifier performance. Many semi-supervised learning methods have been proposed [22], unfortunately, no single method fits all problems.

The semi-supervised learning approaches closest to our proposal are [23] and [24]. Both labeled and unlabeled data are clustered by means of (variations of) known clustering algorithms (k-means in [23] and SOM in [24]). Next, labeled data in each cluster is exploited to assign labels to unlabeled data. Finally a new classifier is trained on the entire labeled dataset. While we exploit a different, iterative clustering approach to group data, our labeling process is similar to [23]. Due to its iterative refinement process,

the approach adopted in SeLeCT is also particularly suited to model traffic flow changes, because it allows a seamless adaptation of the obtained traffic classes to traffic pattern evolution.

3.1.2 Key features of SeLeCT

Some of the key features of SeLeCT are:

• *Adaptive classification model.* A semi-supervised learning approach allows SeLeCT to learn information from unlabeled data with simplified manual intervention. Once some labels are provided, SeLeCT automatically adapts the model to changes in the traffic.

• *Simple iterative approach.* SeLeCT is based on k-means, a simple yet effective clustering algorithm. It uses k-means as a building block in an iterative clustering refinement process, which allows leveraging specific Internet traffic features such as the server port that cannot be integrated into classical clustering algorithms in a straightforward fashion. This approach yields strongly cohesive clusters and provides an almost complete coverage of the considered flows.

• *Leverages layer-4 features.* SeLeCT relies on the availability of flow level features that can be easily acquired at the beginning of the flow, and it does not assume to see both directions of traffic.

• *Limited complexity.* SeLeCT can run in real time by constantly monitoring the incoming traffic, creating batches of flows, and processing these batches before the next batch accumulates.

3.1.3 Applications to traffic classification

The application of unsupervised techniques is not new in the traffic classification field. [25] is one of the preliminary works and shows that clustering techniques are useful to obtain insights about the traffic. In [26] supervised and unsupervised techniques are compared, demonstrating that unsupervised algorithms can achieve performance similar to the supervised algorithms. Other works compare the accuracy of different and standard unsupervised algorithms [16, 27, 28]. In general, the techniques presented in these works achieve a moderate accuracy and they typically identify several hundreds of clusters, therefore questioning the applicability of this methodology in practice.

Recently, [21, 29–32] have introduced the *semi-supervised* methodology in the context of traffic classification.

[21] is among the first works that proposes also a simple labeling algorithm. It uses the off-the-shelf k-means algorithm and present a performance evaluation considering a trace collected from a Campus and a small residential network. Limited ground truth is available and only coarse classes are considered (e.g., P2P, HTTP, EMAIL, CHAT, etc.). Results show that to achieve good accuracy, a still large number of clusters must

be used ($k \geq 400$) and the labeled dataset must be large (more than 15% of flows must be already labeled). We explicitly compare the performance of SeLeCT against the solution proposed in [21] in Section 3.5.

In [29] the authors propose a simple clustering algorithm based on information entropy to group flows. Clusters are then labeled using some ad-hoc engineered algorithm that can coarsely identify classes like P2P or Client/Server traffic. Limited performance evaluation is provided considering traffic generated by 20 hosts only. Neither learning nor seeding is proposed. In [30], the authors proposed advanced unsupervised and semi-supervised machine learning algorithms to cluster flows. 22 (bi-directional) flow level features are used, which include packet size and inter-arrival time. Performance evaluation considers two small datasets of 4,000 flows each. Accuracy reaches 85%. [31] proposes a semi-supervised method which extends [21]. As features, the destination IP address, server port and transport protocol are considered. k-means is used as basic building block. Accuracy, evaluated considering two traffic traces, tops 90%. In [32], the authors propose an unsupervised traffic classification that uses both flow features and packet payload. Using a bag-of-words approach and latent semantic analysis, some clusters are identified. Performance is evaluated using a single trace and reaches 90% of accuracy.

In all cases, SeLeCT achieves better results in terms of classification performance, provides finer grained visibility on traffic, and offers a simple self-seeding mechanism that naturally allows the system to increase its knowledge..

3.2 Problem statement

We consider *directed traffic flows* as the objects to classify. A directed flow, or flow for short, is defined as the group of packets that have the same five tuple $F = \{srcIP, dstIP, srcPort, dstPort, protocol\}$. Note that packets going in opposite directions belong to two directed flows. For instance, in a traditional TCP connection, packets sent by the client belong to a directed flow, and packets sent by the server belong to a different flow. Considering directed flows allows the classifier to work even in presence of asymmetric routing (backbone networks for instance).

We assume all packets traversing a link are exposed to the classifier which keeps track of per-flow state. A flow F is identified when the first packet is observed; the flow ends when no packets have been seen for a given time ΔT. TCP signaling segments may be used to detect appropriate flow start and end. As suggested in [33], we consider a conservative value of $\Delta T = 5min$.

For each flow F, a set of features $A(F) = \{a_1^{(F)}, a_2^{(F)}, \ldots, a_n^{(F)}\}$ is collected. These features are used by SeLeCT to characterize flows and take the classification decision.

name	DateTime	Place	Type	IP	Flow
Dataset-1	Aug05 1pm	S.America	backbone	108k	527k
Dataset-2	Sep10 10am	Asia	backbone	111k	1.8M
Dataset-3	Aug11 2am	Europe	access	111k	885k
Dataset-4	Aug11 5pm	Europe	access	190k	2.3M

TABLE 3.1: Datasets used in the Chapter for performance evaluation. The table includes flows for which features can be computed.

The goal of SeLeCT is to assign a proper application to each flow F based on the sole knowledge of the flow feature set $A(F)$.

In this work, we choose behavioral features that are well known to carry useful information about the application and protocol used at the application layer [9, 19]. In particular, we select: (i) The server port $srvPrt$, (ii) the length of the first n segments with payload, and (iii) their corresponding inter-arrival-time. Note that only flows that have more than n segments can be classified. The impact of the choice of n is discussed in Section 3.3.

Formally, let $L(i_F)$ be the length of the i-th segment of flow F, and let $t(i_F)$ be its arrival time. The i-th inter-arrival time $\Delta t(i_F)$ is $\Delta t(i_F) = t(i_F) - t(i_F - 1),\ i_F > 1$. Then

$$A(F) = \{srvPrt,\ L(i_F),\ \Delta t(i_F)\ \forall i_F \leq n \wedge L(i_F) > 0\}$$

The choice of which features to consider is a matter of optimization and several works in the literature have proposed and investigated possible alternatives. Our choice stems from the following intuitions: (i) keep the feature set limited, (ii) include generic layer-4 features that can be easily computed, and (iii) use features that can be collected during the beginning of a flow so that we can classify flows in real-time (i.e., minimize the time required for identification). It is out of the scope of this Chapter to compare and choose which are the most suitable features to use. We will consider this as a part of our future work. However, given the high accuracy of SeLeCT, we believe that it may be difficult to improve it by considering a wider/different set of features.

3.3 Datasets to evaluate SeLeCT

In this section, we briefly describe the datasets that we collected and used to evaluate SeLeCT. We provide more details in Section 3.5. Table 3.1 summarizes the main characteristics of the datasets. We collected four different traces from access and backbone networks of large ISPs[2]. Each dataset is a 1-hour long complete packet trace including the packet payloads. We selected these traces to create a very heterogeneous benchmarking set. They include backbone and access scenarios, day and night time periods, different years, and users from three different continents.

[2]Due to NdA with ISPs we are not allowed to share the original traffic traces.

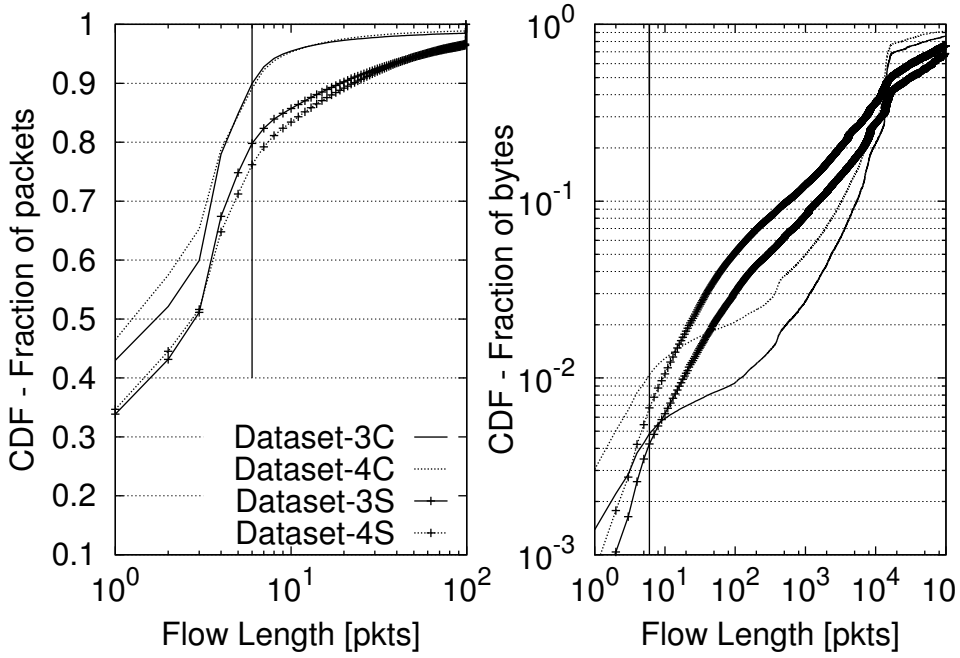


FIGURE 3.1: CDF of the flow length in packets (on the left), and bytes (on the right). The vertical line is in correspondence of 6 data packets.

In this work, we focus on TCP traffic only as most applications today rely on TCP. The extension of SeLeCT to UDP traffic is straightforward and is not further investigated in this Chapter.

For each trace, we generate two separate datasets - the set of flows originated by clients (i.e., hosts actively opening the TCP connection) and the set of flows originated by servers (i.e., hosts that replied to the connection request). A letter 'C' (client-to-server) or 'S' (server-to-client) is appended at the dataset name when needed. Overall, the oldest trace - Dataset-1 - was collected in 2005 from a major ISP in South America; it contains more than half million TCP flows involving more than $100,000$ hosts. Dataset-2 was collected from the peering link of an ISP in Asia in September 2010. Finally, Dataset-3 and Dataset-4 were collected at different times of the day from the same vantage point in Europe during August 2011. Dataset-3 was collected at 2am in the night, while Dataset-4 was collected at 5pm. The latter contains about 2.3 million flows directed to more than $190,000$ hosts. We will primarily use the last two datasets for deeper investigation in the rest of the Chapter.

Only flows that have at least n data packets can be considered by SeLeCT. So the first question to answer is how much traffic can be classified by SeLeCT for different values of n. Figure 3.1 reports the Cumulative Distribution Function (CDF) of the number of packets (on the left) and bytes (on the right) carried by flows of different length. For the sake of simplicity, let us focus on Dataset-3 and Dataset-4, which are the two most recent datasets. The CDF of the fraction of packets (on the left plot) shows that the large

```
1: Main()
2: Output: set C of labeled clusters
3: S = ∅
4: while (newbatch B) do
5:    ProcessBatch(B, U, S, C, NS)
6:    S = NS
7: end while
8:
9: ProcessBatch(B, U, S, C, NS):
10: Input: Set B of new flows, set S of seeds
11: Output: set C of labeled clusters, set NS of new seeds
12: B' = B ∪ S ∪ U /* Merge new flow, seeding set,
13:                    and past outliers */
14: C' = doIterativeClustering(B');
15: C = doLabeling(C');
16: NS = extractSeeds(C);
```

Algorithm 1: SeLeCT Main loop.

majority of the flows are "mice", i.e., flows with few packets. For instance, 90% of client flows have no more than 6 data packets (highlighted by the vertical bar). However, by looking at the CDF of bytes (reported on right plot), we observe that the mice account for no more than 1% of the volume of traffic (notice the log scale on y-axis). Thus, by considering flows that have at least 6 data packets: (*i*) we allow a richer description of each flow characteristics (*ii*) we are discarding the large majority of mice flows and (*iii*) we are looking at more than 99% of traffic volume. Based on these observations, in the rest of the Chapter we use $n = 6$. Thus, as any statistical classifier, SeLeCT targets long-lived flows.

3.4 The SeLeCT algorithm

We consider a scenario in which traffic is sniffed in real time and new flows enter the system continuously. Flows are processed in *batches*. A new batch $\mathcal{B}$ is formed as soon as a given number of valid flows is observed. The probe monitors packets and rebuilds flows. For a given flow, as soon as 6 data packets are observed the flow identifier and features are dispatched to a buffer where the batch is being formed. When the batch reaches the target number of flows, it is dispatched to the classification algorithm, and a new batch starts.

SeLeCT analyzes each batch of newly collected flows via the **ProcessBatch()** function shown in the pseudo-code reported in Alg. 1. This function takes in input

- $\mathcal{B}$, the batch of new flows;

- $\mathcal{U}$, the set of previous outliers that were not assigned to any class when processing the previous batch;
- $\mathcal{S}$, the set of *seeding flows*, i.e., flows already analysed in past batches for which SeLeCT was able to provide a label;

As output, it produces

- $\mathcal{C}$, the set of clusters;
- $\mathcal{NS}$, the set of new seeds that are extracted from each cluster;
- $\mathcal{U}$, which contains the set of new outliers;

Its main steps (see Alg. 1) are (i) clustering batch data to get homogeneous subsets of flows (function **doIterativeClustering()**), (ii) flow label assignment (function **doLabeling()**), and (iii) extraction of a new set of seeds (function **extractSeeds()**).
Note that flows that are not assigned to any cluster are returned in the $\mathcal{U}$ set. Those flows are then aggregated in the next batch, so that they can eventually be aggregated to some cluster[3]. In the following we detail each step of the batch processing.

3.4.1 Iterative clustering

Clustering algorithms group objects with similar characteristics [13]. Objects are described by means of features which map each object to a specific position in a hyperspace. The similarity between two objects is based on their *distance*. The closer the two objects are, the more likely they are similar and thus should to be grouped in the same cluster. Typically, the Euclidean distance is used.
Iterative clustering is the core of SeLeCT. It exploits the k-means clustering algorithm [13] to group flows into subsets or *clusters* which are possibly generated by the same applications.
We selected the k-means algorithm since it is well understood and it has been previously used in previous works. We tested also other clustering algorithms like DBSCAN [13]. Results are similar or worse, with a trickier sensitivity to parameter settings.
In this context, it is natural to consider two flows with similar packet lengths and inter-arrival times to be close (i.e., to be likely generated by the same application). However, the same property does not hold for the *srvPort* feature. For instance, two flows directed to port 25 and to port 80 are not more likely to be similar than two flows directed to port 80 and to port 62000. The *srvPort* feature is a nominal feature [13], thus it cannot be included in Euclidean distance computations.

[3]It would be possible to limit the number of batches some flows may be still in the $\mathcal{U}$ set and output them in a "unclassifiable" set to avoid delaying classification process.

Still, the *srvPort* is an important feature for traffic classification [9]. Two cases can be distinguished: protocols and applications i) running on one (or more) specific *srvPort* on servers, or ii) running on a random *srvPort* selected by each server. We denote them as *dominatedPort* and *randomPort* protocols respectively. In both cases, the *srvPort* carries valuable information if applied as a *filter*.
In the past, several researchers have applied clustering algorithms to traffic analysis [21, 27]. However, to the best of our knowledge, none of the previous works exploited the specific characteristic of the *srvPort* feature in a clustering process. This is mainly related to the fact that port numbers are not embedded in a metric space. Thus ingenuity is required to smartly include them. In our work we engineer an iterative procedure to identify clusters of flows in which the *srvPort* information is used to *filter* elements in each cluster. As reported in Alg. 3, we devise an iterative process, in which clustering phases and filtering phases alternate. We use a set-based notation. Names of the sets are defined in the pseudo code.

3.4.1.1 The filtering procedure

1: **doFiltering**($\mathcal{I}$, $\mathcal{C}$, $\mathcal{U}$, $\mathcal{DP}$, $portFraction$, $DominatingPhase$)
2: *Input*: cluster $\mathcal{I}$ of flows to be filtered, $DominatingPhase$ flag to select the filtering
3: *Output*: set $\mathcal{C}$ of clusters, set $\mathcal{U}$ of outliers, set $\mathcal{DP}$ of dominant ports
4: $\mathcal{DP} = \emptyset$
5: **if** $||\mathcal{I}|| < minPoints$ **then**
6: $\quad \mathcal{U} = \mathcal{U} \cup \mathcal{I}$; return
7: **end if**
8: **if** $DominatingPhase ==$ TRUE **then**
9: $\quad$ /* Processing dominatedPort cluster */
10: $\quad$ **if** (topPortFreq($\mathcal{I}$) $> portFraction$) **then**
11: $\quad\quad \mathcal{C}' =$ getFlows($\mathcal{I}$,DP)
12: $\quad\quad \mathcal{C} = \mathcal{C} \cup \mathcal{C}'$ /* Add the filtered cluster to $\mathcal{C}$ /*
13: $\quad\quad \mathcal{R} = \mathcal{I} \setminus \mathcal{C}'$
14: $\quad\quad \mathcal{U} = \mathcal{U} \cup \mathcal{R}$ /* Put discarded flows in $\mathcal{U}$ */
15: $\quad\quad dp =$ dominantPort($\mathcal{I}$)
16: $\quad\quad \mathcal{DP} = \mathcal{DP} \cup \{dp\}$ /* Record dominant port */
17: $\quad$ **else**
18: $\quad\quad \mathcal{U} = \mathcal{U} \cup \mathcal{I}$ /* $\mathcal{I}$ flows must be reclustered */
19: $\quad$ **end if**
20: **else**
21: $\quad \mathcal{C} = \mathcal{C} \cup \mathcal{I}$ /* $\mathcal{I}$ is a good cluster at last */
22: **end if**

Algorithm 2: Filtering of clusters.

The filtering procedure is reported in Alg. 2. Filtering is performed on the cluster $\mathcal{I}$ provided as input. First, **doFiltering()** discards clusters which have less than $minPoints$

```
1: doIterativeClustering(B)
2: Input: Set B of flows to be clustered
3: Output: set of clusters C, set of outliers U
4: U = B, DP = ∅
5: for (step=1; step ≤ itermax; step++) do
6:    C' = k-means(U)
7:    U = ∅
8:    for I in C' do
9:       /* look for dominatedPort clusters first */
10:      doFiltering(I,C,U,DP,portFraction,true)
11:   end for
12: end for
13: /* Last step: process random port clusters */
14: for dp in DP do
15:   delFlows(U,dp) /* Discard flows still to DP */
16: end for
17: C' = k-means(U)
18: for I in C' do
19:   /* look for randomPort clusters now */
20:   doFiltering(I,C,U,DP,0,false)
21: end for
22: return C, U
```

Algorithm 3: Iterative Clustering

flows to avoid dealing with excessively small clusters. Discarded flows are returned in set $\mathcal{U}$, the set of unclustered flows that will undergo a subsequent clustering phase (lines 5-7).

DominatingPhase is a flag that is used to select the type of filtering: when it is TRUE, the filtering processes only *dominatedPort* clusters.

To this aim, the *srvPort* distribution is checked. If the fraction of flows with the most frequent *srvPort* in $\mathcal{I}$ exceeds the threshold $portFraction$, the cluster is a *dominatedPort* cluster. The flows involving the dominant *srvPort* are clustered together and added to the set $\mathcal{C}$ of final clusters (line 11-12), while flows not involving the dominant *srvPort* are removed and put in $\mathcal{U}$ (lines 13-14). The dominant port *dp* is included in the set $\mathcal{DP}$ of dominant ports (lines 15-16). If there is no dominant port, all flows from $\mathcal{I}$ are put in $\mathcal{U}$ (lines 17-18).

When *DominatingPhase* is FALSE, *randomPort* clusters are handled. In this case, cluster $\mathcal{I}$ (with all its flows) is simply added to the set of final clusters (line 21).

3.4.1.2 The iterative clustering procedure

The iterative clustering procedure is reported in Alg. 3 which receives as input the current batch $\mathcal{B}$ of flows. It iteratively generates dominated port clusters alternating

clustering and filtering phases. At last, it generates random port clusters. More specifically, the set of flows $\mathcal{B}$ to be clustered is processed for *itermax* iterations. At each iteration the set $\mathcal{U}$ of flows that are not yet assigned to any cluster is processed (lines 5-12). k clusters are formed using the well-known k-means algorithm that returns the set $\mathcal{C}'$ of k clusters. Each cluster in $\mathcal{C}'$ undergoes a filtering phase (lines 8-11), which is looking for *dominatedPort* clusters at this stage. The **doFiltering()** procedure returns in $\mathcal{U}$ flows that did not pass the filter and must be processed at the next iteration.
After *itermax* iterations, *randomPort* clusters are handled. In this case, the information carried by the dominant port has been already exploited in previous phases. The set $\mathcal{DP}$ of dominant ports contains the *srvPort* that appeared as dominant in the past. Intuitively, if a *srvPort* emerged as dominant port, then flows that have not been already put into *srvPort* dominated clusters should be considered outliers.
As such, we first remove from the set $\mathcal{U}$ of flows to be clustered all those flows directed to any dominating port that has been found in the previous iterations (lines 14-16). Then, the final clustering is completed (line 17-21).

3.4.2 Labeling

Once flows have been clustered, the **doLabeling($\mathcal{C}'$)** procedure (see Alg. 1 - line 15) assigns a label to each cluster. For each cluster $\mathcal{I}$ in $\mathcal{C}'$, flows are checked.
If $\mathcal{I}$ contains some *seeding flows*, i.e., flows (extracted from $\mathcal{S}$) that already have a label, a simple majority voting scheme is adopted: the seeding flow label with the largest frequency will be extended to all flows in $\mathcal{I}$, possibly over-ruling a previous label for other seeding flows. More complicated voting schemes may be adopted (e.g., by requiring that the most frequent label wins by 50% or more). However, performance evaluation shows that the homogeneity of clusters produced by the iterative clustering procedure is so high that simple schemes work very nicely in practice as shown in Section 3.7.

3.4.2.1 Bootstrapping the labeling process

If no seeding flows are present, $\mathcal{I}$ is labeled as "unknown" and passed to the system administrator that should manually label the cluster. This will clearly happen during the bootstrapping of SeLeCT, when no labeled flows are present.
To address this issue, several solutions can be envisioned. For example, labels can be manually assigned by using the domain knowledge of the system administrator, supported by all the available information on the flows in the cluster (e.g., port number, server IP addresses or even the flow payload, if available). We show how easily this can be done in Section 3.7. A second option is to use a bootstrapping flow set from some active experiments in which traffic of a targeted application is generated. Similarly, a

set of bootstrapping flows can be generated by providing labels obtained by some other available traffic classification tools, (as in [21]).
In all cases, the complexity of the labeling process is reduced to the analysis of few clusters, instead of hundred of thousands of flows. This mechanism can be also automated as suggested by [34], but this is outside the scope of this Chapter.

3.4.3 Self-seeding

Once some clusters have been labeled, SeLeCT is able to automatically reuse this information to process next batches. This is simply achieved by extracting some *seeding flows* from labeled clusters by means of the **extractSeeds($\mathcal{C}$)** procedure (see Alg. 1 - line 16).
It implements a *stratified sampling technique*, i.e., from each cluster, the number of extracted seeds is proportional to the cluster size. Stratified sampling ensures that at least one observation is picked from each of the cluster, even if probability of it being selected is far less than 1. Thus, it guarantees that in the seeding set there are representatives of each cluster and avoids the bias due to classes having much more flows than others. Let $numSeeds$ be the target number of seeding flows, i.e., $numSeeds = ||\mathcal{NS}||$.
For each labeled cluster $\mathcal{I}$, a number NS_I of labeled flows proportional to the cluster size is extracted at random. That is $NS_I = 1 + \left(\frac{||\mathcal{I}||}{numSeeds}\right)$ flows are randomly selected from each cluster $\mathcal{I}$.
This mechanism enforces a self training process that allows the system to grow the set of labeled data and thus augment the coverage of the classification process. Section 3.7 provides some evidence to support this statement.

3.5 Experimental results

3.5.1 Experimental dataset

We performed several experiments to assess the performance of SeLeCT using the datasets described in Section 3.3. All traces have been processed to generate directed flow level logs. Recall that we only consider TCP flows in this work. We use two separate advanced DPI classifiers to label flows and use these labels as our ground truth. The first one is provided by the NarusInsight[4] professional tool, and the second one is implemented in Tstat [10], the Open Source traffic monitoring developed at Politecnico di Torino. A total of 23 different protocols are identified including web (HTTP/S, RTSP,

[4] http://www.narus.com/

TLS), mail (SMTP/S, POP3/S, IMAP/S), chat (XMPP, MSN, YAHOOIM), peer-to-peer (BitTorrent, eMule, Gnutella, Fasttrack, Ares) and other protocols (SMB, FTP, Telnet, IRC).

To be conservative, we label as "unknown" those flows that do not match any of the DPI rules, or for which DPIs' labels are different. Each dataset has a different share of application labels, with a typical bias toward most popular protocols like HTTP and/or P2P that dominate the datasets; we do not report these details for the sake of brevity.

3.5.2 Performance metrics

We consider two metrics to characterize the output of the iterative clustering algorithm: *number of clusters* and *clustered flows percentage* (i.e., the ratio of flows $||\mathcal{C}'||$ clustered by **doIterativeClustering($\mathcal{B}'$)** to the total number of flows $||\mathcal{B}'||$ provided as input expressed in percentage).

In order to evaluate classification performance, we use the *confusion matrix*. The confusion matrix is a matrix in which each row represents the instances in a predicted class (i.e., the decision of SeLeCT), while each column represents the instances in an actual class (i.e., the ground truth). The name stems from the fact that it highlights cases in which the system is confusing two classes (i.e., it is mislabeling one as another). To evaluate the classification performance of SeLeCT, we use three metrics: *overall accuracy, recall,* and *precision*.

• *Accuracy* is the ratio of the sum of elements in the main diagonal (i.e., the total true positives) of the confusion matrix to the sum of all elements (i.e., the total samples). Accuracy does not distinguish among classes and is biased towards dominant classes in the dataset. For instance, consider a scenario where 90% of flows are HTTP flows. A classifier that always returns the "HTTP" label will have accuracy of 90%, despite completely missing all the other classes. Although accuracy is an important metric, it does not capture all the characteristics of the classifier.

• *Recall* for the i-th class, is the ratio of the element (i, i) (i.e., the true positives) in the confusion matrix to the sum of all elements in the i-th column (i.e., the total samples belonging to the i-th class). It measures the ability of a classifier to select instances of class i from a data set. In the same example as before, always returning "HTTP" would have a recall of 0% for all classes except for "HTTP".

• *Precision*, for the i-th class, is the ratio of the element (i, i) in the confusion matrix to the sum of all the elements in the i-th row (i.e., the true positives plus the false positives). It measures the ability of the classifier in assigning only correct samples to class i. In the example above, always returning "HTTP" would have a precision of 90% for the HTTP class and of 0% for the other classes.

In the rest of this section, we consider the following parameter settings: Batch size $||\mathcal{B}|| = 10,000$, number of flows used for seeding $numSeeds = 8,000$, $minPoints = 20$, $itermax = 3$, $portFraction = 0.5$ for $step < itermax$, and $portFraction = 0.2$ for step $itermax$. Extensive parameter sensitivity is carried over in Section 3.8. For the k-means algorithm, we set $k = 100$, number of iterations smaller than $1,000,000$ and, to avoid the initial centroid placement bias, we execute 10 independent runs and select the result with the best Sum of Squared Errors (SSE) [13].

3.5.3 Iterative clustering performance

We first evaluate the benefits of the iterative clustering procedure in SeLeCT.
We compare the accuracy against i) simple port-based classifier and ii) classic k-means as proposed in [21]. The simple port-based classifiers uses the *srvPort* to label flows. It considers well-known ports for the most common protocols, and port 4662 for eMule.
Experiments here consider, for each dataset, the first batch of 10,000 flows only. For both algorithms, labeling is performed by the **doLabeling()** procedure. The labeling process adopts a simple majority voting scheme: given a cluster, the most frequent label among seeding flows in the cluster is extracted, and used to label all flows (mimicking [21]). The assigned label is then compared to the original label that the DPI assigned to each flow.
Figure 3.2 reports results for all datasets.
Flow-wise and byte-wise accuracy are reported in top and bottom plot, respectively. The former is computed as the percentage of the correctly classified flows, while the latter is computed as the percentage of the bytes carried by correctly classified flows. Results highlight the benefit of the iterative clustering process for which the accuracy is about 97.5% on average, with a worst case of 94.2% for Dataset-3C considering flow-wise accuracy.
The simple k-means adopted in [21] results in no more than 85% flow-wise accuracy, which is in line to the findings in [21, 27]. The port-based classifier performs poorly in some scenarios where protocols not using a well-known port. This is the case for Dataset-1C where the presence of P2P traffic is predominant.
SeLeCT is the only classifier that offers excellent results for all datasets, and considering both flow-wise and byte-wise accuracy. Given the marginal differences of the two metrics, in the following we consider only flow-wise performance indexes.
An interesting observation in Figure 3.2 is that the Server datasets show better accuracy than the Client datasets. The reason is that layer-4 features carry more valuable information to differentiate between classes when considering packets sent by servers rather than by clients, e.g., the typical lengths of packets sent by HTTP and SMTP servers

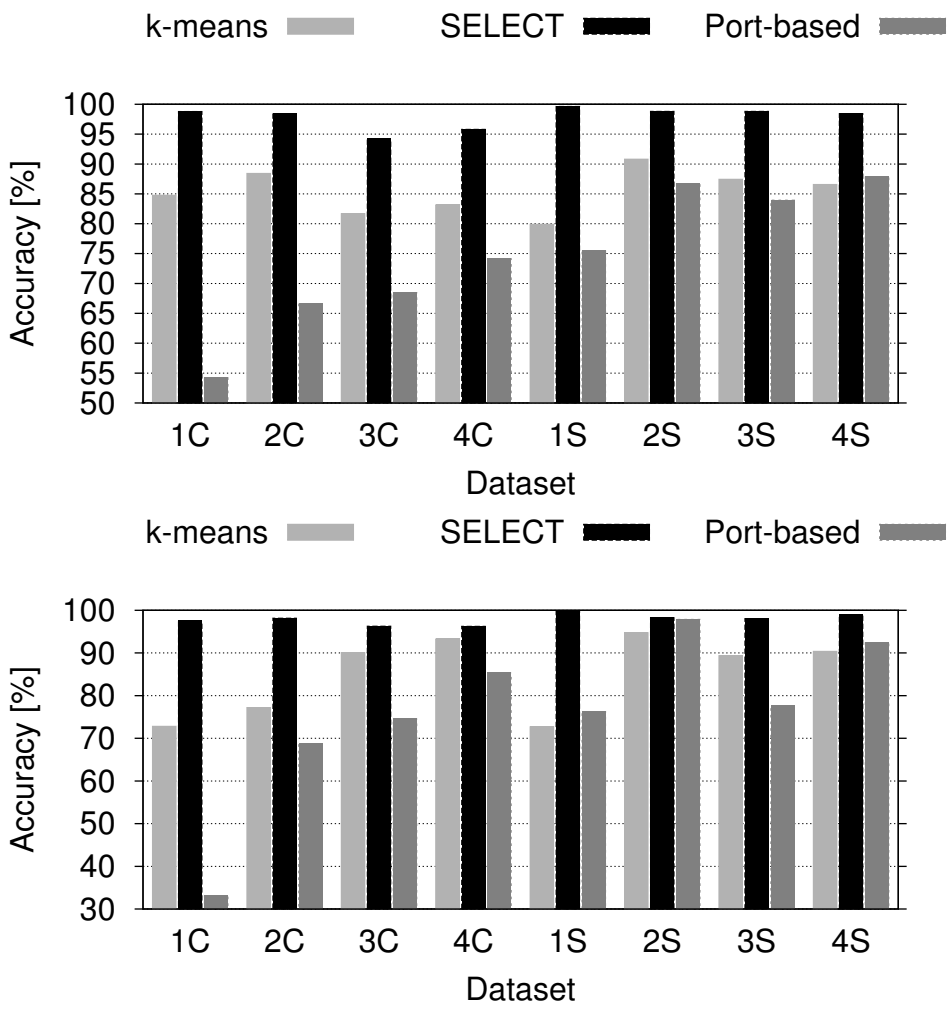


FIGURE 3.2: Accuracy of the clusters for simple port-based classifier, classic k-means and SeLeCT. Accuracy computed per flows on the top, per byte on the bottom. Results reported for all datasets.

are different, while the client queries could be more similar. The intuition is that server responses have more peculiar lengths than client queries.

Table 3.2 shows the confusion matrix for Dataset-2S, which represents the *best case* for the k-means based classifier. The bold font highlights true positives. First, notice that the HTTP, SMTP, and Unknown classes are clearly predominant, possibly causing a "capture effect" so that other classes vanish. In fact, most of the flows of other classes are misclassified as one of these three predominant classes, impairing recall and precision, even if the accuracy is still high (90% in this case - see Figure 3.2). For example, POP3S and Telnet have 0% for both recall and precision. For the HTTPS flows - which are a non negligible fraction of samples - precision is 74% and recall is as low as 54%, i.e., about half of the HTTPS flows are misclassified. Finally, the predominant class performance is impaired as well. For example, SMTP precision drops to 78% because of the high number of false positives. In summary, the standard k-means clustering exhibits poor

	BT	HTTP	HTTPS	MSN	POP3	POP3S	SMB	SMTP	SSH	Telnet	UNK	XMPP
BT	**34**	9	2	0	0	0	0	1	0	0	11	0
HTTP	18	**5829**	175	10	1	2	0	27	0	0	118	1
HTTPS	3	18	**345**	5	0	0	0	18	0	0	65	0
MSN	0	0	0	**0**	0	0	0	0	0	0	0	0
POP3	3	6	1	0	**16**	2	0	3	0	0	14	0
POP3S	0	0	0	0	0	**0**	0	0	0	0	0	0
SMB	0	0	0	0	0	0	**0**	0	0	0	0	0
SMTP	21	18	85	14	45	53	18	**2247**	0	43	276	5
SSH	0	0	0	0	0	0	0	0	**0**	0	0	0
Telnet	0	0	0	0	0	0	0	0	0	**0**	0	0
UNK	21	35	35	6	0	1	0	29	9	0	**214**	0
XMPP	0	0	0	0	0	0	0	0	0	0	0	**0**

TABLE 3.2: Confusion matrix of a classifier based on the simple k-means for Dataset-2S. Columns give the ground truth.

	BT	HTTP	HTTPS	MSN	POP3	POP3S	SMB	SMTP	SSH	Telnet	UNK	XMPP
BT	**3**	0	0	0	0	0	0	0	0	0	3	0
HTTP	0	**5769**	0	0	0	0	0	0	0	0	30	0
HTTPS	0	0	**530**	0	0	0	0	0	0	0	0	0
MSN	0	0	0	**7**	0	0	0	0	0	0	1	0
POP3	0	0	0	0	**42**	0	0	0	0	0	0	0
POP3S	0	0	0	0	0	**46**	0	0	0	0	0	0
SMB	0	0	0	0	0	0	**8**	0	0	0	0	0
SMTP	0	0	0	0	0	0	0	**2217**	0	0	102	0
SSH	0	0	0	0	0	0	0	0	**9**	0	0	0
Telnet	0	0	0	0	0	0	0	0	0	**43**	0	0
UNK	4	0	0	2	0	0	0	0	0	0	**83**	0
XMPP	0	0	0	0	0	0	0	0	0	0	0	**5**

TABLE 3.3: Confusion matrix of the SeLeCT classifier for Dataset-2S. Columns give the ground truth.

performance for not dominant classes.

SeLeCT significantly boosts performance as depicted in Table 3.3[5]. The overall accuracy tops to 98.82% and the confusion matrix exhibits almost perfect results. Interestingly, only flows in the Unknown class have been (possibly) misclassified.

For example, 102 flows that the DPI labeled as Unknown are instead labeled as SMTP by SeLeCT. We manually cross-checked these flows, and found that 97 out of 102 flows are indeed SMTP flows which the DPI was not able to correctly classify because the SMTP banner sent by the server was not the usual one, and its pattern was not included

[5]Totals are different than in Table 3.2 since SeLeCT adopts a conservative approach by deferring the clustering of "noise" flows to next batches.

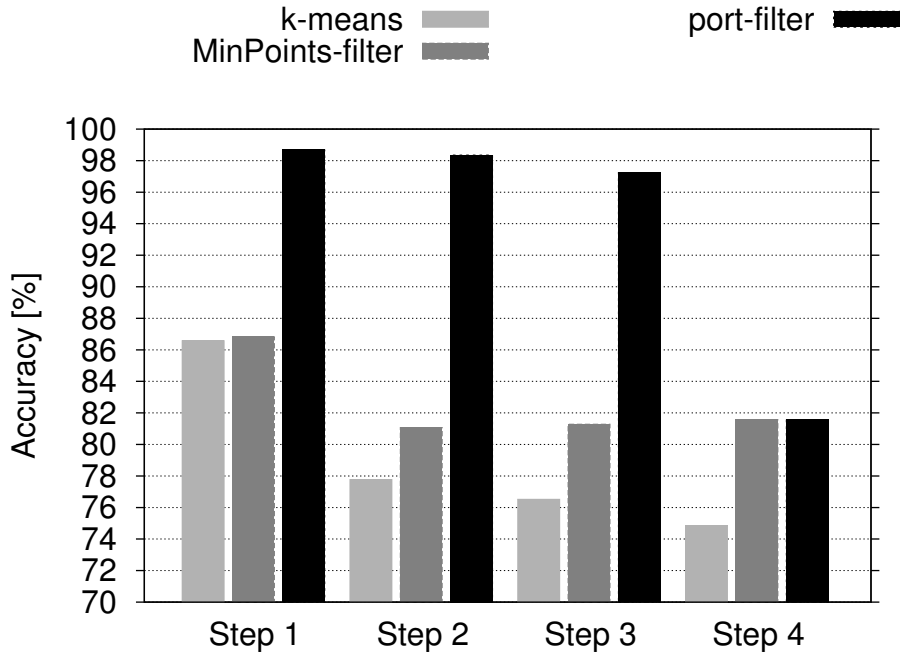


FIGURE 3.3: Accuracy before and after the different filtering steps for Dataset-4S.

in the DPI engine signature set. Double checking unknown flows that SeLeCT classified as HTTP, we also verified that the DPI was fooled by some HTTP messages which included non-English text (recall this dataset was collected from an ISP in the far east). This shows that SeLeCT is able to automatically adapt classes to small variations of features.

SeLeCT is more robust than the DPI-based classifier because layer-4 features are less sensitive to small feature changes than the DPI pattern matching rules. The latter can be fooled by a simple character change.

Figure 3.3 gives more insights about the benefits of the filtering steps in the iterative clustering process. It reports the overall accuracy after (*i*) running the k-means only (line 6 of Alg. 3), (*ii*) after all clusters with less than *minPoints* samples have been discarded (lines 4-6 of Alg. 2), and (*iii*) after the final port based filtering is performed (lines 7-18 of Alg. 2). Accuracy is evaluated at each of the four steps independently of the others, i.e., the results are not cumulative. The first 10,000 flows in the first batch of the Dataset-4S trace are considered. In this case, flows are labelled by the original DPI label; flows in a cluster are then re-assigned the majority label, and the original and the new label are then compared. . Results show that discarding clusters with less than *minPoints* provides small improvements, while the port-based filtering is the key to boost accuracy to 98% when *dominatedPort* clusters are selected. Only at the last step, when *randomPort* clusters are considered and the port-based filtering is disabled, accuracy lowers to 82%. In this case, discarding the clusters smaller than *minPoints* helps improving recall and precision for all classes (see Table 3.3). This last

step is important since it allows to properly look for Peer-to-Peer (P2P) protocols that typically do not run on standard server ports.
These results show the benefits of the iterative clustering approach. In particular, they highlight the befits of the filtering mechanisms that allows exploiting the information carried by the *srvPort*, which was not leveraged by previous clustering approaches.

3.6 Interesting findings enabled by SeLeCT

One of the interesting possibilities offered by SeLeCT is its ability to automatically group flows in homogeneous clusters. It is thus interesting to verify if the clusters offer more fine-grained classification than traditional protocol classification. We first investigate *dominatedPort* clusters whose DPI inherited label is "Unknown" for all datasets. We found:
• $srvPort = 5223$ - the **Apple push notification server over TLS** is identified in Dataset-3 and Dataset-4;
• $srvPort = 5152$ - **Backdoor.Laphex.Client** traffic is identified in Dataset-1;
• $srvPort = 12350$ - the **Skype proprietary authentication** protocol is identified in Dataset-3 and Dataset-4;
SeLeCT automatically unveils clusters of traffic generated by services that appear as real unknown to the network administrator. This is the case of the *Apple Push Notification* system for iOS devices and iCloud enabled devices, which is based on the SSL/TLS protocol, but running on a non standard $srvPort = 5223$. All flows in this cluster are labeled by the DPI as SSL/TLS protocol. To find the correct label, a *whois* lookup for the $srvIP$ addresses reveals that the servers are all registered to Apple Inc. By running an active experiment, it is possible to confirm that all flows in this cluster are related to Apple Push Notification and iCloud services.
A second cluster of unknown flows aggregates traffic generated by the malware *Backdoor.Laphex.Client Bot/Trojan*. Manual inspection of flows payload confirms this assumption. Similarly the cluster of flows directed to $srvPort = 12350$ turns out to unveil *Skype Authentication* protocol traffic. Also in this case, the $srvIP$ of all flows reveals strong clues about the application. All flows are directed to $srvIP$ in the subnet 213.146.189.0/24, registered to Skype Inc.
We then analyze clusters labeled as HTTP traffic. There are several tens of them in each dataset, and some share some clear threat. As proposed in [35], the $srvIP$ feature reveals interesting information. For instance, $srcIP$ addresses in some clusters clearly belong to the same subnet. By means of a simple *whois* query, it is possible to identify clusters containing only Google, Dailymotion or Amazon services, respectively. Similarly, a POP3S cluster refers to *mail.google.com* servers scattered in 4 different

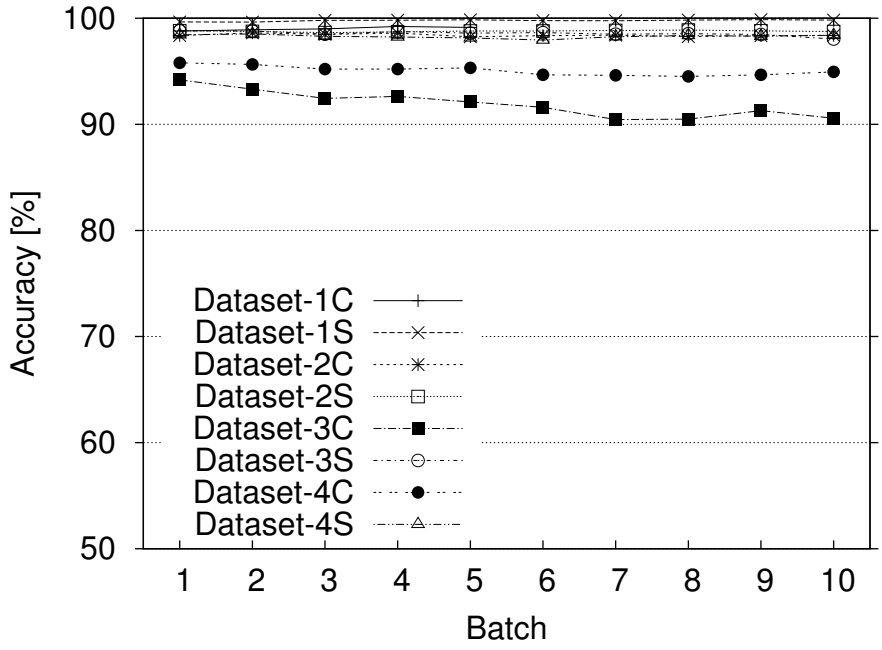


FIGURE 3.4: Accuracy over different batches.

subnets in Dataset-4, while a second POP3S cluster aggregates together all flows of other mail providers.

These examples confirm the ability of SeLeCT to automatically reveal new classes of traffic that would be hard to highlight by means of any supervised technique. Once SeLeCT is augmented with this knowledge by injecting these labels, flows are correctly classified in all subsequent batches thanks to the seeding mechanism.

Overall, we were able to find labels for about 90% of unknown clusters. The remaining 10% of clusters contains flows that appear to be encrypted, and for which the IP addresses refer to end-user addresses assigned by ISPs to modems. We suspect those could be Skype flows, but we are not able to confirm this assumption.

3.7 Exploring the seeding process

So far we have analyzed the performance of SeLeCT considering a single batch provided as input. We are interested now in analyzing the performance of the seeding process. To accomplish this, we run SeLeCT on ten successive batches of flows. As previously done, the bootstrapping at batch 1 is done using the DPI labels. Then, for the subsequent batches, **extractSeeds()** is used to seed the labeling process from batch n to batch $n+1$. Each batch performance is evaluated by comparing the DPI labels in the ground truth with the labels provided by SeLeCT.

	BT	eMule	HTTP	HTTPS	IMAPS	POP3	POP3S	UNK
BT	**157**	105	0	0	0	0	0	8
eMule	122	**3556**	0	0	0	0	4	24
HTTP	0	0	**10815**	0	0	0	0	5
HTTPS	0	0	1	**1291**	0	0	0	14
IMAPS	0	0	0	0	**53**	0	0	0
POP3	0	0	0	0	0	**145**	0	3
POP3S	0	0	0	0	0	0	**25**	0
UNKNOWN	0	0	18	0	0	0	0	**196**

TABLE 3.4: Confusion matrix at batch 10 for Dataset-3C.

3.7.1 Self-seeding

Figure 3.4 shows the results for all datasets. First, notice that the accuracy of SeLeCT is extremely high and stable over time for all server datasets. As we already mentioned before, this is due to the better representativeness of the layer-4 features for server flows. Other metrics (i.e., the number of clusters and the percentage of clustered flows) remain unchanged over different batches and hence we do not report these results.

For client Dataset-3C and Dataset-4C, the accuracy slightly decreases over time. For instance, in Dataset-3C it decreases to about 90% during the first 7 batches, then it stabilizes. Investigating further, we notice that both recall and precision of SeLeCT are higher than 98% for all classes of traffic except for BitTorrent and eMule protocols which tend to be confused with each other. This is detailed by the confusion matrix of the 10-th batch in Table 3.4. Note that the total number of flows exceeds the batch size, since at step 10 SeLeCT processes also seeding flows. The relative higher fraction of P2P traffic in the Dataset-3C (collected at 2am) results in a global decrease in the overall accuracy. Similar considerations hold for the Dataset-4C which refers to peak time. However, in this case the fraction of P2P flows is smaller than during the night and thus it has less impact on the overall accuracy. An important and desirable property is that confusion actually happens among P2P protocols only. The lack of dominating port for P2P protocols makes it more challenging for SeLeCT to clearly distinguish the traffic.

Based on the results of our experiments, we believe that SeLeCT shows very good performance in terms of accuracy, precision, and recall. For most protocols, SeLeCT correctly classifies flows for which labels have been provided with no confusion.

3.7.2 Bootstrapping

As we noted before, SeLeCT requires manual intervention to provide labels to clusters. When a label for a few flows is introduced, SeLeCT will carry on these labels for future

%scriptsize

SrvPort	25	80	**88**	110	443	995	**1935**	**4662**	**5223**	**12350**
# cluster	1	46	1	3	30	2	1	51	1	1
Label	SMTP	HTTP	HTTP	POP3	HTTPS	POP3S	RTMP	eMule	Apple	Skype

TABLE 3.5: *dominatedPort* clusters at batch 1. Bold font highlights clusters on non-standard ports.

classification. In the previous experiments we used the labels provided by a DPI to bootstrap the classification and seeding process. We now investigate how difficult it can be to manually bootstrap the system. We assume that a network operator is offered clusters of flows, and s/he has to use her/his domain knowledge to provide labels.
We consider the Dataset-4S trace and ignore all the DPI labels. In other words, no labels are provided to SeLeCT. At the end of the first batch, the operator has to analyze the clusters that have been formed to label them.

3.7.2.1 *dominatedPort* Clusters

To assign a label, the information provided by the *srvPort* for *dominatedPort* clusters proves to be very valuable. Table 3.5 reports the *srvPort* and the number of corresponding *dominatedPort* clusters on the first and second row, respectively, while the third row reports the class label that we assigned. Overall, protocols running on well-known ports are straightforward to identify. Notice that SeLeCT can identify several clusters that refer to the same protocol (e.g., 46 clusters of HTTP flows). In general, the number of clusters is proportional to i) the number of flows, and ii) the variability of the services offered on a given protocol.
It is interesting that SeLeCT naturally created some clusters whose protocol was not known to the DPI. These clusters are highlighted using bold fonts. By simply searching the web, protocols are easily identified: Port 1935 is used by the Macromedia flash server to stream videos using the RTMP protocol; port 4662 is the default eMule port. At last, port 5223 is used by Apple push notification service for iOS devices running over TLS, and port 12350 cluster contains flows going to Skype Inc. managed servers (see above). Following this approach, 136 clusters can be immediately labeled. Only one cluster dominated by $srvPort = 88$ remains ambiguous. Looking at the closest cluster, it reveals that flows in this cluster are very likely to be HTTP flows, since the 6 closest clusters are HTTP clusters. A simple packet inspection on some flows confirms this hypothesis. This process can be possibly automated in the future.
Once SeLeCT is augmented with the knowledge of these labels, flows are correctly classified in all subsequent batches thanks to the seeding mechanism. From Figure 3.7, we can see that more than 80% of flows are typically clustered in *dominatedPort* clusters

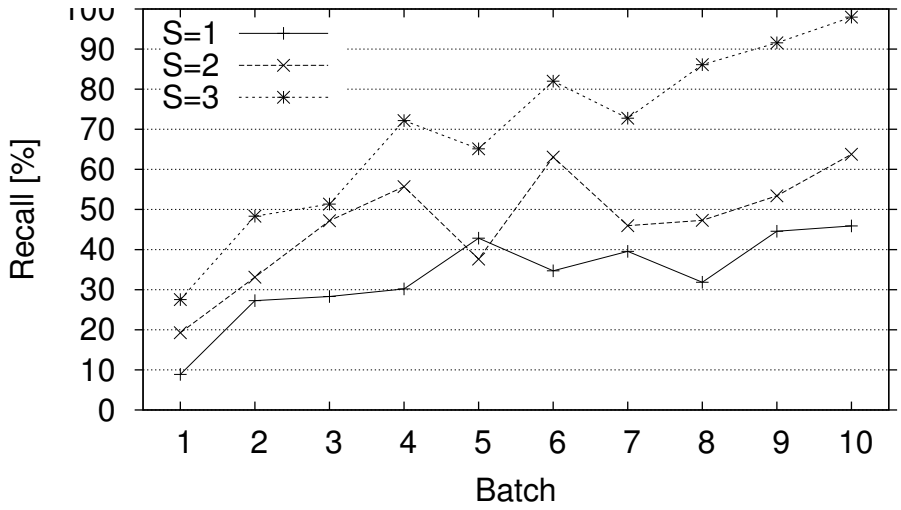


FIGURE 3.5: eMule recall when only S labeled clusters are used as bootstrap at batch 1 for Dataset-4S.

at the end of step 3. In other words, more than 80% of flows can be easily labeled using simple information obtained from the dominating *srvPort*, whose accuracy is close to 100% (refer to Figure 3.3).

3.7.2.2 *randomPort* clusters

At the last iteration, SeLeCT disables the port filters in **doClustering()** and the remaining 10-20% of flows are clustered in *randomPort* clusters. The analysis of those clusters is expected to be more complicated since the *srvPort* information is, by construction, providing limited information. First of all, it is easy to see whether a cluster is grouping some P2P protocol or traditional client-server protocols by looking at the $srcIP, dstIP$ of flows, as proposed in [15, 36].

Interestingly, *srvPort* analysis still provides vital clues about the protocol when analyzing the port number frequency distribution by considering all flows in a cluster together. For instance, consider a P2P protocol in which the user can manually change the port used by the application. It is very likely that the port the user would choose is "similar" to the default number offered by the application, therefore biasing the port frequency distribution. Consider a cluster in which the topmost ports are 4664, 4661, 8499, 7662, 6662, 5662, 4663, 64722, ... The intuition suggests to label flows in that cluster as eMule whose default port is 4662 (which turns out to be the correct label). On the contrary, clusters in which port numbers are uniformly distributed clearly suggest that the application itself is enforcing a random port selection, as done, e.g., by most popular BitTorrent applications.

At last, packet inspection can been considered as another option to label *randomPort* clusters. Unlike traditional per-flow analysis, the inspection of clustered flows simplifies the identification of signatures since a set of flows is exposed and can be analyzed in

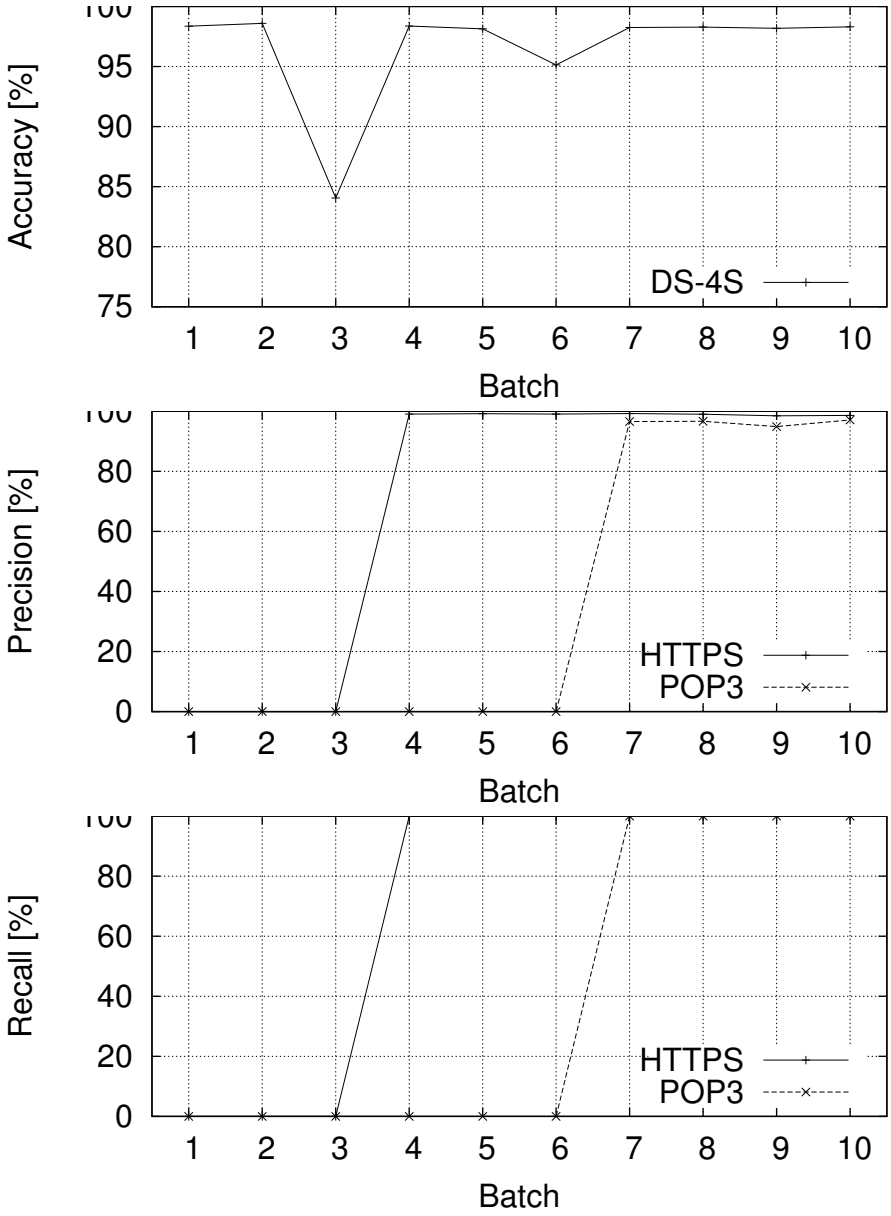


FIGURE 3.6: New protocols suddenly appear: HTTPS traffic is added at batch 3, and POP3 traffic is added at batch 6 in Dataset-4S.

parallel to identify common headers. Once a label has been found, SeLeCT extend it to all the flows in the same cluster.

3.7.3 Seeding evolution

To show the ability of SeLeCT to increase its knowledge over time, we perform the following experiment. Consider Dataset-4S and focus on the eMule flows not having the default 4662 *srvPort* (which are clustered as *dominatedPorts* clusters). At the end of batch 1 processing, only the largest S *randomPort* clusters are manually labeled as eMule (e.g., by checking the port number distribution as above). Labeled flows are

then used to bootstrap the seeding process. Figure 3.5 reports the recall evolution over the different batches for different values of S. For $S = 3$, corresponding to only 28% flows selected as bootstrap at the end of batch 1, SeLeCT already achieves 98% of recall at batch 10. Worst case precision is 98.6%. These results show that SeLeCT seeding process is successfully bootstrapped even if only $S = 1$ cluster is used as initial seed.

We now perform another experiment in which we simulate the sudden appearance of a new class of traffic. We consider the Dataset-4S trace, from which we removed all POP3 and HTTPS flows. Then, during the third and sixth batch, HTTPS and POP3 traffic is injected to simulate the sudden birth of new protocols. We run SeLeCT over all 10 batches. Results are reported in Fig. 3.6. The top plot reports the overall accuracy, while middle and bottom plots report precision and recall, respectively. Notice how SeLeCT rapidly detects the presence of new traffic classes. In particular, at batch 3, accuracy severely drops since HTTPS flows are labeled as "Unknown". We then bootstrap the HTTPS seeding as before, i.e., by labeling the largest Unknown traffic cluster as HTTPS. Bootstrapping in this case is much faster then for eMule thanks to the purity of HTTPS clusters. Indeed, at batch 4, accuracy returns to 97.5%, and HTTPS precision and recall approach 100%.

At batch 6, the same transient is observed when POP3 flows are injected. Being their number small, the impairment on accuracy is less evident. Then, from batch 7 on, the bootstrapping of the POP3 protocol is completed so that accuracy, recall and precision get back to excellent values.

These examples show that SeLeCT allows an easy identification of protocols that, in our example, were not detected by the DPI because no signature was present. This enhances the operator's network visibility by providing homogeneous clusters of flows whose analysis is much easier, due to the aggregated information provided by the flows in the cluster.

3.8 Parameter sensitivity analysis

In this section we present an extended set of experiments to evaluate the impact of the parameter choices on SeLeCT. In general, SeLeCT is very robust to various parameter settings and its behavior is stable in different scenarios. In this section, we report some of the most interesting findings.

3.8.1 Setting filtering parameters

Figure 3.7 reports the percentage of clustered flows during different iterations of the iterative clustering. Only Server datasets are considered for the sake of simplicity. As we can see, SeLeCT clusters most of the flows during step 1, when there are many *dominatedPort* clusters (i.e., clusters in which most of the flows involve the same port).

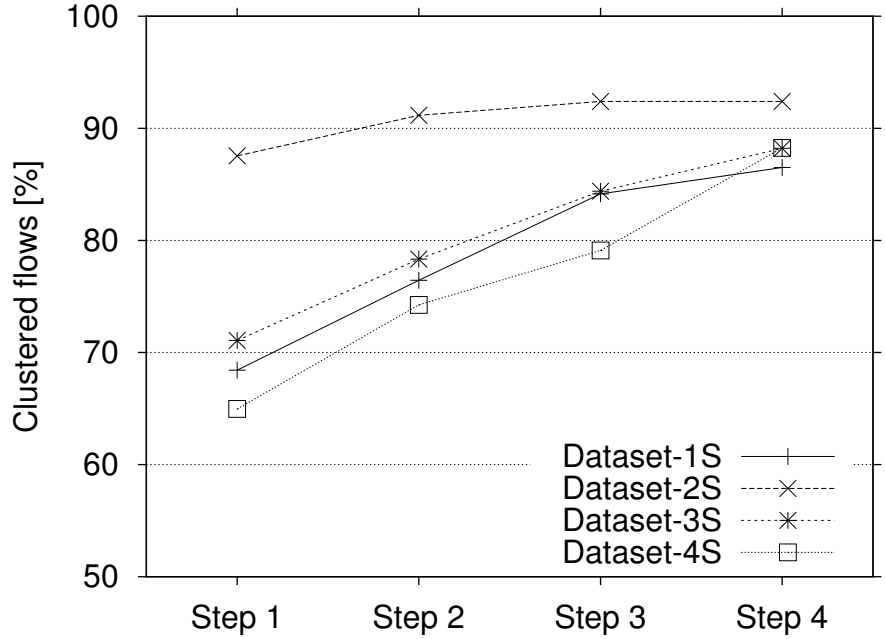


FIGURE 3.7: Fraction of clustered flows at each step.

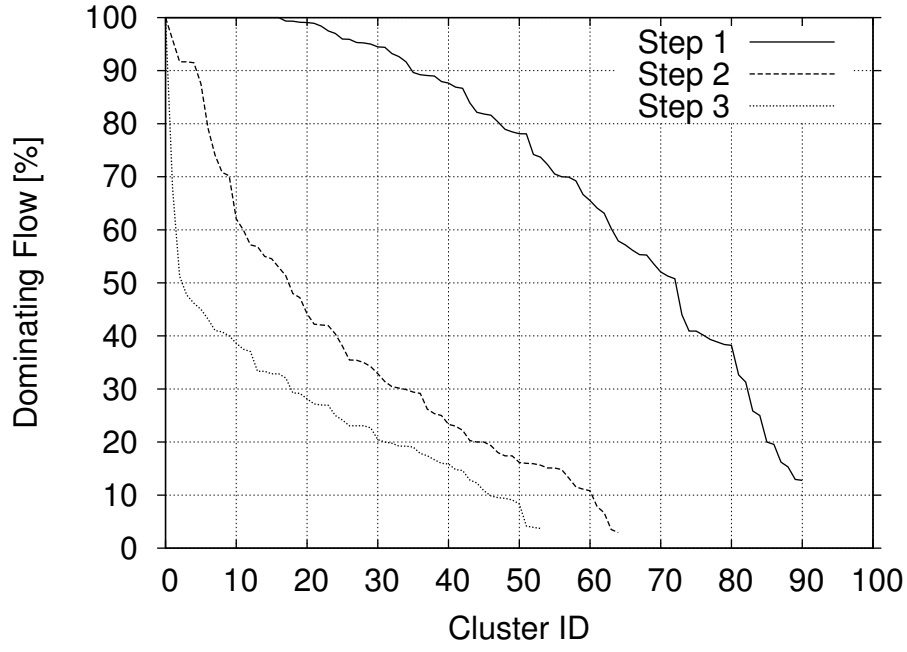


FIGURE 3.8: Fraction of flows directed to the dominating *srvPort* in each cluster for different steps for Dataset-4S.

Small clusters and outlier flows are discarded and passed to step 2. At this point, an additional fraction of *dominatedPort* clusters are identified, allowing to add about 10-15% more flows. This filtering is repeated one more time at step 3 when another 5-10% of flows is clustered. As a last step, SeLeCT looks for *randomPort* clusters and an additional fraction of flows gets properly clustered (e.g., P2P protocols). As the curves

suggest, the benefit of adding more *dominatedPort* filtering phases is limited, and little improvement is achieved by setting *itermax* larger than 3.

To confirm this intuition, Figure 3.8 reports, for each step, the fraction of flows directed to the dominating port in each cluster with more than *minPoints* flows. Clusters are sorted in decreasing fraction for ease of visualization. The number of *dominatedPort* clusters is large during step 1, with 70 clusters having more than 50% of flows that are directed to the same *srvPort*. Given $portFraction = 0.5$, SeLeCT picks flows in these clusters. In step 2, the number of *dominatedPort* clusters decreases, and only 17 clusters pass the $portFraction = 0.5$ filter. In step 3, very few *dominatedPort* clusters are present. This confirms the intuition that it is useless to add more than 3 steps because the information carried by the *srvPort* has already been exploited. In addition, the intuition suggests to relax the $portFraction$ threshold during the last step, thus we set $portFraction = 0.2$.

3.8.2 Sensitivity to $portFraction$

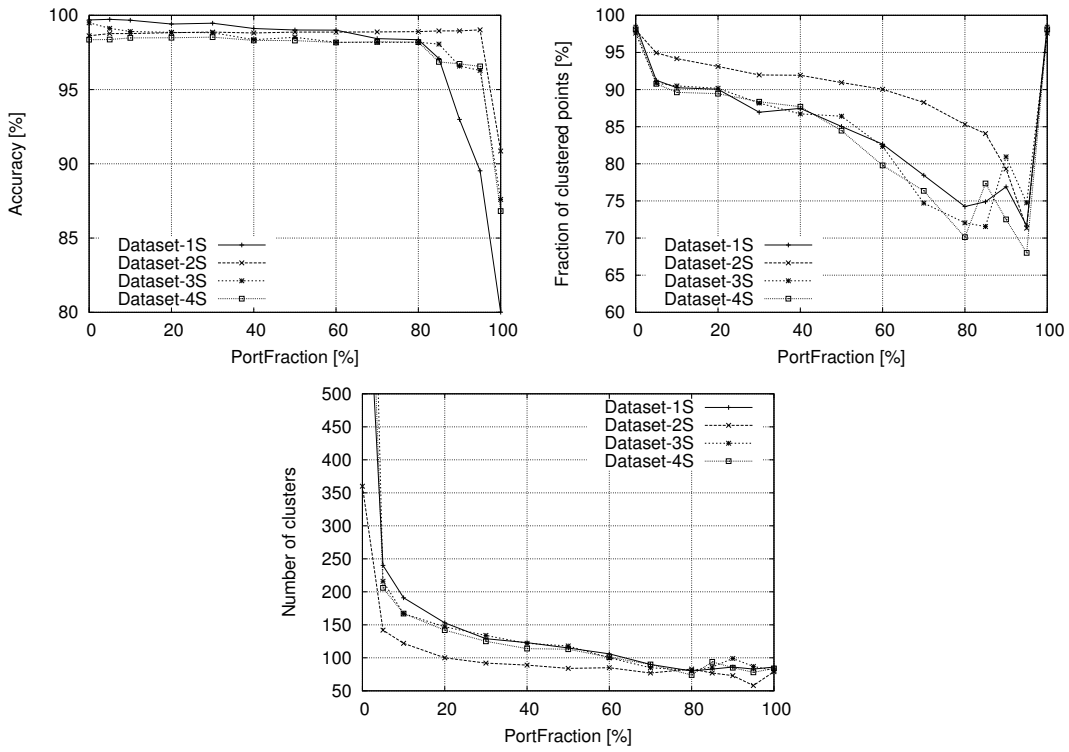


FIGURE 3.9: Sensitivity analysis to *portFraction*: accuracy, fraction of clustered flows and number of clusters in left, middle and right plot.

To complete the sensitivity analysis, Fig. 3.9 shows how the choice of *portFraction* impacts performance. More specifically, the left plot, which reports the overall accuracy, shows that the impact on accuracy is limited, and only values larger than 80% exhibit some severe degradation on accuracy (note the y-range). The middle plot, which shows

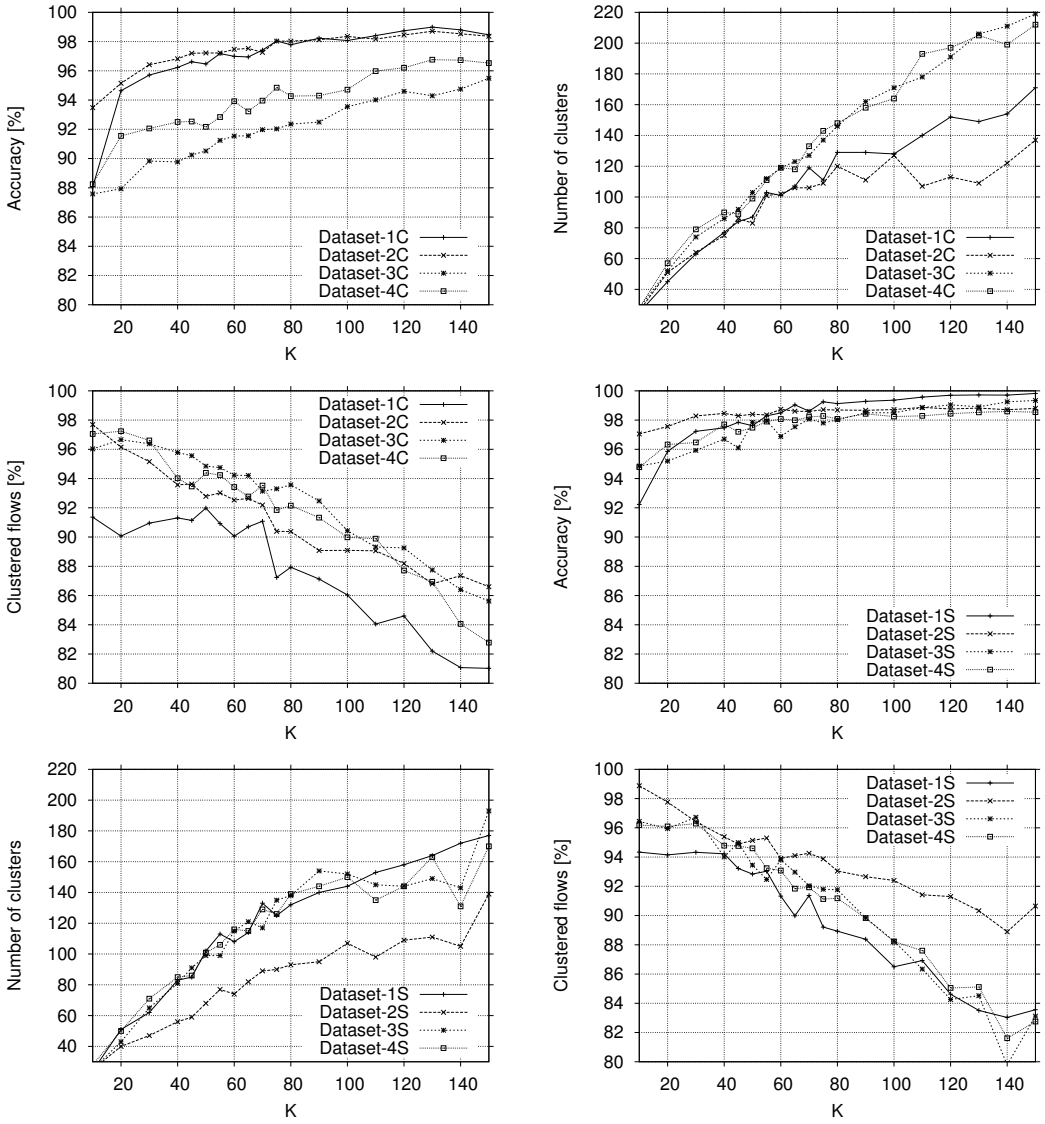


FIGURE 3.10: Sensitivity to k.

the fraction of clustered points, suggests to select smaller values for $portFraction$, since this results in a larger fraction of clustered flows. However, a trade-off is shown in the right plot, because the number of clusters notably increases for small values of $portFraction$. Small values cause the algorithm to accept a lot of clusters in the first filtering steps (refer to Fig. 3.8), causing the total number of clusters to increase rapidly. Values of $0.3 < portFraction < 0.8$ offer a good trade-off.

3.8.3 Sensitivity to k and $minPoints$

Finally, we show the sensitivity of k and $minPoints$ in Figures 3.10 and 3.11, respectively. Plots report the overall accuracy, number of clusters, and the fraction of clustered flows from left to right, the Client and Server flows on the top and bottom plots, respectively. Figure 3.10 shows that accuracy is typically higher than 90% except for very

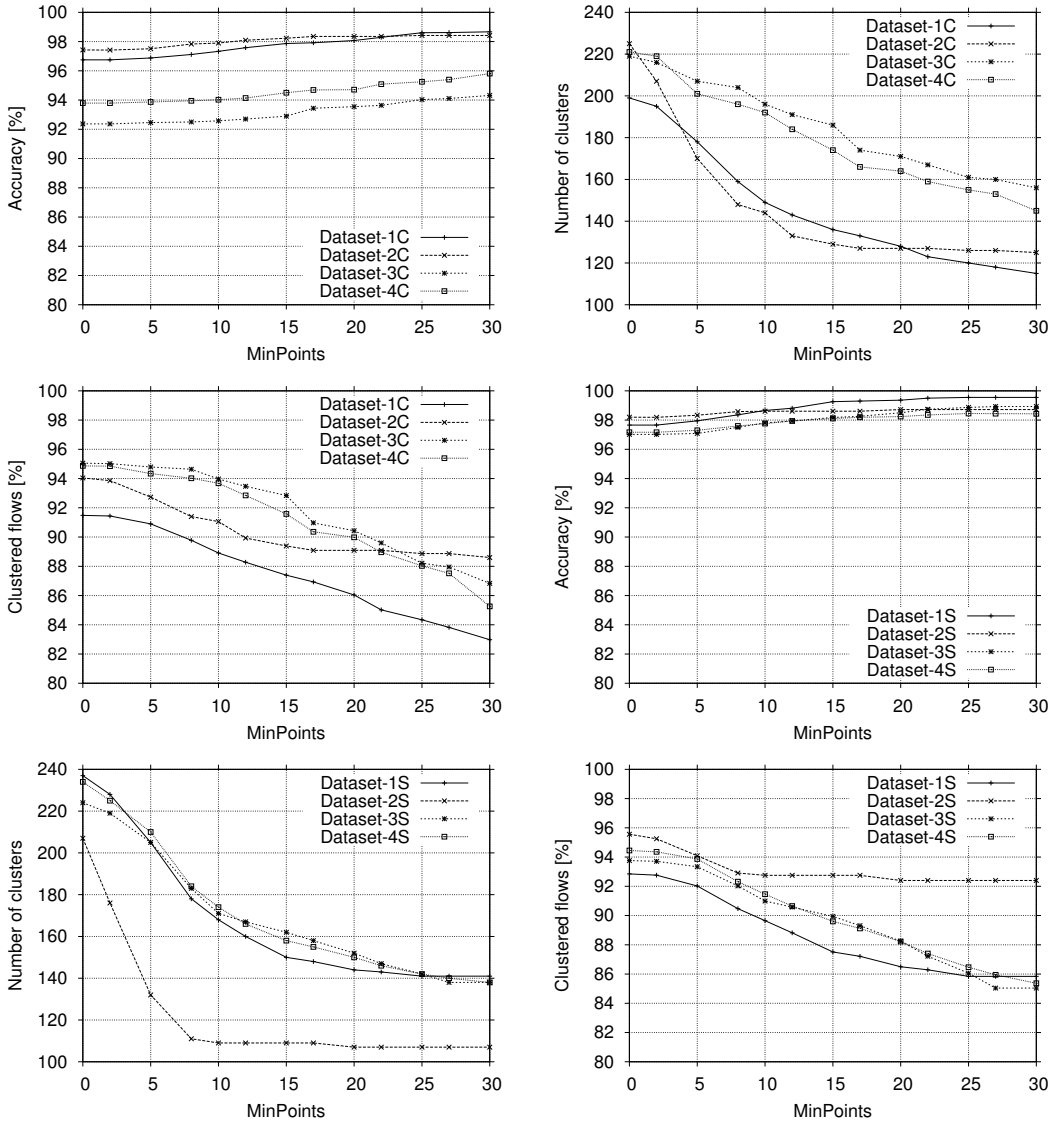


FIGURE 3.11: Sensitivity to $MinPoints$.

small values of k. Larger values of k improve accuracy, since SeLeCT is allowed to form more clusters. This is confirmed by the total number of clusters which increases almost linearly with k up to a saturation point. However, fragmenting flows into many clusters causes cluster size to be small. Hence, the parameter setting, $minPoints = 20$, filters a larger fraction of flows, causing the percentage of clustered flows to decrease. Finally, notice that Dataset-3C and Dataset-4C are the two most critical scenarios due to the mix of protocols that is present in this network and the relatively weaker descriptiveness of the layer-4 features for client flows.

A similar reasoning applies when varying $minPoints$. It has limited impact on the overall accuracy as already noticed in Figure 3.3, while the number of clusters and the fraction of clustered flows exhibit an inverse dependence on $minPoints$: small values cause both of these metrics to grow quickly, while $minPoints$ higher than 15-20 starts

showing a saturation. This is true especially for the Server datasets.
Overall, the choice of k and $minPoints$ is not critical; choosing $k = 100$ and $minPoints = 20$ allows a good trade-off between high accuracy, limited number of clusters, and large fraction of clustered flows.

3.8.4 Complexity

The complexity of SeLeCT is mainly driven by the complexity of the k-means algorithm. To find the optimal solution considering n objects, k clusters, and a d dimensional space, the problem can be optimally solved in $O(n^{dk+1} \log n)$, which would turn out to be definitively too much for real time applications. However, by considering the centroids computation and re-clustering steps for a fixed number of iterations, the computational time is deterministic. In our case, we choose the number of iteration to be smaller than 1,000,000, and we repeat the k-means 10 times to avoid possible bias do to bad initial centroid choice. Considering these settings, for Dataset-4S, the scenario with the highest flow arrival rate, SeLeCT was able to complete the processing of batch n before the collection of flows of batch $n + 1$ was complete, thus enabling real-time operation even if the current prototype is not optimised. Notice that only flows that have at least 6 data packets are passed to SeLeCT, i.e., 70-90% of flows are actually not considered in practice, see Fig. 3.1. As a final note, several functions of SeLeCT can also be run in parallel.

Chapter 4

Analysis of Twitter Data Using a Multiple-Level Clustering Strategy

In recent years, social networks and online communities have become a powerful source of knowledge. Social network users are used to publish and continuously update multimedia resources, posts, blogs, etc. Actions undertaken by Web users reflect their habits, personal interests, and professional skills. Hence, the analysis of the user-generated content coming from social networks has received an increasingly high attention in several application contexts. For instance, data mining techniques have already been applied to recommend personalized services and products based on social annotations [37], [38], [39], organize and make social knowledge accessible [40], and perform email spamming based on social networks [41]. In particular, data mining from UGC published on the popular Twitter micro-blogging Website has achieved promising results in the analysis of most notable user behaviors [42], [43] and topic trends [44].

Twitter textual data (i.e., tweets) can be analysed to discover user thoughts associated with specific events, as well as aspects characterizing events according to user perception. Clustering techniques can provide a coherent summary of tweets, which can be used to provide summary insight into the overall content of the underlying corpus. Nevertheless clustering is a widely studied data mining problem in the text domain, clustering twitter messages imposes new challenges due to their inherent sparsness.

This Chapter proposes a data analysis framework to discover, in a data collection with a variable distribution, cohesive and well-separated groups of tweets. Our framework exploits a multiple-level clustering strategy that iteratively focuses on disjoint dataset portions and locally identifies clusters. The density-based DBSCAN algorithm [45] has been adopted because it allows the identification of arbitrarily shaped clusters, is less susceptible to noise and outliers, and does not require the specification of the number

of expected clusters in the data. To highlight the relevance of specific words for a given tweet or set of tweets, they have been represented in the Vector Space Model (VSM) [46] using the TF-IDF weighting score [46]. The cluster content has been compactly represented with the most representative words appearing in their tweets based on the TF-IDF weight. Association rules representing word correlations are also discovered to point out in a compact form the information characterizing each cluster. To our knowledge, this work is the first study addressing a jointly exploitation of a multiple-level clustering strategy with association rules for tweet analysis.
As a reference case study, the proposed framework has been applied to two real datasets retrieved from Twitter. The results showed that, starting from a tweet collection, the framework allows the identification of clusters containing similar messages posted on an event. The multiple-level strategy iterated for three levels compute clusters that progressively contain longer tweets describing the event through a more varied vocabulary, talking about some specific aspects of the event, or reporting user emotions associated with the event.

4.1 Motivating example

Tweets are short, user-generated, textual messages of at most 140 characters long and publicly visible by default. For each tweet a list of additional features (e.g., GPS coordinates, timestamp) on the context in which tweets have been posted is also available.
This Chapter focuses on the analysis of the textual part of Twitter data (i.e., on tweets) to provide summary insight into some specific aspects of an event or discover user thoughts associated with specific events. Clustering techniques are used to identify groups of similar tweets. Cluster analysis partitions objects into groups (clusters) so that objects within the same group are more similar to each other than those objects assigned to different groups [13]. Each cluster is then compactly described through the most representative words occurring in their tweets and the association rules modeling correlations among these words. Association rules [47] identify collections of itemsets (i.e., sets of words in the tweet analysis) that are statistically related in the underlying dataset. Association rules are usually represented in the form $X \rightarrow Y$, where X and Y are disjoint itemsets (i.e., disjoint conjunctions of words).
A simplified example of the textual part of two Twitter messages is shown in Figure 4.1. Both tweets regard the Paralympic Games that took place in London in year 2012. As described in Section 4.3.1, to suit the textual data to the subsequent data mining steps, tweets are preprocessed in the framework by removing links, stopwords, no-ascii chars, mentions, and replies.
Our proposed framework assigns the two example tweets to two different clusters, due to their quite unlike textual data. Both example tweets contain words as

$\{paralympics, olympic, stadium\}$, overall describing the paralympics event. In addition, $\{fireworks, closingceremony\}$ and
$\{amazing, athletics\}$ are the representative word sets for Tweets 1 and 2, respectively, reporting the specific subject of each message. The association rules
$\{closingceremony \rightarrow fireworks\}$ and
$\{amazing \rightarrow athletics\}$ model correlations among representative words in the two tweets. They allow us to point out in a compact form the representative information characterizing the two messages. While the first tweet talks about a specific event in the closing ceremony (i.e., the fireworks), the second one reports a positive opinion of people attending the event.

```
TWEET 1 - text: {Fireworks on! paralympics closingceremony at Olympic Stadium}

TWEET 2 - text: {go to Olympic Stadium for amazing athletics at Paralympics}
```

FIGURE 4.1: Two simplified example tweets

4.2 Related work

The application of data mining techniques to discover relevant knowledge from the User Generated Content (UGC) of online communities and social networks has become an appealing research topic. Many research efforts have been devoted to improving the understanding of online resources [42, 48], designing and building query engines that fruitfully exploit semantics in social networks [49, 50], and identifying the emergent topics [43, 51]. Research activity has been carried out to on Twitter data to discover hidden co-occurrences [42] and associations among Twitter UGC [44, 52, 53], and analyse Twitter UGC using clustering algorithms [54–56].
Specifically, in [42] frequently co-occurring user-generated tags are extracted to discover social interests for users, while in [53] association rules are exploited to visualize relevant topics within a textual document collection. [44] discovers trend patterns in Twitter data to identify users who contribute towards the discussions on specific trends. The approach proposed in [52], instead, exploits generalized association rules for topic trend analysis. A parallel effort has been devoted to studying the emergent topics from Twitter UGC [43, 51]. For example, in [43] bursty keywords (i.e., keywords that unexpectedly increase the appearance rate) are firstly identified. Then, they are clustered based on their co-occurrences.
Research works also addressed the Twitter data analysis using clustering techniques. [54] proposed to overcome the short-length tweet messages with an extended feature vector along with a semi-supervised clustering technique. The wikipedia search has

been exploited to expand the feature set, while the bisecting k-Means has been used to analyze the training set. In [55], the Core-Topic-based Clustering (CTC) method has been proposed to extract topics and cluster tweets. Community detection in social networks using density-based clustering has been addressed in [56] using the density-based OPTICS clustering algorithm.
Unlike the above cited papers, our work jointly exploits a multiple-level clustering technique and association rules mining to compactly point out, in tweet collections with a variable distribution, the information posted on an event.

4.3 The Proposed Multiple-Level Clustering Framework

The proposed framework to analyse Twitter data is shown in Figure 4.2 and detailed in the following subsections.
The textual content of Twitter posts (i.e., the tweets) is retrieved through the Twitter Stream APIs (Application Programming Interfaces) and preprocessed to make it suitable for the subsequent mining steps. The multiple-level clustering approach is then applied to discover, in a dataset with a variable distribution, groups of tweets with a similar informative content. The DBSCAN algorithm has been exploited for the cluster analysis. Clustering results are evaluated through the Silhouette [57] quality index, balancing both intra-cluster homogeneity and inter-cluster separation. To analyse tweets contained in the cluster set, each cluster has been characterized with the most representative words appearing in its tweets and the association rules modeling correlations among these words. We validated both the meaning and the importance of the information extracted from the tweet datasets with the support of news available on the web. This allows us to properly frame the context in which tweets were posted.

4.3.1 Twitter Data Collection and Preprocessing

Tweet content and their relative contextual data are retrieved through the Stream Application Programming Interfaces (APIs). Data is gathered by establishing and maintaining a continuous connection with the stream endpoint.
To suit the raw tweet textual to the following mining process, some preliminary data cleaning and processing steps have been applied. The textual message content is first preprocessed by eliminating stopwords, numbers, links, non-ascii characters, mentions, and replies. Then, it is represented by means of the Bag-of-Word (BOW) representation [46].
Tweets are transformed using the Vector Space Model (VSM) [46]. Each tweet is a vector in the word space. Each vector element corresponds to a different word and is associated with a weight describing the word relevance for the tweet. The Term Frequency (TF) - Inverse Document Frequency (IDF) scheme [46] has been adopted to weight word

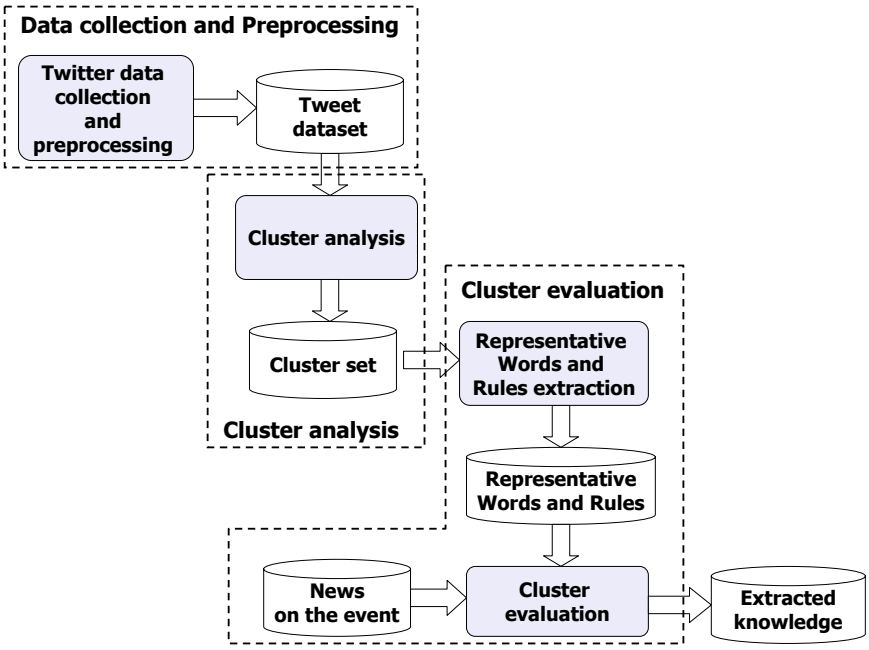


FIGURE 4.2: The proposed multiple-level clustering framework for tweet analysis

frequency. This data representation allows highlighting the relevance of specific words for each tweet. It reduces the importance of common terms in the collection, ensuring that the matching of tweets is more influenced by discriminative words with relatively low frequency in the collection. In short-messages as tweets, the TF-IDF weighting score could actually boild down to a pure IDF due to the limited word frequency within each tweet. Nevertheless, we preserved the TF-IDF approach to consider also possible word repetitions.

The tweet collection is then partitioned based on trending topics, identified by analysing the most frequent hashtags. A dataset partition is analyzed as described in the following sections.

4.3.2 Cluster Analysis

Differently from other clustering methods, density-based algorithms can effectively discover clusters of arbitary shape and filter out outliers, thus increasing cluster homogeneity. Additionally, the number of expected clusters in the data is not required. Tweet datasets can include outliers as messages posted on some specific topics and clusters can be non-spherical shaped. Besides, the expected number of clusters can be hardly guessed a priori, because our aim is discovering groups of similar tweets through an explorative data analysis. For these reasons, the DBSCAN density-based method has been selected for tweet cluster analysis.

In the DBSCAN algorithm [45], clusters are identified as dense areas of data objects surrounded by an area of low density. Density is evaluated based on the user-specified

parameters Eps and $MinPts$. A dense region in the data space is a n-dimensional sphere with radius Eps and containing at least $MinPts$ objects. DBSCAN iterates over the data objects in the collection by analyzing their neighborhood. It classifies objects as being (i) in the interior of a dense region (a core point), (ii) on the edge of a dense region (a border point), or (iii) in a sparsly occupied region (a noise or outlier point). Any two core points that are close enough (within a distance Eps of one another) are put in the same cluster. Any border point close enough to a core point is put in the same cluster as the core point. Outlier points (i.e., points far from any core point) are isolated.

One single execution of DBSCAN discovers dense groups of tweets according to one specific setting of the Eps and $MinPts$ parameters. Tweets in lower density areas are labeled as outliers and not assigned to any cluster. Hence, different parameter settings are needed to discover clusters in datasets with a variable data distribution as the one considered in this study.

In application domains where data collections have a variable distribution, clustering algorithms can be applied in a multiple-level fashion [58]. In this study we coupled a *multiple-level clustering* approach with association rule mining to discover representative clusters and the information characterizing them. Our approach iteratively applies the DBSCAN algorithm on different (disjoint) dataset portions. The whole original dataset is clustered at the first level. Then, at each subsequent level, tweets labeled as outliers in the previous level are re-clustered. The DBSCAN parameters Eps and $MinPts$ are properly set at each level by addressing the following issues. To discover representative clusters for the dataset, we aim at avoiding clusters including few tweets. In addition, to consider all different posted information, we aim at limiting the number of tweets labeled as outliers and thus unclustered.

The cosine similarity measure has been adopted to evaluate the similarity between tweets represented in the VSM model using the TF-IDF method. This measure has been often used to compare documents in text mining [46].

4.3.3 Cluster Evaluation

The discovered cluster set is evaluated using the Silhouette index [59]. Silhouette allows evaluating the appropriateness of the assignment of a data object to a cluster rather than to another by measuring both intra-cluster cohesion and inter-cluster separation.

The silhouette value for a cluster C is the average silhouette value on all its tweets. Negative silhouette values represent wrong tweet placements, while positive silhouette values a better tweet assignments. Clusters with silhouette values in the range [0.51,0.70] and [0.71,1] respectively show that a reasonable and a strong structure have been found [59]. The cosine similarity metric has been used for silhouette evaluation, since this measure was used to evaluate tweet similarity in the cluster analysis (see Section 4.3.2).

Each cluster has been characterized in terms of the words appearing in its tweets and the association rules modeling strong correlations among these words. News available on the web are used to properly frame the context in which tweets were posted and validate the extracted information. Specifically, the most representative words for each cluster are highlighted. These words are the relevant words for the cluster based on the TF-IDF weight. They occur with higher frequency in tweets in the cluster than in tweets contained in other clusters.
The quality of an association rule $X \rightarrow Y$, with X and Y disjoint itemsets (i.e., sets of words in this study), is usually measured by rule support and confidence. Rule support is the percentage of tweets containing both X and Y. Rule confidence is the percentage of tweets with X that also contain Y, and describes the strength of the implication. To rank the most interesting rules, we also used the lift index [13], which measures the (symmetric) correlation between sets X and Y. Lift values below 1 show a negative correlation between sets X and Y, while values above 1 indicate a positive correlation. The interest of rules having a lift value close to 1 may be marginal. In this work, to mine association rules representing strong word correlations, rules with high confidence value and lift grater than one have been selected.

4.4 Experimental results

This section presents and discusses the preliminary results obtained when analysing two real collections of twitter messages with the proposed framework.

4.4.1 Datasets

We evaluated the usefulness and applicability of the proposed approach on two real datasets retrieved from Twitter. Our framework exploits a crawler to access the Twitter global stream efficiently. To generate the real Twitter datasets we monitored the public stream endpoint offered by the Twitter APIs over a 1-month time period and tracked a selection of keywords ranging over two different topics, i.e., Sport and Music. The crawler establishes and maintains a continuous connection with the stream endpoint to collect and store Twitter data.
For both Twitter data collections, we analyzed the most frequent hashtags to discover trending topics. Among them, we selected the following two reference datasets for our experimental evaluation: the *paralympics* and the *concert* datasets. The *paralympics* dataset contains tweets on the Paralympic Games that took place in London in year 2012. The *concert* dataset contains tweets on the Madonna's concert held in September 6, 2012, at the Yankee Stadium located at The Bronx in New York City. Madonna is an American singer-songwriter and this concert was part of the "Mdna 2012 World Tour". Tweets in each dataset are preprocessed as described in Section 4.3.1. Hashtags used for

tweets selection have been removed from the corresponding dataset, because appearing in all its tweets.
The main characteristics of the two datasets are as follows. The paralympics dataset contains 1,696 tweets with average length 6.89. The concert dataset contains 2,960 tweets with average length 6.38.

4.4.2 Framework Configuration

In the proposed framework, the procedures for data transformation and cluster evaluation have been developed in the Java programming language. These procedures transform the tweet collection into the VSM representation using the TF-IDF scheme and compute the silhouette values for the cluster set provided by the cluster analysis. The DBSCAN [45] and FPGrowth [47] algorithms available in the RapidMiner toolkit [14] have been used for the cluster analysis and association rule extraction, respectively.
To select the number of iterations for the multiple-level clustering strategy and the DBSCAN parameters for each level, we addressed the following issues. We aim at avoiding clusters including few tweets, to discover representative clusters, and at limiting the number of unclustered tweets, to consider all posted information. For both datasets we adopted a three-level clustering approach, with each level focusing on a different dataset part. The Eps and $MinPts$ values at each iteration level for the two datasets are reported in Section 4.4.3.
To extract association rules representing strong correlations among words appearing in tweets contained in each cluster, we considered a minimum confidence threshould greater than or equal to 80%, lift greater than 1, and a minimum support threshold greater than or equal to 10%.

4.4.3 Analysis of the Clustering Results

Starting from a collection of Twitter data related to an event, the proposed framework allows the discovery of a set of clusters containing similar tweets. The multiple-level DBSCAN approach, iterated for three levels, computed clusters progressively containing longer tweets, that (i) describe the event through a more varied vocabulary, (ii) focus on some specific aspects of the event, or (ii) report user emotions and thoughts associated with the event.
First-level clusters contain tweets mainly describing general aspects of the event. Second-level clusters collect more diversified tweets that describe some specific aspects of the event or express user opinions about the event. Tweets become progressively longer and more focused in third-level clusters, indicating that some additionally specific aspects have been addressed. Since at each level clusters contain more specific messages, a lower number of tweets are contained in each cluster and the cluster size tends to

reduce progressively. By further applying the DBSCAN algorithm on the subsequent levels, fragmented groups of tweets can be identified. Clusters show good cohesion and separation as they are characterized by high silhouette values. Both the meaning and the importance of the information extracted from the two datasets has been validated with the support of news on the event available on the web.

Cluster properties are discussed in detail in the following subsections. Tables 4.1 and 4.2 report, for each first- and second-level cluster in the two datasets, the number of tweets, the average tweet length, the silhouette value, and the most representative words. Representative association rules are also reported, pointing out in a compact form the discriminative information characterizing each cluster. Clusters are named as C_{i_j} in the tables, where j denotes the level of the multiple-level DBSCAN approach providing the cluster and i locally identifies the cluster at each level j.

4.4.3.1 Tweet Analysis in the Paralympics Dataset.

First-level clusters can be partitioned into the following groups: clusters containing tweets that (i) post general information about the event (clusters C_{1_1} and C_{2_1}), (ii) regard a specific discipline (C_{3_1}) or team (C_{4_1} and C_{5_1}) among those involved in the event, (iii) report user emotions (C_{6_1}), and (iv) talk about the closing ceremony (C_{7_1}).

Specifically, clusters C_{1_1} and C_{2_1} mainly contains information about the event location (rule $\{london\} \rightarrow \{stadium, olympics\}$). Clusters C_{4_1} is about the Great Britain team taking part in the Paralympics event (rule $\{teamgb\} \rightarrow \{olympic\}$). Clusters C_{3_1} and C_{6_1} focus on the athletics discipline. While cluster C_{3_1} simply associates athletics with the Olympic event, users in cluster C_{6_1} express their appreciation on the athletics competitions they are attending (rule $\{athletics\} \rightarrow \{amazing, day\}$). Finally, tweets in cluster C_{7_1} talk about the seats of people attending the final ceremony (rule $\{closingceremony, stadium\} \rightarrow \{seats\}$).

Second-level clusters contain more diversified tweets. The following categories of clusters can be identified: clusters with tweets posting information on (i) specific events in the closing ceremony (clusters C_{1_2} and C_{2_2}), (ii) specific teams (cluster C_{3_2}) or competitions (cluster C_{4_2}) in Paralympics, and (iii) thoughts of people attending Paralympics (cluster C_{5_2}).

More in detail, cluster C_{1_2} focuses on the flame that was put out on the day of the closing celebration (rule $\{stadium, london\} \rightarrow \{flame, closingceremony\}$), while cluster C_{2_2} is on the fireworks that lit up London's Olympic stadium in the closing ceremony (rule $\{stadium, closingceremony\} \rightarrow \{fireworks\}$). Cluster C_{3_2} is about the Great Britain team taking part to athletics discipline (rule $\{teamgb, park\} \rightarrow \{athletics\}$). Tweets in cluster C_{4_2} address the final basketball competition in the North Greenwich Arena. They contain the information about the event location and the German

women's team involved in the competition (rules $\{final\} \rightarrow \{north, germany\}$ and $\{final\} \rightarrow \{basketball, germany\}$). Tweets in cluster C_{5_2} show an enthusiastic feeling on Paralympics (rule $\{stadium, olympic\} \rightarrow \{london, fantasticfriday\}$) and the desire to share pictures on them (rule $\{pic, dreams\} \rightarrow \{stadium, time\}$).
Third-level clusters (with DBSCAN parameters $MinPts = 15$, $Eps = 0.65$) show a similar trend to second-level clusters. For example, clusters contain tweets on some specific aspects of the closing ceremony, as the participation of the ColdPlay band (rule $\{london\} \rightarrow \{coldplay, watching\}$), or tweets about a positive feeling on the Paralympics event (rules $\{love\} \rightarrow \{summer, olympics\}$ and $\{gorgeous\} \rightarrow \{day\}$). By stopping the multiple-level DBSCAN approach at this level, 808 tweets labeled as outliers remain unclustered, with respect to the initial collection of 1,696 tweets.

4.4.3.2 Tweet Analysis in the Concert Dataset.

Among first-level clusters, we can identify groups of tweets mainly posting information on the concert location (clusters C_{1_1}, C_{2_1}, and C_{3_1} with rule $\{concert, mdna\} \rightarrow \{yankee\}$). The remaining clusters talk about some aspects of the concert. For example, cluster C_{4_1} regards the opening act (rule $\{yankee, stadium\} \rightarrow \{opening, act\}$). Cluster C_{5_1} is on the participation of the Avicii singer (rule $\{wait\} \rightarrow \{yankee, avicii\}$), cluster C_{6_1} on the "forgive" writing on Madonna's back (rule $\{forgive\} \rightarrow \{stadium, nyc\}$), and cluster C_{7_1} is about the raining weather (rule $\{rain\} \rightarrow \{yankee, stadium\}$). Finally, cluster C_{8_1} regards people sharing concert pictures (rule $\{queen\} \rightarrow \{instagram\}$).
In second-level clusters, tweets focus on more specific aspects related to the concert. For example tweets in cluster C_{2_2} refer to Madonna with the "madge" nickname typically used by her fans (rule $\{singing\} \rightarrow \{stadium, madge\}$).
Similar to the paralympics dataset, also in the concert dataset third-level clusters (with DBSCAN parameter Eps=0.77 and $MinPts$=23) show a similar trend to second-level clusters. For example, clusters contain tweets regarding some particular songs. At this stage, 1660 tweets labeled as outliers remain unclustered, with respect to the initial collection of 2,960 tweets considered at the first level.

4.4.4 Performance Evaluation

Experiments were performed on a 2.66 GHz Intel(R) Core(TM)2 Quad PC with 8 GB main memory running linux (kernel 3.2.0). The run time of DBScan at the first, second, and third level is respectively 2 min 9 sec, 1 min 9 sec, and 48 sec for the paralympics dataset, and 4 min 4 sec, 1 min 53 sec, and 47 sec for the concert dataset. The run time progressively reduces because less tweets are considered at each subsequent level. The time for association rule extraction is about 24 sec for the cluster set at each level.

TABLE 4.1: First- and second-level clusters in the paralympics dataset (DBSCAN parameters $MinPts$=30, Eps=0.39 and $MinPts$=25, Eps=0.49 for first- and second-level iterations, respectively)

First-level clusters					
Cluster	**Tweets**	**Avg Length**	**Avg Sil**	**Words**	**Association Rules**
C_{1_1}	70	3	1	olympic, stadium	olympic→ stadium
C_{2_1}	30	7.33	0.773	olympics, london, stadium	london→ stadium, olympics
C_{3_1}	124	4.47	0.603	london, park, athletics, day	london, day→ athletics olympic→ park, athletics
C_{4_1}	30	6.67	0.710	heats, teamgb, olympic	teamgb→ olympic heats→ teamgb
C_{5_1}	30	5.67	0.806	mens, olympic, stadium	mens→ olympic
C_{6_1}	40	6	0.620	day, pic, amazing, athletics	athletics→ amazing, day day, pic→ stadium
C_{7_1}	36	5.72	0.804	closingceremony, seats, park, stadium	closingceremony, stadium→ seats olympic, park→ closingceremony
Second-level clusters					
Cluster	**Tweets**	**Avg Length**	**Avg Sil**	**Words**	**Association Rules**
C_{1_2}	90	5.67	0.398	flame, closingceremony, london, stadium	stadium,london→ flame,closingceremony
C_{2_2}	36	6.67	0.616	fireworks, closingceremony, hart, stadium	stadium,closingceremony→ fireworks fireworks, hart→ stadium
C_{3_2}	26	6.08	0.722	teamgb, athletics, park, olympic, london	teamgb, park→ olympic teamgb, park→ athletics olympic, park→ teamgb, london
C_{4_2}	34	9.65	0.502	greenwich, north, arena, basketball germany, final, womens	final→ north, germany final→ basketball, germany final→ womens, germany
C_{5_2}	40	6.5	0.670	fantasticfriday, dreams, time, pic olympic, london, stadium	pic, dreams→ stadium,time stadium,olympic→ london, fantasticfriday

TABLE 4.2: First- and second- level clusters in the concert dataset (DBSCAN parameters $MinPts$=40, Eps=0.41 and $MinPts$=21, Eps=0.62 for the first- and second-level iterations, respectively)

First-level clusters					
Cluster	**Tweets**	**Avg Length**	**Avg Sil**	**Words**	**Association Rules**
C_{1_1}	148	5.05	0.817	concert, mdna, yankee, stadium	concert, yankee→ stadium concert, mdna→ yankee
C_{2_1}	340	4	1	bronx, yankee, stadium	yankee, stadium→ bronx
C_{3_1}	160	3	1	yankee, stadium	stadium→ yankee
C_{4_1}	40	6	0.950	opening, act, mdna, yankee, stadium	act→ opening yankee, stadium→ opening, act
C_{5_1}	60	6	0.779	avicii, wait, concert	wait→ yankee, avicii
C_{6_1}	84	6.19	0.794	forgive, nyc, mdna, stadium	forgive→ stadium, nyc
C_{7_1}	40	7	0.986	rain, yankee, stadium	rain→ yankee, stadium
C_{8_1}	40	6	0.751	queen, instagram, nyc	queen→ instagram
Second-level clusters					
Cluster	**Tweets**	**Avg Length**	**Avg Sil**	**Words**	**Association Rules**
C_{1_2}	60	6.67	0.523	raining, mdna, stop	raining→ mdna, stop
C_{2_2}	40	7	0.667	madge, dame, named, singing	singing→ stadium, madge madge, singing, named→ stadium, dame
C_{3_2}	44	7.64	0.535	surprise, brother, birthday, avicii, minute	yankee, stadium, surprise→ birthday
C_{4_2}	22	8.55	0.893	style, way, vip, row livingthedream	style→ vip, livingthedream

Chapter 5

Analyzing Twitter User-Generated Content Changes

In the previous Chapter, we proposed a novel analysis framework to discover, in a data collection with a variable distribution, cohesive and well-separated groups of tweets. Besides that, Twitter user-generated content consists of a large collection of short textual messages (i.e., the tweets) posted by Web users and their contextual information (e.g., publication time and date). Since the Twitter user-generated content and contextual data continuously evolve over time, a relevant research issue is the application of data mining techniques to discover most significant pattern changes. Dynamic itemset mining [60] entails discovering itemsets that (i) frequently occur in the analyzed data, and (ii) may change from one time period to another. The history of the main itemset quality indexes reflects the most relevant temporal data correlation changes. However, the sparseness of the analyzed data makes dynamic itemset mining from UGC a challenging task. In fact, potentially relevant itemsets discovered at a certain time period are likely to become infrequent (i.e., their support value becomes lower than a given threshold) in at least another one. Hence, the information associated with the discovered itemsets may be lost, unless lowering the support threshold and mining a huge amount of other (potentially redundant) itemsets.

This Chapter presents the TwiChI (Twitter Change mIner) system that aims at supporting experts in the analysis of Twitter UGC changes targeted to user behavior and topic trend analysis. TwiChI exploits the Twitter Application Programming Interfaces (APIs) to retrieve both tweet textual contents and their contextual features (i.e., publication date, time, place). Data crawling is continuously executed using the Twitter Public stream endpoint to track the temporal evolution of the frequent itemsets occurring in the analyzed data. The retrieved data is analyzed by the proposed HiGen Miner algorithm [61], which discovers compact patterns, named the History Generalized Patterns (HiGens). HiGens represent the evolution of frequent itemsets across consecutive time

Timestamped tweet dataset	*ItemSet*	*Support(%)*
$D_{Jan2012}$	(Place, New York City), (Time, 3.45 p.m.)	20%
	(Keyword, Obama), (Place, New York City)	10%
$D_{Feb2012}$	(Place, New York State), (Time, from 3 to 6 p.m.)	50%
	(Keyword, President of USA), (Place, New York City)	16%

TABLE 5.1: Example of HiGens extracted by enforcing the minsup = 10%.

periods. To avoid the discarding of rare but potentially relevant knowledge, itemsets that become infrequent in a certain time period with respect to the minimum support threshold are generalized at a higher level of abstraction by exploiting a taxonomy (i.e., a set of is-a hierarchies built on data items). A generalized version of a traditional itemset is an itemset that represents the same knowledge at a higher level of aggregation according to a given taxonomy [62]. Hence, the knowledge associated with itemsets that rarely occur at certain time periods is still maintained by replacing the low level itemset versions with their frequent generalizations with least abstraction level.

Consider, for instance, tweet messages and related contextual information (e.g., publication time, geographical location) retrieved in the period January and February 2012. The tweet collection may be partitioned into two distinct monthly time periods. Analyzing the two sub-collections, the TwiChI framework may discover the HiGens reported in Table 5.1. Suppose that *(Keyword, Obama), (Place, New York City)* is the reference itemset under analysis. Since it occurs frequently in January 2012 according to the enforced minimum support threshold (i.e., a minimum frequency of occurrence in the source data), then it is reported for the corresponding time period as is. Instead, since in February 2012 the reference itemset becomes infrequent, it is generalized by exploiting an analyst-provided taxonomy. In particular, item *(Keyword, Obama)* is generalized as the corresponding government role and the corresponding high level version of the reference itemset *(Keyword, President of USA), (Place, New York City)* is reported. Note that by generalizing the reference itemset at a higher level of abstraction, its associated information becomes frequent with respect to the support threshold and is kept instead of the infrequent version.

Experiments, performed on real Twitter datasets, show the applicability of the proposed system to real-life use-cases. For instance, the HiGen reported in Table 5.1 may be used to discover which Twitter message topics (e.g., politics) are more likely to be matter of contention in certain time slots. The achieved experimental results show that TwiChI is particularly suitable for supporting domain expert analysis targeted to user behavior analysis and topic trend detection.

5.1 Related work

This section overviews main state-of-art approaches related to (i) generalized itemset mining, (ii) dynamic data mining, and (iii) data mining from user-generated content.

5.1.1 Generalized itemset mining

Frequent itemset mining is a widely exploratory data mining technique that allows the identification of hidden and interesting correlations among data. Introduced in the context of market basket analysis, this mining activity nowadays finds applications in a wide range of different contexts (e.g., network traffic characterization [63], context-aware applications [64]. However, the suitability of data mining approaches for business decisions strictly depends on the abstraction level of the analyzed data. Traditional frequent itemset mining algorithms (e.g., Apriori [65], FP-Growth [47]) are sometimes not effective in mining valuable knowledge, because of the excessive detail level of the mined information. In fact, to make the mining processcomputationally tractable a minimum support threshold is commonly enforced to select only the patterns that frequently occur in the analyzed data. Hence, rare but potentially relevant knowledge is discarded.
Generalized itemsets [62] are patterns that represent high level correlations among data. By exploiting a taxonomy (i.e., a set of is-a hierarchies) that aggregates data items into upper level generalizations, generalized itemsets are generated by combining items belonging to different abstraction levels. Generalized itemsets may allow better supporting the expert decision process than traditional ones, because they provide a high level view of the analyzed data and also represent the knowledge covered by their low level infrequent descendants.
The first generalized association rule mining algorithm, namely Cumulate, was presented in [62]. It is an Apriori-based algorithm that generates generalized itemsets by considering, for each item, all its parents in the hierarchy. One step further towards a more efficient extraction process for generalized association rule mining was based on new optimization strategies [66, 67]. In [67] a faster support counting is provided by exploiting the TID intersection computation, which is common in rule mining algorithms designed for the vertical data format. Differently, in [66] an optimization based on top-down hierarchy traversal and multiple-support thresholds is proposed. It aims at identifying in advance generalized itemsets that cannot be frequent by means of an Apriori-like approach. To further increase the efficiency of generalized rule mining algorithms, in [68] a FP-tree based algorithm is proposed, while in [69] both subset-superset and parent-child relationships in the lattice of generalized itemsets are exploited to avoid generating meaningless patterns. More recently, in [70] authors propose an algorithm that performs support-driven itemset generalization, i.e., a frequent generalized itemset is extracted only if it has at least an infrequent (rare) descendant.

This work focuses on analyzing the temporal evolution of generalized itemsets mined from Twitter data. The mining algorithm integrated in TwiChI [61] extends the generalization procedure first proposed in [70] to a dynamic context. However, unlike [70], it does not extract all frequent generalizations of an infrequent low level itemset, but considers only the ones characterized by minimum abstraction level.

5.1.2 Dynamic data mining

Traditional itemset and association rule mining approaches do not take the temporal evolution of the extracted itemsets/association rules into account. Instead, dynamic data mining focuses on tracing the evolution of the main itemset and/or association rule quality indexes to figure out the most significant temporal changes.

The problem of discovering relevant changes in the history of itemsets or association rules has already been addressed by a number of previous works [60, 61, 71–74]. For instance, active data mining [60] has been the first attempt to represent and query the history collection of the discovered association rule quality indexes. Rules are mined from datasets collected at consecutive time periods and evaluated based on well-known quality indexes (e.g., support, confidence). Then, the analyst is in charge of specifying a history pattern in a trigger which is fired when such a pattern trend is exhibiting. The history patterns are exploited to track most notable pattern index changes. More recently, other time-related data mining frameworks tailored to monitor and detect changes in rule quality measures have also been proposed [71, 72, 74]. For instance, in [72], patterns are evaluated and pruned based on both subjective and objective interestingness measures, while in [71] authors focus on monitoring pattern mining with a limited computational effort. To this aim, new patterns are observed as soon as they emerge, while old patterns are removed from the rule base as soon as they become extinct. Furthermore, at one time period a subset of rules is selected and monitored, while data changes that occur in subsequent periods are measured by their impact on the rules being monitored. Similarly, the work proposed in [74] also addresses itemset change mining from time-varying data streams. Differently, in [73] authors deal with rule change mining by discovering two main types of rules: (i) stable rules, i.e., rules that do not change a great deal over time and, thus, are likely to be reliable and could be trusted, and (ii) trend rules, i.e., rules that indicate some underlying systematic trends of potential interest.

Since all the above-mentioned approaches do not consider itemsets/rules at different abstraction levels, their ability in capturing relevant data correlation changes may be biased by the support threshold enforcement. In fact, some relevant trends may be discarded, because the underlying recurrences become infrequent at a certain time period. To overcome this issue, in [61] a dynamic itemset mining approach has recently been

proposed. It discovers History Generalized Patterns, which represent a sequence of generalized itemsets extracted in consecutive time periods. Each HiGen is mainly focused on a reference itemset, whose support index values are traced in the consecutive time periods. In case an itemset becomes infrequent in a certain time period, its generalization with least abstraction level is maintained to avoid discarding potentially relevant knowledge. This Chapter proposes a data mining system that discovers History Generalized Patterns from Twitter UGC and exploits them to drive the knowledge discovery process.

5.1.3 Data mining from user-generated content

The proliferation of the UGC, posted by Web users in different data formats (e.g., posts, tags, videos), has increased the attention of the research community in developing new methods to manage and analyze this huge amount of information. The UGC coming from social networks and online communities is a powerful resource of information which can be analyzed by means of different data mining approaches. Even if the most significant research efforts have been devoted to improving the performance of recommendation and categorization systems, in the last several years the analysis and the identification of the evolution of the UGC content, user behaviors and interests have been received more and more attention by the research community. In particular, the proposed approaches are mainly addressed to (i) improve the knowledge discovery processes from online resources, (ii) discover topic trends of the news published online and (iii) understand the dynamics behind social networks and online communities.

One of the main research directions is the discovery of most relevant online community user behaviors [75, 76]. For instance, in [75] common user activities (e.g., universal searches, message sending, and community creation) are discovered by means of clickstream data analysis. Differently, [76] study the UGC lifetime by empirically analyzing the workloads coming from three popular knowledge-sharing online social networks, i.e., a blog system, a social bookmark sharing network, and a question answering social network.

The UGC published on social networks, such as Facebook and Twitter, can be very useful for profiling user behaviors and discovering patterns valuable for further analysis. In particular, several new approaches have been proposed to support knowledge discovery from Twitter by means of data mining techniques. For instance, TwitterMonitor [43] is focused on the detection of topic trends from Twitter streams. This system first identifies and clusters the "bursty" keywords (i.e., keywords that appear in tweets at unusually high rate), and then performs contextual knowledge extraction to compose an accurate description of the identified trends. Trend patterns can also be exploited to support decision-making and recommendation processes. For instance, [44] analyze the trend of

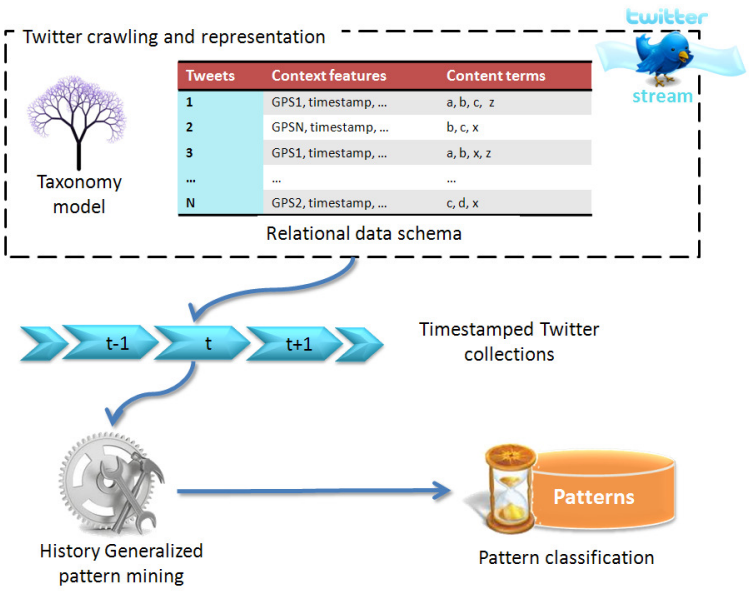


FIGURE 5.1: The TwiChI framework

the topics and the demographics of the sets of Twitter users who contribute towards the discussion of particular trends to support decision-making activities. Differently, [77] combine RSS news and UGC coming from microblogs into a news recommendation system. In particular, they mine Twitter message content to identify emerging topics and breaking events. The RSS stories have been ranked based on a weighted score that takes the Lucene tf-idf score of each article term and the information provided by tweets into account.

Similarly, this Chapter also presents a data mining system to perform knowledge discovery from messages posted on Twitter. Unlike previous approaches it exploits both the content and the contextual information associated with Twitter posts to perform user behavior and topic trend analysis. To this aim, it extracts generalized dynamic patterns that represent the evolution of the most relevant patterns over consecutive time periods at different abstraction levels.

5.2 The TwiChI framework

The TwiChI (Twitter Change mIner) is a data mining system aimed at supporting the discovery of dynamic patterns that represent the historical evolution of the most valuable correlations among textual content and publication context of messages posted on Twitter (tweets). The extracted patterns can represent the changes in user behaviors and/or topic trends. In Figure 5.1 the TwiChI framework architecture is shown, while the main blocks of the system are briefly described in the following.

Twitter data crawling and representation. This block aims at retrieving and preprocessing user-generated messages (tweets) posted on Twitter. Tweets are partitioned

in a sequence of collections according to their publication date. For each collection, the main tweet features are modeled into two different representations: (i) a relational data schema, and (ii) a taxonomy model. The relational data schema includes both content (i.e., the message words) and contextual (e.g., the geographical location) features. The taxonomy model is composed of a set of hierarchies built over the tweet contextual and content features and is generated by a semi-automatic process. In particular, aggregation functions based on hierarchical models are exploited to aggregate values of lower level features (e.g., the GPS coordinates) into their higher level aggregations (e.g., cities and regions). Aggregation functions may be generated by exploiting either established knowledge bases (e.g., WordNet) or Extraction, Loading, and Transformation (ETL) processes.

History Generalized pattern mining. This block focuses on discovering History Generalized patterns (HiGens) from the sequence of timestamped tweet collections by exploiting the recently proposed HiGen Miner algorithm [61]. HiGens represents the most significant data correlation changes by also considering knowledge at different abstraction levels.

Pattern classification. The last block focuses on categorizing the extracted HiGens based on their main characteristics to ease the expert in-depth analysis. HiGens are classified as (i) stable HiGens, (ii) monotonous HiGens, and (iii) oscillatory HiGens, according to the time-related trend. In particular, the evolution trend of the abstraction level at which patterns are represented within each time period is considered as discriminative feature.

In the following sections a more detailed description of the main TwiChI framework blocks is given.

5.2.1 Twitter data crawling and representation

This block addresses the retrieval and preprocessing of the tweets posted on Twitter. User-generated tweets are at most 140 characters long and publicly visible by default. Moreover, they are enriched by several contextual features (e.g., publication location in terms of GPS coordinates, date, and hour) which are peculiar characteristics of the context in which tweets are posted. Since data retrieved by Twitter Stream APIs (Application Programming Interfaces) is not suitable for being directly analyzed by a dynamic miner, an ad-hoc crawling procedure and a preprocessing phase are needed. In the following, the data representation and the Twitter crawler of the TwiChI system are presented.

5.2.1.1 Twitter data representation

Given a collection of retrieved tweets, we define two different data representations which will be exploited by the subsequent TwiChI mining step: (i) a relational data schema, and (ii) a taxonomy model. In the following each data representation is better formalized. ***Relational data schema.*** Tweets belonging to a retrieved collection are composed of the textual message and a set of contextual features (e.g. publication date, time, location). To represent tweets into a relational schema both message words and contextual feature values are modeled as data items, where an item (l_i, v_i) is a couple (attribute, value) and the value vi belongs to the discrete domain attribute of the attribute l_i. When coping with continuous attributes, the value range is discretized into intervals and the intervals are mapped to consecutive positive integers. Items represent either the textual message content, (e.g., text word "travel"), or a contextual feature value (e.g., Date, 2012-07-28). A tweet could be represented as a set of items, called record, as stated in the following.

Definition 1. Record. Let $L = l_1, l_2, \ldots, l_n$ be a set of attributes and $\Omega = \Omega_1, \Omega_2, \ldots, \Omega_n$ the corresponding domains. A record r is a set of items that contains at most one item for each attribute in L. Each record is characterized by a time stamp t.

The time stamp t is defined by the analyst during the crawling process and may represent the tweet publication date or time. A set of records (tweets) whose time stamps belong to a fixed time period T is called timestamped relational tweet collection.

Definition 2. Timestamped relational tweet collection. Let $L = l_1, l_2, \ldots, l_n$ be a set of attributes and $\Omega = \Omega_1, \Omega_2, \ldots, \Omega_n$ the corresponding domains. A relational tweet collection D_T is a collection of records, where each record r has a time stamp that belongs to the time period T.

For instance, when considering as timestamp the tweet publication date and as time period T = [July 1st 2012, July 31st 2012] each crawled tweet that has been published in July 2012 is included in the timestamped tweet collection relative to T.

To enable the dynamic mining process, tweets are organized in a sequence of timestamped relational tweet collections relative to consecutive time periods. For instance, tweets crawled in the first trimester of the year 2012 may be partitioned in a sequence of three timestamped collections, each one related to a distinct monthly time period.

Taxonomy model. Semantic relationships between attribute values belonging to a tweet collection are usually not defined in the relational data schema. To drive the generation of generalized itemsets we define a taxonomy, which is a hierarchical model that represents the is-a relationships holding between data instances (i.e., the data items) relative to the same concept (i.e., the attributes). To aggregate attribute values into higher level concepts, we introduce the notion of aggregation tree, i.e., an aggregation hierarchy built on the domain of one attribute of the relational tweet collection.

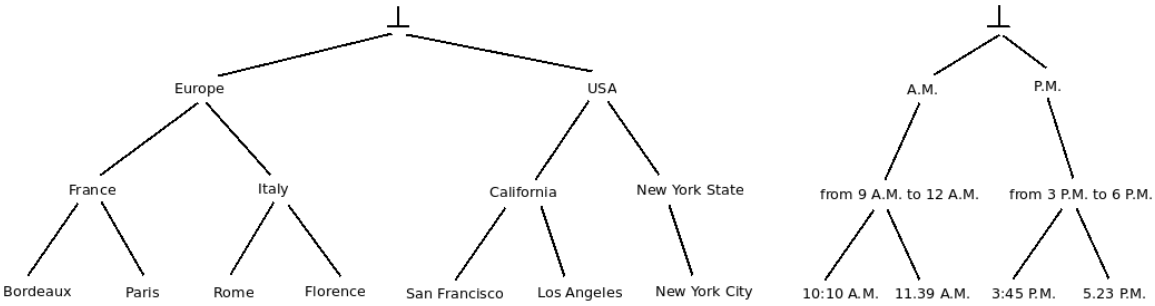


FIGURE 5.2: Examples of aggregation trees.

Definition 3. Aggregation tree. Let l_i be an attribute and Ω_i its domain. An aggregation tree A_i is a tree representing a predefined set of aggregations over values in Ω_i. Ai leaves are all the values in Ω_i. Each non-leaf node in A_i is an aggregation of all its children. The root node $\perp$ aggregates all values for attribute l_i.

Figure 5.2 reports two examples of aggregation trees built on the *Place* and *Time* attributes, respectively.

We define a taxonomy as a set of aggregation trees built over distinct data attributes. Despite a taxonomy may potentially include many aggregation trees over the same attribute, for the sake of simplicity in the following we exclusively consider taxonomies that contain at most one aggregation tree A_i $in\rho$ per attribute l_i inL. Given a taxonomy Δ, we formalize the concept of generalized item as an item (l_i, e_i) such that e_i is a non-leaf node in some A_i $in\Delta$.

Definition 4. Generalized item. Let l_i be an arbitrary attribute, Ω_i its domain, and A_i an aggregation tree built on values in Ω_i. A generalized item (l_i, e_i) assigns the value e_i to attribute l_i . e_i is a non-leaf node in A_i which defines an aggregation value over values in Ω_i. $leaves(e_i) \subseteq \Omega_i$ is the set of items whose values are leaf nodes descendant of e_i in A_i.

The support of a generalized item (l_i, e_i) in a relational tweet collection d_t is the (observed) frequency of leaves(e_i) in D_T.

For instance, if the words “Boots” and “Tennis Shoes” occur, respectively, in half and one third of the tweets of a collection, their supports are 50% and 33%. If “Boots” and “Tennis Shoes” are the only descendants of the common generalization “Shoes”, according to a given taxonomy, the support of “Shoes” is 50%.

The two data representations are generated by the Twitter crawler described in the following, which also partitions the retrieved data into collections based on the publication timestamp.

5.2.1.2 Twitter crawler

Twitter APIs are general-purpose tools that allow the efficient retrieval of tweets from the Web. However, tweets inherent to the submitted queries are retrieved disregarding the

temporal and semantic relationships among their content. Moreover, tweets are provided in a data format which is commonly unsuitable for further analysis. For instance, the tweet geographical provenance is provided as a couple of GPS coordinates, but the related city, region, and/or state are usually missing. Furthermore, it may be not easy to differentiate between tweets published in close time periods (e.g., during the last 12 hours) from the ones that are rather far from (e.g., the tweets published the day before). Since our system addresses the analysis of the dynamic data correlation changes that occur in the messages posted by the community, we exploit a tweet crawler that automatically collects and organizes timestamped relational tweets relative to a sequence of given time periods. To this aim, D is defined as the original set of tweets collections and D_T is a collection of tweets whose time stamps are contained in the time period T. The tweet crawler has the following parameters: (i) the sequence of time periods whereby tweets are partitioned, and (ii) a set of filtering parameters. Filtering parameters include all the parameters provided by Twitter APIs, such as the selection of keywords and the geographical radius used to select the tweets of interest from the Public stream. The crawler continuously monitors the stream and retrieves tweets according to the search parameters. At the end of a given time period, a new collection D_T is defined according to the predefined time scheduling.

Since data is retrieved in the JSON format, a preprocessing step is applied to suit tweets to the two-way data representation (see Section "Twitter data representation"). The relational data schema is generated by a data cleaning process which discards useless and redundant information and correctly manages missing values. For each tweet, the textual message is tailored to the Bag-of-Word (BOW) representation. It includes only the terms selected by a stemming algorithm. The stemming method integrated in the TwiChI system discards noisy data such as stopwords, numbers, and links. The relational data schema, composed of the set of distinct terms belonging to the BOW representation, is then enriched with the set of contextual information (e.g., GPS coordinates, publication date, Twitter username) provided by the Twitter APIs.

To build a taxonomy over the Twitter relational data distinct aggregation trees are built over each tweet feature (e.g, spatial information, and message words). To properly manage data associated with distinct attributes, the aggregation values used for generalizing low level item values are extracted by means of semi-automatic procedures called *aggregation functions*. In particular, we exploit a set of ad-hoc aggregation functions tailored to each attribute domain. To prevent discarding useful information and enrich the tweet features, the aggregation functions can exploit established semantics-based models, such as controlled vocabularies or lexical/domain-specific databases. For instance, an aggregation function that accesses a geographical database is used to define the relationship between the GPS coordinates and their corresponding region or state. Similarly, the WordNet lexical database (http://wordnet.princeton.edu) is queried to

retrieve the most relevant semantic relationships holding between tweet term couples. More specifically, we focus on the hyponyms (i.e., is-a-subtype-of relationships). Terms belonging to these relationships are considered as generalizations of the original term. Consider, as an example, the term "dog". Since the semantic relationship <dog> is-a-subtype-of <domestic animal> is retrievable from the WordNet database, then the term "domestic animal" is selected as the upper level generalization of the term "dog". To enrich the aggregation tree built over textual features, the database querying process is deepened to find all the possible upper level aggregations (e.g., < dog > is-a-subtype-of <animal>). If no semantics-based model is available for a given attribute, the aggregation functions may extract is-a relationships by simply parsing the corresponding attribute domain values, by exploiting an approach similar to the Extraction, Transformation and Load (ETL) processes used in data warehousing (Kimball et al., 2002). Consider, for instance, the " Date" attribute and its high level aggregation "Semester". The corresponding mapping may be simply derived by parsing the lower level "Date" domain values (e.g., 2012-07-28) and generating upper level concepts (e.g.,2^{nd} Semester 2012) according to the corresponding aggregation function (i.e., Date $\rightarrow$ Semester). The generalization hierarchies extracted by means of the above-mentioned aggregation functions are combined in a taxonomy, which will be used to drive the dynamic generalized itemset mining process, as described in the following.

5.2.2 History Generalized pattern mining

This block aims at discovering from the collection of timestamped relational tweet collections dynamic patterns, namely the History Generalized patterns (HiGens) that represent the evolution of the most notable data correlation changes.

Correlations among the tweet content and context collected within each time period are represented in the form of *generalized itemsets*. A formal definition of generalized itemset follows.

Definition 5. (Generalized) itemset. Let L be a set of attributes, Ω the corresponding domains, and Δ a taxonomy defined on values in Ω. An itemset I is a set of items (l_k, e_k) in which each attribute $l_k \in L$ may occur at most once. A generalized itemset is an itemset that includes at least a generalized item (t_k, e_k) such that $e_k \in \Delta$.

For instance, (Place, New York), (date, October 2010) is a generalized itemset of length 2 (i.e., a generalized 2-itemset).

A (generalized) itemset covers a given record (tweet) with timestamp t if all its (possibly generalized) items $x \in X$ are either contained in r or ancestors of items $i \in r$ (i.e., i *leaves(x)*, i ∈ r). The support of a (generalized) itemset X in a timestamped relational tweet collection D_T is given by the number of tweets $r \in D_T$ covering X divided by the cardinality of D_T.

The generalization level of a (generalized) itemset is affected by the highest generalized item level according to the given taxonomy.

Definition 6. (Generalized) itemset level. Let $X = \{(t_1, e_1), \ldots, (t_k, e_k)\}$ be a (generalized) k-itemset. Its level *L[X]* is the maximum item generalization level by considering items in X, i.e., $L[X] = max_{1 \leq j \leq k} L[(l_j, e_j)]$.

It follows that the level of a not generalized itemset is 1.

A descendant of an itemset represents part of its knowledge at a lower aggregation level.

Definition 7. (Generalized) itemset descendant/ancestor. Let Q be taxonomy. A (generalized) itemset X is a descendant of a generalized itemset Y if (i) X and Y have the same length and (ii) for each item $y \in Y$ there exists at least an item $x \in X$ that is a descendant of y with respect to Q. If X is a descendant of Y then Y is an ancestor of X.

Consider the generalized itemset (Place, New York State), (date, from 3 to 6 p.m.). According to the taxonomy reported in Table 5.1, its level is 2 because (Place, New York) and (date, from 3 to 6 p.m.) have levels 2. Furthermore, it is an ancestor of (Place, New York City), (date, 3.45 p.m.). If (Place, New York State), (date, from 3 to 6 p.m.) covers half of the tweets contained in the analyzed timestamped collection, its support is 50%.

The generalized itemset mining task entails discovering all itemsets (generalized and not) that satisfy a minimum support threshold *minsup*, i.e., the itemsets whose frequency of occurrence is above or equal to *minsup*. Itemsets satisfying the above constraint are said to be *frequent*.

To analyze changes in the evolution of the extracted itemsets in consecutive time periods, TwiChi discovers the dynamic patterns, namely the History Generalized patterns (HiGens), proposed in [61].

Definition 8. HiGen. Let $D = \{D_1, \ldots, D_n\}$ an ordered sequence of timestamped relational tweet collections, Δ a taxonomy built on D, it a not generalized itemset, named*reference itemset*, and *minsup* a minimum support threshold. A HiGen *HGit* relative to it is an ordered sequence of generalized itemsets $g_1, \ldots, gn$ such that:

- if *it* is frequent in $D_i \in D$ then $g_i = it$
- else g_i is an frequent ancestor characterized by minimal generalization level with respect to Δ among the frequent ancestors of *it*

Each HiGen is associated with a (not generalized) reference itemset and describes its evolution, in terms of its main quality indexes, from one time period to another. Notice that, by Definition 8, each not generalized itemset may be associated with one or more HiGens. In case the considered reference itemset becomes infrequent with respect to the support threshold in a given time period, it is substituted by its generalization(s)

with minimal level. Hence, the knowledge covered by the considered pattern is still maintained at a higher level of abstraction for this time period.
For instance, the HiGens, reported in Table 5.1, may represent the evolution of the reference itemset {(Place, New York City), (Time, 3.45 p.m.)} over two example timestamped relational tweet collections $D_{Jan2012}$ and $D_{Feb2012}$, retrieved in two consecutive monthly time period (January and February 2012, respectively), by enforcing a minimum support threshold equal to 20% and by exploiting the taxonomy reported in Figure 5.2. Since the reference itemset, which is frequent in $D_{Jan2012}$, becomes infrequent in $D_{Feb2012}$ with respect to the support threshold its frequent generalization {(Place, New York State), (Time, from 3 to 6 p.m.)} is kept in place of it.
A brief description of the algorithm exploited to extract HiGens is given in the following.

5.2.3 The HiGen miner algorithm

Given a sequence of timestamped relational tweet collections, a taxonomy, and a minimum support threshold, HiGen Miner discovers all HiGens, according to Definition 8.
To avoid extracting HiGens as a postprocessing step that follows the traditional generalized itemset mining phase, HiGen Miner exploits an Apriori-based support-driven generalized itemset mining approach in which the generalization procedure is triggered on infrequent itemsets only. The generalization process does not generate all possible ancestors of an infrequent itemset at any abstraction level, but it stops at the generalization level in which at least a frequent ancestor occurs. Furthermore, the taxonomy evaluation procedure over a pattern is postponed after its support evaluation in all timestamped collections to avoid multiple (computationally expensive) evaluations.
A pseudo-code of the HiGen MINER is reported in Algorithm 1. At an arbitrary iteration k, HiGen MINER performs the following three steps: (i) k-itemset generation from each timestamped collection in D (line 3), (ii) support counting and generalization of infrequent (generalized) k-itemsets of increasing level (lines 6-37), (iii) generation of candidate itemsets of length k+1 by joining k-itemsets and infrequent candidate pruning (line 39). After being generated, frequent k-itemsets are included in the corresponding HiGens contained in the HG set (line 9), while infrequent ones are generalized by means of the taxonomy evaluation procedure (line 17). Given an infrequent itemset c of level l and a taxonomy, the taxonomy evaluation procedure generates a set of generalized itemsets of level *l+1* by applying, on each item the corresponding generalization hierarchy. All the itemsets obtained by replacing one or more items in c with their generalized versions of level textitl+1 are generated and included into the Gen set (line 21). Finally, generalized itemset supports are computed by performing a dataset scan (line 26). Frequent generalizations of an infrequent candidate c, characterized by level *l+1*, are first

added to the corresponding HiGen set and then removed from the Gen set when their lower level infrequent descendants in each time period have been fully covered (lines 27- 32). In such a way, their further generalizations at higher abstraction levels are prevented. Hence, the taxonomy evaluation over an arbitrary candidate of length k is postponed when the support of all candidates of length k and generalization level l in each timestamped dataset is known. The sequence of support values of an itemset that is infrequent in a given time period is store and reported provided that (i) it has at least a frequent generalization in the same time period, and (ii) it is frequent in at least one of the remaining time periods. The generalization procedure stops, at a certain level, when the *Gen* set is empty, i.e., when either the taxonomy evaluation procedure does not generate any new generalization or all the considered generalizations are frequent in each time period and, thus, have been pruned (line 30) to prevent further knowledge aggregations. The algorithm ends the mining loop when the set of candidate itemsets is empty (line 40).

5.2.4 Pattern classification

Domain experts are usually in charge of analyzing the results of the data mining process to discover patterns valuable for targeted analysis. TwiChi provides to the experts a selection of dynamic generalized patterns, i.e., the HiGens, which represents potentially valuable Twitter data correlation changes. However, the amount of the discovered patterns may be large, especially when low support threshold values are enforced. Hence, a preliminary pattern classification is desirable to ease the knowledge discovery process. TwiChi categorizes the extracted HiGens based on their time-related trend in the sequence of timestamped relational collection. In particular, to better highlight the temporal evolution of the knowledge associated with the HiGen reference itemset, HiGens are classified as: (i) stable HiGens, i.e., HiGens that include generalized itemsets belonging to the same generalization level, (ii) monotonous HiGens, i.e., HiGens that include a sequence of generalized itemsets whose generalization level shows a monotonous trend, and (iii) oscillatory HiGens, i.e., HiGens that include a sequence of generalized itemsets whose generalization level shows a variable and non-monotonous trend.
Since a generalized itemset of level l may have several generalizations of level l + 1 and taxonomies may have unbalanced data item distributions, stable HiGens are further partitioned in: (i) strongly stable HiGens, i.e., stable HiGens, in which items contained in its generalized itemsets and belonging to same data attribute, are characterized by the same generalization level, and (ii) weakly stable HiGens, i.e., stable HiGens in which items contained in its generalized itemsets and belonging to the same attribute, may be characterized by different generalization levels.

Input: sequence of timestamped relational tweet collection $D = D_1, D_2, .., D_g$, minimum support threshold *minsup*, taxonomy Δ
Output: set of HIGENs HG

```
1: k = 1 // Candidate length
2: HG = HiGen set;
3: C_k = set of distinct k-itemsets in D
4: repeat
5:   for all c in C_K do
6:     scan all D_i in D and count the support of c in D_i
7:   end for
8:   L_k^i = itemsets c in C_k that satisfy minsup for any D_i
9:   HG = update HIGEN set(Lik, HG)
10:  l = 1 // Candidate generalization level
11:  Gen = generalized itemset container
12:  repeat
13:    for all c in C_k of level l do
14:      D_c^inf = D_i in D — c is infrequent in D_i
15:      if D_c^inf is empty then
16:        gen(c) = set of new generalizations of itemset c of level l+1
17:        gen(c) = taxonomy evaluation(c,Δ)
18:        for all gen in gen(c) do
19:          gen.desc = c
20:        end for
21:        Gen = Gen ∪ gen(c)
22:      end if
23:    end for
24:    if Gen is not empty then
25:      for all gen 2 Gen do
26:        scan all D_i in D^inf gen.desc and count the support of gen in D_i
27:        for all gen frequent in any D_i in D_gen.desc^inf do
28:          HG = update HIGEN set(gen, HG)
29:          if gen is frequent in all Di in D_gen.desc^inf then
30:            remove gen from Gen
31:          end if
32:        end for
33:      end for
34:      C_k = C_k ∪ Gen
35:    end if
36:    l = l + 1
37:  until Gen is empty
38:  k = k + 1
39:  C_{k+1} = candidate_generation(C_k)
40: until C_k is empty
41: return HG
```

Algorithm 4: The HiGen Miner algorithm

Collection	*ItemSet*	*Support(%)*
	Strongly Stable HiGen **Reference itemset:** *(Place, New York City), (Time, 3.45 p.m.)*	
$D_{Jan2012}$	(Place, New York City), (Time, 3.45 p.m.)	20%
$D_{Feb2012}$	(Place, New York City), (Time, 3.45 p.m.)	50%
$D_{Mar2012}$	(Place, New York City), (Time, 3.45 p.m.)	25%
	Weakly Stable HiGen **Reference itemset:** *(Place, New York City), (Time, 4.00 p.m.)*	
$D_{Jan2012}$	(Place, New York State), (Time, from 3 to 6 p.m.)	27%
$D_{Feb2012}$	(Place, New York City), (Time, from 3 to 6 p.m.)	21%
$D_{Mar2012}$	(Place, New York State), (Time, from 3 to 6 p.m.)	25%
	Monotonous HiGen **Reference itemset:** *(Place, New York City), (Time, 5.00 p.m.)*	
$D_{Jan2012}$	(Place, New York City), (Time, from 5.00 p.m.)	28%
$D_{Feb2012}$	(Place, New York City), (Time, from 3 to 6 p.m.)	25%
$D_{Mar2012}$	(Place, New York City), (Time, p.m.)	21%
	Oscillatory HiGen **Reference itemset:** *(Place, New York City), (Time, 6.00 p.m.)*	
$D_{Jan2012}$	(Place, New York City), (Time, from 6.00 p.m.)	20%
$D_{Feb2012}$	(Place, New York City), (Time, from 3 to 6 p.m.)	24%
$D_{Mar2012}$	(Place, New York City), (Time, from 6.00 p.m.)	21%

TABLE 5.2: HiGen examples. Minsup = 20%.

In Table 5.2 a HiGen example relative to each category is reported. HiGens have been extracted from an example sequence of tweet collections by enforcing a minimum support threshold equal to 20% and by exploiting the taxonomy reported in Figure 5.2. For each HiGen the corresponding reference itemset is also reported. Notice that the itemsets contained in the strongly and weakly stable HiGens are all characterized by the same generalization level (i.e., 1 and 2, respectively) while for the monotonous HiGen the level of the reported itemsets increases from 1 to 3 from January to March 2012. Finally, for the oscillatory HiGen the generalization level varies with a non-monotonous trend.
Examples of HiGens mined from a real-life Twitter dataset are reported in Section "Expert validation".

5.3 Experimental Results

In the previous sections, we introduced and thoroughly described the TwiChI framework. To assess the effectiveness of the devised approach, in this section we report and describe a set of experiments we performed on real datasets coming from Twitter.
All the experiments were performed on a 3.2 GHz Pentium IV system with 8 GB RAM, running Ubuntu 12.04.

5.3.1 Evaluated datasets and taxonomy

The TwiChi frameworks exploits a crawler to effectively access to Twitter's global stream of Tweet data. We monitored the public streams endpoint offered by the Twitter API, covering the time period from 2012-07-07 to 2012-07-23 and tracking a selection of keywords ranging over different topics (e.g., weather, finance, sport). The crawler establishes and maintains a continuous connection with the stream endpoint to collect and store the Twitter data. As described in Section Twitter crawler, the tweets are preprocessed to represent the data into the relational data format and extract the taxonomies over content and context features.

In our crawling session, we collected 5047 tweets over 13 consecutive days in the time period [07/07/2012, 23/07/2012] posted by 708 distinct users located in 101 different GPS coordinates. To build the taxonomy model over the tweet textual content, we used the semantic generalizations of 3-levels Wordnet hyponym (i.e., is-a-subtype-of). Similarly, over the spatial attribute, a geographical hierarchy, which aggregates single locations into larger regions (province, region, state, continent) was built as well. Since the tweets contain only the GPS coordinates from which tweet are posted, we mapped the coordinates to the nearest location (i.e., city). Finally, the twitting date and time are analyzed by the aggregation functions to derive a hierarchy over the corresponding attributes (i.e., time, day, period).

5.3.2 Characteristics of the mined patterns

TwiChi analyzes sequences of timestamped tweet collections to discover the most significant pattern changes. We analyzed the characteristics of the patterns generated by TwiChi by setting two different temporal configurations: the former configuration, denoted in the following as Configuration A, aggregates tweets relative to the 13 considered time periods as follows: [2012-07-07, 2012-07-12], [2012-07-13, 2012-07-17], [2012-07-18, 2012-07-23]. The latter configuration (Configuration B) aggregates tweets based on the following time periods: [2012-07-07, 2012-07-09], [2012-07-10, 2012-07-13], [2012-07-14, 2012-07-18], [2012-07-19, 2012-07-23].

Figure 5.3 reports the number of HiGens mined from the real-life collections by varying the minimum support threshold in the range [0.5%, 5%] and by setting Configurations A and B. The number of mined HiGens increases more than linearly when lowering the support threshold due to the combinatorial increase of the number of generated combinations. To have a deep insight into the achieved results, we also analyzed the per level distribution of the itemsets contained in the mined HiGens. When rather low support thresholds (e.g., 0.5%) are enforced, many HiGens (53%) exclusively contain level-1 (not generalized) itemsets representing the reference itemset in each considered time period. When increasing the support threshold, the reference itemset becomes infrequent in

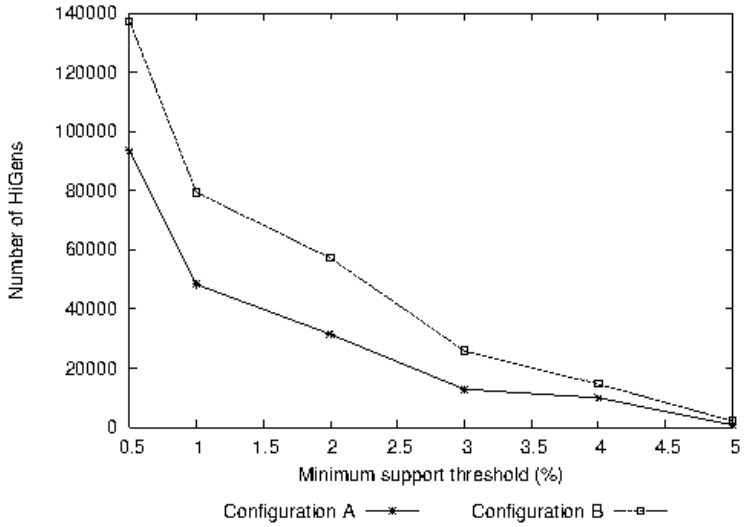


FIGURE 5.3: Number of mined HiGens.

Configuration	Minsup (%)	Number of Stable HiGens (%)			Number of monotonous HiGens (%) (%)	Number of oscillatory HiGens(%)
		Weak	Strong	Total		
A	0.15%	15	41	56	18	26
	1 %	23	14	37	27	36
	5 %	30	11	41	24	45
B	0.5%	13	46	59	11	30
	1%	27	9	36	21	43
	5%	35	8	43	13	44

TABLE 5.3: HiGen per category distribution.

some time periods. Hence, it is generalized by exploiting the given taxonomy and upper level itemsets are also included in the mined HiGens. For instance, at medium support thresholds (e.g., 1%) at least two out of three HiGens contain a generalized itemset and the percentage of level-2 itemsets contained in the mined HIGens is rather high (66%). When high support thresholds are enforced (e.g., 5%) most of the mined HiGen (78%) exclusively contain generalized itemsets and the number of itemsets with level higher than 2 becomes significant (39%). Notice that the high level information covered by the generalized itemsets is representative of the one associated with the low level reference itemset discarded due to the support threshold enforcement.

Since TwiChi classifies the extracted dynamic patterns based on their temporal trends (see Section "Pattern classification"), we also analyzed the per category distribution of the extracted HiGens. Table 5.3 reports the percentages of HiGens classified as strongly stable, weakly stable, monotonous, and oscillatory mined by enforcing three different support thresholds, i.e., 0.5%, 1%, and 5%.

When low support thresholds are enforced, the majority of the extracted patterns are stable because, in many cases, the knowledge covered by the reference itemset remains frequent in all the considered time periods. Differently, when medium and large support thresholds are enforced, the number of monotonous and oscillatory HiGens increases due to the higher selectivity of the support threshold. At high support thresholds (e.g., 5%) the number of stable HiGen still slightly increases because some of the extracted HiGens contain (possibly generalized) itemsets with the same level in all the considered time periods. The percentages of extracted monotonous and oscillatory HiGens are also affected by the number of considered time periods, as comes out from the comparison between Configuration A (3 time periods) and B (4).

5.3.3 Real-life use-case study

In this section, we present two real use-cases for the TwiChi system targeted to user behavior and topic trend analysis. Examples of the discovered HiGens are also given.

5.3.3.1 Weather forecasting service profiling

Consider an application scenario for the TwiChi system in which experts are interested in discovering peculiar user behaviors in order to shape service provisioning to the actual user interests and needs. Through the TwiChi system, analysts may automatically retrieve tweet collections posted by users coming from different cities in consecutive time periods and figure out the most relevant data correlation changes.
Consider, as an example, the real-life collections and taxonomy described in Section "Evaluated datasets and taxonomy". By setting the configuration A (see Section "Characteristics of the mined patterns") and the minimum support threshold to 1% the HiGens 1 and 2 reported in Table 5.3 are extracted. Users coming from Los Angeles (California, USA) frequently posted weather information during the analyzed time period. Hence, they may be likely to be interested in receiving automatic weather forecasting information. Similarly, people from Philadelphia frequently posted information about daily temperatures. The information may be deemed useful for profiling weather forecasting services to actual user needs. Notice that the interest about temperature information decreases in the second and third time periods. However, the weather topic, which is a generalization of the former one, remains of interest in the considered city.

5.3.3.2 Service shaping

Consider again the previous application scenario. Suppose that analysts are now interested in shaping the bandwidth of an online weather forecast service to improve the efficiency of the provided service. Analysts may focus on the HiGens that show a

Time period	*ItemSet*	*Support(%)*
	Strongly Stable HiGen 1 **Reference itemset:** *(Place, Los Angeles), (Word, Rain)*	
[07-07, 07-12]	(Place, Los Angeles), (Word, Rain)	1%
[07-13, 07-17]	(Place, Los Angeles), (Word, Rain)	1.3%
[07-18, 07-23]	(Place, Los Angeles), (Word, Rain)	1%
	Monotonous HiGen 2 **Reference itemset:** *(Place, Philadelphia), (Word, Temperature)*	
[07-07, 07-12]	(Place, Philadelphia), (Word, Temperature)	1.2%
[07-13, 07-17]	(Place, Philadelphia), (Word, Weather)	1.6%
[07-18, 07-23]	(Place, Philadelphia), (Word, Weather)	1%
	Monotonous HiGen 3 **Reference itemset:** *(Place, New York City), (Word, Weather)*	
[07-07, 07-12]	(Place, New York State), (Word, Weather)	1%
[07-13, 07-17]	(Place, USA), (Word, Weather)	2.1%
[07-18, 07-23]	(Place, USA), (Word, Weather)	1.8%

TABLE 5.4: HiGen selection. Configuration A. minsup = 1%.

monotonous or oscillatory trend to figure out which user groups, coming from specific cities or regions, are less used to request for weather forecasts.

Consider, for instance, the HiGen 3 reported in Table 5.4. It turns out that the interest in the weather service in the New York State becomes rather low in the second and third time periods. In fact, the location is generalized as USA, because the correlation with the New York State remains infrequent in the considered time periods. Indeed, the discovery of the reported HiGen may prompt service bandwidth reallocation in order to optimize resource usage.

Chapter 6

TUCAN: Twitter User Centric ANalyzer

In Chapter 4 and Chapter 5 we addressed the analysis of user-generated content from the Twitter micro-blogging Website. And, as we have discussed, most of previous works on Twitter focus on the analysis of "a community of twitters", whose tweets are analysed using text and data mining techniques to identify the topics, moods, or interests. [43, 51, 78–80].

In this Chapter we take a different angle: we focus on the analysis of a Twitter *target user*. We consider set of tweets that appear on his Twitter public page, i.e., the target user's timeline, and define a methodology to explore exposed content and extract possible valuable information. Which are the tweets that carry the most valuable information? Which are the topics he/she is interested into? How do this topics change over time? A further goal is to compare the Twitter activity of two (or more) target users. Do they share some common traits? Is there any shared interest? What is the most common interest of these two users, regardless of the time they are interested in it?

We propose a graphical framework which we term as TUCAN - Twitter User Centric ANalyzer. TUCAN highlights correlations among tweets using intuitive visualization, allowing exploration of the information exposed in them, thus enabling the extraction of valuable information from user's timeline. Given a number of limitations on the topic analysis of Twitter messages, such as limited length of messages, prevalent use of non-dictionary words (*i.e.*, abbreviations, mentions, hashtags, re-tweets, slang, and cultural words), and lack of contextual resources (*e.g.*, due to extensive use of Twitter for "private" purposes [81]), lots of ingenuity is required to automatically extract significant information out of tweets. From a methodology stand-point, we build upon text mining techniques, adapting them to cope with the specific Twitter characteristics.

As input, we group a user's tweets based on a window of time (e.g., a day, or a week) so to form *bird songs*, one for each time window. At the next step, filtering is applied to

each bird song using either simple stop-word removal, stemming, lemmatization, or more complicated transformations based on lexical databases. Next, terms in bird songs are scored using classic Term Frequency-Inverse Document Frequency (TF-IDF) [82] to pinpoint those terms that are particularly important for the target user. Each pair of birds songs are finally compared by computing a similarity score, so to unveil those bird songs that contain overlapping, and thus persistent, topics. The output is then represented using a coloured matrix, in which cell colour represents the similarity score. As a result, TUCAN offers a simple and natural visual representation of extracted information that easily unveils the most interesting bird songs and the persistent topics the target user is interested into during a given time period. Moreover, comparisons among bird songs give intuitions on the transition of user interests as well as the significance of topics to the user.

The framework is naturally extended to find and extract similarities among tweets of two or more target users. TUCAN computes and graphically shows the similarity among bird songs generated from the timelines of the pairs of target users, revealing similarities and common interests that are present possibly during different time periods.

TUCAN demonstrates to be useful to highlight correlation among tweets, which in turn proves very valuable in identifying topics of interest in the Twitter timeline of a user. This is very instrumental in generic individual profiling or surveillance applications, where the information hidden inside the target user's flow of tweets has to naturally emerge. TUCAN is also very powerful to compare individuals, to examine their timelines in parallel, hunting for similarities, pinpointing common interests, and observing changes, deviations, etc. For instance, comparing a well-known public profile timeline, e.g., President Barack Obama, against a generic target user would unveil if they share common political interests. Alternatively, two casual targets can be compared to see if some common trait/interest exist (possibly at different time), e.g., to evaluate the success of an Internet dating or marriage.

To demonstrate the effectiveness of TUCAN on real-world microblogs, we applied it to two month long history of 712 Twitter users. Results show that the correlation among tweets turns out to be a key point in the identification and analysis of twitter users over time; analyzing tweet messages of a politician, we were able to confirm that his topics and topic durations well matched with ongoing political events at the time. Comparing his tweets against tweets from the US government, a subset of topics that are in-line with the government's positions were picked up. Analysis on topic changes revealed transitions in users' social relationships.

6.1 Related work

The increasing availability of valuable information from microblogging platforms pushed the research community to investigate efforts for mining textual information from them.

Text topic extraction and modeling. A plurality of works ([81, 83–87]) is based on a well known topic modeling technique called, the Latent Dirichlet Allocation (LDA) [88]. [85] extends LDA to infer descriptions of entities (*e.g.*, authors) separately from their relationships. [81] incorporates supervision to LDA, leveraging hashtags of Twitter for topic labeling. Generalizing topic extraction to Tweets without hashtags, [86] directly applies LDA to individual sentence within each Tweet message.
To further enhance the performance of topic extraction from short and sparse messages, author-topic (AT) model was proposed [89, 90]. By creating topic mixture at the level of authors rather than individual documents, AT is claimed to obtain more stable set of topics than LDA. [80] conducts empirical comparisons of LDA, AT, and simple TF-IDF on aggregates of Tweet messages. The work discovers that the accuracy of the topic models are highly influenced by the length of the documents. It also finds that with long enough documents, the model based approaches become less effective compared to the baseline TF-IDF. Based on the observations, we design TUCAN to flexibly aggregate messages into bird songs. With effectively formed bird songs, TUCAN can provide powerful topic analysis even with generic TF-IDF.

Time-series analysis in microblogs. Many literatures on topic analysis ([51, 81, 86]) focus on detecting emergence of anomalous topics or prominent shifts on topic trends. Leveraging groups of semantically associated document tags, [51] discovers temporally emergent topics from Twitter data stream. [81] defines four types of Tweet categories and classifies streamed messages into them. Because these time-series analysis work on the entire group of users as a whole and do not distinguish single users, they cannot express topical relationships across individuals. We, on the other hand, focus on building dynamic relationships among the users. Aimed at similar goal, [91] proposes to detect topical relationships across entities over time. However, they only focus on time correlated co-occurring events. Instead, TUCAN aims to detect topic correlations even if they occur at different time frames. Recently, [92] proposed an interactive tool to analyse topic extracted from a stream of tweets organized in adjacent time slices of equal length. LDA is applied to mine topics and cosine similarity is leveraged to align them from the different time bins. Again, TUCAN is more user-centric and flexible enough to reveal topic correlations from time periods not strictly consecutive in time.

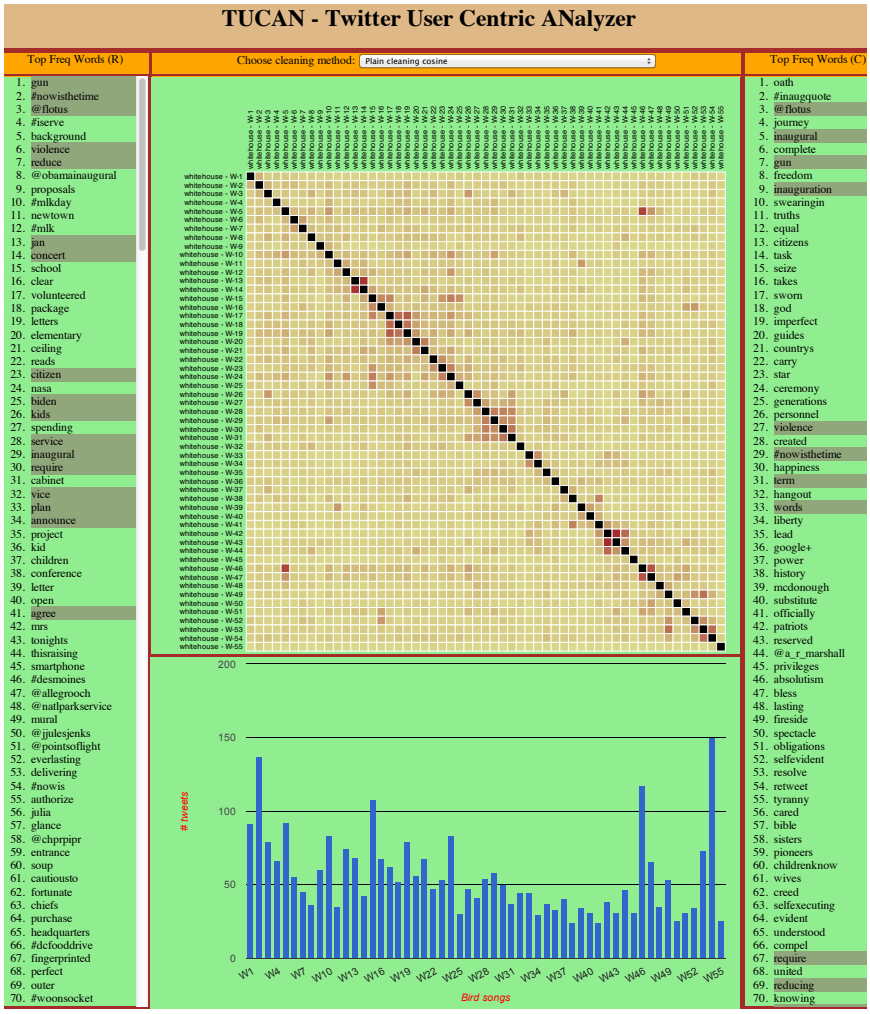
6.2 Framework

The TUCAN architecture includes three modules: (i) bird song generator, (ii) cross-correlation computation engine, and (iii) dashboard visualizer. A set of target Twitter users, *e.g.*, their screen names or user-ids, is provided to the system as an input. The system collects tweets related to such users on which various analytics are executed. Their outcome is visualized to enable the operator to gain knowledge about the users and the topics they are twitting about.

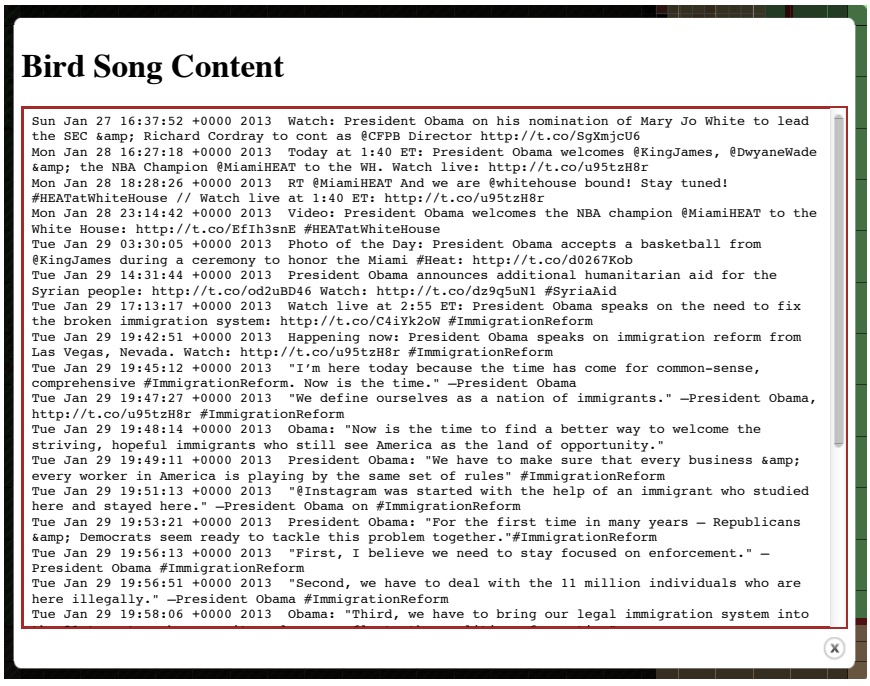
6.2.1 Bird song generation and cleaning process

Let $TW(u)$ be the set of tweets of a single user u that are retrieved from Twitter, time stamped with their generation time, stored and organized in a repository in binary format, to be easily accessed and further analyzed when necessary. Bird songs are created by aggregating tweets from $TW(u)$ generated within a time period T, to then be analyzed. We define the i-th *bird song* for the user u, $BS(u, i)$, as the subset of tweets in $TW(u)$ that appear in the i-th time period of duration T, i.e., the set of tweets that are generated in the $[(i-1)T, (i)T),\ i > 0$ window of time.

A "plain cleaning" pre-processing is applied to bird songs to discard stopwords, HTML tag entities, and links. Plain cleaning can be possibly substituted by more advanced text cleaning mechanisms; the following are also considered in this work: (i) removal of Twitter 'mentions', (ii) stemming, (iii) lemmatization, and (iv) ontology-based lexicon generalization. TUCAN allows the analyst to select the most appropriate cleaning method to take advantage of different effects of them in different contexts. Twitter mentions are words that begins with @ signs representing the mentioning of some named entities. The intuition behind removing the mentions comes from the fact that they do not provide insight in the topics being addressed, being just Twitter-ID of other users. Stemming and lemmatization are common text processing techniques aiming at reducing a word to its root form to lower sparseness present in a text document. The main difference between stemming and lemmatization is that the former is based on the heuristic of removing the trailing part of a word, while the latter brings a word to a canonical form based on a vocabulary and a morphological analysis the word. Here the Porter stemming algorithm [93] was deployed, while lemmatization is derived from the well-established Wordnet lexical database [94]. At last, our ontology-based lexicon generalization method leverages the Wordnet database to derive the most general concept for each word in the bird song. For instance, "gun" and "rifle" are replaced by the more generic term "weapon". The impact of the different cleaning methods will be exemplified by the experimental results presented in Section 6.3.



(a) TUCAN Main Interface



(b) Bird song detail

FIGURE 6.1: TUCAN Web Interface showing the analysis of the WhiteHouse official account. $T = 7$ days, plaing cleaning and Cosine similarity are considered.

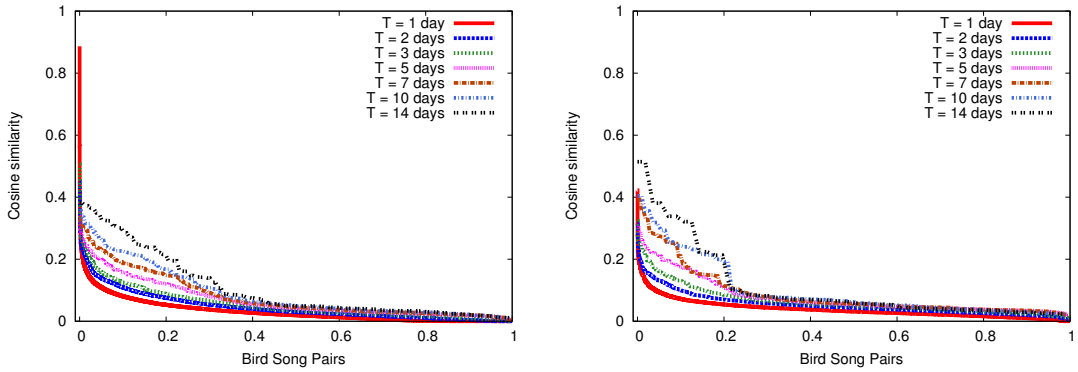
6.2.2 Cross-correlation computation

Each pre-processed bird song is tailored in a Bag-Of-Words (BoW) model, a common representation used in information retrieval and natural language processing. The bird song is tokenized in an unordered set of words, disregarding their sequence and position. Each word is then scored according to a weighting scheme. In this work, the Term Frequency-Inverse Document Frequency (TF-IDF) score is adopted as past literature has shown it to produce good results [80]. TF-IDF is computed as the product of the frequency of a term in its bird song and the inverse of the frequency of the term in the set of documents (i.e., all bird songs) being analyzed. TF-IDF provides a measure of the importance of a term in a specific bird song (first factor) put in perspective with how common the term is in the whole collection of bird songs. The intuition behind this weighting scheme is that, if a word appears in a huge number of bird songs in a given collection, its discriminative power is very low and is probably not useful to represent the content of the bird song, even if it often appears in it. Hence, words that are frequent in a bird song but rare in the collection are assigned with higher weights.
Bird songs are then transformed into a vector space model $VS(u, i)$, in which each word is given a fixed position. In this space, each word in the bird song $BS(u, i)$ is characterized by its TF-IDF score. Words that do not appear in $BS(u, i)$ are characterized by a null score.
To evaluate the similarity $VS(u, i) \otimes VS(v, j)$ among a pair of bird song vectors the Cosine similarity measure is applied.
Given any two term document vectors, the Cosine similarity is the cosine of the angle between them. The closer two vectors are to one other, the smaller the angle between them will be, i.e., the higher their similarity. Intuitively, the Cosine similarity of two very similar bird songs will be close to one. Instead, if no common words appear in two bird songs, their Cosine similarity will be 0.

6.2.3 Dashboard visualizer

In order to pinpoint similarities among bird songs, independently of the time the user posted them, TUCAN computes the similarity score for all possible pairs of bird songs. In total, N^2 similarity scores are computed and stored in a matrix form, where each cell represents $VS(u, i) \otimes VS(u, j), i, j \in [1, N]$. To help identifying correlation, the matrix is presented to the analyst in a graphical format using a web interface. Each cell is represented by a square whose color reflects the similarity score between the i-th and j-th bird songs. In particular, let

$$m = \max_{i,j,i \neq j} VS(u, i) \otimes VS(u, j),$$



(a) Barack Obama (b) User A

FIGURE 6.2: Effect of different time window sizes T. Plain cleaning and Cosine similarity.

cells are colored with different intensity, using a linear scale, so that the cell with similarity equal to m has the darkest color (see Figure 6.1(a) for an example). Bird songs are organized in increasing time window from left to right (and top to bottom).

As shown in Figure 6.1(a), when a cell is clicked, the web interface displays the top-ranked words appearing in $BS(u,i)$ and $BS(u,j), i \neq j$ on the left and right panes next to the matrix. Words that appear in both bird songs are highlighted. When clicking on the cells in the main diagonal (presented always in black[1]), the analyst is offered a popup showing the content of the original tweets of the i-th bird song. The GUI also shows a histogram below the matrix reporting $n(u,i)$ $\forall i$ to allow the analyst to easily gauging variations in the bird song size, e.g., due to the user changing his twitting habits during a holiday period. At the top of the matrix, the analyst is offered a drop-down menu to select the cleaning pre-processing to be applied.

6.3 Experimental results

Applying TUCAN to real world data from Twitter, we conducted an extensive study examining its capability on analyzing user centric topics. We begin by presenting a description on our dataset and how we collected it. Then we provide a series of sensitivity evaluation on various parameters of TUCAN, followed by a number of use cases with emphasis on different aspects of user centric topic analysis.

6.3.1 Dataset description

To perform user centric analysis through TUCAN, we monitor 712 randomly selected Twitter users for two or more months starting from the Summer 2012. The actual Tweet period covered for each user depends on the combination of the user's activity and crawling limitations imposed by Twitter API. Additionally, we monitor 28 well-known

[1]Note that by definition, $VS(u,i) \otimes VS(u,i) = 1$.

public figures, selected among politicians, news media, tech blogs, etc. In total, we collect 740 twitter timelines leveraging Twitter REST APIs[2]. Specifically, we access each user's public timeline and retrieve tweet STATUS objects which contains monograms (messages he puts on his page with no destined user), mentions of other users, conversations with follower/followees, and status updates.

From a total of 810,655 tweets, it emerges that 15% of them contain hashtags, 25% contain replies and 12% hyperlinks to other web pages. Similar proportions of message types are reported in the literature, suggesting our dataset presents no bias towards any particular types of tweets. About 300 users (40%) twitted more than twice in each week. Out of them, 20 users posted more than 400 tweets per week (i.e., more than 57 tweets/day). This already suggests that the window size parameter T has to be tailored to each user twitting habit when forming bird songs. Section 6.3.3 presents sensitivity tests on T.

6.3.2 The TUCAN GUI

Revisiting Figure 6.1, we present how TUCAN GUI is used for our analysis with an example of 56 week long history of official White House tweets. The reader can appreciate the correlation that TUCAN highlights among bird songs along the main diagonal. The darker areas indeed show that the correlation among top-words in bird songs is high, unveiling persistent topics. For instance, the top-words presented in the left and right lists easily allow to see the topics the White House was twitting about, i.e., violence and inauguration (Week-41). Those tweets refer to the second half of January 2013 during (i) the Inaugural Address by President Barack Obama, and (ii) the debate on violence and weapon possession started after the Newtown school tragedy. For reference, consider (part of) the tweets that form the bird song referring the 21st of January 2013 on Figure 6.1(b). Intuitively, extracting and summarizing information from the original tweets is much more complicated than by observing TUCAN output. Other areas of high correlation are clearly visible. Those refer to the Sandy hurricane, London Olympics games, etc. TUCAN allows to easily spot these major events that last for several weeks. Notice the Week-6/Week-46 dot with high similarity. Topics in those weeks refer to bills, insurance, gas price, and cost of education.

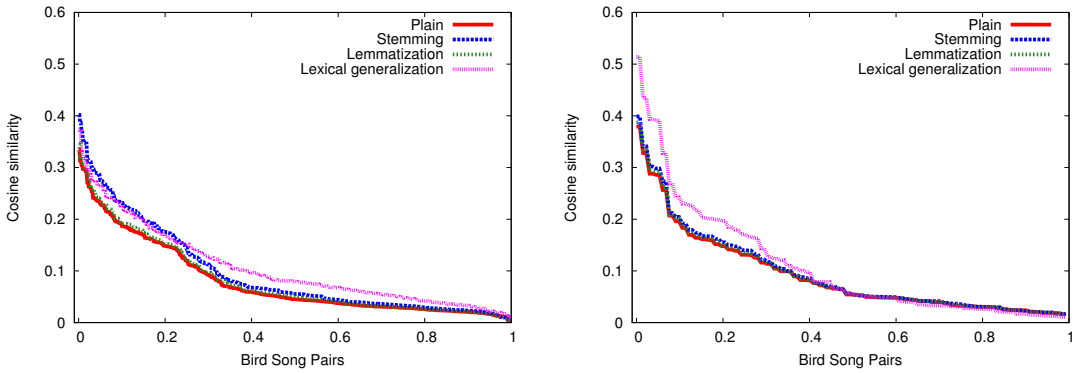
6.3.3 Parameter sensitivity analysis

We begin our analysis on TUCAN by showing effects of tuning different parameters: time window sizes, preprocessing methods, and inclusion of Twitter mentions. Results are presented showing, for all bird songs pairs of user u, the similarity score sorted in

[2]https://dev.twitter.com/docs/api

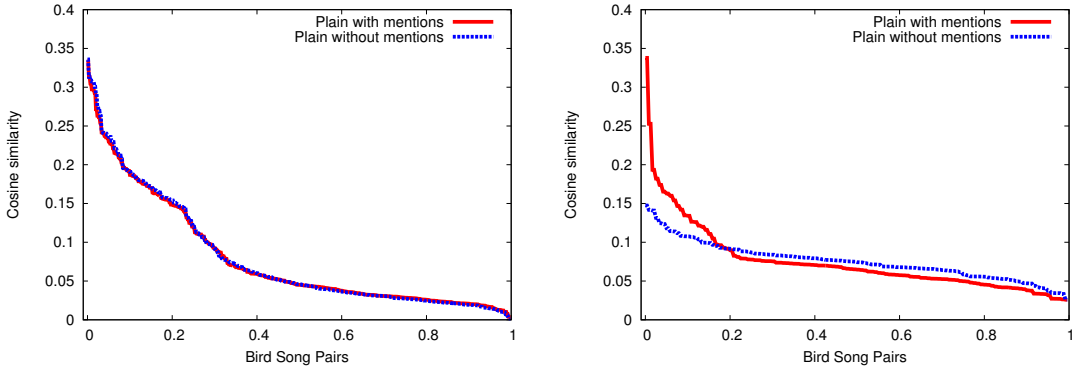
Rank	single Tweet	$T = 1$ day	$T = 7$ days	$T = 14$ days
1	photo	lead	#immigrationreform	#immigrationreform
2	day	international	**immigration**	**gun**
3	bo	@cfpb	**gun**	**immigration**
4	snow	cordray	**violence**	**violence**
5		mary	comprehensive	comprehensive
6		snow	@whlive	@whlive
7		nominates	broken	broken
8		sec	@vp	reform
9		richard	representative	representative
10		white	reform	@vp

TABLE 6.1: Top-words ranked by TF-IDF, Barack Obama.



(a) Barack Obama (b) User B

FIGURE 6.3: Effect of different cleaning methods. Cosine similarity and $T = 7$ days.



(a) Barack Obama (b) User C

FIGURE 6.4: Effect of mention removal. $T = 7$ days, plain cleaning, and Cosine similarity.

decreasing order. The X-axis displays the bird song rank normalized to the number of bird songs $N(u)$. The Y-axis shows absolute values of similarity score.

Effect of different time window sizes. The time window size T determines the size of bird song – a highly important parameter for topic models to perform optimally [80]. Figure 6.2 shows comparisons of time windows sizes for a public figure (Barack Obama, on the left) and a randomly chosen normal user (User A, on the right). As we vary

window size from $T = 1$ day to $T = 14$ days, we expect that the overall similarity scores become strictly higher. Indeed, Figure 6.2 clearly shows this; for instance, for Barack Obama, the average (max) score of $T = 1$ day is 0.03 (0.87), average score of $T = 14$ days is 0.11 (0.38). Same observation holds for normal users as shown in Figure 6.2(b). Notice that higher similarity score is not always welcome; a too large aggregation time window tends to create very large birds songs, in which similarity is artificially inflated, and the analysis blurred. As previously stated, T should be matched to twitting habits of the target.

On the other end, too short aggregation time window makes similarity interesting only on a small subset of bird song pairs, focusing the analysis on a too small groups of bird songs. Artifacts are also possibly created. For instance, notice the high similarity score at $x = 0$ in Figure 6.2(a) when $T = 1$ day. The reason for this outlier is that bird songs are formed by only a handful of terms; if three or four of those happened to co-occur in two bird songs, their similarity score turns out to be extremely high.

Further inspection on topic words also supports the importance of aggregation of tweets into bird songs. Table 6.1 shows up to ten top-words extracted from Barack Obama's bird songs (as previously mentioned). When tweets are used as they are (without aggregation), we not only observe that the number of common words are small, *i.e.*, the tweet has too few words to allow successful analysis; but we also observe that the relationship among the words are loose. Similarly, for $T = 1$ day, no clear topic emerge. In contrary, when $T = 7$ or $T = 14$ days, the top-words are much more coherent (especially between 'gun', 'violence', and 'broken') pinpointing to a clear topic.

In summary, both the general trend of small similarity, and possible existence of outliers suggest to use quite large time window for analysis. As observed in Table 6.1, and from other tests run on a large number of users, $T = 7$ days usually gives the same amount of meaningful keywords as larger window sizes (*e.g.*, $T = 14$ days). For that reason, from here on, we use $T = 7$ days unless otherwise noted. Once similarity has been pinpointed, the analyst can drill down by lowering T.

Effect of different pre-processing methods. Many researchers on information extraction have proposed different pre-processing methods to sanitize original documents. On the particular application to Twitter document analysis, however, no work identified the optimal method. Therefore, we evaluate the performance of three well-known sanitization methods – stemming, lemmatization, lexical generalization – applied on the top of plain cleaning. Figure 6.3 compares the cleaning methods considering one public and one normal Twitter users as before. For the profile of Barack Obama, Figure 6.3(a) suggests that stemming and lexical generalization work better than lemmatization and plain cleaning. However, the overall gap between the two groups of curves is less than 0.05 in similarity score. In the case of a normal user, Figure 6.3(b) shows that lexical generalization tend to perform better than other pre-processing methods (by about

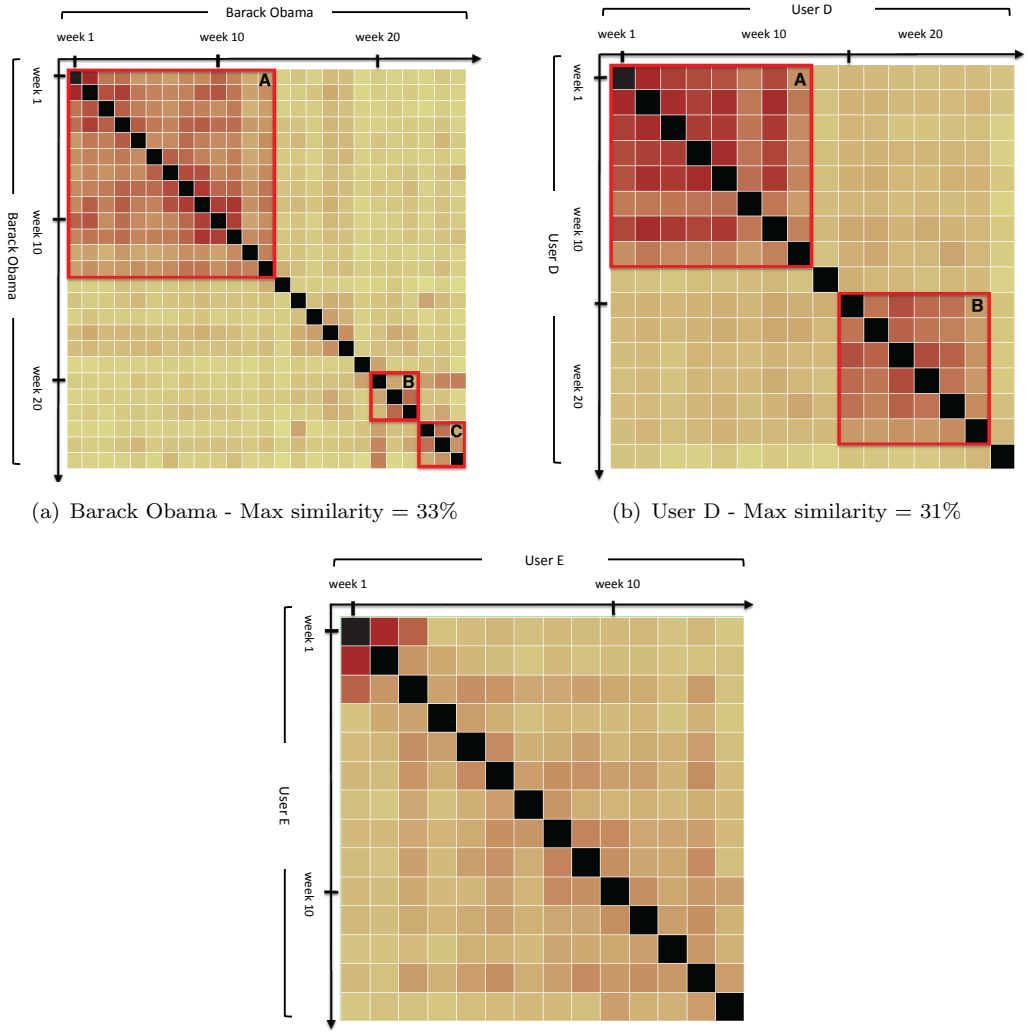
0.05). Notice that the increase in similarity is predominant for those pairs whose similarity is already quite large, and thus possibly less useful. By investigating further, we notice that lexical generalization tends (by definition) to return more general topics.
In summary, we observe small impact of the filtering process, and results are marginally affected by this choice. As such, TUCAN has been designed to offer the analyst the choice of the cleaning method that he consider the best for the case under analysis. Plain cleaning is the default choice.

Effect of including Twitter mentions. Among many specific mechanisms Twitter offers, "mentions" play an important role in the analysis of user conversations [95]. From our analysis, we noticed an interesting contrasts when mentions are included or excluded. As Figure 6.4(a) shows, for public figure's tweets (Barack Obama), results of including and not including mentions do not make much difference in similarity distributions. This is because of the usage of mentions by public profiles: either those are rarely used (e.g., in news media), or they are used to mention i) to lots of different users, or ii) to always the same group of users (this is the case for Barack Obama). However, for a normal user, as seen in Figure 6.4(b), proportion of mentions can get up to 70% and clearly makes distinction on the similarity distribution. The reason for similarity being higher when mentions are included is because the mentions themselves works as keywords (as in the case of '@whlive' or '@vp' from Table 6.1 that are however the Twitter profiles of White House Live and of the Vice President), resulting in (unnaturally) increased similarity scores. In Section 6.3.4, we will demonstrate cases where inclusion of mentions can indicate a particular pattern of a normal user's social relationship. Unless explicitly denoted, however, we include mentions in our analysis.

6.3.4 User centric analysis

To demonstrate the effectiveness of TUCAN on user analysis, we present results of case studies. Unless mentioned otherwise, we use the following settings by default: (i) windows size of 7 days, (ii) pre-processing with plain cleaning, and (iii) similarity scoring using Cosine similarity measure.

Analysis on timeline of a single user. Figure 6.5 shows correlation matrices representing similarities between pairs of bird songs of a single user. Figure 6.5(a) shows a matrix on the bird songs of *Barack Obama*. It highlights three blocks of highly correlated period of Tweets. The larger block [A] at the upper left corner represents Obama tweets during US presidential election in 2012. With a maximum Cosine similarity score of 0.33, it is clear that he has been tweeting a lot on a few correlated topics (voting, Romney, convention, health, etc. being among the most recurrent top terms). Block [B] refers to periods when Obama was interested in fiscal cliff. Finally, block [C] relates



(a) Barack Obama - Max similarity = 33%

(b) User D - Max similarity = 31%

(c) User E - Max similarity = 26%

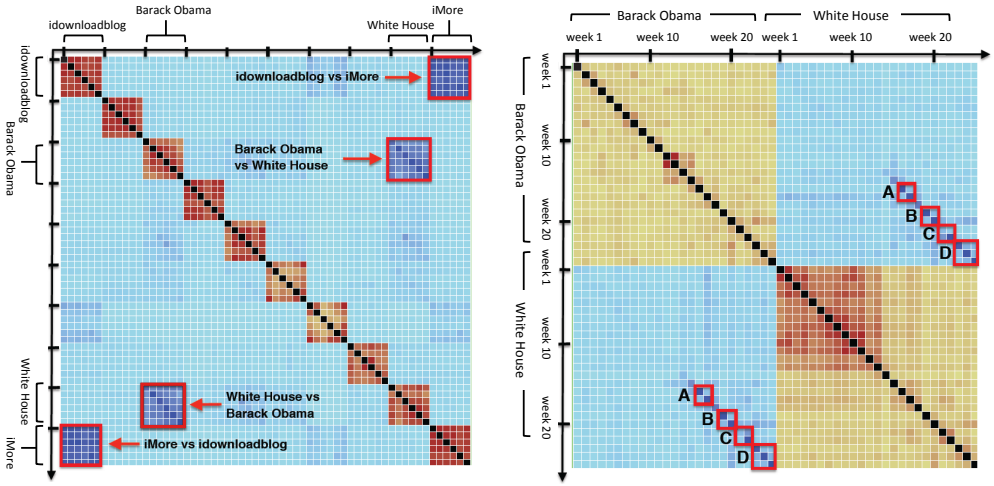
FIGURE 6.5: Similarity among bird songs for different type of users. $T = 7$ days, plain cleaning, Cosine similarity.

to the shooting in the Newtown elementary school, during which Obama's major topic terms were gun, violence, and weapon.

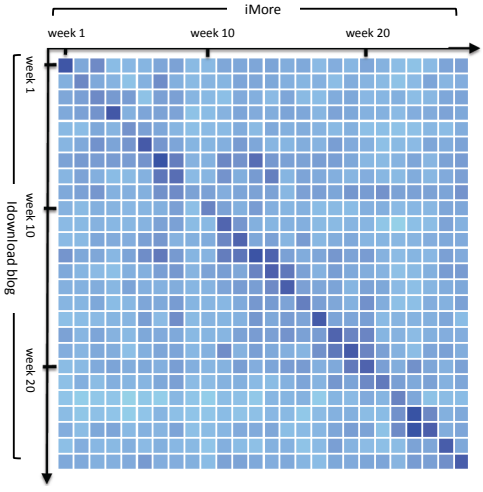
The correlation matrix in Figure 6.5(b) shows an interesting behavior of a normal "user D" (as opposed to a public figure or news media). As discussed in Section 6.3.3, mentions are very frequent among common users. Analyzing user D's bird songs without filtering out mentions, the plot highlights two blocks, [A] and [B]. The similarity of bird songs are dominated by the use of mentions to particular follower/followee of his. Investigating key terms in the time period of block [A], user D was exchanging messages with one of his follower. After one week of pause, in block [B], user D then mentions about another follower of his (and never refers to the follower in [A]). We suppose that user D's sudden change in his mentions indicates a change in his social relationship, e.g., change of his dating partner.

Lastly, Figure 6.5(c) shows a typical correlation matrix of generic "normal users". Compared to a public figure's correlation matrix (Figure 6.5(a)), the size of correlated blocks is small and more uniform. Likewise, the maximum similarity score is also lower at 0.26. This can be explained by different use of Twitter between public figures and normal users; public figures use Twitter to deliver messages with substantial topics ([79, 81]), whereas normal users use Twitter to socialize (with messages on status updates, social signals, messages indicating mood, etc.) as noted in [81].
Finally, TUCAN can also be instrumented to highlight artificial similarity among a user's tweet that were generated by automatic tools like Foursquare check-in, auto-tweet tools, etc. We do not report their examples for sake of brevity.

Analysis across different users. Besides the per-user analysis, TUCAN can infer semantic relationships across a multiple of users when applied to a group of target users. We select ten public figures and media blogs and report the cross-similarity matrix in Figure 6.6. The latest six bird songs with $T = 14$ days are considered, referring to a common period of time. Each bird song is checked against each other. Results are represented as a colored matrix, using different color scales (and normalization) for blocks outside the main diagonal and in the main diagonal (where same-user's bird songs are compared). Focusing on the former, two pairs of users emerge as mostly correlated: {*Barack Obama*, *White House*} and {*idownloadblog*, *iMore*}.
Zooming in and increasing the resolution by selecting $T = 7$ days, Figure 6.6(b) compares {*Barack Obama*, *White House*} in detail over 25 weeks of tweeting. First, notice that during Barack Obama's campaign (ref. Figure 6.5(a)) the correlation with White House is marginal. After elections, four periods of high correlations are pinpointed, highlighting the periods Barack Obama and White House publicize similar topics. The block [A] indicates the period of educational cost cut. [B] indicates the massacre at Newtown. [C] refers to fiscal cliff, and [D] on reformation of US immigration laws. The discovery of both well-correlated and non-correlated periods allows us to quantify periods of time the President spoke for himself (and his political party) and the government of the US. Similar consideration holds when zooming in {*idownloadblog*, *iMore*} comparison in Figure 6.6(c). Both users are blogs reporting news on Apple products. Also in this case $T = 7$ days, for 25 bird songs. Only the cross-similarity macro block is shown for the sake of brevity. Notice the large similarity in the main diagonal; it indicates that the two profiles report the same news, whose duration last for short period of time. The behavior is justified by the fact that both accounts work as sources of technology news.



(a) Famous users vs famous users. $T = 14$ days. (b) Barack Obama vs White House. $T = 7$ days.



(c) idownloadblog vs iMore. $T = 7$ days.

FIGURE 6.6: Similarity among users over different bird songs. Plain cleaning and Cosine similarity.

Chapter 7

Personalized Tag Recommendation Based on Generalized Rules

Recommender systems help users find desirable products or services by analyzing user profiles and their similarities, or by finding products that are similar to those the users expressed interest in. The diffusion of the collaborative tagging systems (e.g., Del.icio.us, Flickr, Zooomr) has recently focused the attention of the research community on the problem of tag recommendation. Tags are keywords that provide meaningful descriptors of a Web resources. Recommending tags to a user who is annotating a resource is a challenging research issue that has been recently investigated in different real-life contexts (e.g., photo annotation [96, 97], blog post tagging [98], bookmark tagging [50]). Given a set of user-defined tags, a relevant research issue is the recommendation of additional tags to partially annotated Web resources. Accomplishing this task effectively has the twofold aim at automating the annotation process by suggesting to the user an ordered set of pertinent tags and improving the effectiveness and the efficiency of querying retrieval systems (e.g., [99–101]). Recommendation of additional tags may be either exclusively based on collective knowledge, i.e., independently of the knowledge about the user who annotated the resources [50, 97, 102], or personalized [96, 98]. To figure out valuable correlations between previously annotated and recommendable tags rule-based approaches have shown to achieve fairly good performance against probabilistic and co-occurrence-based machine learning strategies [50]. To enhance the performance of the tag recommendation systems in the context of photo tag recommendation, the combined usage of user-specific and collective knowledge has also been recently addressed [103]. However, the lack of a controlled vocabulary from which tags could be selected during the annotation process makes the sets of previously assigned annotations very sparse [50, 97] and, thus, unsuitable for being successfully coped with most of the information retrieval

and data mining techniques.

This Chapter presents a novel rule-based recommendation system that addresses the task of recommending additional tags to partially annotated Flickr photos. It combines combines the knowledge provided by the personal and collective contexts, i.e, the history of the past personal and collective photo annotations. To address this issue, it discovers and exploits high level tag correlations, in the form of generalized association rules, from the collections of the past user annotations. To the best of our knowledge, this is the first attempt to exploit generalized rules in tag recommendation. Generalized association rules $X \rightarrow Y$ represent correlations among tag sets X and Y such that (i) frequently occur in the analyzed dataset, i.e., the observed frequency (the support) of $X \cup Y$ is above a given threshold, (ii) almost hold in the source data, i.e., the strength of the implication between X and Y (the confidence) is higher than a given threshold, and (iii) may also include items belonging to different abstraction levels (i.e., tags may be generalized as the corresponding categories). The use of tag generalization hierarchies allows the discovery of relevant tag associations that may remain hidden at the level of individual tags. Hence, it may effectively counteract the issue of data sparsity, thus, allowing the recommendation of meaningful and pertinent tags, as shown in the experimental evaluation (see Section 7.4).

7.1 Motivating example

In the following the use of generalized rules in tag recommendation is explained with the help of a running example.

Motivating example 1 Consider a photo, published on Flickr, of the Guildhall, which is a famous building situated in the center of London (U.K.). Our goal is to recommend to a given user pertinent additional photo tags to annotate, knowing that his first user-specified annotation is *London*. A graphical representation of the considered use-case is shown in Figure 7.1. To perform tag recommendation, we exclusively consider, as preliminary step, the collection of the past user-specified annotations (i.e., the personal knowledge base) while temporarily disregarding the collective knowledge provided by annotations made by the other system users. A traditional association rule mining process may discover the rule $\{London\} \rightarrow \{Guildhall\}$, where *London* and *Guildhall* are tags. Since the user has already annotated the photo with the tag *London*, *Guildhall* is an example of subsequent tag to recommend. The quality of the proposed recommendation could be evaluated in terms of well-known rule quality indexes (e.g., the rule support and confidence [104]). As discussed in [50], the analysis of the strength of the discovered implications is the core part of rule-based recommendation systems. In particular, frequent and high-confidence rules are deemed the most reliable ones for

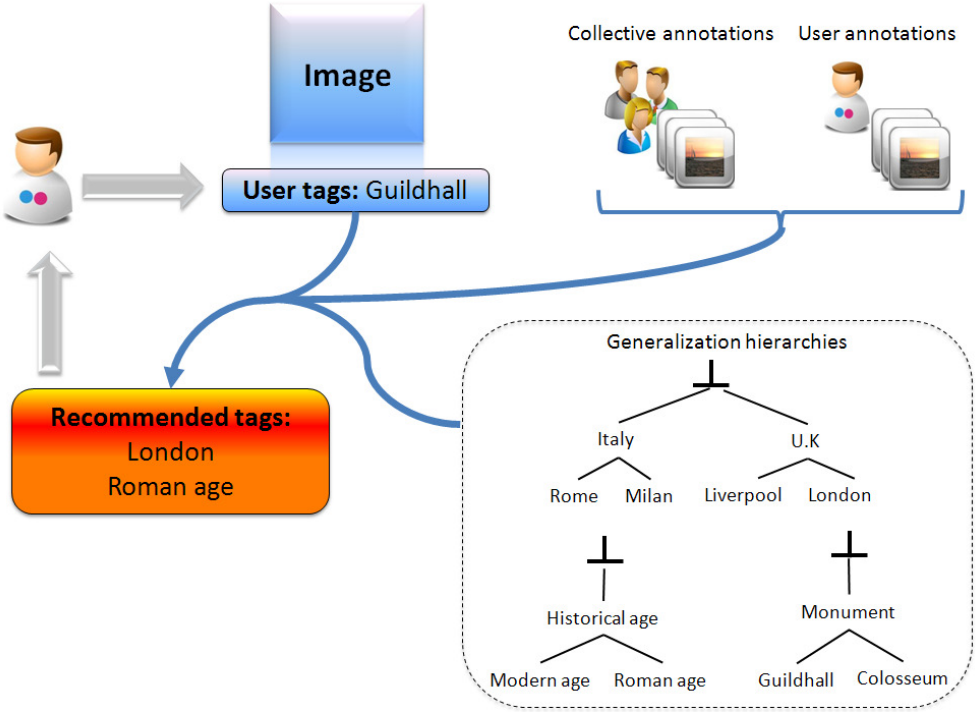


FIGURE 7.1: Example of use-case.

being used in tag recommendation. Enforcing a minimum frequency of occurrence of the selected rules reduces the sensitivity of the rule-based model to noise and data overfitting, However, data sparsity still makes the discovery of potentially relevant rules a computationally intensive task, because specific rules often occur rarely in the analyzed data [50, 102]. The use of generalization hierarchies built over the history tags, as the ones reported in Figure 7.1, may allow the generation of high level tag associations that occur more frequently than their low level versions. For instance, by aggregating the tag *London* into the corresponding state *U.K.* the generalized (high level) rule $\{U.K.\} \rightarrow \{Guildhall\}$ may prompt the suggestion of the same annotation while considering a higher level view of the analyzed pattern.

To discriminate among potentially pertinent tags, two distinct rule sets are generated: (i) a *user-specific rule set*, which represents the personalized knowledge base and includes (generalized) rules extracted from the past annotations made by the user to which the recommendation is targeted, and (ii) a *collective rule set*, which represents the collective knowledge and includes (generalized) rules mined from the past annotations made by the other users. Tags mainly referable to user-specific rules are deemed the most suitable ones for additional tag recommendation. However, their significance strictly depends on user activeness and ability in photo tagging [103]. To overcome this issue, in our system we consider tag recommendations based on collective knowledge as well. Collective knowledge also plays a key role in specializing high level associations discovered from the user-specific context, as shown in the following example.

Motivating example 2 Consider again the use-case shown in Figure 7.1. Suppose now that the first user-specified annotations are *London* and *Roman age*. If the rule $\{London, Roman\ age\} \rightarrow \{Monument\}$ is selected from the user-specified rule set, any descendant of *Monument* (e.g., *Colosseum*, *Guildhall*) is an eligible tag to recommend. The presence in the collective rule set of the rule $\{London, Roman\ age\} \rightarrow \{Guildhall\}$ may push the recommendation of the tag *Guildhall* as deemed worthy of notice by the community.

The effectiveness of the proposed system has been validated on real-life and benchmark photo collections retrieved from Flickr. The use of generalized rules allows significantly improving the performance of state-of-the-art approaches.

7.2 Related work

The success of social networks and online communities has relevantly increased the attention to the problem of recommending Web resource annotations, i.e., the tags. Tag recommendation systems focus on suggesting tags to a user who is annotating a resource by combining the information coming from one or more contexts. In particular, collective tag recommendation analyzes the knowledge provided by the past resource annotations independently of the user who annotated each resource [50, 97, 102], while personalized tag recommendation addresses tag recommendation by considering the user context [96, 98]. This chapter addresses tag recommendation by combining both personalized and collective knowledge.

A significant research effort has been devoted to personalized tag recommendation. For instance, in [98] the author presents a collaborative filtering method to address personalized blog post tag recommendation. It analyzes the information about users' behaviors, activities, or preferences to predict what users will like based on their similarity to other users. Analogously to most of the collaborative filtering methods (e.g., [105]), it assumes that similar users share similar tastes. Similarity between posts, users, and tags is evaluated by exploiting information retrieval techniques. In [106] the combination of a graph-based and collaborative filtering method is proposed. A User-Resource-Tag (URT) graph is indexed by means of an ad-hoc indexing strategy derived from the popular PageRank algorithm [107]. To reduce the sparsity of the generated graphs, the use of Singular Value Decomposition (SVD) methods has been also investigated [108]. Differently, the application of content-based strategies has been studied in [109–111]. They focus on recommending tags that are similar to those that a user annotated in the past (or is annotating in the present). For instance, in [110] the authors present an application for large scale automatic generation of personalized annotations. They automatically select from the main Web page keywords personalized tags based on their relevance to the content of both the considered page and the other documents residing

on the surfer's Desktop. Similarly, in [109, 112, 113] multimedia content related to the annotated Web resource is analyzed and used to drive the tag recommendation process. For instance, in [112, 113] the information discovered from both Web page content and related annotations is exploited for tag recommendation purposes, while, in [109], the authors analyze interpersonal relations, image text, and visual content together. Differently, in [111] an hybrid collaborative filtering method is proposed and integrated in a scalable architecture. The issue of interactive Flickr tag recommendation is addressed in [96]. Suggested tags are first selected from the set of previously assigned ones based on co-occurrence measures. Next, based on the recommendation, the candidate set is narrowed down to make the suggestion more specific. However, co-occurrence methods are challenged by data sparsity as either the computational complexity may increase exponentially with the number of tags or the score associated with each tag may be not directly comparable. Unlike previous approaches, to counteract the sparsity of the tag collections this chapter proposes to exploit generalized rules.

A parallel issue has been devoted to collective tag recommendation [50, 97, 102, 111]. For instance, in [97], additional tags are recommended to partially annotated Flickr photo by using co-occurrence measures to analyze the collective knowledge. The work proposed in [103] extends the previous system by analyzing the knowledge coming from different contextual layers, including the personal and the collective ones. Differently, authors in [50] reformulate the task of content-based tag recommendation as a (supervised) classification problem. Using page text, anchor text, surrounding hosts, and available tag information as training data, they train a classifier for each tag they want to predict. Even though their approach is able to achieve fairly high precision, the overall training time may become significant when the cardinality of the considered tags increases. This work is also the first attempt to address collective tag recommendation by means of association rules. Association rules allow the discovery of strong tag associations that may be profitably exploited in tag recommendation. Similarly, other approaches (e.g., [114]) focus on rule-based collective tag recommendation. However, the commonly high sparsity of the collections of past annotations limits the effectiveness of the proposed approaches as the most specific (and possibly interesting) rules may remain hidden. This chapter proposes to overcome the above issue by discovering tag associations at different abstraction levels. To the best of our knowledge, this is the first attempt to exploit generalized rules in tag recommendation. Authors in [102] also address the same issue by adopting an approach based on Latent Dirichlet Allocation (LDA). The proposed strategy is proved to very effective in tackling the cold start problem for tagging new resources for which no tag has been assigned yet. Differently, this chapter specifically addresses personalized tag recommendation of partially annotated resources.

In recent years, a notable research effort has been devoted to discovering generalized association rules from (possibly large) data collections. Generalized association rules have been first introduced in [62] in the context of market basket analysis as an extension of the traditional association rule mining task [104]. By evaluating a set of hierarchies of aggregation built over the data items, items belonging to the source data are aggregated based on different granularity concepts. Each generalized rule, which is a high level representation of a "lower level" rule, provides a higher level view of a pattern hidden in the analyzed data. The first generalized association rule mining algorithm [62] follows the traditional two-process for generalized rule mining: (i) frequent generalized itemset mining, driven by a minimum support threshold, and (ii) generalized rule generation, from the previously mined frequent itemsets, driven by a minimum confidence threshold. Candidate frequent itemsets are generated by exhaustively evaluating the generalization hierarchies. To reduce the complexity and improve the efficiency of the mining process, several optimizations strategies and more efficient algorithms have been proposed [62, 66, 68–70, 115, 116]. This chapter discovers and exploits generalized rules in personalized tag recommendation by adopting an Apriori-based strategy [62] that integrates, as itemset mining step, the approach recently proposed in [70].

7.3 The recommendation system

This chapter presents a novel personalized photo tag recommendation system. Given a photo and a set of user-defined tags, the system proposes novel pertinent tags to assign to the photo based on both the user-specific preferences (i.e., the tags already annotated by the same user to any photo) and the remaining part of collective knowledge (i.e., the annotations provided by the other users). Its main architectural blocks are shown in Figure 7.2. A brief description of each block follows.

Preprocessing. This block aims at making the collections of the previous tag annotations suitable for the generalized rule mining process. The tag set is tailored to a transactional data format, where each transaction corresponds to the annotations performed by a user to a given photo and includes the corresponding set of assigned tags. Over the history tag collection a set of generalization hierarchies is also derived from the established Wordnet lexical database [94].

Generalized association rule mining. This block focuses on discovering high level tag correlations, in the form of generalized association rules, from the transactional representation of the tag set. The available tag generalization hierarchies are also evaluated to discover tag correlations at different abstraction levels. Two distinct rule sets are generated: (i) a user-specific rule set, which includes generalized rules extracted from the

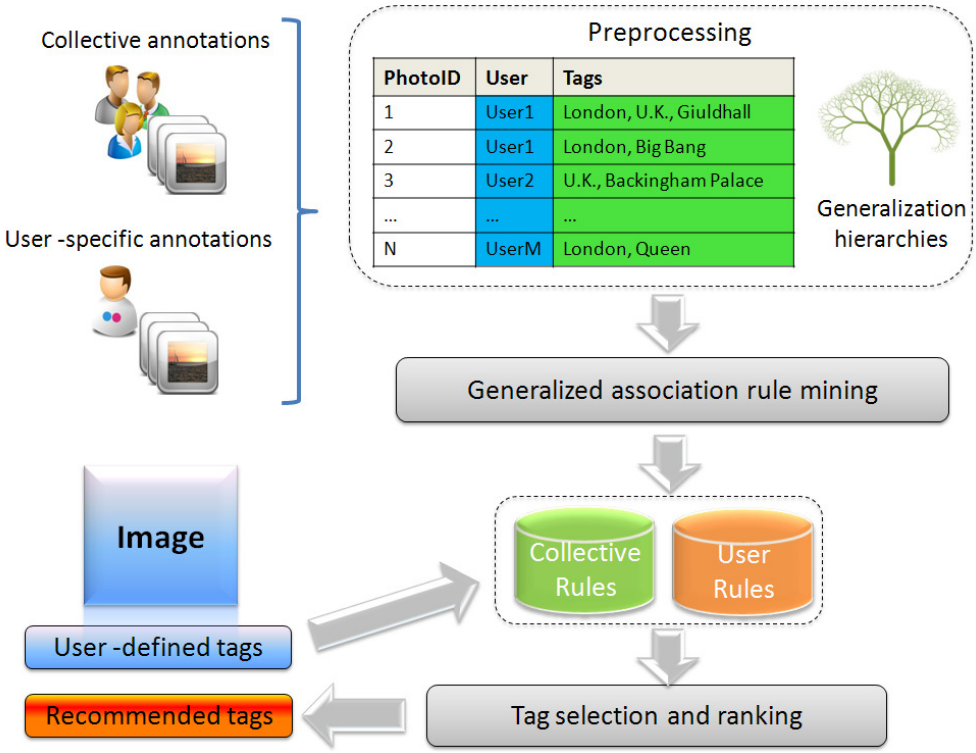


FIGURE 7.2: The recommendation system architecture

past annotations made by the user to which the recommendation is targeted, (ii) a collective rule set, which includes generalized rules mined from the past annotations made by the other users.

Tag selection and ranking. Given a photo and a set of tags already assigned by the user, this block aims at generating a ranked list of additional tags to suggest. To this aim, from the user-specific and collective rule sets generalized rules pertinent to the already assigned tags are selected. The ranked list of suggested tags is derived from the set of selected rules based on their main quality indexes.

7.3.1 Problem statement

Given a set of photos P, a set of tags T, and a set of users U the ternary relation $X = P \times T \times U$ represents the user assignments of tags in T to photos in P. The set $\tau(p_i,u_j) \subseteq T$ includes the tags assigned by user $u_j \in U$ to $p_i \in P$ and could be defined as follows:

$$\tau(p_i, u_j) = \pi_t \sigma_{p_i,u_j} X \tag{7.1}$$

where π and σ are the commonly used projection and selection primitive operators of the relational algebra [117].

To discriminate between past assignments made by the user u_j and collective ones (i.e., $\neg u_j$), the ternary relation X may be partitioned as follows:

$$X(u_j) = \pi_t \sigma_{u_j} X \tag{7.2}$$

$$X(\neg u_j) = \pi_t \sigma_{U \setminus u_j} X \tag{7.3}$$

We denote as user-specific and collective knowledge bases the sets $X(u_j)$ and $X(\neg u_j)$ such that $X(u_j) \cup X(\neg u_j)$=X. Given a set $\tau(p_i,u_j)$ of user-defined tags and the user-specific and collective knowledge bases $X(u_j)$ and $X(\neg u_j)$, the personalized tag recommendation task addressed by this work focuses on suggesting to user u_j new tags in $T \setminus \tau(p_i,u_j)$ for a photo p_i.

7.3.2 Preprocessing

Flickr is an online photo-sharing system whose resources are commonly annotated by the system users. The analysis of the past photo annotations is crucial for recommending novel tags to users who are annotating a photo. However, data retrieved from the Web is commonly unsuitable for being directly analyzed by means of data mining algorithms. Indeed, a preprocessing step is needed to tailor the retrieved tag sets to a suitable data format.

To enable the association rule mining process, the collection of past Flickr photo annotations is tailored to a transactional data format. A transactional dataset is a set of transactions, where each transaction is a set of items of arbitrary size. To map a tag set to a transactional data format, the annotations made by a user to a given photo are considered as a transaction composed of the set of (not repeated) assigned tags. A more formal definition of the transactional tag set is given in the following.

Definition 7.3.1. **Transactional tag set.** Let $X = P \times T \times U$ be the ternary relation representing the assignments of tags in T made by users in U to photos in P. Let $\tau(p_i,u_j) \subseteq T$ be the set of all (distinct) tags assigned by user $u_j \in U$ to $p_i \in P$. A transactional tag set $\mathcal{T}$ is a set of transactions, where each transaction corresponds to a set $\tau(p_i, u_j)$ for a certain combination of user $u_j \in U$ and photo $p_i \in P$ occurring in X.

For instance, if the user u_j assigns to the photo p_i the tags *Guildhall* and *London* the corresponding transaction is $\tau(p_i, u_j)$=$\{$*Guildhall, London*$\}$. The transactional tag set $\mathcal{T}$ including the set of all distinct $\tau(p_i, u_j)$ occurring in X is the full list of all past photo annotations.

Given a user u_j to which the personalized tag recommendation is targeted, the transactional tag set $\mathcal{T}$ is partitioned between the annotations made by u_j and not, i.e., distinct transactional representations of $X(u_j)$ and $X(\neg u_j)$, denoted as $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$

throughout the chapter, are generated. The separate analysis of $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$ allows the discovery of both user-specific and collective tag associations, in the form of generalized rules.
To enable the process of generalized rule mining from $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$, a set of hierarchies of aggregations (i.e., the generalization hierarchies) is built over the transaction tag set $\mathcal{T}$.

Definition 7.3.2. **Generalization hierarchy.** Let T be the set of tags occurring in the transactional tag set $\mathcal{T}$. A generalization hierarchy GH built over $\mathcal{T}$ is a predefined hierarchy of aggregations over T. The leaves of GH are all the tags in T. Each non-leaf node in GH is an aggregation of all its children. The root node (denoted as $\perp$) aggregates all the tags occurring in $\mathcal{T}$.

The Wordnet lexical database [94] is queried to retrieve the most relevant semantic relationships holding between a tag in T and any other term. More specifically, the following semantic relationships are considered: hyponyms (i.e., is-a-subtype-of relationships) and meronyms (is-part-of relationships). Terms to which any selected relationship is directed are considered as generalizations of the original tag. For instance, consider the example tag *London*. If the following semantic relationship is retrieved from the Wordnet database

$$<London>\ \textit{is-part-of}\ <U.K.>$$

then the term *London* is selected as the upper level generalization of the tag *U.K.*. Next, the database querying process is deepened to find possible upper level aggregations (e.g., $<U.K.>$ *is-part-of* $<Europe>$). The above procedure allows the construction of meaningful generalized hierarchies, according to Definition 7.3.2, built over a given transactional tag set. Extracts of some example generalization hierarchies are reported in Figure 7.1. The generalization hierarchies will be used to drive the generalized rule mining process, as described in the following.

7.3.3 Generalized association rule mining

This block focuses on discovering high level associations, in the form of generalized association rules, from the transactional tag sets $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$. Association rules represent significant correlations among the analyzed data [104]. More specifically, an association rule is an implication $A \Rightarrow B$, where A and B are itemsets, i.e., sets of data items. In the transactional representation of the tag set, items are tags in T associated with any photo included in the collection.
Generalized association rules [62] are rules that may include items at higher levels of abstraction, i.e., the generalized items. By considering the generalization hierarchies built over the transactional tag set (Cf. Definition 7.3.2), any concept that aggregates

one or more tags in T at a higher abstraction level is considered as a generalized item. For instance, consider again the semantic relationship <*London*> *is-part-of* <*U.K.*>. If *London* is a tag (item) that occurs in the transactional tag set, *U.K.* is an example of generalized item. Similarly, generalized itemsets are itemsets (tag sets) including at most one generalized item (e.g., {*Guildhall, U.K.*}). A more formal definition follows.

Definition 7.3.3. **Generalized itemset.** Let $\mathcal{T}$ be a transactional tag set and T the corresponding item domain, i.e., the set of tags occurring in $\mathcal{T}$. Let $\rho = \{GH_1, \ldots, GH_m\}$ be a set of generalization hierarchies built over $\mathcal{T}$ and E the set of generalized items (high level tag aggregations) derived by all the generalization hierarchies in ρ. A generalized itemset I is a subset of $T \bigcup E$ including at least one generalized item (high level tag aggregation) in E.

Generalized itemsets are characterized by a notable quality index, i.e., the support, which is defined in terms of the itemset coverage with respect to the analyzed data.

Definition 7.3.4. **Generalized itemset coverage.** Let $\mathcal{T}$ be a transactional tag set and ρ a set of generalization hierarchies. A (generalized) itemset I covers a given transaction $tr \in \mathcal{T}$ if all its (possibly generalized) items (tags) $x \in I$ are either included in tr or ancestors (generalizations) of items (tags) $i \in tr$ with respect to ρ.

The support of a (generalized) itemset I is given by the ratio between the number of transactions $tr \in T$ covered by I and the cardinality of $\mathcal{T}$.

A (generalized) itemset I is said to be a descendant of another generalized itemset Y if (i) I and Y have the same length and (ii) for each item $y \in Y$ there exists at least an item $i \in I$ that is a descendant of y.

The concept of generalized association rule extends traditional association rules to the case in which they may include either generalized or not generalized itemsets. A more formal definition follows.

Definition 7.3.5. **Generalized association rule.** Let A and B be two (generalized) itemsets. A generalized association rule is represented in the form $R : A \Rightarrow B$, where A and B are the body and the head of the rule respectively.

A and B are also denoted as antecedent and consequent of the generalized rule $A \Rightarrow B$. Generalized association rule extraction is commonly driven by rule support and confidence quality indexes. While the support index represents the observed frequency of occurrence of the rule in the transactional tag set, the confidence index represents the rule strength.

Definition 7.3.6. **Generalized association rule support.** Let $\mathcal{T}$ be a transactional tag set and ρ a set of generalization hierarchies. The support of a generalized rule $R : A \Rightarrow B$ is defined as the support (i.e., the observed frequency) of $A \cup B$ in $\mathcal{T}$.

Definition 7.3.7. **Generalized association rule confidence.** Let $\mathcal{T}$ be a transactional tag set and ρ a set of generalization hierarchies. The confidence of a rule $R : A \Rightarrow B$

is the conditional probability of occurrence in $\mathcal{T}$ of the generalized itemset B given the generalized itemset A.

For instance, the generalized association rule $\{U.K.\} \rightarrow \{Guildhall\}$ (s=10%,c=88%) states that the tag generalization *U.K.* co-occurs with the tag *Guildhall* in 10% of the transactions (annotations) of the collection and the implication holds in 88% of the cases.

To address generalized association rule mining task [62] from the tag history collections $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$, we performed the traditional two-step process: (i) generalized itemset mining, driven by a minimum support threshold *minsup* and (ii) generalized association rule generation, from the set of previously extracted itemsets, driven by a minimum confidence threshold *minconf*. A generalized association rule is said to be *strong* if it satisfies both *minsup* and *minconf*.

Given a set of generalization hierarchies built over the tags in X, a minimum support threshold *minsup*, and a minimum confidence threshold *minconf*, the generalized rule mining process is performed on $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$ separately. More specifically, given a photo p_i, a user u_j, and a set of user-specific tags $\tau(p_i,u_j)$, the main idea behind our approach is to treat strong high level correlations related to the annotations made by the user u_j differently from that made by the other users. To this aim, two distinct rule sets are generated: (i) a *user-specific rule set*, which includes all strong generalized rules extracted from the past annotations made by the user to which the recommendation is targeted, (ii) a *collective rule set*, which includes all strong generalized rules mined from the past annotations made by the other users. To accomplish the generalized itemset mining task efficiently and effectively, we exploit our implementation of a recently proposed mining algorithm, i.e., the GenIO algorithm [70]. A brief description of the adopted algorithm is given in Section 7.3.3.1.

7.3.3.1 The GenIO Algorithm

GenIO [70] is a generalized itemset mining algorithm that addresses the discovery of a smart subset of all the possible frequent (generalized) itemsets. Given a source dataset, a set of generalization hierarchies ρ, and a minimum support threshold *minsup* it discovers all frequent not generalized itemsets and all frequent generalized itemsets having at least an infrequent descendant, i.e., a descendant that does not satisfy *minsup*. To achieve this goal, the generalization process is support-driven, i.e., it generalizes an itemset only if it is infrequent with respect to the minimum support threshold. A more through description of the main algorithm steps follows.

GenIO is an Apriori-based algorithm [104] that performs a level-wise itemset generation. More specifically, at arbitrary iteration k, the Apriori-based itemset mining steps are the following: (i) candidate generation, in which all possible k-itemsets are generated

from the $(k-1)$-itemsets and (ii) candidate pruning, which is based on the property that all the subsets of frequent itemsets must also be frequent in the source data, to early discard candidate itemsets that cannot be frequent. Candidate generation is known to be the most computationally and memory intensive step The actual candidate support value is counted by performing a dataset scan. GenIO follows the same level-wise pattern. However, it manages rare itemsets by lazily evaluating the given generalization hierarchies. The generalization process is performed by applying on each item (tag) contained in an (infrequent) itemset I the corresponding generalization hierarchies. All itemsets obtained by replacing one or more items in I with their generalized versions are generalized itemsets of I. Hence, the generalization process on itemset I potentially generates a set of generalized itemsets. The generalization process of I is triggered if and only if I is infrequent with respect to the minimum support threshold. Since the GenIO algorithm has been first proposed in the context of structured datasets, a few straightforward modifications to the original algorithm have been adopted to make it applicable to transactional data as well.

7.3.3.2 Rule generation

The generalized rule generation task entails the discovery of all generalized association rules satisfying a minimum confidence threshold *minconf*, starting from the set of frequent (generalized) itemsets discovered by the GenIO algorithm.
The proposed recommendation system accomplishes the rule generation task by performing the second step of the traditional Apriori algorithm [104]. To achieve this goal, we exploited our more efficient implementation of the generalized rule generation procedure first proposed in [62].

7.3.4 Tag selection and ranking

Given a photo p_i, a set of user-defined tags $\tau(p_i,u_j)$ assigned by user u_j to p_i, and the sets of generalized rules $R_{\mathcal{T}(\sqcap_|)}$ and $R_{\mathcal{T}(\neg\sqcap_|)}$ mined, respectively, from $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$, this block entails the selection and the ranking of the additional tags to recommend to u_j for p_i. For the sake of clarity, in the following we discuss how to effectively tackle the selection and ranking problems separately.

7.3.4.1 Selection

The selection step focuses on selecting additional tags to suggest to user u_j for the partially annotated photo p_i from the rules belonging to the user-specific and the collective rule sets $R_{\mathcal{T}(\sqcap_|)}$ or $R_{\mathcal{T}(\neg\sqcap_|)}$. A pseudo-code of the selection procedure is given in Algorithm 5.

Require: the user-specific rule set $R_{\mathcal{T}(\sqcap_|)}$, the collective rule set $R_{\mathcal{T}(\neg\sqcap_|)}$, and the user-specified tags $\tau(p_i,u_j)$
Ensure: the tag selection C
1: covered_rules(u_j) = select_pertinent_user-specific_rules($R_{\mathcal{T}(\sqcap_|)}$, $\tau(p_i,u_j)$)
2: covered_rules($\neg u_j$) = select_pertinent_collective_rules($R_{\mathcal{T}(\neg\sqcap_|)}$, $\tau(p_i,u_j)$)
3: **for all** user-specific rules R in covered_rules(u_j) **do**
4: insert tags in $R.consequent$ into C
5: **for all** generalized tags g in C **do**
6: **for all** collective rules R_2 in covered_rules($\neg u_j$) **do**
7: **if** $R_2.consequent$ includes any tag t^* in $g.leafdescendant$ **then**
8: insert t^* in C
9: **end if**
10: **end for**
11: **end for**
12: **end for**
13: remove generalized tags from C
14: **return** C

Algorithm 5: Tag selection

To select tags that are strongly associated with the user-specified ones, only a subset of the extracted rules is deemed worth considering for additional tag recommendation. More specifically, the strong generalized rules in $R_{\mathcal{T}(\sqcap_|)}$ and $R_{\mathcal{T}(\neg\sqcap_|)}$ whose rule antecedent covers, at any level of abstraction, the user-specified tag set $\tau(p_i,u_j)$ or any of its subsets are selected and included in the corresponding rule sets *covered_rules*(u_j) and *covered_rules*($\neg u_j$) (see lines 1-2). According to Definition 7.3.4, the coverage of (a portion of) the tag set $\tau(p_i,u_j)$ may be due to the presence in the rule antecedent of either an exact matching (i.e., the same tags) or one of its generalized versions. Any rule that does not fulfill the above-mentioned constraint is not considered in subsequent analysis.

TABLE 7.1: Generalized rules used for recommending to user u_j tags subsequent to *Rome*.

ID	**Generalized rule**	**Support** (%)	**Confidence** (%)
Annotations made by user u_j			
1	$\{London\} \Rightarrow \{Guildhall\}$	2.5%	100%
2	$\{London\} \Rightarrow \{Historical\ age\}$	1.4%	85%
3	$\{U.K.\} \Rightarrow \{Royal\ family\}$	1.8%	91%
Annotations made by the other users			
4	$\{London\} \Rightarrow \{Guildhall, Royal\ family\}$	1.5%	95%
5	$\{U.K.\} \Rightarrow \{Roman\ Age\}$	1.3%	80%
6	$\{London\} \Rightarrow \{Tourism\}$	1.2%	72%

Consider, for instance, a photo p_i annotated by the user u_j with the tag *London*. In Table 7.1 is reported the selection of generalized rules taken from the set of rules mined from, respectively, the past user annotations $T(u_j)$ and $T(\neg u_j)$ by exploiting the generalization hierarchies reported in Figure 7.1 and by enforcing, respectively, a minimum support threshold equal to 1% and a minimum confidence threshold equal to 50%. Notice that any selected rule contains the tag *London* or its generalization *U.K.* as rule

antecedent. Consider now the case in which the set of user-specified tags $\tau(p_i,u_j)$ is {*London*, *Roman age*}. Rules including either {*London*, *Roman age*}, {*U.K.*, *Roman age*}, {*London*, *Historical age*}, or {*U.K.*, *Historical age*} as rule antecedent are considered as well together with that covering only one of the user-specified tags *London* or *Roman age* or their relative generalizations.

Not generalized tags belonging to the consequent of the selected user-specific or collective rules in $R_{\mathcal{T}(\sqcap_|)}$ or $R_{\mathcal{T}(\neg\sqcap_|)}$ are eligible tags to recommend. Since we consider the tag associations mainly referable to the user-specific context the most reliable ones for personalized tag recommendation, we first select the collection C of generalized and not generalized tags contained in the consequent of any rule in $R_{\mathcal{T}(\sqcap_|)}$ (line 4). Then, we refine the selection by replacing generalized tags with the most pertinent not generalized descendants derivable from the collective knowledge base (line 8).

Recalling the previous example, the set C of candidate tags is first initialized as follows: {*Guildhall*, *Historical age*, *Royal family*}. Readers could notice that *Guildhall* and *Royal family* are tags, while *Historical age* is an upper level generalization. Since the generalization *Historical age* could not directly recommended, it is replaced with one (or more) of its low level tags. The selection of the eligible descendants of any generalization in C is driven by the collective knowledge. For instance, since, among the two low level descendants of *Historical age* (i.e., the tags *Roman Age* and *Modern age*), only *Roman Age* occurs at least once in the consequent of any of the selected collective rules in $R_{\mathcal{T}(\neg\sqcap_|)}$ (see Table 7.1), the tag *Historical age* is exclusively replaced by its leaf descendant *Roman Age*, as it is strongly recommended by the community.

The selection procedure performs two nested loops. The outer loop (lines 5-11) iterates over the generalizations occurring in the candidate set C, while the inner one (lines 6-10) iterates over the collective rule sets and selects the leaf descendants of any generalization in C. While leaf descendants are included in C as pertinent additional tags to recommend (line 8), any generalization in C is discarded (line 13). Finally, the updated set C of selected candidate tags is returned (line 14).

7.3.4.2 Ranking

The last but not the least task in tag recommendation is the ranking of the candidate recommendable tags in C. Tag ranking should reflect (i) the tag significance with respect to the user-defined tags in $\tau(p_i,u_j)$, (ii) the tag relevance according to the past user-specific preferences, and (iii) the tag relevance based on the past collective knowledge related to other system users.

To evaluate the significance with respect to $\tau(p_i,u_j)$ we propose a tag ranking strategy that considers the interestingness of the rules in $R_{\mathcal{T}(\sqcap_|)}$ and $R_{\mathcal{T}(\neg\sqcap_|)}$ from which they have been selected. Generalized rule interestingness is evaluated in terms of its confidence

index value [104], i.e., the rule strength in the analyzed dataset (Cf. Definition 7.3.7) in both the personal and collective knowledge base.

Formally speaking, let $c \in C$ be an arbitrary candidate tag and $R^c_{\mathcal{T}(\sqcap_|)} \subseteq R_{\mathcal{T}(\sqcap_|)}$, $R^c_{\mathcal{T}(\neg\sqcap_|)} \subseteq R_{\mathcal{T}(\neg\sqcap_|)}$ be, respectively, the subsets of rules in $R_{\mathcal{T}(\sqcap_|)}$ and $R_{\mathcal{T}(\neg\sqcap_|)}$ whose antecedent covers c (at any level of abstraction). The ranking score of c in $\mathcal{T}(\sqcap_|)$ and $\mathcal{T}(\neg\sqcap_|)$ is defined as the average confidence of the rules in $R^c_{\mathcal{T}(\sqcap_|)}$ and $R^c_{\mathcal{T}(\neg\sqcap_|)}$, respectively.

$$rankscore(c, \mathcal{T}(\sqcap_|)) = \frac{\sum_{\nabla_{\sqcap_|} \in \mathcal{R}^{\rfloor}_{\mathcal{T}(\sqcap_|)}} \rfloor\wr\backslash\{(\nabla_{\sqcap_|})}{|\mathcal{R}^{\rfloor}_{\mathcal{T}(\sqcap_|)}|}$$

$$rankscore(c, \mathcal{T}(\neg\sqcap_|)) = \frac{\sum_{\nabla_{\neg\sqcap_|} \in \mathcal{R}^{\rfloor}_{\mathcal{T}(\neg\sqcap_|)}} \rfloor\wr\backslash\{(\nabla_{\sqcap_|})}{|\mathcal{R}^{\rfloor}_{\mathcal{T}(\neg\sqcap_|)}|}$$

Roughly speaking, the ranking scores $rankscore(c, \mathcal{T}(\sqcap_|))$ and $rankscore(c, \mathcal{T}(\neg\sqcap_|))$ reflect the average significance of the tag c in the personal and collective contexts. To combine the individual tag ranks achieved in different contexts in a unified ranking list we adopted an aggregation method based on the Borda Count group consensus function [118]. The chosen approach first assigns descending integer scores to the elements of each individual rank and then combines the voting scores to generate a unique ranking. To effectively deal with ranking lists of different lengths, in our Borda Count implementation we assign to the first element of each rank the same value equal to the length of the longest of all the input ranks.

The recommendation system returns the ranked list of candidate tags in C produced by the Borda Count method.

7.4 Experimental results

We performed a large set of experiments addressing the following issues: (i) a performance comparison between our system and a set of recently proposed methods, (ii) a discussion about the impact of the generalization process on the recommendation performance, (iii) an analysis of a real-life use-case for our system and the discovered generalized tag associations, and (iv) the analysis of the impact of the main system parameters on the recommendation performance.

7.4.1 Photo collections

To test recommendation system performance, we used a benchmark and a real-life dataset.

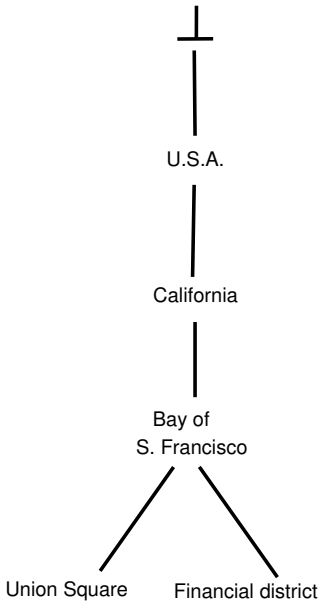


FIGURE 7.3: Portion of an example generalization hierarchy built over the photo collection tags

The used benchmark dataset is the MIR Flickr 2008 image collection, which was offered by the LIACS Medialab at Leiden University and introduced by the ACM MIR Committee in 2008 [119]. It collects 25,000 images and the related annotating users and tags.

The real-life collection is generated by retrieving, by means of the Flickr APIs, 5,000 real photos. The selected photos were chosen based on a series of high level geographical topics, i.e., *New York*, *San Francisco*, *London*, and *Vancouver*. The retrieved dataset is made available for research purposes[1].

Since for both the benchmark and the real-life datasets the majority (i.e, around 80%) of the contained photos have at least 5 tags, to perform a fair performance evaluation (see Section 7.4.2) we focus our analysis on this photo subset.

By following the strategy described in Section 7.3.2 a set of generalization hierarchies is derived from the Wordnet lexical database over the collected photo tags. A portion of one of the generated generalization hierarchies is reported in Figure 7.3.

7.4.2 Experimental design

Our system retrieves a ranked list of pertinent additional tags based on the extracted frequent generalized rules to tackle the tag recommendation ranking problem. Given a photo p_i and a set of user-defined tags $\tau(p_i,u_j)$, the system has to recommend tags that describe the photo based on both user-specific and collective past annotations. To

[1] http://dbdmg.polito.it/wordpress/research/recommendation-systems/

perform personalized recommendation, from both the tested photo collections the user-specific annotations made by 10 users who annotated at least 15 photos are considered separately. Once a user-specific annotation subset is selected, the rest of the collection is considered as the collective set. For each analyzed user collection, the evaluation process performs a hold-out train-test validation, i.e., the user-specific collection is partitioned in a training set, including the 75% of the whole annotations, whereas the remaining part is chosen as test set. To evaluate the additional tag recommendation performance of our system, for each test photo two random tags are selected as initial (user-specified) tag set and the recommended tag list is compared with the held-out test tags. A recommended tag is judged as correct if it is present in the held-out set. Since held-out tags need not to be the only tags that could be assigned to the photo, the evaluation method actually gives a lower bound of the system performance.

To evaluate the performance of both our recommendation system and its competitors, we exploited three standard information retrieval metrics, previously adopted in [97, 103] in the context of additional Flickr tag recommendation. The selected measures are deemed suitable for evaluating the system performance at different aspects. Let Q be the set of relevant tags, i.e. the tags really assigned by the user to the test photo, and C the tag set recommended by the system under evaluation. The adopted evaluation measures are defined as follows.

Mean Reciprocal Rank (MRR). This measure captures the ability of the system to return a relevant tag (i.e., a held-out tag) at the top of the ranking. The measure is averaged over all the photos in the testing collection and is computed by:

$$MRR = max_{q \in Q} \frac{1}{c_q} \tag{7.4}$$

where c_q is the rank achieved by the relevant tag q.

Success at rank k (S@k). This measure evaluates the probability of finding a relevant tag among the top-k recommended tags. It is averaged over all the test photos and is defined as follows:

$$S@k = \begin{cases} 1 & \text{if } Q \cap C_k \neq \emptyset, \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

where $q \in Q$ is a relevant tag and C_k is the set of the top-k recommended tags.

Precision at rank k (P@k). This metric evaluates the percentage of relevant tags over the set of retrieved ones. The measure, averaged over all test photos, is defined as follows:

$$P@k = \frac{|Q \cap C_k|}{|Q|} \tag{7.6}$$

Notice that the combined use of precision and success highlights the system ability to get a set of tags that is globally appreciable from the user's point of view, while MRR measures the quality of the top tag selection. To perform a fair evaluation, on each test photo measure estimates are averaged over several runs, where, within each run, a different (randomly generated) held-out tag set ranking is considered.

7.4.3 Performance comparison

The aim of this section is twofold. First, it experimentally demonstrates the effectiveness of our system against a state-of-the-art approach. Secondly, it evaluates the impact of the generalization process on the recommendation performance. To achieve these goals, we compared the performance of our system, in terms of the evaluation metrics described in Section 7.4.2, on both benchmark and real-life datasets with: (i) five different variants of the recently proposed personalized Flickr tag recommendation system [103], which specifically addresses the problem of additional photo tag recommendation given a set of user-specified tags, and (ii) a baseline version of our approach, which does not exploit generalized knowledge.

The system presented in [103] is a personalized recommender system that proposes additional photo tags, pertinent to a number of different user contexts, among which the personal and the collective ones. The system generates a list of recommendable tags based on a probabilistic co-occurrence measure for each context and then aggregates the results achieved within each context in a final recommended list by exploiting the Borda Count group consensus function [118]. To the best of our knowledge, it is the most recent work proposed on the topic of personalized additional Flickr tag recommendation. To perform a fair comparison, we evaluated the performance of the approach presented in [103] (denoted as *Probabilistic prediction* in the following) when coping with the combination of collective and personalized contexts. Moreover, within each context (personalized or collective), we tested different co-occurrence measures as well. More specifically, we also integrated and tested four co-occurrence measures, i.e., Sum, $Vote$, Sum^+ (Sum + Promotion), and $Vote^+$ (Vote + Promotion), previously proposed by the same authors in [97] in the context of collective additional tag recommendation. The additional measures are taken as representatives of different co-occurrence measures that could be adopted to aggregate and select tags pertinent to each context.

To demonstrate the usefulness of generalized rules in tag recommendation, we also compared the performance of our system with that of a baseline version, which exploits traditional (not generalized) association rules [104] solely. More specifically, the baseline method performs the same steps of the proposed approach, while disregarding the use of tag generalizations in discovering significant tag associations (see Section 7.3.4.1).

To test the performance of our approach we consider as standard configurations for the tested datasets the following settings: $minsup$=50% and $minconf$=40% for the real-life dataset and $minsup$=20% and $minconf$=35% for the benchmark dataset. A more detailed analysis of the impact of the above-mentioned parameters on the proposed recommendation performance is reported in Section 7.4.5. Even for the baseline version of our system we tested several support and confidence threshold values. For the sake of brevity, in the following we select as representative and report just the configuration that achieved the best results in terms of MMR measure (i.e., minimum support and confidence thresholds equal to 50%).

The overall results achieved by the performance evaluation session on the real-life and the benchmark datasets are summarized in Tables 7.2 and 7.3, respectively. They report the success and the precision at ranks from 1 to 5 (i.e., S@k, P@k k $\in$ [1,5]) as well as the Mean Reciprocal Rank (MRR) achieved by both our system and all the tested competitors. Similarly to what previously done in [97, 103], for the sake of brevity we choose not to report ranks with k higher than 5. To validate the statistical significance of the achieved performance improvements the Student t-test has been adopted [120] by using as p-value 0.05. Significant worsening in the comparisons between our system and the other tested competitors are starred in Tables 7.2 and 7.3. For each tested measure, the result(s) of the best system(s) is written in boldface.

Our recommendation system significantly outperforms both its baseline version and all the other tested competitors in terms of MRR, S@1, S@2, and P@k (for any tested value of k) on the real-life dataset and in terms of MRR, S@k, and P@k for $k > 1$ on the benchmark dataset. Furthermore, it performs as good as *Probabilistic prediction* [103], $Vote^+$, Sum, and Sum^+ in terms of S@k for $k \geq 3$ on the real-life dataset and as good as *Probabilistic prediction* in terms of P@1/S@1 on the benchmark dataset. Performance improvements in terms of P@k remain statistically significant for for any $k \leq 9$ on the real-life dataset, while in terms of P@k and S@k they are significant for any tested value of k in the range [2,10] on the benchmark dataset.

To have a deep insight into the achieved results, in Figures 7.4 and 7.5 we also plot the variations of the precision and the success at rank k by varying k in the range [1,5] for the real-life and the benchmark datasets, respectively. Results achieved on the real-life crawled data show that our approach performs best for any tested value of k in terms of precision at rank k (see Figure 7.4(b)). Furthermore, it also performs best for k equal to 1 and 2 in terms of success, while its performance is comparable to the one of the other approaches for $k \geq 3$. A slightly different performance trend comes out on the benchmark dataset. Our system is slightly less accurate than its best competitor in first tag prediction, while it performs significantly better than all the others (including *Probabilistic prediction*) in recommending all the subsequent tags.

TABLE 7.2: Real-life dataset. Performance comparison in terms of S@k, P@k, and MRR metrics. Statistically relevant worsening in the comparisons between our system and the other approaches are starred.

	Probabilistic Prediction	**Vote**	**Vote$^+$**	**Sum**	**Sum$^+$**	**Baseline**	**Generalized rule-based**
Precision at rank k							
P@1	0.6956*	0.5652*	0.6521*	0.6086*	0.6086*	0.6956*	**0.8044**
P@2	0.6195*	0.4782*	0.5543*	0.5760*	0.5543*	0.6630*	**0.7282**
P@3	0.5434*	0.4202*	0.5289*	0.5000*	0.5434*	0.6086*	**0.6667**
P@4	0.4619*	0.3858*	0.4619*	0.4891*	0.4782*	0.5434*	**0.6087**
P@5	0.4434*	0.3869*	0.4173*	0.4739*	0.4391*	0.4826*	**0.5304**
Success at rank k							
S@1	0.6956*	0.5652*	0.6521*	0.6086*	0.6086*	0.6956*	**0.8044**
S@2	0.8043*	0.7608*	0.8043*	0.7826*	0.8043*	0.7608*	**0.8478**
S@3	**0.8478**	0.7826*	0.8260	0.8043*	0.8260	0.7608*	**0.8478**
S@4	**0.8478**	0.8043	**0.8478**	**0.8478**	**0.8478**	0.7826*	**0.8478**
S@5	**0.8478**	0.8260	**0.8478**	**0.8478**	**0.8478**	0.7826*	**0.8478**
MRR							
	0.7681*	0.6837*	0.7429*	0.7159*	0.7219*	0.7337*	**0.8261**

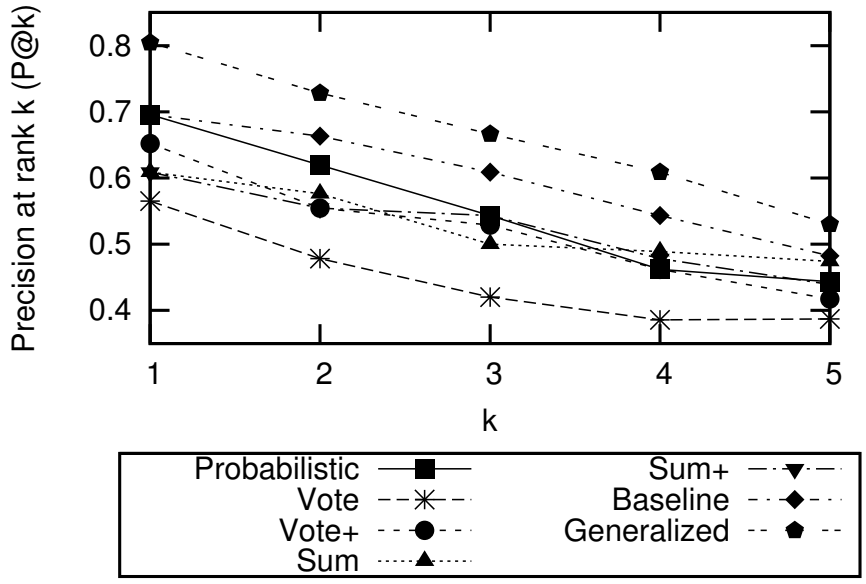
TABLE 7.3: Benchmark dataset. Performance comparison in terms of S@k, P@k, and MRR metrics. Statistically relevant worsening in the comparisons between our system and the other approaches are starred.

	Probabilistic Prediction	**Vote**	**Vote$^+$**	**Sum**	**Sum$^+$**	**Baseline**	**Generalized rule-based**
Precision at rank k							
P@1	**0.7660**	0.4468*	0.4894*	0.4681*	0.4894*	0.5319*	0.7447
P@2	0.6809	0.3936*	0.4255*	0.3936*	0.4255*	0.4787*	**0.6996**
P@3	0.6170*	0.3333*	0.3759*	0.3475*	0.3789*	0.4468*	**0.6383**
P@4	0.5638*	0.2979*	0.3298*	0.3032*	0.3298*	0.3989*	**0.5904**
P@5	0.4978*	0.2681*	0.2851*	0.2638*	0.2851*	0.3574*	**0.5191**
Success at rank k							
S@1	**0.7660**	0.4468*	0.4894*	0.4681*	0.4894*	0.5319*	0.7447
S@2	0.7872*	0.4894*	0.5106*	0.4894*	0.5106*	0.5957*	**0.8723**
S@3	0.8298*	0.5106*	0.5319*	0.5106*	0.5319*	0.5957*	**0.8936**
S@4	0.8298*	0.5106*	0.5319*	0.5106*	0.5319*	0.5957*	**0.8936**
S@5	0.8298*	0.5106*	0.5319*	0.5106*	0.5319*	0.5957*	**0.8936**
MRR							
	0.7908*	0.4752*	0.5071*	0.4858*	0.5071*	0.5638*	**0.8156**

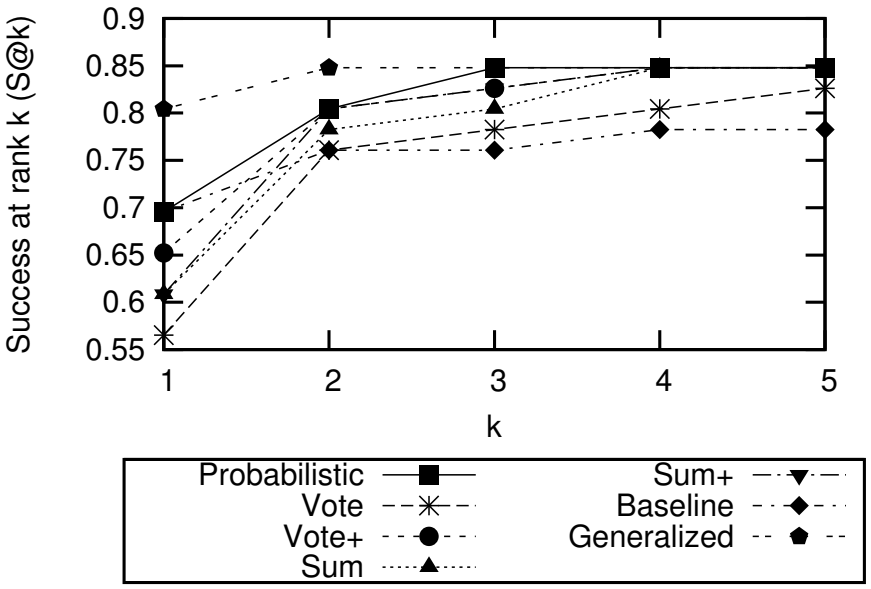
In summary, results show that our approach, on average, selects the most suitable recommendable tags at the top of the ranking and precisely identify the potential user interests.

7.4.4 Real-life use-case

In this section we analyze the results achieved by our system in a real-life use-case. Consider a user that is annotating a Flickr photo of the St. Mary Church, located at the Financial District of San Francisco (California, U.S.A.) nearby the Financial Center. The photo is taken from the real-life photo collection described in Section 7.4.1. Over the photo annotations a set of generalization hierarchies, whose extract is shown in Figure 7.3, is built by our recommendation system (see Section 7.3.2).

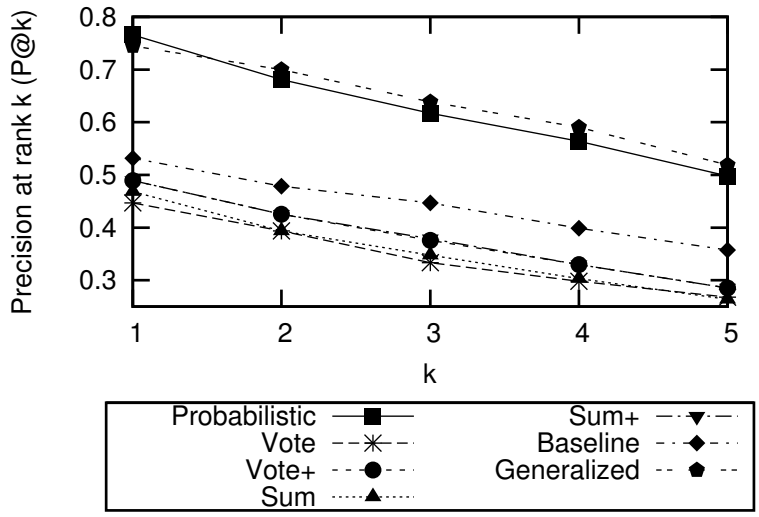


(a) Precision at rank k (P@k).

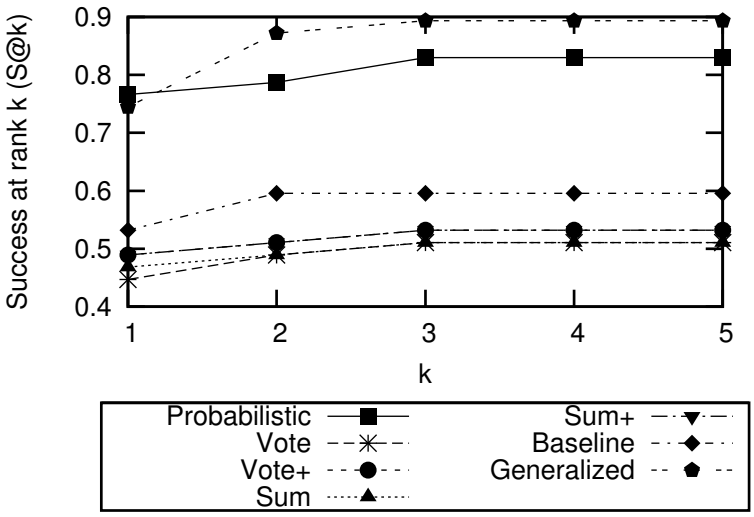


(b) Success at rank k (P@k).

FIGURE 7.4: Real-life dataset. Performance comparison by varying the reference rank k.



(a) Precision at rank k (P@k).



(b) Success at rank k (P@k).

FIGURE 7.5: Benchmark dataset. Performance comparison by varying the reference rank k.

The user is interested in tagging the photo with good descriptors so that the Flickr querying system may effectively retrieve its content based on the user-provided information. Suppose that the user has already annotated the photo with the following tags $\tau(p_i, u_j)$={*St. Mary Square, Financial District*}. The system analyzes the user-specific and collective knowledge bases to suggest additional tags to recommend. By setting the standard configuration (minimum support threshold $minsup$=50%, minimum confidence threshold $minconf$=40%) the following strong rule is discovered by our system from the collective transactional tag set:

1. {*St. Mary Square, Financial District*} $\Rightarrow$ {*Financial center*} (support = 40%, confidence = 100%).

Hence, *Financial center* is a candidate additional tag to recommend suggested by the community. However, due to the sparsity of the user-specific knowledge base none of the not generalized rules includes {*St. Mary Square, Financial District*} as rule antecedent since the combination of the two tags rarely occurs in the analyzed collection. Nevertheless, the following strong generalized rules are extracted:

2. {*San Francisco Bay, Financial District*} ⇒ {*St. Mary*} (support = 42%, confidence = 99%)

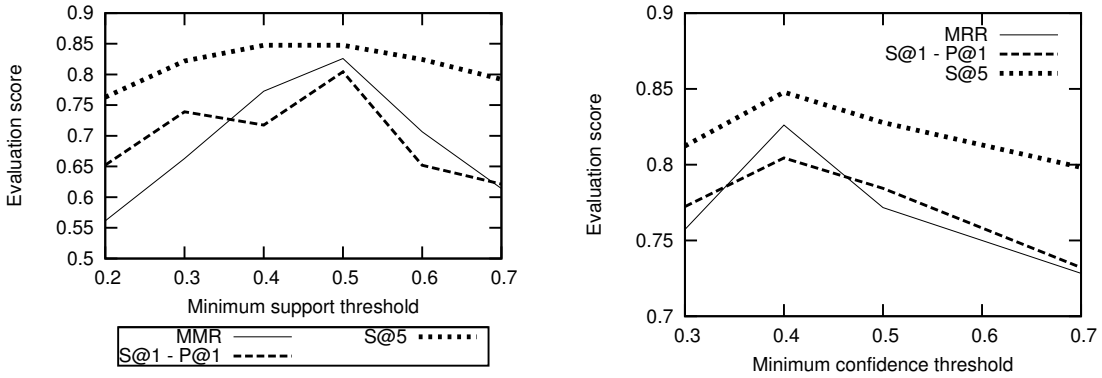
3. {*San Francisco Bay, Financial District*} ⇒ {*Business*} (support = 55%, confidence = 100%)

Both rules represent correlations between the previously assigned and the potentially relevant future tag annotations at a higher abstraction level. Rule (A) suggests the recommendation of the pertinent tag *St. Mary* as additional tag, while rule (B) highlights a high level tag category that is worth considering in the recommendation process. In particular, the latter rule states that, among the past user annotations, a correlation between the category *Business* and the previously annotated tags holds. Indeed, the user would willingly annotate the photo with a tag belonging to that category. The knowledge about the community behavior addresses the system to recommend the tag *Financial center* as it is a lower level descendant of the category *Business*.

7.4.5 Parameter analysis

We also analyzed the impact of the main system parameters on the tag recommendation performance. To this aim, in Figures 7.6(a) and 7.6(b) we plot the average MRR, S@1/P@1, and P@5 measures, as representatives among all the tested measures (see Section 7.4.2), achieved by our system on the real-life collection by varying the minimum support and confidence threshold enforced during the generalized rule mining process, respectively. Curves, not reported here for the sake of brevity, relative to different evaluation measures and dataset show similar trends.

When relatively high support thresholds (e.g., 70%) are enforced, the percentage of not generalized rules is quite limited (e.g., 13% of the user-specific rule set mined from the training photo collection described in Section 7.4.1) and many informative rules (generalized and not) are discarded. Nevertheless, the use of generalizations may prevent the discarding of the most informative recurrences thanks to the extraction of high level associations from the user-specific knowledge base. In the opposite case, i.e., when relatively low support thresholds (e.g., 20%) are enforced, many low level tag associations become frequent (e.g., 1.0% of the user-specific rule set from the same training data) and, thus, are extracted by our system. However, the sparsity of the analyzed tag collections still left some of the most peculiar associations among tags hidden. Aggregating tags



(a) Impact of the support threshold. $minconf$=40%. (b) Impact of the confidence threshold. $minsup$=50%

FIGURE 7.6: Parameter analysis. MRR, S@1/P@1, and P@5 measures.

into high level categories allows achieving the best balancing between specialization and generalization of the discovered associations and, thus, improves recommender system performance.

The confidence threshold may slightly affect the recommendation system performance. By enforcing very low confidence threshold values (e.g., 30%), a large amount of (possibly misleading) low-confidence rules is selected. Indeed, the quality of the rule-based model, at the top of which the recommendation system is built, worsens. Differently, when increasing the confidence threshold a more selective pruning of the low quality rules may allow enhancing the recommender system performance. As an extreme case, when enforcing very high confidence thresholds (e.g., 90%), rule pruning selectivity becomes too high to generate a considerable amount of interesting patterns.

Best values of support and confidence threshold actually depend on the analyzed data distribution. For instance, when coping with the benchmark dataset the best minimum support threshold values are around 20%, because the analyzed dataset is relatively sparse.

Success at rank k (e.g., see S@5 in Figures 7.6(a) and 7.6(b)) is shown to be, on average, less affected by support and confidence thresholds than precision at rank k, because the probability of finding a relevant tag in the top-k recommended tags is more weakly influenced by the rule-based model quality than the percentage of retrieved relevant tags.

Chapter 8

Misleading Generalized Itemset Discovery

Generalized itemset mining [62] is an established data mining technique that focuses on discovering knowledge hidden in the analyzed data at different abstraction levels. By exploiting a taxonomy (i.e. a set of is-a hierarchies built over the analyzed data) the mining process entails discovering patterns, i.e. the frequent generalized itemsets, that (i) have a frequency of occurrence (support) in the analyzed data higher than or equal to a given threshold and (ii) can include items at any level of abstraction. Low-level itemsets represent rather specific and detailed data correlations for which the corresponding support is unlikely to exceed the given threshold. On the other hand, high-level (generalized) itemsets provide a high-level view of the underlying data correlations. Hence, they could represent, at a high granularity level, the knowledge that remains hidden at a lower abstraction level. The interestingness of an itemset is commonly measured in terms of the strength of the correlation between its items [121–123]. To evaluate itemset correlation, in this Chapter we exploit an established correlation measure, i.e. the Kulczynsky (Kulc) correlation measure [124]. This measure has recently been adopted to perform high-level itemset correlation analysis [125]. Itemset correlation values are usually clustered in three different correlation types. Specifically, if an itemset X occurs less than expected in the analyzed data (i.e. the item correlation value is between 0 and a given threshold *max_neg_cor*) then X is said to be *negatively correlated*; if it occurs more than expected (i.e. the item correlation value is above a given threshold *min_pos_cor*) then X shows a *positive correlation*, otherwise (i.e. whenever there is neither a positive nor a negative item correlation) X is said to be *not correlated*. Unfortunately, to support domain experts in making decisions not all of the mined high-level patterns can be trusted. Indeed, some misleading high-level itemsets could be included in the mining result. A generalized itemset X is, to some extent, misleading if (some of) the low-level X's descendants have a correlation type in contrast to those of X.

For example, let us consider the structured dataset that is reported in Table 8.1. Each record contains the record identifier (rid), the city, and the product description. The itemset mining process can be driven by the taxonomy in Figure 8.1, which generalizes cities and products as the corresponding nations and product categories. Table 8.2 reports the set of frequent generalized itemsets that are mined by enforcing a support threshold min_sup=1 and two correlation thresholds max_neg_cor=0.65 and min_neg_cor=0.8. The frequent generalized itemset X={(Product, Wearing), (City, Italy)} has a positive correlation type, whereas its frequent low-level descendant itemset Y={(Product, T-shirt),(City, Rome)} is negatively correlated (see Table 8.2). To estimate the extent to which X is misleading we evaluate the percentage of dataset records that are covered by both X and any of its contrasting low-level correlations. For example, the record with rid 3 is covered by both X and Y. In other words, 25% of the records that are covered by {(Product, Wearing), (City, Italy)} are in common with those covered by {(Product, T-shirt),(City, Rome)}.

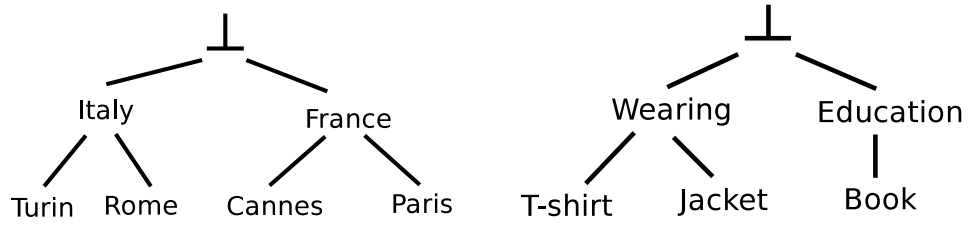
In this Chapter we propose: (i) a novel generalized itemset type, namely the Misleading Generalized Itemset (MGI); (ii) a MGI quality measure called Not Overlapping Degree (NOD) which indicates the extent to which the high-level pattern is misleading compared to its low-level descendants; and (iii) an approach to discovering a worthwhile subset of MGIs with NOD less than or equal to a maximum threshold max_NOD. Specifically, each MGI, hereafter denoted as $X \triangleright \mathcal{E}$, represents a frequent generalized itemset X and its set $\mathcal{E}$ of low-level frequent descendants for which the correlation type is in contrast to those of X. Experts need to analyze the misleading high-level data correlations separately from the traditional generalized itemsets and exploit such knowledge by making different decisions. To make this analysis possible, MGIs are extracted only if the low-level descendant itemsets that represent contrasting correlations cover almost the same portion of data as the high-level (misleading) ancestor X, i.e only if X represents a "clearly misleading" pattern. To do so, a maximum NOD constraint is enforced during the MGI mining process. Hence, unlike previous approaches (e.g. [122, 125]), we evaluate the degree of overlapping between the sets of records that are covered by a generalized itemset and its low-level (descendant) contrasting correlations. An algorithm to mine MGIs at the top of traditional generalized itemsets is also proposed.

The effectiveness of the proposed approach and the usability of the discovered patterns for supporting domain expert decisions are demonstrated by experiments performed on real-life data coming from two mobile applications and the UCI data repository [126]. Furthermore, the scalability of the algorithm has also been evaluated on synthetic datasets.

TABLE 8.1: Example dataset $\mathcal{D}$.

Id	City	Product
1	Turin	T-shirt
2	Turin	T-shirt
3	Rome	T-shirt
4	Paris	Jacket
5	Paris	Jacket
6	Cannes	Book
7	Turin	T-shirt

FIGURE 8.1: Example taxonomy built on $\mathcal{D}$'s attributes



(a) Aggregation tree $AT_{location}$ defined on the Location attribute

(b) Aggregation tree $AT_{product}$ defined on the Product attribute

8.1 Related work

The generalized itemset and association rule mining problem was first introduced in [62] in the context of market basket analysis. The authors proposed an Apriori-based algorithm [65] to discover frequent itemsets and association rules at different abstraction levels from datasets that were supplied with taxonomies. However, since the mining process evaluates the input taxonomy exhaustively a large number of (possibly redundant) item combinations is generated. A step beyond towards the generation of a more compact and humanly manageable pattern set has been made in [69, 116, 127, 128]. The proposed approaches enforce mining constraints to discover a worthwhile subset of frequent generalized itemsets or association rules. For example, in [116] the authors propose to push boolean constraints, which enforce the presence or the absence of an arbitrary item combination, into the mining process. In [69] the authors also take subset-superset and parent-child taxonomic relationships into account to avoid generating all the item combinations. More recently, an important research effort has also been devoted to discovering closed and maximal generalized itemsets [127, 128], which represent notable itemset subsets [129]. The authors in [70] propose to select only the frequent generalized itemsets that have at least one infrequent descendant to also consider rare but potentially interesting knowledge. Unlike [62, 69, 116, 127, 128], our approach does not focus on itemset pruning but rather it addresses the complementary issue of highlighting misleading high-level itemsets, which are represented, to a large extent, by their low-level contrasting correlations.

TABLE 8.2: MGI mined from $\mathcal{D}$. min_sup = 1, max_neg_cor= 0.65, min_pos_cor= 0.80, and max_NOD = 100%.

Frequent generalized itemset (level≥2) [correlation type (Kulc value)]	**Frequent descendant [correlation type (Kulc value)]**	**Not overlapping degree (%)**
{(City, Italy)} [positive (1)]	{(City, Turin)} [positive (1)] {(City, Rome)} [positive (1)]	-
{(City, France)} [positive (1)]	{(City, Paris)} [positive (1)] {(City, Cannes)} [positive (1)]	-
{(Product, Wearing)} [positive (1)]	{(Product, T-shirt)} [positive (1)] {(Product, Jacket)} [positive (1)]	-
{(Product, Education)} [positive (1)]	{(Product, Book)} [positive (1)]	-
{(Product, Wearing), (City, Italy)} [positive (5/6=0.83)]	**{(Product, T-shirt), (City, Turin)} [positive (7/8=0.88)] {(Product, T-shirt), (City, Rome)} [negative (5/8=0.63)]**	**75**
{(Product, Wearing), (City, France)} [negative (1/2=0.50)]	**{(Product, Jacket), (City, Paris)} [positive (1)]**	**0**
{(Product, Education), (City, France)} [negative (2/3=0.66)]	**{(Product, Book), (City, Cannes)} [positive (1)]**	**0**

A significant effort has also been devoted to discovering frequent item correlations among large datasets [121–123, 125]. In this context, a pioneering work [122] proposes to evaluate association rule significance via the chi square test for correlation. The authors also exploit the upward closure of the chi square measure to discard some uninteresting candidate itemsets early. To extract negatively correlated item correlations, which are usually characterized by low support value [130], in [121, 123, 131] two novel itemset correlation measures, namely *collective strength* and *support expectation*, have also been proposed and used to perform indirect negative association rule mining. To evaluate item correlation independently of the dataset size in [124] a null-invariant Kulczynsky measure has also been proposed [132]. In [125] the same measure has been exploited to discover flipping correlations among data that were supplied with taxonomies. Flipping correlations are itemsets for which the correlation type flips from positive to negative (or vice versa) when items are generalized to a higher level of abstraction for *every* generalization step. However, when coping with real-life data, item correlation flippings are not likely to occur at every generalization step. Furthermore, a generalized itemset may have many low-level contrasting correlations which are worth considering all together. Unlike [125], this chapter addresses the complementary issue of discovering a worthwhile subset of misleading high-level itemsets which are covered, to a large extent, by contrasting correlations at lower abstraction levels.

Parallel research efforts have also been devoted to proposing optimization strategies to efficiently address generalized itemset mining [66–68, 133]. While the authors in [66] propose an Apriori-based top-down traversal of the search space, an FP-Growth-like

approach to generalized itemset mining [68] and a mining algorithm [67] that exploits the vertical data format [134] have also been presented. In contrast, in [133] an efficient data structure is used to store and generalize low-level itemsets and association rules. Furthermore, the discovery of a succinct and non-redundant subset of frequent itemsets [135–138] has also been investigated. Since the above approaches do not address misleading generalized itemset mining, their goal is somehow related to but different from those addressed by this work.

8.2 Preliminary definitions and notations

This chapter addresses the problem of generalized itemset mining from structured data that are supplied with taxonomies. A structured dataset is a set of records. Each record is a set of items, which are defined as pairs (attribute_name, value). While attribute_name is the description of a data feature, value represents the associated information and belongs to the corresponding attribute domain. Since continuous attribute values are unsuitable for use in itemset mining, continuous values are discretized by a traditional preprocessing step [13]. For instance, Table 8.1 reports an example of structured dataset D that is composed of 3 attributes: the record identifier (rid), the city, and the product description.

A taxonomy is a set of is-a hierarchies built over the data attribute items. It consists of a set of aggregation trees, one or more for each dataset attribute, in which the items that belong to the same attribute domain are aggregated in higher level concepts. For example, let us consider the taxonomy that is reported in Figure 8.1. It includes two aggregation trees, one for each attribute in $\mathcal{D}$. By construction, we disregard the rid attribute for the subsequent analysis. For each aggregation tree the leaf nodes are labeled with values belonging to the corresponding attribute domain, whereas each non-leaf node aggregates (a subset of) lower level nodes and is labeled with a value that is not in the attribute domain. Aggregation tree root nodes are labeled with the special value $\perp$. A pair (attribute_name, aggregation_value), where aggregation_value is a non-leaf node label, is called *generalized item*. For instance, (City, France) is a generalized item that corresponds to a taxonomy non-leaf node which aggregates all of the French cities that occur in $\mathcal{D}$ (see Table 8.1 and Figure 8.2(a)). For the sake of simplicity, hereafter we consider only taxonomies that are composed of one aggregation tree per attribute.

A k-itemset (i.e. an itemset of length k) is defined as a set of k distinct items [104]. For instance, {(City, Turin), (Product, T-shirt)} is an example of itemset that occurs in $\mathcal{D}$ (see Table 8.1). Similarly, when dealing with structured datasets that are supplied with taxonomies, a generalized k-itemset is a set of k distinct items or generalized items. For instance, given the taxonomy reported in Figure 8.1, {(City, Italy), (Product, Wearing)} is an example of generalized 2-itemset.

Generalized itemsets are characterized by many properties [62]. For our purposes, we recall some notable properties in the following.

Coverage and support. A generalized itemset I is said to *cover* a given record $r_i \in \mathcal{D}$ if all of its (generalized) items are either contained in r_i or ancestors of items in r_i. I's support in $\mathcal{D}$ is defined as the ratio between the number of records in $\mathcal{D}$ that are covered by I and the total number of records in $\mathcal{D}$ [62]. A generalized itemset for which the support exceeds a given threshold min_sup is said to be *frequent*. For example,{(City, Italy), (Product, Wearing)} has support $\frac{4}{7}$ in $\mathcal{D}$ because it covers the records with rids 1, 2, 3, and 7 (see Table 8.1). Given a set of generalized itemsets $\mathcal{I}$, for our purposes we also define the coverage of $\mathcal{I}$ with respect to $\mathcal{D}$, hereafter denoted as $cov(\mathcal{I}, \mathcal{D})$, as the ratio between the number of records in $\mathcal{D}$ that are covered by *any* itemset in $\mathcal{I}$ and the total number of records in $\mathcal{D}$. For example, together the itemsets {(City, Italy), (Product, Wearing)} and {(City, France), (Product, Wearing)} have coverage $\frac{6}{7}$ in $\mathcal{D}$, because they cover all records in $\mathcal{D}$ except for the one with rid 6. Given a single generalized itemset, from the above definitions it trivially follows that its coverage and support values in $\mathcal{D}$ are the same.

Level-sharing itemset. The level of an arbitrary (generalized) item i_j with respect to a taxonomy Γ is defined as the height of the Γ's subtree rooted in i_j. It indicates the item abstraction level according to the given taxonomy. Similar to [66, 125], we target the item correlations at same abstraction level, i.e. the itemsets that exclusively contain items with the same level. Such patterns are denoted as *level-sharing itemsets* [66]. The level of a level-sharing itemset I with respect to the taxonomy Γ, i.e. $\mathrm{L}[I, \Gamma]$, corresponds to that of any of its items.

Experts are expected to provide balanced taxonomy trees to effectively highlight contrasting correlations at different taxonomy levels. If the experts do not provide balanced taxonomy trees, as in [125], we rebalanced those taxonomy aggregation trees in the performed experiments. Specifically, given a taxonomy with maximal aggregation tree height H_{max}, for each aggregation tree with height $H < H_{max}$ we performed a depth-first visit. For each tree branch with depth less than H_{max} we added multiple copies of the top-level item i as i's ancestors up to depth H_{max}.

Descent relationship. Given two generalized k-itemsets I_1 and I_2, I_1 is said to be a descendant of I_2, i.e. $I_1 \in \mathrm{Desc}[I_2, \Gamma]$ if for every item $i_j \in I_1$ there exists an item $i_k \in I_2$ such that either $i_j{=}i_k$ or i_j is a descendant of i_k with respect to the given taxonomy. For example, {(City, Turin), (Product, T-shirt)} is a descendant of {(City, Italy), (Product, Wearing)}.

Correlation. The itemset correlation measures the strength of the correlation between its items. In this chapter, similar to [125], we evaluate the correlation of a generalized k-itemset I by means of the Kulczynsky (Kulc) correlation measure [124], which is defined as follows:

$$\text{kulc(I)} = \frac{1}{\text{k}} \sum_{j=1}^{\text{k}} \frac{sup(I, \mathcal{D})}{sup(i_j, \mathcal{D})} \tag{8.1}$$

where $sup(I, \mathcal{D})$ is I's support in $\mathcal{D}$ and i_j $[1 \leq j \leq k]$ is the j-th item in I.
From Equation 8.1 it follows that Kulc values range between 0 and 1. Unlike many other traditional itemset correlation measures, Kulc has the null (transaction)-invariant property, which implies that the correlation measure is independent of the dataset size [124]. By properly setting maximum negative and minimum positive Kulc thresholds, hereafter denoted as *max_neg_cor* and *min_pos_cor*, the generalized itemsets may be classified as negatively correlated, uncorrelated, or positively correlated itemsets according to their correlation value. More specifically, generalized itemsets for which Kulc is between max_neg_cor and min_pos_cor consist of items that are not correlated with each other (i.e. their items are statistically independent), generalized itemsets for which Kulc is below max_neg_cor show negative item correlation, whereas generalized itemsets for which Kulc is above min_pos_cor indicate a positive item correlation, i.e. their items co-occur more than expected. For the sake of brevity, we hereafter denote the above-mentioned correlation types as *uncorrelated*, *negative*, and *positive*, respectively.

8.3 The Misleading Generalized Itemset mining problem

Given a structured dataset $\mathcal{D}$ that is supplied with a taxonomy Γ and a minimum support threshold min_sup, the traditional frequent generalized itemset mining problem entails discovering all of the frequent generalized itemsets from $\mathcal{D}$.
Frequent generalized itemsets represent data correlations at different abstraction levels. On the one hand, low-level itemsets commonly represent rather specific and detailed data correlations. Unfortunately, they are unlikely to be frequent with respect to the enforced minimum support threshold. On the other hand, high-level itemsets provide a high-level viewpoint of the analyzed data, which could be useful for representing the infrequent knowledge at a higher abstraction level. However, some high-level itemsets could be deemed to be misleading, because their correlation type is in contrast to that of their low-level descendants. For instance, consider the example dataset and taxonomy reported in Table 8.1 and Figure 8.1, respectively. The frequent generalized itemset {(Product, Wearing), (City, Italy)} has a positive correlation type, whereas its frequent low-level descendant itemset {(Product, T-shirt), (City, Rome)} is negatively correlated (see Table 8.2). Since the type of the mined data correlation changes unexpectedly while performing a drill-down, the high-level itemset is, to some extent, misleading.
To allow domain experts to discover and analyze the misleading high-level itemsets separately, we propose a new generalized pattern type, namely the M*isleading* G*eneralized* I*temset* (MGI). MGIs are patterns in the form $X \triangleright \mathcal{E}$, where X is a frequent generalized

itemset of level $l \geq 2$ with either positive or negative correlation type, while $\mathcal{E}$ is the set of frequent level-$(l-1)$ X's descendants for which the correlation type is in contrast to that of X. A more formal definition follows.

Definition 8.1. MGI**.** Let $\mathcal{D}$ be a structured dataset and Γ a taxonomy. Let min_sup be a minimum support threshold and max_neg_cor and min_pos_cor a maximum negative and a minimum positive correlation threshold. Let $\mathcal{LSGI}$ be the subset of frequent level-sharing generalized itemsets in $\mathcal{D}$ that are either positively or negatively correlated. Given a frequent level-sharing generalized itemset $X \in \mathcal{LSGI}$ of level $l \geq 2$, let Desc$^*[X,\Gamma]$ be the subset of level-$(l-1)$ X's descendants for which the correlation type is in contrast to that of X. An MGI is a pattern in the form $X \triangleright \mathcal{E}$, where $X \in \mathcal{LSGI}$ and $\mathcal{E}$=Desc$^*[X,\Gamma]$.

For example, setting min_sup = 1, max_neg_cor=0.65, and min_pos_cor=0.8, the MGI {(Product, Wearing), (City, Italy)} $\triangleright$ {(Product, T-shirt), (City, Rome)} is mined from the example dataset in Table 8.1, because {(Product, Wearing), (City, Italy)} has a positive correlation (0.83), whereas its descendant itemset {(Product, T-shirt), (City, Rome)} is negatively correlated (0.63).
We define the level of an MGI $X \triangleright \mathcal{E}$ with respect to the input taxonomy Γ as X's level, i.e. $\mathrm{L}[X \triangleright \mathcal{E}, \Gamma] = \mathrm{L}[X, \Gamma]$. For example, {(Product, Wearing), (City, Italy)} $\triangleright$ {(Product, T-shirt), (City, Turin)} is a level-2 MGI because {(Product, Wearing), (City, Italy)} has level 2.
Since a generalized itemset could have many low-level descendants that represent contrasting correlations, we evaluate the interest of an MGI $X \triangleright \mathcal{E}$ as the relative difference between the support of the ancestor generalized itemset X and the coverage of its low-level contrasting data correlations in $\mathcal{E}$. We denote this measure as the *Not Overlapping Degree* (NOD).

Definition 8.2. MGI**'s NOD measure.** Let $X \triangleright \mathcal{E}$ be an MGI. Let $sup(X, \mathcal{D})$ be X's support in $\mathcal{D}$ and $cov(E, \mathcal{D})$ the coverage of $\mathcal{E}$ in $\mathcal{D}$. The Not Overlapping Degree (NOD) of $X \triangleright \mathcal{E}$ is defined by: $\frac{\mathrm{sup}(X,\mathcal{D}) - \mathrm{cov}(\mathcal{E},\mathcal{D})}{\mathrm{sup}(X,\mathcal{D})}$.

Since the inequality $sup(X, \mathcal{D})$-$cov(\mathcal{E}, \mathcal{D}) \geq 0$ holds, it trivially follows that the MGI NOD values are between 0 and 1. The lower the NOD value is, the more significant the degree of overlapping between the contrasting low-level correlations in $\mathcal{E}$ and their common ancestor X. As an extreme case, when the contrasting descendant itemsets cover *every* record covered by X the MGI NOD value is 0. For example, the MGI {(Product, Wearing), (City, Italy)} $\triangleright$ {(Product, T-shirt), (City, Rome)} has a NOD value equal to $\frac{4-1}{4} = \frac{3}{4}$ because {(Product, Wearing), (City, Italy)} covers four records in $\mathcal{D}$ (i.e. the records with rids 1, 2, 3, and 7), whereas its descendant {(Product, T-shirt), (City, Rome)} covers one of them (i.e. the record with rid 3).

Experts could be interested in analyzing only the MGIs with a relatively low NOD value, because they represent clearly misleading high-level data correlations. Hence, we enforce a maximum NOD constraint to select only the subset of MGIs with a NOD value less than or equal to a maximum NOD threshold max_NOD. As shown in Section 8.5, this worthwhile MGI subset is useful for supporting the expert-driven knowledge discovery process in a real-life application scenario.

Problem statement. Given a structured dataset $\mathcal{D}$, a taxonomy, a minimum support threshold, a maximum negative, and a minimum positive correlation threshold, and a maximum NOD threshold max_nod, the mining task addressed by this chapter entails discovering from $\mathcal{D}$ all of the MGIs for which the NOD value is less than or equal to max_NOD.

8.4 The Misleading Generalized Itemset Miner algorithm

The Misleading Generalized Itemset Miner (MGI Miner) algorithm addresses the MGI mining problem that is stated in Section 8.3. The MGI extraction process entails the following steps: (i) Traditional frequent level-sharing generalized itemset mining and (ii) MGI extraction at the top of the previously extracted itemsets. MGI extraction is performed level-wise, i.e. level-1 MGIs are generated first. Next, at each step, MGIs with increasing level are generated until the top of the taxonomy is reached. Algorithm 6 reports a pseudo-code for the MGI Miner algorithm.

Frequent level-sharing itemset mining. Frequent level-sharing generalized itemsets are used to drive the MGI mining process (see line 1) because each MGI consists of a combination of them (see Definition 8.1). The traditional itemset extraction task is accomplished by an established projection-based itemset miner, i.e. the LCMv2 algorithm [139], which is an extension of the traditional FP-Growth algorithm [47]. Projection-based itemset mining relies on the following steps: (i) creation and in-memory storage of an FP-tree-based dataset representation and (ii) frequent itemset extraction by recursively visiting the conditional FP-tree projections. We applied the following main modifications to a traditional FP-tree-based itemset miner [139]: (1) To efficiently cope with structured dataset, the itemset miner prevents the generation of the candidate itemsets that include couples of items corresponding to the same attribute. (2) To suit the traditional LCM implementation to generalized itemset mining, we adopted the strategy, first proposed in [62], of extending the dataset records by appending to each record all of its item generalizations in Γ. (3) To prevent the generation of not level-sharing itemsets, the generation procedure of the conditional FP-tree projections related to a level-l item disregards the not level-l items. Frequent level-sharing generalized itemsets are stored in $\mathcal{LSGI}$ (line 1).

Require: a structured dataset $\mathcal{D}$, a taxonomy Γ, a maximum NOD threshold max_NOD, a minimum support threshold min_sup, a maximum negative and a minimum positive Kulc thresholds max_neg_cor and min_pos_cor
Ensure: the subset of all the MGIs $\mathcal{MGI}$
1: $\mathcal{LSGI}$ = mineTraditionalLevelSharingGeneralizedItemsets($\mathcal{D}$, Γ, min_sup)
2: $\mathcal{MGI} = \emptyset$
3: {Generate MGIs $X \triangleright \mathcal{E}$ with level $l > 1$}
4: **for** l=2 to *maxlevel* **do**
5: {for each frequent level-sharing generalized itemset one candidate MGI is generated}
6: **for all** X in $\mathcal{LSGI}[l]$ **do**
7: {Create a level-l candidate MGI $X \triangleright \mathcal{E}$}
8: insert the candidate MGI ($X \triangleright \mathcal{E}$) in $C[l]$
9: **end for**
10: {Populate the $\mathcal{E}$ set of the level-l candidate MGI}
11: **for all** it in $\mathcal{LSGI}[l-1]$ **do**
12: {Retrieve the candidate itemset *genit* of level l that is ancestor of it and update $genit.\mathcal{E}$}
13: *genit* = retrieveAncestor($\mathcal{LSGI}[l]$,it,l,Γ);
14: *cor_type_genit*=ComputeKulc(*genit*,$\mathcal{D}$,max_neg_cor,min_pos_cor)
15: *cor_type_it*=ComputeKulc(it,$\mathcal{D}$,max_neg_cor,min_pos_cor)
16: {If the level-$(l-1)$ itemset it has a correlation type different from its ancestor *genit* then it must be added to $genit.\mathcal{E}$}
17: **if** *cor_type_genit* $\neq$ *cor_type_it* **then**
18: insert it into $genit.\mathcal{E}$
19: **end if**
20: **end for**
21: {Select the level-l candidate MGIs with NOD less than or equal to max_NOD}
22: **for all** c in $C[l]$ **do**
23: $c.NOD$ = ComputeNOD(c,$\mathcal{D}$,Γ);
24: **if** $c.NOD \leq$ max_NOD **then**
25: insert c into $\mathcal{MGI}[l]$
26: **end if**
27: **end for**
28: **end for**
29: **return** $\mathcal{MGI}$

Algorithm 6: MGI Miner algorithm

MGI **mining:** Once all the frequent level-sharing itemsets X are extracted, MGI Miner generates candidate MGIs in the form $X \triangleright \mathcal{E}$ and populates their $\mathcal{E}$ part with X's descendants for which the correlation type is in contrast to those of X. MGIs are mined by following a level-wise approach, i.e. climbing up the taxonomy stepwise until the top of the taxonomy is reached (lines 4-28). Performing a level-wise taxonomy evaluation prevents the need for multiple itemset scans. Indeed, level-l MGIs are generated from the sets of level-l and level-$(l-1)$ frequent level-sharing itemsets $\mathcal{LSGI}[l]$ and $\mathcal{LSGI}[l-1]$. While the level-l itemsets are used to populate the X part (lines 6-9), the level-$(l-1)$ itemsets that represent contrasting correlations are used to fill the $\mathcal{E}$ set (lines 11-20). Hence, while mining level-l MGIs all of the traditional frequent itemsets that have a level strictly less than $l-1$ can be discarded early. Finally, level-l MGIs for which the NOD value is less than or equal to max_NOD are selected and added to the output set (lines 22-27).

8.5 Experimental results

We performed a large suite of experiments to evaluate: (i) the usefulness of the MGIs mined from data that were acquired from a real-life context with the help of a domain

expert (see Section 8.5.2); (ii) the impact of the algorithm parameters on the MGI MINER performance on benchmark datasets (see Section 8.5.3); and (iii) the MGI MINER algorithm scalability on synthetic datasets (see Section 8.5.4).
The experiments were performed on a 3.30 GHz Intel® Xeon® CPU E31245 PC with 16 GB main memory running Linux (kernel 3.2.0).

8.5.1 Datasets

A brief description of the evaluated datasets is reported in the following paragraphs.

Real-life mobile datasets

To validate the usefulness of the proposed patterns, we ran experiments on two real mobile datasets that were collected by a research hub of an international leader in the telecommunication area. The two datasets were acquired by logging the user requests for two different mobile applications, namely *Recs* and $TeamLife$. The applications provide users with a set of services (e.g. weather forecasting, restaurant recommendations, and photo and movie uploads) through their mobile devices (i.e. smartphones or tablet PCs). Service requests coming from each application were collected in a separate log file (i.e. dataset). A more thorough description of the analyzed datasets and their corresponding taxonomies follows.

Recs The *Recs* application is a recommender system that provides recommendations to users on entertainment activities (e.g. restaurants and museums). Each user can request a recommendation, vote for an item (i.e. an entertainment center), update a vote, upload a file or a photo to provide useful information about an item (i.e. a restaurant or a museum), and post a comment. Hence, a set of services is provided to the end users to perform the described operations/services. The dataset contains the user requests that were submitted and that were obtained by logging the user requests over the time period of three months. For *Recs*, the following aggregation trees have been considered:

- date → month → trimester → year
- time stamp → hour → time slot (2-hour time slots)→ day period (AM/PM)
- user → gender
- service → service category

TeamLife The *TeamLife* dataset was generated by logging the *TeamLife* application requests. *TeamLife* users can upload files, photos, and videos, share them with other

system users, and post short messages. The uploading services (i.e. file, photo, and video uploading services) are aggregated into the UploadData service category. The dataset collects the user requests that were submitted over a time period of three months. For *TeamLife* we used a taxonomy that is similar to the one previously described for the *Recs* dataset.

UCI benchmark datasets

To analyze the MGI Miner algorithm performance we exploited a set of UCI benchmark datasets [126] with different characteristics in terms of number of records and attributes. The main dataset characteristics are summarized in Table 8.3.

The taxonomies built over the UCI datasets were generated as follows. To build the aggregation trees over the continuous data attributes, we applied several equi-depth discretization steps with finer granularities [13]. Specifically, the finest discretized values were considered to be the data item values and thus became the taxonomy leaf nodes, while the coarser discretizations were exploited to aggregate the corresponding low-level values into higher level values. For the UCI datasets reported in Table 8.3 we created a 3-level taxonomy by applying a 10-bin equal frequency discretization to generate the level-1 items and a 5-bin discretization to generate the level-2 items. At the top of the hierarchy, the level-2 items were aggregated into the root node. In contrast, the aggregation trees built over the nominal data attributes were analyst-provided. The items for which no meaningful aggregation is available were aggregated directly into the root node.

A description of a representative UCI dataset coming from the census domain and its corresponding taxonomy follows.

Adult dataset. Adult collects census data about American people (e.g. education, occupation, marital status, race, and sex). We defined the following aggregation trees for the nominal attributes Education, Marital-status, and Native-country.

- Education (Preschool, 1st-4th grades, ..., 12th grade → Pre High School / HS-grad → High School / Assoc-acdm, Assoc-voc, Some College, Bachelors, Masters, Prof-school, Doctorate → Post High School)
- Marital-status (Civil married, Church married → Married / Separated, Divorced, Widowed → Unmarried)
- Native-country (England, Germany, ... → Europe / China, Japan, Thailand, ... → Asia / United-States, Canada, Mexico, ... → America / South Africa, ... → Africa)

For the remaining nominal attributes no item aggregations (disregarding the root node) have been defined.

TABLE 8.3: UCI and real mobile dataset characteristics and number of mined MGIs with max_neg_cor=0.6 and min_pos_cor=0.7.

	Dataset	Rec.	Attr.	Number of items with level=1	Number of items with level>1	min_sup	Gen. Itemsets (level>1)	max_NOD	MGIs
UCI	**Adult**	32,561	15	166	135	1%	353,622	1%	26
								5%	33
	Breast	699	11	742	45	1%	11,454	1%	9
								5%	36
	Cleve	303	14	110	20	1%	240,941	1%	1
								5%	1
	Crx	690	16	98	27	1%	1,457,397	1%	32
								5%	36
	Glass	214	11	306	44	1%	24,872	1%	25
								5%	25
	Heart	270	14	73	25	1%	630,495	1%	1
								5%	1
	Letter recognition	20,000	17	160	80	1%	503,328	1%	6
								5%	6
	Pima	768	9	85	40	1%	10,596	1%	3
								5%	3
	Pendigits	10,992	17	160	80	1%	437,364	1%	1
								5%	2
	Shuttle	43,500	10	89	42	1%	6,747	1%	81
								5%	108
	Vehicle	846	19	154	90	1%	4,717,399	1%	32
								5%	40
	Waveform	5,000	22	89	54	1%	13,589,519	1%	11
								5%	11
	Wine	178	14	133	65	1%	2,406,612	1%	5
								5%	5
Mobile	TeamLife	1,197	4	1,293	31	1%	225	10%	1
								15%	2
	Recs	5,668	4	3,979	39	0.15%	475	10%	1
								15%	1

Synthetic datasets

We used the function 2 of the Quest IBM synthetic dataset generator [140], which was first exploited in [141] in the context of data classification, to generate synthetic data. The data generator automatically produces structured datasets that are composed of a user-specified number of records and attributes. To automate the taxonomy generation procedure we extended the data generator source code as follows. Once a user has specified the required taxonomy height H, for each attribute the item values are treated as taxonomy leaf nodes, sorted into lexicographical order, and clustered into a subset of equal-frequency bins. Each bin is associated with a generalized item that aggregates all group members. Next, the high-level bins are further aggregated to each other and the procedure iterates until all the items are clustered in a unique node (i.e. the root node). At each generalization level the bin frequency is automatically derived from the taxonomy height and the attribute domain cardinality. For example, setting H to 3 a 27-value attribute domain is partitioned into 9 equal-frequency bins at level 1, 3 equal-frequency bins at level 2 and a unique bin at level 3. The extended generator code is available at [142].

8.5.2 Expert-driven MGI validation in a mobile application scenario

We evaluated the usefulness of the MGIs mined from the real-life data taken from a mobile scenario with the help of a domain expert. Table 8.4 reports two MGIs that were extracted from the TeamLife dataset by enforcing min_sup=1%, max_neg_cor=0.6, min_pos_cor=0.7, and max_NOD=15%. As an example, in this section we discuss their usability for supporting experts in planning marketing campaigns and resource allocation.

TABLE 8.4: Examples of MGIs mined from TeamLife.

ID	MGI	**support (%)**	**NOD (%)**	X**'s correlation type (Kulc value)**
MGI 1	{(User, Male), (Service, UploadData)} ▷ { {(User,UserA), (Service,Photo)}, {(User,UserB), (Service,Photo)}, ... {(User,UserZ), (Service,File)}, {(User,UserZ), (Service,Photo)} }	70.0%	7.63%	positive (0.86)
MGI 2	{(Date, May), (Service, UploadData)} ▷ { {(Date,2009-05-01), (Service,Photo)}, {(Date,2009-05-06), (Service,File)}, ... {(Date,2009-05-29), (Service,Photo)}, {(Date,2009-05-31), (Service,Photo)} }	58.3%	12.9%	positive (0.76)

Each of the MGIs reported in Table 8.4 consists of a positively correlated level-2 generalized itemset X and a set $\mathcal{E}$ of negatively correlated low-level (descendant) itemsets. Let us consider the MGI 1 first. The traditional high-level itemset X={(User, Male), (Service, UploadData)} indicates that the UploadData mobile services (i.e. Photo, File, and Video) are frequently requested by male users. The domain expert could exploit such information for marketing purposes. For instance, he may recommend to male users services that belong to the UploadData category, while disregarding the specific type of service each user is actually interested in. However, analyzing MGI 1 the above pattern turns out to be misleading. In fact many of X's descendants (i.e. many combinations of a user with a specific UploadData service) show an opposite trend. Specifically, many male users appear to be negatively correlated with at least one of the services that belong to the UploadData category. Hence, recommending to male users all the UploadData services indiscriminately could be a suboptimal choice for marketing purposes. On the contrary, the expert should consider the correlation type of each descendant itemset separately in order to perform targeted recommendations. Note that the NOD value of the MGI 1 is 7.63%. Hence, the service requests that are covered by itemsets that represent contrasting correlations are approximately 92% of those that are covered by the (misleading) high-level itemset. Therefore, discovering MGIs rather than traditional itemsets allows analysts to avoid planning non-personalized and possibly ineffective marketing campaigns. On the other hand, the domain expert may further investigate the

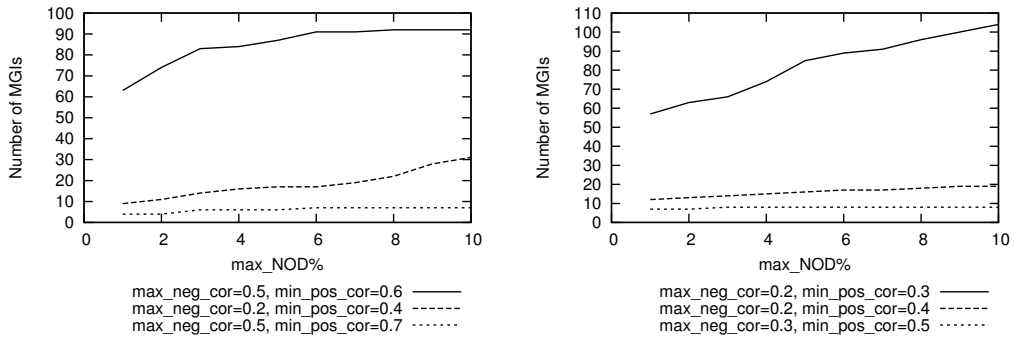
interest of some specific users that show a contrasting correlation with one or more services of the UploadData category in order to offer them personalized promotions.
The domain expert deemed the MGI 2 to be interesting to support resource allocation/shaping. The generalized itemset X={(Date, May), (Service, UploadData)} indicates a positive correlation between the UploadData category and a specific month. Experts could exploit such knowledge to allocate dedicated resources to the UploadData service category in May, while disregarding the individual user's interests. However, analyzing MGI 2 may prompt the expert to perform a more conservative and accurate resource allocation and shaping. More specifically, it turns out that some UploadData services are requested less than expected during the early days of May. Since the subset of negatively correlated frequent descendants covers approximately 87% of the records that are covered by {(Date, May), (Service, UploadData)}, the high-level ancestor could be considered to be a misleading high-level pattern. Rather than allocating the resources for all the UploadData services indiscriminately, the network resource manager should perform a more selective resource allocation according to the actual daily service usage. For example, since the Photo service appears to be, on average, underused during most of the days of May, the manager should allocate a portion of its currently dedicated bandwidth to any other service of the same category (e.g., File or Video).

8.5.3 Algorithm parameter analysis

We analyzed the impact of the main MGI Miner algorithm parameters on the number of MGIs mined from the UCI datasets. In the following section, the effect of each algorithm parameter will be discussed separately.

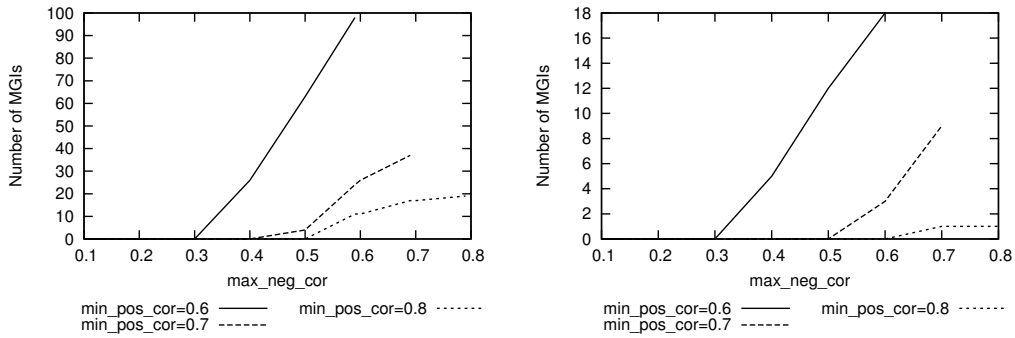
8.5.3.1 Effect of the maximum NOD threshold

The maximum NOD threshold allows experts to select only the MGIs that represent an unexpected and clearly misleading pattern. It indicates the maximum portion of data that are covered by a generalized itemset and that are not covered by any of its low-level contrasting correlations.
Figures 8.3(a) and 8.3(b) plot the number of MGIs mined by varying the max_NOD value in the range [0, 10%] on two representative UCI datasets, i.e. Adult and Pima, respectively. As expected, lowering the maximum NOD threshold the number of extracted MGIs decreases more than linearly because of the higher selectivity of the enforced constraint. Note that even setting a relatively high max_NOD threshold value (e.g. 10%) the number of extracted MGIs remains limited (e.g. around 130 for the Adult dataset). Similar results were obtained for the mobile dataset and the other UCI datasets. To provide an insight into the achieved results, Table 8.3 summarizes the results that were achieved by setting two representative max_NOD values.



(c) Adult (d) Pima

FIGURE 8.2: Impact of the maximum NOD threshold on the number of mined MGIs. min_sup=1%.



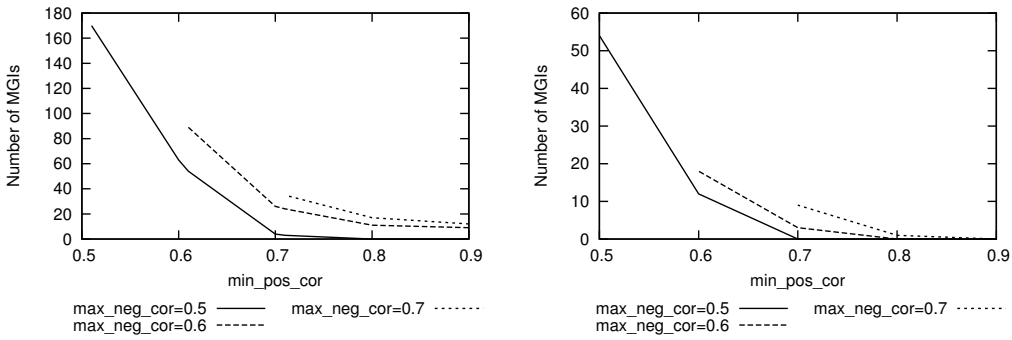
(a) Adult (b) Pima

FIGURE 8.3: Impact of the maximum negative threshold max_neg_cor on the number of mined MGIs. max_NOD=1%, min_sup=1%.

8.5.3.2 Effect of the correlation thresholds

Enforcing different maximum negative and minimum positive correlation thresholds can affect MGI MINER algorithm performance and the characteristics of the mined patterns. To analyze the impact of the positive and negative correlation thresholds separately, in Figures 8.3(a) and 8.3(b) we plotted the number of MGIs mined by varying max_neg_cor and by setting three representative min_pos_cor values on Adult and Pima, while in Figures 8.4(a) and 8.4(b) we analyzed the opposite situation, i.e. we varied min_pos_cor by setting three representative values for max_neg_cor on the same datasets. Note that since max_neg_cor $<$ min_pos_cor some curve points are missing.

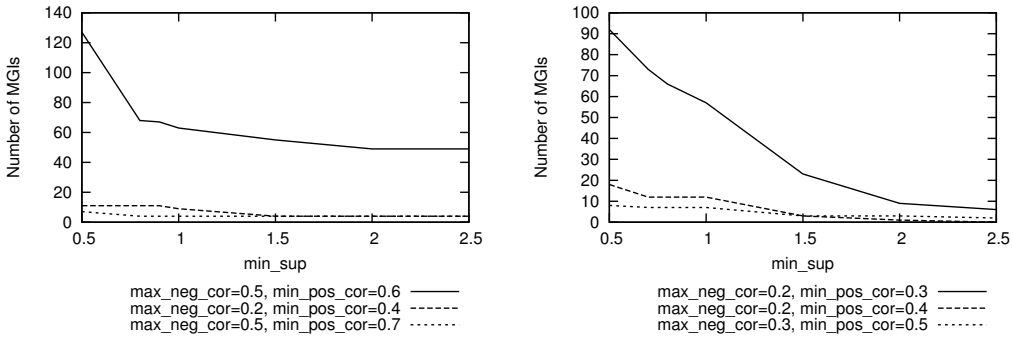
As expected, the itemset correlation changes occur more frequently while setting closer max_neg_cor and min_pos_cor values. Moreover, the itemset correlation values appear to be unevenly distributed among the analyzed data. Specifically, the majority of the frequent generalized itemsets have a Kulc value between 0.4 and 0.7. Hence, setting max_neg_cor and min_pos_cor in such a value range yields a significant increase in the number of extracted MGIs, because the generalization process is likely to change the



(a) Adult (b) Pima

FIGURE 8.4: Impact of the minimum positive threshold min_pos_cor on the number of mined MGIs. max_NOD=1%, min_sup=1%.

correlation type. On the other hand, setting the positive and the negative thresholds out of the above-mentioned value range yields a mined set cardinality reduction.



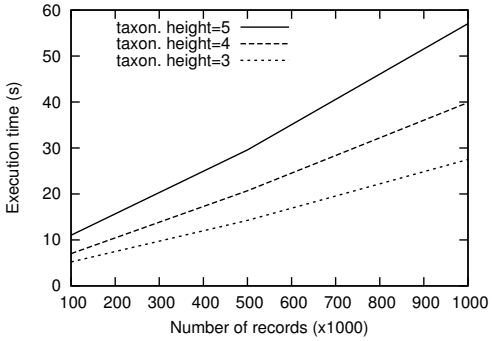
(a) Adult (b) Pima

FIGURE 8.5: Impact of the minimum support threshold min_sup on the number of mined MGIs. max_NOD=1%.

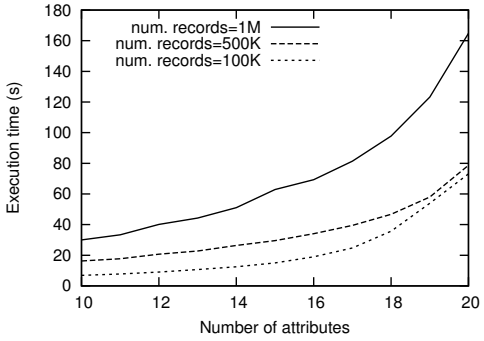
8.5.3.3 Effect of the minimum support threshold

The minimum support threshold min_sup significantly affects the characteristics of the results of the traditional itemset mining algorithms (e.g. Apriori [104], FP-Growth [47]). For this reason, we also analyzed the impact of min_sup on the characteristics of the mined patterns. Figures 8.5(a) and 8.5(b) report the number of MGIs extracted from Adult and Pima by varying min_sup in the range [1%,10%] and by setting three representative pairs of min_pos_cor and max_neg_cor values.

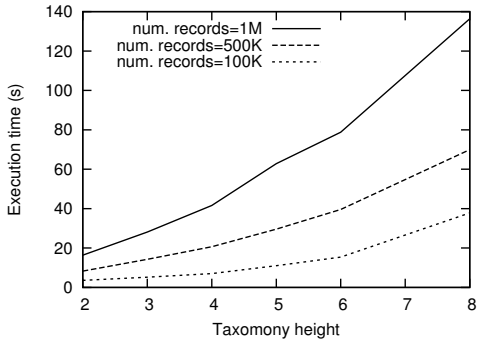
The number of mined MGIs increases while lower min_sup values are enforced. This trend is mainly due to the combinatorial increase in the number of generated item combinations which yields a super-linear increase in the number of frequent traditional



(a) Scalability with the number of records. Num. of attributes=15.



(b) Scalability with respect to the number of attributes. Taxonomy height=5.



(c) Scalability with respect to the taxonomy height. Num. of attributes=15.

FIGURE 8.6: MGI MINER scalability. min_sup=1%, max_NOD=1%, max_neg_cor=0.6, min_pos_cor=0.7.

generalized itemsets. Although the curve slopes depend on the analyzed data distribution and the enforced correlation thresholds (see Section 8.5.3.2), the results that were achieved on datasets with different characteristics show rather similar trends.

8.5.4 Scalability

We analyzed the MGI MINER algorithm scalability, in terms of execution time, on synthetic data with (i) the number of records, (ii) the number of attributes, and (iii) the taxonomy height.

To evaluate the scalability with the number of records we varied the data cardinality in the range $[10^5, 10^6]$ while setting the number of attributes to 15 and three representative taxonomy height values (i.e, 3, 4, and 5). The results, reported in Figure 8.6(a), show that the MGI MINER execution time scales roughly linearly with the number of records, because the data distribution remains approximately unchanged while increasing the dataset cardinality.

We also analyzed the impact of the number of attributes and the taxonomy height on the MGI MINER execution time. In the former case, we varied the dataset dimensionality in the range [10, 20], while considering a 5-level taxonomy and three representative dataset

cardinalities (i.e. 10^5, $5{\times}10^5$, 10^6). In the latter, we varied the taxonomy height between 2 and 8, while considering three 15-attribute datasets with different size. The results, reported in Figures 8.6(b) and 8.6(c), show that the MGI Miner execution time scales more than linearly with both the number of attributes and the taxonomy height because of the combinatorial increase in the number of generated combinations. However, the execution time remains acceptable even when coping with rather complex datasets and taxonomies (e.g. approximately 140s for a 15-attribute dataset with 10^6 records and an 8-level taxonomy).

Chapter 9

SeaRuM: a Cloud-Based SErvice for Association RUle Mining

The capability of modern applications (e.g., social networks, computer networks, wireless sensor applications) of generating and collecting data has been rapidly increasing, as we already discussed in previous chapters. Since data mining focuses on studying effective and efficient algorithms to transform huge amounts of data into useful and actionable knowledge, the interest in data mining is continuously growing both in the industrial and research domains. Industries are attracted by the business opportunities arising from the exploitation of the extracted knowledge, while researchers are interested in the challenging issues coming from the application of data mining techniques to new scenarios. Different data analytics tools rely on data mining algorithms to gain interesting insights from large volumes of semi-structured or unstructured data. Data mining techniques allow extracting previously unknown interesting patterns such as groups of data objects (cluster analysis), unusual objects (anomaly detection) and dependencies (association rule mining) [13].

When dealing with huge data collections like those of "Big Data", the computational cost of the data mining process (and in some cases the feasibility of the process itself) can potentially become a critical bottleneck in data analysis. For example, association rule mining algorithms, which find application in a wide range of different domains including medical images [143], biological data [144], and network traffic data [63], are characterized by computationally intensive tasks. Thus, parallel and distributed approaches can be adopted to increase the mining efficiency and improve algorithm scalability. The current trend explored by cloud providers, such as Windows Azure, is offering an increasingly heterogeneous portfolio of online services in the cloud, spanning traditional IaaS (Infrastructure-as-a-Service) and PaaS (Platform-as-a-Service) as well as business analytics-oriented cloud-based tools.

Association rule mining is a two-step process: (i) frequent itemset extraction and (ii)

association rule generation from frequent itemsets [65]. Since the first phase represents the most computationally intensive knowledge extraction task, effective solutions have been widely investigated to parallelize the itemset mining process both on multi-core processors [145–148] and with a distributed architecture [149–152]. However, when a large set of frequent itemsets is extracted, the generation of association rules from this set becomes a critical task.

We aim to design and develop an association rule mining framework as a SaaS (Software-as-a-Service) service model, to offer a data analytics service to cloud users. To our knowledge, the association rule generation from frequent itemsets on a distributed architecture has not been investigated yet, and represents a core contribution of this work. More specifically, this Chapter presents a cloud-based service, named SeARuM (a Cloud-Based SErvice for Association RUle Mining), to efficiently mine association rules on a distributed computing model. SeARuM consists of a series of distributed MapReduce jobs run in the cloud. Each job performs a different step in the association rule mining process.

As a reference case study, the proposed approach has been applied to two real network datasets. The analysis of the large amount of Internet traffic data is an important task, since a huge amount of interesting knowledge can be automatically mined to effectively support both service providers and Internet applications. To profile network communications, we analyzed traffic metrics and statistical measurements computed on traffic flows. The results showed the effectiveness and efficiency of the SeARuM architecture in mining interesting patterns on a distributed computing model.

9.1 Related work

The mining task in association rule extraction entails two subtasks, the frequent itemset generation and the subsequent association rule extraction. The former subtask is more computationally intensive than the latter [65], and it has been paid more attention by the research community, leading to the definition of efficient algorithms.

Initial parallel and distributed itemset extraction stemmed from algorithms based on the Apriori approach [145]. Further improvements include FP-Tree [47], which exploits prefix-tree-like structures, and a multi-tree approach proposed in [146]. Parallel processing flows perform a sequence of three steps: firstly a horizontal subset of the data is analyzed, then a local FP-Tree is built, finally the mining process is carried out on the local FP-Tree. The candidate pattern bases from different processing flows are then merged together. The good efficiency and scalability of this approach are counterbalanced by large memory requirements and node traversal redundancy with a high number of parallel processing flows. [149] proposes an enhanced version of the merging algorithm specifically addressing the cluster computing environment. [153] discusses a

tree partition approach based on a single tree. In the context of very large datasets, different constraints have been proposed in the parallel itemset extraction algorithm by [150], leading to good scalability on a 32-processor cluster. Better hardware resource exploitation and improvement of the itemset extraction on multiple-core processors have been addressed by [147, 148]. A path tiling technique has been devised to enhance the FP-Tree algorithm in terms of performance, by improving the temporal locality of data accesses at different memory levels. Cache hint optimization was firstly addressed by this technique in [148], which then found application also in the parallel itemset mining context [147].

Large-scale mining based on the MapReduce paradigm [154] is based on algorithms that distribute data and computation across a distributed architecture [151, 152]. A parallel FP-Growth algorithm has benn proposed in [151]. The algorithm, named PFP, aimed at supporting efficient query recommendation for search engines. PFP consists of two separate MapReduce jobs (i.e., no computational dependencies exists between them) and achieves an almost linear speedup. [152] proposes some improvements with respect to [151], particularly addressing efficient load balance strategies, thus achieving higher speedup and better performance.

9.2 Problem statement

Let $\mathcal{D}$ be a dataset whose a generic record r is a set of features. Each feature, also called *item*, is a couple *(attribute, value)*. Since we are interested in analyzing statistical features computed on traffic flows, each feature models a measurement describing the network flow (e.g., Round-Trip-Time (RTT), number of hops).

An *itemset* is a set of features. The *support count* of an itemsets I is the number of records containing I. The *support* $s(I)$ of an itemset I is the percentage of records containing I. An itemset is *frequent* when its support is greater than, or equal to, a minimum support threshold $MinSup$. *Association rules* identify collections of itemsets (i.e., set of features) that are statistically related (i.e., frequent) in the underlying dataset. Association rules are usually represented in the form $X \rightarrow Y$, where X (also called rule antecedent) and Y (also called rule consequent) are disjoint itemsets (i.e., disjoint conjunctions of features). Rule quality is usually measured by rule support and confidence. *Rule support* is the percentage of records containing both X and Y. It represents the prior probability of $X \cup Y$ (i.e., its observed frequency) in the dataset. *Rule confidence* is the conditional probability of finding Y given X. It describes the strength of the implication and is given by $\mathrm{c}(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)}$ [13].

Given a dataset $\mathcal{D}$, a support threshold $MinSup$, and a confidence threshold $MinConf$, the mining process discovers all association rules with support and confidence greater than, or equal to, $MinSup$ and $MinConf$, respectively.

Furthermore, to rank the most interesting rules, we used the lift index [13], which measures the (symmetric) correlation between antecedent and consequent of the extracted rules. The lift of an association rule $X \rightarrow Y$ is defined as [13]

$$\text{lift(X, Y)} = \frac{\text{c}(X \rightarrow Y)}{\text{s(Y)}} = \frac{\text{s}(X \rightarrow Y)}{\text{s(X)s(Y)}} \tag{9.1}$$

where $\text{s}(X \rightarrow Y)$ and $\text{c}(X \rightarrow Y)$ are respectively the rule support and confidence, and $\text{s}(X)$ and $\text{s}(Y)$ are the supports of the rule antecedent and consequent. If lift(X,Y)=1, itemsets X and Y are not correlated, i.e., they are statistically independent. Lift values below 1 show a negative correlation between itemsets X and Y, while values above 1 indicate a positive correlation. The interest of rules having a lift value close to 1 may be marginal. In this work the mined rules are ranked according to their lift value to focus on the subset of most (positively or negatively) correlated rules.

9.3 The SeARuM architecture

SeARuM consists of a series of distributed jobs run in the cloud. Each job receives as input the result of one or more preceding jobs and performs one of the steps required for association rule mining. Currently, each job is performed by one or more MapReduce tasks run on a Hadoop cluster.
The SeARuM architecture contains the following jobs, described in details in the subsequent sections:

- Network measurement acquisition
- Data pre-processing
- Item frequency computation
- Itemset mining
- Rule extraction
- Rule aggregation and sorting

Since our case study is based on network traffic analysis, we thoroughly describe the SeARuM architecture in this application scenario. Furthermore, in the current design, SeARuM addresses a specific class of association rules, whose consequent consists of a single item.

9.3.1 Network measurement acquisition

The first step to analyse network traffic is is collecting network measurements. To this aim, a passive probe is located on the access link (vantage point) that connects an edge

network to the Internet. The passive probe sniffs all incoming and outgoing packets flowing on the link, i.e., packets directed to a node inside the network and generated by a node in the Internet, and vice versa. The probe runs Tstat [10, 155], a passive monitoring tool allowing network and transport layer measurement collection. Tstat rebuilds each TCP connection by matching incoming and outgoing segments. Thus, a flow-level analysis can be performed [155]. A TCP flow is identified by snooping the signaling flags (SYN, FIN, RST). The status of the TCP sender is rebuilt by matching sequence numbers on data segments with the corresponding acknowledgement (ACK) numbers.

To evaluate the SeARuM cloud-based service in a real-world application, we focus on a subset of measurements describing the traffic flow among the many provided by Tstat. The most meaningful features, selected with the support of domain experts, are detailed in the following:

- the *Round-Trip-Time* (RTT) observed on a TCP flow, i.e., the minimum time lag between the observation of a TCP segment and the observation of the corresponding ACK. RTT is strongly related to the distance between the two nodes
- the *number of hops* (Hop) from the remote node to the vantage point observed on packets belonging to the TCP flow, as computed by reconstructing the IP Time-To-Live[1]
- the *flow reordering probability* ($P\{reord\}$), which can be useful to distinguish different paths
- the *flow duplicate probability* ($P\{dup\}$), that can highlight a destination served by multiple paths[2]
- the total *number of packets* ($NumPkt$), the total *number of data packets* ($DataPkt$), and the total *number of bytes* ($DataBytes$) sent from both the client and the server, separately (the client is the host starting the TCP flow)
- the minimum ($WinMin$), maximum ($WinMax$), and scale ($WinScale$) values of the *TCP congestion window* for both the client and the server, separately
- the *TCP port* of the server ($Port$)
- the *class of service* ($Class$), as defined by Tstat, e.g., HTTP, video, VoIP, SMTP, etc.

[1] The initial TTL value is set by the source, typical values being 32, 64, 128 and 255.

[2] $P\{reord\}$ and $P\{dup\}$ are computed by observing the TCP sequence and acknowledgement numbers carried by segments of a given flow. We refer the reader to [155] for more details.

Based on measurements listed above, an input data record is defined by the following features: RTT, Hop, $P\{reord\}$, $P\{dup\}$, $NumPkt$, $DataPkt$, $DataBytes$, $WinMax$, $WinMin$, $WinScale$, $Port$, $Class$. To obtain reliable estimates on reordering and duplicate probabilities, only TCP flows which last more than $P = 10$ packets are considered. This choice allow focusing the analysis on long-lived flows, where the network path has a more relevant impact, thus providing more valuable information.

9.3.2 Data pre-processing

This step performs the following two activities:

- Value discretization
- Transactional format conversion

Associaton rule mining requires a transactional dataset of categorical values. The discretization step converts continuously valued measurements into categorical bins. Then, data are converted from the tabular to the transactional format. An example is reported in Table 9.1.
Automatic discretization approaches can exploit state-of-the-art techniques (e.g., clustering, statistical-based algorithms, etc.) to select appropriate bins depending on data distribution. These approaches yielded poorly significant bins on network data considered in this study. More specifically, the most frequent values were split into too many bins with respect to the real applicative interest. Hence, discretized bins are fixed-size and determined by domain experts based on the significance in the networking context. The fixed-size bins have been determined as follows:

- RTT: a bin each 5 ms for values from 0 ms to 200 ms, an additional bin for values higher than 200 ms.
- Hop: a bin for each value from 1 to 20, an additional bin for values exceeding 20.
- $P\{reord\}$: a bin each 0.1 from 0 to 1.
- $P\{dup\}$: a bin each 0.1 from 0 to 1.
- $NumPkt$, $DataPkt$, and $DataBytes$: logarithmic bins, base 10, e.g., 5432 falls in the 3-4 bin since the value is between 10^3 and 10^4.
- $WinMax$ and $WinMin$: a bin for each multiple N of 4 Kb, where N is a power of 2, e.g., the bin 8-16 means that the TCP window is between 8 and 16 times 4 Kb.
- $WinScale$, $Port$, and $Class$: a bin for each value (no discretization).

Both the value discretization and the transactional format conversion are performed by a single map only job. Each record is processed by the map function and, if the number of packets is above the threshold (10 packets), the corresponding discretized transaction is emitted as a result of the mapping. This task entails an inherently parallel elaboration, considering that can be applied independently to each record.

	RTT	$NumPkt$	$P\{reord\}$
original	7	5432	0.88
discretized	5-10	3-4	0.9
transactional	RTT=5-10	$NumPkt$=3-4	$P\{reord\}$=0.9

Table 9.1: Pre-processing example

9.3.3 Item frequency computation

A second job is exploited to compute the item frequency from the transactions emitted by the pre-processing phase. An example is reported in Tables 9.2 and 9.3. Table 9.2 has three sample transactions that represent a possible output of the pre-processing phase. A map function is exploited to process each transaction: the map emits a (*key*, *value*) pair for each item in the transaction, where the *key* is the item itself (e.g., RTT=5-10), and the *value* is its count, i.e., always 1. A reduce function is then executed to sum all the values for each *key*, hence computing the support count of each item. This is a typical group-by query performed as a distributed MapReduce job. As a running example, we will consider the sample result of this job reported in Table 9.3, as obtained by the sample transactions in Table 9.2.

transaction 1	RTT=5-10	$NumPkt$=3-4	Hop=10
transaction 2	RTT=5-10		Hop=11
transaction 3	RTT=5-10	$NumPkt$=3-4	
transaction 3	RTT=5-10	$NumPkt$=3-4	Hop=11

Table 9.2: Sample transactions

item	**sup count**	**sup**
RTT=5-10	4	100%
$NumPkt$=3-4	3	75%
Hop=10	1	25%
Hop=11	2	50%

Table 9.3: Sample items

9.3.4 Itemset mining

A third job performs the itemset mining by exploiting the parallel FP-growth algorithm, as described in [151]. This step consists of multiple MapReduce tasks. From the sample items of Table 9.3, a result of this job is reported in Table 9.4, where only itemsets with support higher than 50% have been extracted.

ID	itemset			sup count	sup
1	RTT=5-10			4	100%
2	RTT=5-10	$NumPkt$=3-4		3	75%
3	RTT=5-10		Hop=11	2	50%
4		$NumPkt$=3-4		3	75%
5			Hop=11	2	50%

TABLE 9.4: Sample itemsets

9.3.5 Rule extraction

The rule extraction step, to the best of the authors knowledge, is a novel contribution as a distributed cloud-based service. It consists of a MapReduce job, as detailed in the following. For each itemset of length k (k-itemset), the map function emits:

- a (key,$value$) pair with
  - key: the k-itemset itself
  - $value$: the k-itemset support count
- for each $(k-1)$-itemset, a (key,$value$) pair with
  - key the $(k-1)$-itemset
  - $value$ the pair (k-itemset, support count of the k-itemset).

Then, the reduce function performs the actual rule extraction. Since each $(k-1)$-itemset emitted as key contains its k-itemset and the k-itemset support count as value, the missing item in the $(k-1)$-itemset with respect to the k-itemset is the rule consequent (head), whereas the $(k-1)$-itemset is the antecedent (rule body). The support count values of the k-itemset, the $(k-1)$-itemset and the consequent item are used to compute the support, confidence, and lift of the rule, as defined in Section 9.2. Table 9.5 reports the rules extracted from the itemsets of the running example (see Table 9.4).

9.3.6 Rule aggregation and sorting

A final step is executed by means of a MapReduce job to sort and aggregate the rules according to the consequent and the quality measure. As discussed in Section 9.2, we

rule	sup count	sup	conf	lift
RTT=5-10 → $NumPkt$=3-4	3	75%	75%	0.75
$NumPkt$=3-4 → RTT=5-10	3	75%	100%	1.33
RTT=5-10 → Hop=11	2	50%	50%	0.50
Hop=11 → RTT=5-10	2	50%	100%	2.00

Table 9.5: Sample rules

selected the lift as rule quality measure. Sorting and aggregating on the consequent helps in analyzing the extracted rules for finding significant correlations. A sample output based on our running example is reported in Table 9.6.

antecedent		consequent
Hop=11, $NumPkt$=3-4	→	RTT=5-10
RTT=5-10	→	$NumPkt$=3-4
RTT=5-10	→	Hop=11

Table 9.6: Sample rules, sorted and aggregated

9.4 Experimental results

A set of preliminary experiments have been performed analyzing SeARuM behavior on real datasets. SeARuM has been applied to two real datasets obtained by performing different capture stages on a backbone link of a nation-wide ISP in Italy that offers us three different vantage points. ISP vantage points expose traffic of three different Points-of-Presence (POP) in different cities in Italy; each PoP aggregates traffic from more than 10,000 ISP customers, which range from home users to Small Office Home Office (SOHO) accessing the Internet via ADSL or Fiber-To-The-Home technology. It represents therefore a very heterogeneous scenario. We will refer to each dataset as D1 or D2 as shown in Table 9.7, where the number of TCP flows and the size of each dataset are also reported.

Dataset	Number of TCP flows	Size [Gbyte]
D1	11,325,006	5.28
D2	413,012,989	192.56

Table 9.7: Network traffic datasets

MapReduce jobs of the SeARuM workflow (see Section 9.3) have been developed in Java using the Hadoop Java API. Part of the code has been developed as an extension of the Apache Mahout project [156], which provides a limited implementation of the parallel itemset mining algorithm FP-Growth to mine the top-k closed itemsets [151].

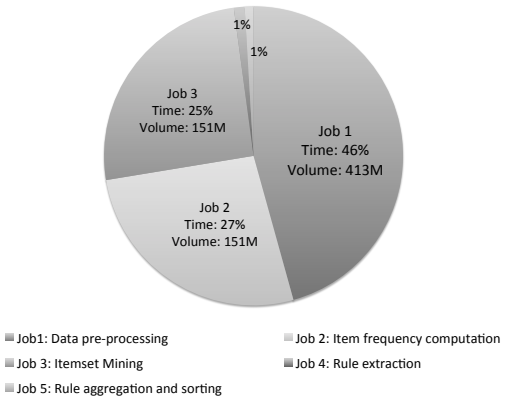


FIGURE 9.1: Dataset D2: Execution time distribution among jobs for MinSup=30% and MinConf=50%

Rule extraction, instead, is a novel contribution of the authors. Experiments have been performed on a cluster of 5 nodes running Cloudera's Distribution of Apache Hadoop (CDH4). Each cluster node is a 2.67 GHz six-core Intel(R) Xeon(R) X5650 machine with 32 Gbyte of main memory running Ubuntu 12.04 server with the 3.5.0-23-generic kernel. All reported execution times are real times, including both system and user time, obtained from the Cloudera Hadoop web administration control panel.

9.4.1 Execution time distribution among jobs

Since SeARuM consists of a sequential workflow, we analyzed how much time is spent at each step. Figure 9.1 shows the percentage of the total execution time for each job of the SeARuM architecture.
The pre-processing job represents the most expensive step with almost 50% of the time. This behavior is due to the higher data volume to be processed at this stage with respect

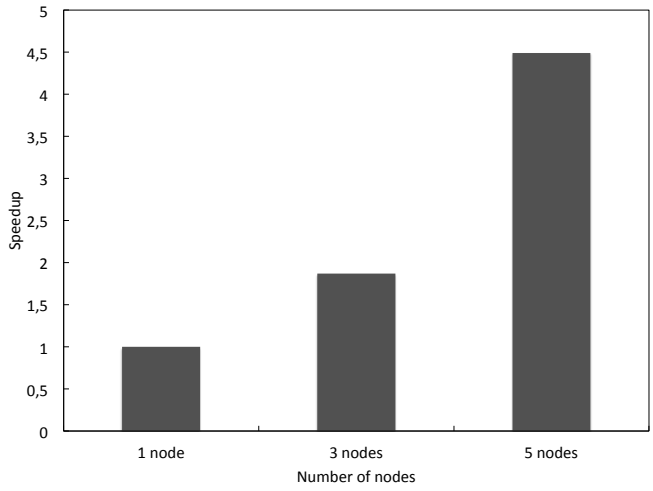
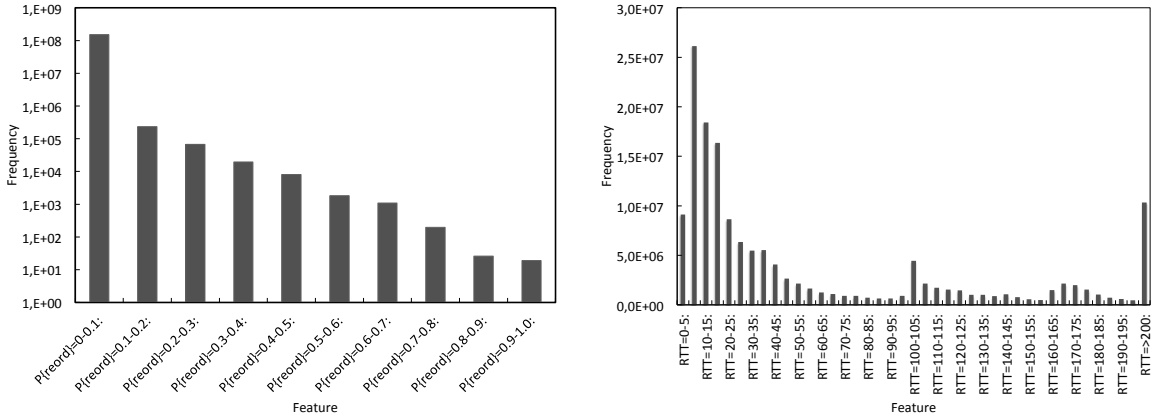


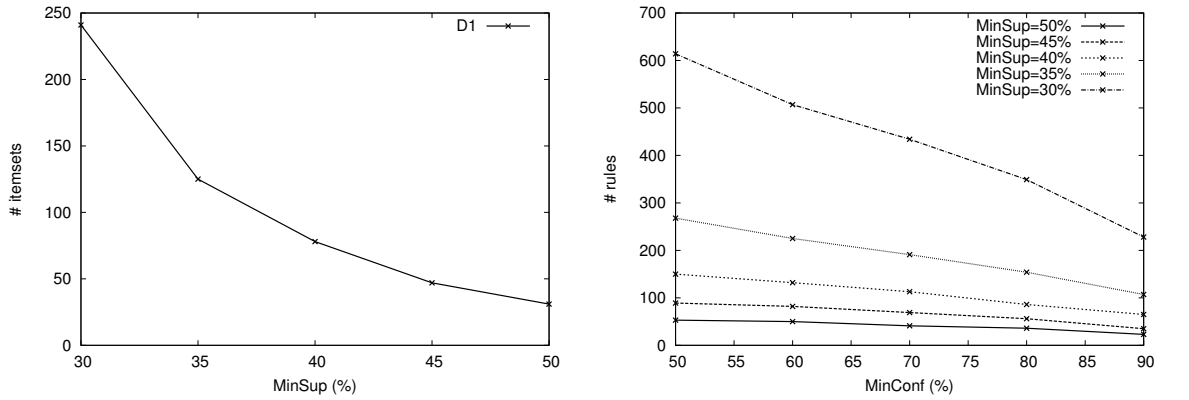
FIGURE 9.2: SeARuM speedup on D2 dataset



(a) Item distribution for the $P\{reord\}$ feature

(b) Item distribution for the RTT feature

FIGURE 9.3: Dataset D2



(a) Number of extracted itemsets for different $MinSup$ values

(b) Number of extracted rules for different values of $MinConf$ and $MinSup$

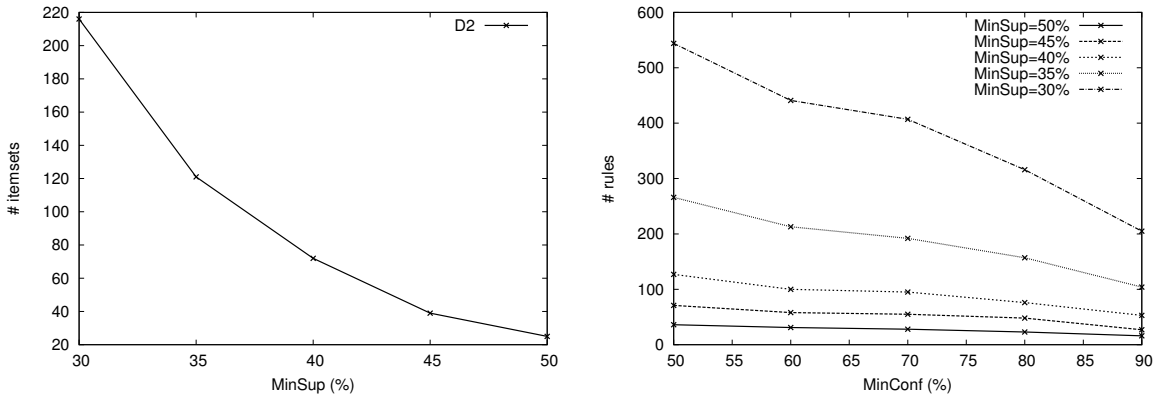
FIGURE 9.4: Dataset D1: Effect of $MinSup$ and $MinConf$ thresholds

to the subsequent steps: pre-processing filters flows with less than 10 packets, thus reducing the data volume from 413 millions of records to 151 millions of transactions. Note that the pre-processing job is executed only once for each discretization bin set, while it provides results that can be used many times by subsequent jobs, e.g., to mine itemsets and rules with different $MinSup$ and $MinConf$ constraints.

When the $MinSup$ value decreases, the computational complexity of the itemset mining job may significantly increase, since a very large number of itemsets may be generated. As a reference, Figure 9.1 reports times for MinSup=30% and MinConf=50% on dataset D2.

9.4.2 Evaluation of association rule mining

To evaluate the effectiveness of SeaRuM in association rule mining, we measured the achieved speedup for different numbers of nodes. We considered 3 configurations: 1



(a) Number of extracted itemsets for different values of $MinSup$

(b) Number of extracted rules for different values of $MinConf$ and $MinSup$

FIGURE 9.5: Dataset D2: Effect of $MinSup$ and $MinConf$ thresholds

node, 3 nodes, and 5 nodes. Dataset D2 has been used since it is the largest. Figure 9.2 reports the speedup when $MinSup = 30\%$ and $MinConf = 50\%$ are applied.

The first histogram in the figure (i.e., 1 node) corresponds to an execution of SeARuM on a single node. The speedup evaluation for increasing numbers of nodes is compared to the single-node performance.

The achieved speedup is the result of job distribution. The contribution of the former is especially relevant when considering large datasets and/or low minimum support thresholds ($MinSup$), which make the mining activity more computationally intensive. As reported in Figure 9.2 the SeARuM performance progressively improves when distributing the mining task on an increasing number of nodes. For instance, a 5-node cluster achieves a speedup of 4.5, showing that SeARuM has promising attitude to scale in larger clouds.

9.4.3 Network knowledge characterization

We evaluated the effectiveness of the proposed approach on real network traffic traces. In particular, we analyzed: (i) the usefulness of the extracted association rules in supporting the knowledge discovery process, and (ii) the item frequency distribution.

As example, the following two rules R1 and R2 are generated from dataset D1 and D2, respectively. Both rule have high confidence values and lift greater than 1 (rule support, confidence, and lift are reported in brackets after each rule).

$R_1 : \{Port = 80,\ P\{reord\} = 0 - 0.1,\ DataPkt = 1 - 2,\ DataBytes = 4 - 5\} \rightarrow Class = HTTP$ (0.313, 0.999, 1.765)

$R_2 : \{P\{dup\} = 0 - 0.1,\ NumPkt \leq 1,\ DataPkt \leq 1,\ Class = SSL\} \rightarrow Port = 443$ (0.013, 0.993, 4.944)

Based on rule R_1, the HTTP protocol is mainly used to transmit a set of TCP flows sent by the server through the TPC port 80. For these flows, the number of packets is in the range 10÷100 and a large number of bytes is transmitted (from 10,000 to 100,000). These flows can be generated when very large files are downloaded (e.g., YouTube videos).
Rule R_2 reports that the TCP *Port* 443 (HTTPS) is mainly used to transmit flows with SSL/TLS coded protocol and less than 10 packets. These flows can be generated when logging into websites through a secure connection (e.g., Facebook, Twitter).
We also analyzed the item frequency distribution to characterize the network activity. Figure 9.3 considers the Round-Trip-Time (RTT) and the flow reordering probability ($P\{reord\}$), which are discussed as representative features.
The item distribution for the $P\{reord\}$ feature is characterized by a very frequent item which models most TCP flows: they have a very low $P\{reord\}$, i.e., from 0 to 0.1. This data distribution analyzed over time and for different (sub)networks may be exploited to identify periods of time or (sub)networks that become less reliable or whose packets change path more frequently than usual.
The item distribution for the RTT, instead, shows four peaks:

- the first peak around 5-20 ms may represent local network traffic
- the second peak around 100 ms may represent external traffic inside the same ISP or in the same geographical zone (e.g., country, continent)
- the third peak around 170 ms may represent traffic towards long-distance destinations (e.g., other continents)
- finally, the last peak over 200 ms may represent network problems or unresponsive services

9.4.4 Effect of the support and confidence thresholds

Minimum support ($MinSup$) and confidence ($MinConf$) thresholds significantly affect the number of extracted itemsets and association rules.
When decreasing the $MinSup$ value, the number of frequent itemsets grows non linearly [65] and the complexity of the frequent itemset extraction task significantly increases. High $MinConf$ values represent a tighter constraint on rule selection. Consequently, when increasing $MinConf$ less rules are mined, but these rules tend to represent stronger correlations among data. High $MinConf$ values should be often combined with low $MinSup$ values to lead the extraction of peculiar (i.e., not very frequent) but highly correlated rules.
Figures 9.4(a) and 9.5(b) plot, for the two reference datasets, the number of extracted itemsets when varying $MinSup$. Figure 9.4(b) and 9.5(b) report the number of association rules for different $MinConf$ values.

Chapter 10

Conclusions

The Internet is a complex and dynamic system with new protocols and applications that arise at a constant pace. All these characteristics designate the Internet a valuable and challenging data source and application domain for a research activity, both looking at Transport layer, analyzing network traffic flows, and going up to Application layer, studying the ever-growing next generation services: blogs, micro-blogs, on-line social networks, photo sharing services and many other applications (*e.g.*, Twitter, Facebook, Flickr, etc.).

In this thesis work we focused on the study, design and development of novel algorithms and frameworks to support large scale data mining activities over huge and heterogeneous data volumes, with a particular focus on Internet data as data source and targeting network traffic classification, on-line social network analysis, recommendation systems and cloud services and Big data.

At the beginning we addressed our research to the Transport layer and we presented a novel Hierarchical classifier, based on classification algorithms, that achieves excellent results even when considering fine grained classification of Internet TCP flows. We implemented a proper feature selection, selected the best approach among 7 different classification algorithms, tested the approach by ten-fold cross validation and t-test to check the significance of the experiments. Considering real traffic traces captured from operative networks, we have shown the benefit of using a hierarchical approach: i) classification performance is boosted, ii) computational complexity is reduced, iii) robustness to the training set is achieved. Results demonstrate that behavioral classifiers can be finally considered a reliable means for fine grained traffic classification in the real world.

Afterwards, to cope with the limitations of a supervised approach, we introduced SeLeCT, a semi-automated Internet flow traffic classifier which leverages unsupervised clustering algorithms to automatically groups flows into clusters. Given that using unsupervised clustering algorithms does not result in high accuracy, we showed that adding

a filtering phase after clustering significantly improves the performance and coverage. Moreover, alternating the clustering and filtering phases further results in very homogeneous clusters, while providing very high coverage. Labels for different clusters in SeLeCT can be bootstrapped using several different approaches (DPI, behavioral techniques, or human-in-the-middle). Once labels for some flows are provided, SeLeCT inherits previously labeled flows to automatically label new clusters. Furthermore, it adapts the model to traffic changes, and is able to automatically increase its knowledge. Extensive experiments showed that SeLeCT simplifies the manual bootstrapping of labels that, once provided to the system, lead to excellent performance: accuracy is close to 98% in most datasets, with worst case still higher than 90%. Furthermore, SeLeCT was able to automatically identify classes of traffic that an advanced DPI-based classifier was ignoring like, e.g., the Apple iOS push notification protocol, or some Bot/Trojan traffic.

Next, we moved our attention to the Application Layer. In this context, in the last years, on-line social network analysis has attracted the attention of different research communities, including database, information retrieval, pattern recognition, and data mining. Many research efforts have significantly contributed to improve the applicability of data mining techniques to social network and online community analysis.

Hence, we focus our research on the micro-blogging service Twitter. First, we described a framework for the analysis of Twitter data aimed at discovering, in a compact form, the information posted by users about an event as well as the user perception of the event. Our experimental evaluation, performed on two real datasets, shows the effectiveness of the approach in discovering interesting knowledge. Other interesting future research directions, to further improve the performance of our framework, will be considering also the additional features (e.g., GPS coordinates) available in Twitter data. Furthermore, a real-time and distributed analysis of Twitter data can be addressed to support the analysis of huge data collection, also regarding parallel events.

A related research topics is the analysis of the dynamics behind the evolution of social network data. We proposed TwiChi, a data mining system that addresses Twitter user-generated content mining targeted to user behavior and topic trend analysis. A dynamic data mining algorithm is exploited to mine patterns which represent the evolution of most significant correlations among data in consecutive time periods. To avoid discarding relevant patterns that suddenly become infrequent in a certain time period, a taxonomy is used to generalize patterns at a higher level of abstraction and represent their associated knowledge at the proper generalization level. TwiChi performance has been evaluated on real-life Twitter datasets. The usability and functionality of the proposed system have been validated in different use cases, among which topic trend detection and user behavior analysis. The discovered HiGens have shown to be particularly suitable for supporting the experts in the analysis of the Twitter UGC, because they

may be exploited to drive domain experts in performing a number of different targeted actions. The proposed approach could be extended in a number of directions, among which (i) the automatic generation of taxonomies suitable for driving the generalization process, (ii) the pushing of user-provided constraints to reduce the amount of extracted patterns, and (iii) the application of the proposed framework to perform context-aware user profiling and shaping of social network services. The taxonomies exploited to generalize items are usually provided by a domain expert. Since the taxonomy generation process is a complex task, automatic or semi-automatic taxonomy generation algorithms should be developed and integrated in the proposed system. Finally, the discovered patterns may be also deemed suitable for performing social network service shaping. In particular, based on the context in which the user has published the UGC, the proposed system may suit service provision to the actual user needs.

Moreover, taking a different angle from the mainstream of previous works, we presented TUCAN, a framework to graphically represent semantic correlations of individual Twitter users' timelines. Building on text mining techniques, TUCAN analyses "bird songs", *i.e.*, group of tweets belonging to the same time period, and compares their similarity. The analyst is offered a GUI to investigate the impact of different pre-processing. Experiments conducted on actual Twitter users show the ability to pinpoint recurrent topics, or correlations among users. There are several avenues for future work. First, we would like to expand our framework to be able to model patterns of topic durations and transitions. Leveraging the measurements revealing the correlation durations of topics, accumulating the statics for long-term can reflect changes in the user's interests. Second, we are interested in inferring users' social relationships based on their topical relations.

A further challenging research topic addressed in the Application layer during this thesis work is Tag recommendation, that is the task of predicting folksonomy tags for a given user and item, based on past user behavior (and possibly other information). Tag recommendation is focused on recommending useful tags to a user who is annotating a Web resource. We presented a novel personalized tag recommendation system that performs additional tag recommendations to partially annotated Flickr photos by exploiting generalized association rules extracted from the collections of the past personal and collective annotations. The use of high level associations is focused on counteracting the impact of data sparsity, as it may highlight correlations among tags that could remain hidden at the level of individual tags. A set of experiments has been conducted on real-life Flickr photo collections. The effectiveness of the proposed approach has been validated against a recently proposed tag recommendation system. Experiments show that the use of the generalizations in rule-based tag recommendation yields significant performance improvements. Our system has so far not been concerned with the analysis of the textual content related to the annotated Web resources (e.g., photo descriptions, related blogs or articles). We plan to extend it by also considering the user-generated

textual content coming from social networks and online communities. Furthermore, to take the evolution of photo annotations over time we will investigate the integration of incremental rule mining approaches as well.

As we have seen through many of our proposed approach, frequent generalized itemset mining is a data mining technique utilized to discover a high-level view of interesting knowledge hidden in the analyzed data. By exploiting a taxonomy, patterns are usually extracted at any level of abstraction. However, some misleading high-level patterns could be included in the mined set. We proposed a novel generalized itemset type, called Misleading Generalized Itemsets (MGIs), from structured datasets that are supplied with taxonomies. MGIs represent misleading frequent high-level data correlations that are worth analyzing apart from traditional itemsets. Each MGI represents a frequent generalized itemset X and its subset of low-level frequent descendants for which the correlation type is in contrast to those of X. An MGI interestingness measure, named Not Overlapping Degree (NOD), is also proposed to select only the (misleading) high-level itemsets that represent almost the same portion of data that is covered by their low-level contrasting correlations. Furthermore, an algorithm to mine MGIs at the top of the traditional itemsets has also been proposed. The experimental results demonstrate the usefulness of the proposed approach for discovering interesting misleading itemsets from real mobile datasets. We plan to extend our research work in the following directions: (i) the study of the applicability of the proposed approach to other real-life contexts (e.g. social network analysis [76], medical data analysis [157]), (ii) the use of different correlation measures (e.g. coherence [124]), and (iii) the pushing of more complex mining constraints into the MGI extraction process.

Finally, to cope with the large volume of data that are being produced by various modern web applications we presented our first implementation of a cloud-based service for association rule mining. Both the horizontal scalability of the approach and the meaningfulness of the extracted knowledge have been addressed. Since preliminary experiments performed on two real datasets showed promising results, a more complete and optimized implementation of the proposed approach as a SaaS (Software-as-a-Service) service model may be envisioned in the long term to offer an efficient association rule mining algorithm to cloud users. In particular, future works will aim at optimizing the MapReduce workflow and expanding the workbench of the cluster architecture.

Bibliography

[1] Luigi Grimaudo, Marco Mellia, and Elena Baralis. Hierarchical learning for fine grained internet traffic classification. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2012 8th International*, pages 463–468. IEEE, 2012.

[2] Luigi Grimaudo, Marco Mellia, Elena Baralis, and Ram Keralapura. SeLeCT: Self-learning classifier for internet traffic. *Network and Service Management, IEEE Transactions on*, 2014.

[3] Elena Baralis, Tania Cerquitelli, Silvia Chiusano, Luigi Grimaudo, and Xin Xiao. Analysis of twitter data using a multiple-level clustering strategy. In *Model and Data Engineering*, pages 13–24. Springer, 2013.

[4] Luca Cagliero, Luigi Grimaudo, and Alessandro Fiori. Analyzing twitter user-generated content changes.

[5] Luigi Grimaudo, Han Song, Mario Baldi, Marco Mellia, and Maurizio Munafo. Tucan: Twitter user centric analyzer. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1455–1457. ACM, 2013.

[6] Luca Cagliero, Alessandro Fiori, and Luigi Grimaudo. Personalized tag recommendation based on generalized rules. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):12, 2013.

[7] Luca Cagliero, Tania Cerquitelli, Paolo Garza, and Luigi Grimaudo. Misleading generalized itemset discovery. *Expert Systems with Applications*, 41(4):1400–1410, 2014.

[8] Daniele Apiletti, Elena Baralis, Tania Cerquitelli, Silvia Chiusano, and Luigi Grimaudo. Searum: a cloud-based service for association rule mining. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pages 1283–1290. IEEE, 2013.

[9] Hyunchul Kim, Kimberly C Claffy, Marina Fomenkov, Dhiman Barman, Michalis Faloutsos, and KiYoung Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proceedings of the 2008 ACM CoNEXT conference*, page 11. ACM, 2008.

[10] Alessandro Finamore, Marco Mellia, Michela Meo, Maurizio M Munafo, and Dario Rossi. Experiences of internet traffic monitoring with tstat. *Network, IEEE*, 25 (3):8–14, 2011.

[11] Tstat home page. URL http://tstat.polito.it.

[12] Marcin Pietrzyk, Jean-Laurent Costeux, Guillaume Urvoy-Keller, and Taoufik En-Najjary. Challenging statistical classification for operational usage: the adsl case. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 122–135. ACM, 2009.

[13] Pang-Ning Tan et al. *Introduction to data mining*. Pearson Education, 2006.

[14] Ingo Mierswa, Michael Wurst, Ralf Klinkenberg, Martin Scholz, and Timm Euler. Yale: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 935–940. ACM, 2006.

[15] Thomas Karagiannis, Konstantina Papagiannaki, and Michalis Faloutsos. Blinc: multilevel traffic classification in the dark. In *ACM SIGCOMM Computer Communication Review*, volume 35, pages 229–240. ACM, 2005.

[16] Laurent Bernaille, Renata Teixeira, and Kave Salamatian. Early application identification. In *Proceedings of the 2006 ACM CoNEXT conference*, page 6. ACM, 2006.

[17] David J Hand. Principles of data mining. *Drug safety*, 30(7):621–622, 2007.

[18] Hanchuan Peng, Fuhui Long, and Chris Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005.

[19] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008.

[20] Thomas G Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural computation*, 10(7):1895–1923, 1998.

[21] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, Ira Cohen, and Carey Williamson. Offline/realtime traffic classification using semi-supervised learning. *Performance Evaluation*, 64(9):1194–1213, 2007.

[22] Xiaojin Zhu. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, 2:3, 2006.

[23] Ayhan Demiriz, Kristin P Bennett, and Mark J Embrechts. Semi-supervised clustering using genetic algorithms. *Artificial neural networks in engineering (ANNIE-99)*, pages 809–814, 1999.

[24] Rozita Dara, SC Kremer, and DA Stacey. Clustering unlabeled data with soms improves classification of labeled real-world data. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 3, pages 2237–2242. IEEE, 2002.

[25] Anthony McGregor, Mark Hall, Perry Lorier, and James Brunskill. Flow clustering using machine learning techniques. In *Passive and Active Network Measurement*, pages 205–214. Springer, 2004.

[26] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Internet traffic identification using machine learning. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE*, pages 1–6. IEEE, 2006.

[27] 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.

[28] Yu Wang, Yang Xiang, and Shun-Zheng Yu. An automatic application signature construction system for unknown traffic. *Concurrency and Computation: Practice and Experience*, 22(13):1927–1944, 2010.

[29] Jing Yuan, Zhu Li, and Ruixi Yuan. Information entropy based clustering method for unsupervised internet traffic classification. In *Communications, 2008. ICC'08. IEEE International Conference on*, pages 1588–1592. IEEE, 2008.

[30] Pedro Casas, Johan Mazel, and Philippe Owezarski. Minetrac: mining flows for unsupervised analysis & semi-supervised classification. In *Proceedings of the 23rd International Teletraffic Congress*, pages 87–94. ITCP, 2011.

[31] Jun Zhang, Chao Chen, Yang Xiang, Wanlei Zhou, and A Vasilakos. An effective network traffic classification method with unknown flow detection. 2013.

[32] Jun Zhang, Yang Xiang, Wanlei Zhou, and Yu Wang. Unsupervised traffic classification using flow statistical properties and ip packet payload. *Journal of Computer and System Sciences*, 79(5):573–585, 2013.

[33] Gianluca Iannaccone, Christophe Diot, Ian Graham, and Nick McKeown. Monitoring very high speed links. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 267–271. ACM, 2001.

[34] Ionut Trestian, Supranamaya Ranjan, Aleksandar Kuzmanovic, and Antonio Nucci. Googling the internet: profiling internet endpoints via the world wide web. *IEEE/ACM Transactions on Networking (TON)*, 18(2):666–679, 2010.

[35] Pedro Casas, Pierdomenico Fiadino, and Arian Bar. Ip mining: Extracting knowledge from the dynamics of the internet addressing space. In *Teletraffic Congress (ITC), 2013 25th International*, pages 1–9. IEEE, 2013.

[36] Marios Iliofotou, Michalis Faloutsos, and Michael Mitzenmacher. Exploiting dynamicity in graph-based traffic analysis: techniques and applications. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 241–252. ACM, 2009.

[37] Qing Li, Jia Wang, Yuanzhu Peter Chen, and Zhangxi Lin. User comments for news recommendation in forum-based social media. *Information Sciences*, 180 (24):4929–4939, 2010.

[38] Andriy Shepitsen, Jonathan Gemmell, Bamshad Mobasher, and Robin Burke. Personalized recommendation in social tagging systems using hierarchical clustering. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 259–266. ACM, 2008.

[39] Yuan Xue, Chen Zhang, Changzheng Zhou, Xun Lin, and Qing Li. An effective news recommendation in social media based on users' preference. In *Education Technology and Training, 2008. and 2008 International Workshop on Geoscience and Remote Sensing. ETT and GRS 2008. International Workshop on*, volume 1, pages 627–631. IEEE, 2008.

[40] Gjergji Kasneci, Maya Ramanath, Fabian Suchanek, and Gerhard Weikum. The yago-naga approach to knowledge discovery. *ACM SIGMOD Record*, 37(4):41–47, 2009.

[41] Ho-Yu Lam and Dit-Yan Yeung. *A learning approach to spam detection based on social networks.* PhD thesis, Hong Kong University of Science and Technology, 2007.

[42] Xin Li, Lei Guo, and Yihong Eric Zhao. Tag-based social interest discovery. In *Proceedings of the 17th international conference on World Wide Web*, pages 675–684. ACM, 2008.

[43] Michael Mathioudakis and Nick Koudas. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1155–1158. ACM, 2010.

[44] Marc Cheong and Vincent Lee. Integrating web-based intelligence retrieval and decision-making from the twitter trends knowledge base. In *Proceedings of the 2nd ACM workshop on Social web search and mining*, pages 1–8. ACM, 2009.

[45] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, volume 96, pages 226–231, 1996.

[46] Michael Steinbach, George Karypis, Vipin Kumar, et al. A comparison of document clustering techniques. In *KDD workshop on text mining*, volume 400, pages 525–526. Boston, 2000.

[47] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM SIGMOD Record*, volume 29, pages 1–12. ACM, 2000.

[48] Zhijun Yin, Rui Li, Qiaozhu Mei, and Jiawei Han. Exploring social tagging graph for web object classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 957–966. ACM, 2009.

[49] Matthias Bender, Tom Crecelius, Mouna Kacimi, Sebastian Michel, Thomas Neumann, Josiane Xavier Parreira, Ralf Schenkel, and Gerhard Weikum. Exploiting social relations for query expansion and result ranking. In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 501–506. IEEE, 2008.

[50] Paul Heymann, Daniel Ramage, and Hector Garcia-Molina. Social tag prediction. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 531–538. ACM, 2008.

[51] Foteini Alvanaki, Sebastian Michel, Krithi Ramamritham, and Gerhard Weikum. See what's enblogue: real-time emergent topic identification in social media. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 336–347. ACM, 2012.

[52] Luca Cagliero and Alessandro Fiori. Discovering generalized association rules from twitter. *Intelligent Data Analysis*, 17(4):627–648, 2013.

[53] AA Lopes, Roberto Pinho, Fernando Vieira Paulovich, and Rosane Minghim. Visual text mining using association rules. *Computers & Graphics*, 31(3):316–326, 2007.

[54] Qing Chen, Timothy Shipper, and Latifur Khan. Tweets mining using wikipedia and impurity cluster measurement. In *Intelligence and Security Informatics (ISI), 2010 IEEE International Conference on*, pages 141–143. IEEE, 2010.

[55] Sungchul Kim, Sungho Jeon, Jinha Kim, Young-Ho Park, and Hwanjo Yu. Finding core topics: Topic extraction with clustering on tweet. In *Cloud and Green Computing (CGC), 2012 Second International Conference on*, pages 777–782. IEEE, 2012.

[56] Kumar Subramani, Alexander Velkov, Irene Ntoutsi, P Kroger, and H Kriegel. Density-based community detection in social networks. In *Internet Multimedia Systems Architecture and Application (IMSAA), 2011 IEEE 5th International Conference on*, pages 1–8. IEEE, 2011.

[57] 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.

[58] Dario Antonelli, Elena Baralis, Giulia Bruno, Tania Cerquitelli, Silvia Chiusano, and Naeem Mahoto. Analysis of diabetic patients through their examination history. *Expert Systems with Applications*, 2013.

[59] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. Wiley. com, 2009.

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

[61] Luca Cagliero. Discovering temporal change patterns in the presence of taxonomies. *Knowledge and Data Engineering, IEEE Transactions on*, 25(3):541–555, 2013.

[62] Ramakrishnan Srikant and Rakesh Agrawal. Mining generalized association rules. In *VLDB*, volume 95, pages 407–419, 1995.

[63] Mario Baldi, Elena Baralis, and Fulvio Risso. Data mining techniques for effective and scalable traffic analysis. In *Integrated Network Management, 2005. IM 2005. 2005 9th IFIP/IEEE International Symposium on*, pages 105–118. IEEE, 2005.

[64] Elena Baralis, Luca Cagliero, Tania Cerquitelli, Paolo Garza, and Marco Marchetti. Context-aware user and service profiling by means of generalized association rules. In *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 50–57. Springer, 2009.

[65] 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. ISBN 1-55860-153-8. URL http://dl.acm.org/citation.cfm?id=645920.672836.

[66] Jiawei Han and Yongjian Fu. Mining multiple-level association rules in large databases. *Knowledge and Data Engineering, IEEE Transactions on*, 11(5):798–805, 1999.

[67] Jochen Hipp, Andreas Myka, Rüdiger Wirth, and Ulrich Güntzer. *A new algorithm for faster mining of generalized association rules.* Springer, 1998.

[68] Iko Pramudiono and Masaru Kitsuregawa. Fp-tax: Tree structure based generalized association rule mining. In *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 60–63. ACM, 2004.

[69] Kritsada Sriphaew and Thanaruk Theeramunkong. A new method for finding generalized frequent itemsets in generalized association rule mining. In *Computers and Communications, 2002. Proceedings. ISCC 2002. Seventh International Symposium on*, pages 1040–1045. IEEE, 2002.

[70] Elena Baralis, Luca Cagliero, Tania Cerquitelli, Vincenzo D'Elia, and Paolo Garza. Support driven opportunistic aggregation for generalized itemset extraction. In *Intelligent Systems (IS), 2010 5th IEEE International Conference*, pages 102–107. IEEE, 2010.

[71] Steffan Baron, Myra Spiliopoulou, and Oliver Günther. Efficient monitoring of patterns in data mining environments. In *Advances in Databases and Information Systems*, pages 253–265. Springer, 2003.

[72] Mirko Böttcher, Detlef Nauck, Dymitr Ruta, and Martin Spott. Towards a framework for change detection in data sets. In *Research and Development in Intelligent Systems XXIII*, pages 115–128. Springer, 2007.

[73] Bing Liu, Yiming Ma, and Ronnie Lee. Analyzing the interestingness of association rules from the temporal dimension. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 377–384. IEEE, 2001.

[74] Yingying Tao and M Tamer Özsu. Mining frequent itemsets in time-varying data streams. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 1521–1524. ACM, 2009.

[75] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user navigation and interactions in online social networks. *Information Sciences*, 195:1–24, 2012.

[76] Lei Guo, Enhua Tan, Songqing Chen, Xiaodong Zhang, and Yihong Eric Zhao. Analyzing patterns of user content generation in online social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 369–378. ACM, 2009.

[77] Owen Phelan, Kevin McCarthy, Mike Bennett, and Barry Smyth. Terms of a feather: Content-based news recommendation and discovery using twitter. In *Advances in Information Retrieval*, pages 448–459. Springer, 2011.

[78] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65. ACM, 2007.

[79] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM, 2010.

[80] Liangjie Hong and Brian D Davison. Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*, pages 80–88. ACM, 2010.

[81] Daniel Ramage, Susan T Dumais, and Daniel J Liebling. Characterizing microblogs with topic models. In *ICWSM*, 2010.

[82] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

[83] Daniel Ramage, David Hall, Ramesh Nallapati, and Christopher D Manning. Labeled lda: A supervised topic model for credit attribution in multi-labeled corpora. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 248–256. Association for Computational Linguistics, 2009.

[84] Yan Liu, Alexandru Niculescu-Mizil, and Wojciech Gryc. Topic-link lda: joint models of topic and author community. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 665–672. ACM, 2009.

[85] Jonathan Chang, Jordan Boyd-Graber, and David M Blei. Connections between the lines: augmenting social networks with text. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM, 2009.

[86] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In *Advances in Information Retrieval*, pages 338–349. Springer, 2011.

[87] Xuan-Hieu Phan, Le-Minh Nguyen, and Susumu Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *Proceedings of the 17th international conference on World Wide Web*, pages 91–100. ACM, 2008.

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

[89] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494. AUAI Press, 2004.

[90] Michal Rosen-Zvi, Chaitanya Chemudugunta, Thomas Griffiths, Padhraic Smyth, and Mark Steyvers. Learning author-topic models from text corpora. *ACM Transactions on Information Systems (TOIS)*, 28(1):4, 2010.

[91] Anish Das Sarma, Alpa Jain, and Cong Yu. Dynamic relationship and event discovery. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 207–216. ACM, 2011.

[92] Alison Smith, Jianyu Li, Panagis Pano Papadatos, and Sana Malik. Topicflow: Visualizing topic alignment of twitter data over time. 2012.

[93] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[94] Christiane Fellbaum. Wordnet: An electronic lexical database. *WordNet is available from http://www. cogsci. princeton. edu/wn*, 1998.

[95] C Honey and Susan C Herring. Beyond microblogging: Conversation and collaboration via twitter. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–10. IEEE, 2009.

[96] Nikhil Garg and Ingmar Weber. Personalized, interactive tag recommendation for flickr. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 67–74. ACM, 2008.

[97] Börkur Sigurbjörnsson and Roelof Van Zwol. Flickr tag recommendation based on collective knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 327–336. ACM, 2008.

[98] Gilad Mishne. Autotag: a collaborative approach to automated tag assignment for weblog posts. In *Proceedings of the 15th international conference on World Wide Web*, pages 953–954. ACM, 2006.

[99] Shenghua Bao, Guirong Xue, Xiaoyuan Wu, Yong Yu, Ben Fei, and Zhong Su. Optimizing web search using social annotations. In *Proceedings of the 16th international conference on World Wide Web*, pages 501–510. ACM, 2007.

[100] Ritendra Datta, Weina Ge, Jia Li, and James Ze Wang. Toward bridging the annotation-retrieval gap in image search. *IEEE MultiMedia*, 14(3):24–35, 2007.

[101] Pavel A Dmitriev, Nadav Eiron, Marcus Fontoura, and Eugene Shekita. Using annotations in enterprise search. In *Proceedings of the 15th international conference on World Wide Web*, pages 811–817. ACM, 2006.

[102] Ralf Krestel, Peter Fankhauser, and Wolfgang Nejdl. Latent dirichlet allocation for tag recommendation. In *Proceedings of the third ACM conference on Recommender systems*, pages 61–68. ACM, 2009.

[103] Adam Rae, Börkur Sigurbjörnsson, and Roelof van Zwol. Improving tag recommendation using social networks. In *Adaptivity, Personalization and Fusion of Heterogeneous Information*, pages 92–99. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 2010.

[104] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, volume 22, pages 207–216. ACM, 1993.

[105] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[106] Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in folksonomies. In *Knowledge Discovery in Databases: PKDD 2007*, pages 506–514. Springer, 2007.

[107] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1):107–117, 1998.

[108] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. Tag recommendations based on tensor dimensionality reduction. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 43–50. ACM, 2008.

[109] Stefanie Lindstaedt, Viktoria Pammer, R Morzinger, Roman Kern, H Mulner, and Claudia Wagner. Recommending tags for pictures based on text, visual content and user context. In *Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on*, pages 506–511. IEEE, 2008.

[110] Paul-Alexandru Chirita, Stefania Costache, Wolfgang Nejdl, and Siegfried Handschuh. P-tag: large scale automatic generation of personalized annotation tags for the web. In *Proceedings of the 16th international conference on World Wide Web*, pages 845–854. ACM, 2007.

[111] Marek Lipczak and Evangelos Milios. Efficient tag recommendation for real-life data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(1):2, 2011.

[112] Ziyu Guan, Jiajun Bu, Qiaozhu Mei, Chun Chen, and Can Wang. Personalized tag recommendation using graph-based ranking on multi-type interrelated objects. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 540–547. ACM, 2009.

[113] Yu-Ta Lu, Shoou-I Yu, Tsung-Chieh Chang, and Jane Yung-jen Hsu. A content-based method to enhance tag recommendation. In *IJCAI*, pages 2064–2069, 2009.

[114] Guilherme Vale Menezes, Jussara M Almeida, Fabiano Belém, Marcos André Gonçalves, Anísio Lacerda, Edleno Silva De Moura, Gisele L Pappa, Adriano Veloso, and Nivio Ziviani. Demand-driven tag recommendation. In *Machine Learning and Knowledge Discovery in Databases*, pages 402–417. Springer, 2010.

[115] Jeremy Mennis and Jun Wei Liu. Mining association rules in spatio-temporal data: An analysis of urban socioeconomic and land cover change. *Transactions in GIS*, 9(1):5–17, 2005.

[116] Ramakrishnan Srikant, Quoc Vu, and Rakesh Agrawal. Mining association rules with item constraints. In *KDD*, volume 97, pages 67–73, 1997.

[117] Elmasri Ramez. *Fundamentals of database systems.* Pearson Education India, 1994.

[118] Merijn Van Erp and Lambert Schomaker. Variants of the borda count method for combining ranked classifier hypotheses. In *IN THE SEVENTH INTERNATIONAL WORKSHOP ON FRONTIERS IN HANDWRITING RECOGNITION.*

2000. AMSTERDAM LEARNING METHODOLOGY INSPIRED BY HUMAN'S INTELLIGENCE BO ZHANG, DAYONG DING, AND LING ZHANG. Citeseer, 2000.

[119] Mark J Huiskes and Michael S Lew. The mir flickr retrieval evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 39–43. ACM, 2008.

[120] Mark Sanderson and Justin Zobel. Information retrieval system evaluation: effort, sensitivity, and reliability. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 162–169. ACM, 2005.

[121] Charu C Aggarwal and Philip S Yu. A new framework for itemset generation. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 18–24. ACM, 1998.

[122] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: generalizing association rules to correlations. In *ACM SIGMOD Record*, volume 26, pages 265–276. ACM, 1997.

[123] Ashok Savasere, Edward Omiecinski, and Shamkant Navathe. Mining for strong negative associations in a large database of customer transactions. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 494–502. IEEE, 1998.

[124] Tianyi Wu, Yuguo Chen, and Jiawei Han. Re-examination of interestingness measures in pattern mining: a unified framework. *Data Mining and Knowledge Discovery*, 21(3):371–397, 2010.

[125] Marina Barsky, Sangkyum Kim, Tim Weninger, and Jiawei Han. Mining flipping correlations from large datasets with taxonomies. *Proceedings of the VLDB endowment*, 5(4):370–381, 2011.

[126] Uci repository of machine learning databases. URL http://archive.ics.uci.edu/ml.

[127] Daniel Kunkle, Donghui Zhang, and Gene Cooperman. Mining frequent generalized itemsets and generalized association rules without redundancy. *Journal of Computer Science and Technology*, 23(1):77–102, 2008.

[128] Kritsada Sriphaew and Thanaruk Theeramunkong. Fast algorithms for mining generalized frequent patterns of generalized association rules. *IEICE Transactions on Information and Systems*, 87(3):761–770, 2004.

[129] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Database Theory—ICDT'99*, pages 398–416. Springer, 1999.

[130] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 32–41. ACM, 2002.

[131] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. *Indirect association: Mining higher order dependencies in data*. Springer, 2000.

[132] Robert J Hilderman and Howard J Hamilton. Knowledge discovery and measures of interest. 2001.

[133] Chieh-Ming Wu and Yin-Fu Huang. Generalized association rule mining using an efficient data structure. *Expert Systems with Applications*, 38(6):7277–7290, 2011.

[134] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286, 1997.

[135] Komate Amphawan, Philippe Lenca, and Athasit Surarerks. Mining top-¡ i¿ k¡/i¿ regular-frequent itemsets using database partitioning and support estimation. *Expert Systems with Applications*, 39(2):1924–1936, 2012.

[136] Toon Calders and Bart Goethals. Mining all non-derivable frequent itemsets. In *Principles of Data Mining and Knowledge Discovery*, pages 74–86. Springer, 2002.

[137] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 573–581. ACM, 2011.

[138] Nikolaj Tatti. Probably the best itemsets. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 293–302. ACM, 2010.

[139] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, volume 19, page 30, 2004.

[140] IBM Quest Synthetic Data Generation Code, 2009. URL http://www.almaden.ibm.com/.

[141] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. Sliq: A fast scalable classifier for data mining. In *Advances in Database Technology—EDBT'96*, pages 18–32. Springer, 1996.

[142] Database and data mining group website, paper page, year = 2013, url = http://dbdmg.polito.it/wordpress/research/misleading-generalized-itemsets/.

[143] Maria-Luiza Antonie, Osmar R Zaiane, and Alexandru Coman. Application of data mining techniques for medical image classification. In *MDM/KDD*, pages 94–101, 2001.

[144] Gao Cong, Anthony K. H. Tung, Xin Xu, Feng Pan, and Jiong Yang. Farmer: finding interesting rule groups in microarray datasets. In *ACM SIGMOD '04*, 2004.

[145] Mohammed J Zaki. Parallel and distributed association mining: A survey. *Concurrency, IEEE*, 7(4):14–25, 1999.

[146] Osmar R Zaïane, Mohammad El-Hajj, and Paul Lu. Fast parallel association rule mining without candidacy generation. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 665–668. IEEE, 2001.

[147] Li Liu, Eric Li, Yimin Zhang, and Zhizhong Tang. Optimization of frequent itemset mining on multiple-core processor. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1275–1285. VLDB Endowment, 2007.

[148] Amol Ghoting, Gregory Buehrer, Srinivasan Parthasarathy, Daehyun Kim, Anthony Nguyen, Yen-Kuang Chen, and Pradeep Dubey. Cache-conscious frequent pattern mining on modern and emerging processors. *The VLDB Journal*, 16(1): 77–96, 2007.

[149] Iko Pramudiono and Masaru Kitsuregawa. Tree structure based parallel frequent pattern mining on pc cluster. In *Database and Expert Systems Applications*, pages 537–547. Springer, 2003.

[150] Mohammad El-Hajj and Osmar R Zaïane. Parallel bifold: Large-scale parallel pattern mining with constraints. *Distributed and Parallel Databases*, 20(3):225–243, 2006.

[151] 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.

[152] Le Zhou, Zhiyong Zhong, Jin Chang, Junjie Li, JZ Huang, and Shengzhong Feng. Balanced parallel fp-growth with mapreduce. In *Information Computing and Telecommunications (YC-ICT), 2010 IEEE Youth Conference on*, pages 243–246. IEEE, 2010.

[153] Dehao Chen, Chunrong Lai, Wei Hu, WenGuang Chen, Yimin Zhang, and Weimin Zheng. Tree partition based parallel frequent pattern mining on shared memory systems. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, pages 8–pp. IEEE, 2006.

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

[155] Marco Mellia, Michela Meo, Luca Muscariello, and Dario Rossi. Passive analysis of tcp anomalies. *Computer Networks*, 52(14):2663–2676, 2008.

[156] The apache mahout machine learning library, 2013. URL http://mahout.apache.org/.

[157] Jesmin Nahar, Tasadduq Imam, Kevin S Tickle, and Yi-Ping Phoebe Chen. Association rule mining to detect factors which contribute to heart disease in males and females. *Expert Systems with Applications*, 2012.