POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

ASSOCIATION RULE MINING ON SOCIAL NON-STRUCTURED DATA USING CLUSTERING AND GRAPH DATABASES

Relatore:
Prof. Alessandro Campi

Tesi di Laurea di:
Corrado Palese
Matr. 820448

Anno accademico 2015/2016

ABSTRACT

We currently live in the big data era, in which user-generated content is more than we can analyze day to day in real time because of noise, dynamism and size, without a systematic method. Today the scientific community seems to accept data mining as the most appropriate technique for generating useful information from big dataset, but the way of doing it with unstructured data is still a challenge since a form of preprocessing is required.

In this scenario, the need to efficiently analyze this kind of data is increasing because of characteristics of such big data, especially their huge and sometimes unpredictable variety. Twitter alone, with 320 M active users every month and more than 500 M tweets per day, could represent an important source of information[1].

For this research, we are just focusing on social networks. The reason for this choice is that they are increasingly becoming a platform where people comfortably update states and share or retrieve information about the world in real time (Mooney, Bunescu 2005). Sometimes news is spreading on them faster than in traditional common channels[2] because user capillarity worldwide makes it possible (Kwak, Lee et al. 2010).

In particular, we will focus on Twitter, because its micro-blogging nature makes it suitable for this kind of purpose. It questions the concept of little and private

[1] https://about.twitter.com/it/company Retrieved Nov 28, 2016
[2] ibidem

community of friends in favor of broadcast communications, sometimes less private and of common interest (Naaman, Boase et al. 2010).

Another reason why we chose Twitter is because of hashtag semantic value, their power in summarizing tweet content and the spreading model through the social network that allows us to highlights clusters of topics just focusing on them (Romero, Meeder et al. 2011).

One of the objectives of this thesis is to show how data mining can provide useful techniques to deal with these huge datasets for retrieving information to detect and analyze trending topics and the corresponding user's interactions with them. We identified in Association Rules identification and evolution in time, a systematic approach for conduct the analysis.

SOMMARIO

La mole di dati generata nel contesto attuale, meglio conosciuto con il nome di *big data*, è tale da compromettere spesso la capacità di analisi con i metodi classici a causa del rumore, del dinamismo e delle dimensioni che caratterizzano tale sorgente d'informazione. Attualmente la comunità scientifica sembra riconoscere nelle tecniche di *data mining*, i metodi più appropriati per generare informazione a partire da insiemi particolarmente grandi di dati. Il contesto legato ai Social Network, inoltre, evidenzia un'ulteriore problematica legata alla natura stessa del dato che risulta essere non strutturato e sul quale gli algoritmi standard non possono essere eseguiti direttamente.

D'altro canto, in questo scenario, cresce la necessità di rendere l'analisi sempre più efficiente considerando le dimensioni dei bacini di informazione rappresentati dalle piattaforme social. Solo Twitter, con 320 milioni di utenti attivi ogni mese e 500 milioni di *tweet* pubblicati giornalmente, rappresenta un bacino più che interessante.

In questa ricerca, infatti, ci focalizzeremo sui social network in quanto riconosciamo la natura delle piattaforme social come mezzo di comunicazione ed espressione alla portata di tutti e come mezzo di informazione riguardo eventi e notizie su scala mondiale in tempo reale (Mooney, Bunescu 2005). È formalmente riconosciuto, infatti, che le notizie sui social tendono a propagarsi più velocemente rispetto ai tradizionali canali di informazione (Kwak, Lee et al. 2010).

Nello specifico, ci focalizzeremo su Twitter, in quanto la sua natura di micro-blog incontra la necessità di riconoscere gli eventi e tracciarne gli andamenti. Twitter mette

in discussione il concetto di piccola community privata di amici offrendo un canale di comunicazione di pubblico dominio (Naaman, Boase et al. 2010). Inoltre, l'utilizzo della citata piattaforma nasce dall'idea di sfruttare il valore semantico nell'uso degli hashtag e la loro capacità di riassumere il contenuto di ogni singolo tweet, che permette di evidenziare aree di interesse solo sulla base di parole chiave (Romero, Meeder et al. 2011).

Uno degli obiettivi della tesi è quello di mostrare come l'adattamento delle tecniche di data mining classiche possa fornire uno strumento per il riconoscimento di eventi, della loro evoluzione nel tempo e del coinvolgimento degli utenti attraverso la rete. Abbiamo infine identificato nel calcolo delle regole di associazione, un approccio sistematico per condurre l'analisi.

ACKNOWLEDGMENTS

This research project could not have been possible without the contribution of many people that supported me during these months of work. I would like to express my gratitude to the professor Alessandro Campi, that after offering me the collaboration, supports our work with precious advices and his full availability.

Furthermore, I would like to thank my dad Antonio, my mum Francesca and my sister Ludovica because of their unconditional support. They shared with my both grateful and difficult moments without stopping believing on myself.

My thoughts go to all my friends I met in this amazing journey. I am sure this moment would not have been the same without each of them. I thank who is still in my life and who unfortunately is far in this moment, like all the people I met in my Erasmus experience.

A very big hug go to the Svoltastudenti association because gives me the chance to meet amazing people who finally shared the best moments in my university career as if it had been my family.

Finally, I very warm though go to some special people that are not here anymore. I am sure they would have been proud of me as always: my grandparents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Interest in social networks analysis is recently growing since they represent one of the biggest resource into the well-known big data context. The main innovation coming from them, is related to the capability of providing a channel of data generation used by billions of users everyday (Wu, Zhu et al. 2014). However, data does not mean information. Data is raw and it should be processed to gain useful and human understandable content.

When we deal with social data, some considerations are needed because as well as being a very big source, it is non-structured, dynamic and normally contains noise. Moreover, the information we can extract from them is usually related to real life events. This means that the analysis should be done in real time to be considered reasonable.

The scientific community seems to accept data mining as the most appropriate technique for generating useful information from such big dataset but a preprocessing phase is normally needed in a non-structured context.

For this reasons, our research, will focus on social network data providing by Twitter with the aim of finding a systematic way to make the analysis efficient and scalable. We find in association rule mining the best approach to extract useful information in order to identify and tracking events in real time. We took inspiration from the market basket analysis (Agrawal, Imieliński et al. 1993a) where

the method was used to find correlations between items bought together a certain number of time.

We transpose the idea of transactions into the social context transforming tweets into itemsets composited of hashtags they contain. We take advantage of the semantic values of hashtags in summarizing tweet content through keywords in order to avoid any text mining phase in preprocessing.

In particular, we will examine two key point of our work that makes the whole analysis interesting in terms of time and computational resources.

One is the implementation choice of using graph databases to persist data pulled from the network. This new technology greatly fit the need to model the real-world entity-relationship mechanism especially into a relationships-centered environment, as social networks are. They allow to explicitly save relationships and to query them without executing join operations that compromise performances when they deal with huge datasets.

The other key point concerns the methodology we use to choose the datasets on which generate association rules. We do not apply any filter, except form the language, in pulling the network but we developed a method to cluster tweets before executing the association rules mining. The clustering process drastically reduce the attributes space made by hashtags in every single execution of rules mining with the effect of reducing execution time and computational resources needed, without losing interesting information.

1.1 Organization of the thesis

This section gives a brief overview on the whole methodology, deeply exposed within the document from a theoretical to a more practical point of view. In particular, the document is composited of other five chapters:

- Chapter 2 gives an overview about the context in which we place our work. It focuses on existing methodologies evaluating their strengths and weaknesses. Moreover, it examines graph databases key concepts and the association rules mining process introducing reasons of our decision to use them.
- Chapter 3 gives a formal description of technologies we used to develop our method in terms of APIs and tools.
- Chapter 4 gives the formal definition of the problem and our resolution proposal. It presents the data model, the architecture of the application and the algorithm workflow particularly focusing on differences with respect to other approaches.
- Chapter 5 describes two test cases in which we applied the methodology, presenting results obtained and giving a data-driven demonstration of correctness.
- Chapter 6 concludes the thesis with discussions and possible future works.

2 STATE OF THE ART

2.1 Social Networks Analytics

2.1.1 Twitter as microblogging service

Twitter is a social network microblogging service where users spread information in small posts of 140 characters called *tweets*. Twitter users are organized in communities by the *follower-followee* relationship that is not necessarily bidirectional. That form of "friendship" allows users to receive updates by all users they follow in their own feed section.

Because of the length of messages, real-time updates and the number of users world wide, the microblogging is a new way to communicate. Instead of writing daily articles in blogs, people feel more comfortable sharing information even more than once per day (Java, Song et al. 2007).

The presence of mobile device in our lives make social networks one of the easiest and fastest way to communicate and retrieve information. We need to consider that about the 80% of posted tweets come from this kind of device.

The use of hashtags

The way of Twitter users use hashtags as topic detection and search keywords is quite interesting. A hashtag is a keyword preceded by # symbol. This tags mechanism was proposed and officially approved by an early Twitter user who took inspiration from IRC chat that commonly use keywords preceded by # to distinguish between different discussion topics.

The use of hashtags is the only way Twitter provides to create discussion around a specific topic. Users who want to express their own opinion on a trending topic and to spread it across the network are encourage to use a proper hashtag. Most popular hashtags are also published on a specific area on the Twitter user interface.

Moreover, companies or events organization agencies promote official hashtags to increase the media coverage of the event through user discussions on Twitter platform.

Despite the great importance of hashtags, two criticisms should be considered: the choice to use a new hashtag instead of use the most popular ones and the way users categorized their tweets (Feng, Wang 2014).

The first point explains why, only using a filter on most popular hashtags in order to underline trending topic could generate a too simplified model. We propose a clustering algorithm that allow us to find which are the most popular hashtags first and then to find which are all the related keyword around them.

This way, after defining a correlation index for filtering everything that statistically represent noise, our model considers not only what is official but also what users consider relevant for describing a certain topic.

The second point is naturally solved since we deal with a statistical approach that treats these cases as exceptions.

2.1.2 Tweet filtering

Detecting trending topics on social networks means, above all, to find a reasonable way to distinguish between news about real world events and conversations between small communities of users.

For this purpose, we found two approaches that differ basically for the initial assumption. The first one, used by Adedoyin-Olowe in her research, retrieves useful tweets by using hashtags as search keywords. The method assumes the power of hashtags and their semantic value as summary of the tweet content as the main hypothesis. The fact that most important events encourage the use of official hashtags, nowadays, makes the hypothesis quite interesting.

This approach implies a previous knowledge of the event that she wants to detect in order to filter only useful tweets (Adedoyin-Olowe 2015).

Instead of using only hashtags, sometimes a dictionary of keywords related to the event is used (Corney, Martin et al. 2014, Asur, Huberman 2010, Mathioudakis, Koudas 2010).

The second one, starting from a clustering of tweets streaming in real time, proposes some methods to understand which clusters refer to real events (Becker, Naaman et al. 2011).

2.1.3 Trend detection and tracking

Once tweets are identified using one of the approaches proposed above, the new challenge is to detect interesting topics in the dataset in order to study their evolution and find out hints for the decision-making process.

In this perspective, Asur and Sitaram, propose a method for predicting box office revenue of movies only be using tweet analysis. They made a linear regression model on the tweet rate in a certain time window that outperforms the Hollywood Stock Exchange (http://www.hsx.com/), a popular play-money market, where the

prices for movie stocks can accurately predict real box office results (Asur, Huberman 2010).

In the same way, Twittermonitor project and Corney, Martin, and Goker work, use an algorithm based on detecting bursty keywords and clustering them in real time, highlighting breaking news and emerging topics (Corney, Martin et al. 2014, Mathioudakis, Koudas 2010).

The main intuition of Adedoyin-Olowe in her research was to try to detect and track an event by mining association rules using Apriori algorithm. She built a dataset as a transactional database where each entry is the set of hashtags of tweets. She then classified founded rules in order to distinguish breaking news and track the evolution of an event by just checking how rules change in time. The method is called *TRCM* and is able to detect all kind of events on twitter with the same method at once (Adedoyin-Olowe 2015).

2.1.4 Graph inspection

In addition to discriminating between tweets of global interest and confidential ones, we now focus on the necessity to discover most authoritative users. The goal is satisfied by ranking systems for both tweets and users built using graphs.

An authoritative user is a person who usually spreads useful information fast.

To find them is the objective Turank project (Yamaguchi, Takahashi et al. 2010). Instead of focusing on user relationships, communities' identification and the behavior of users among them, it builds a tweet-user graph which models information flow. aObjectRank (Balmin, Hristidis et al. 2004) is then applied to evaluate user authority scores. It is based on the number of tweets retweeted by other users, since retweet action is often related with information spreading.

Of course, this is not the only purpose of retweeting, so the algorithm tends to assign a higher score where retweet chains are long in order to favor a broadcast approach instead of a retweeting action for answering and commenting in a more

private conversation. The idea to not only work on user *follower-followee* relationships borns from the assumption that sometimes it does not have any semantic value, being just a form of courtesy since it does not require any kind of "handshake".

Starting from the same idea of building a user-tweet graph, *Yang, Lee, Lee and Reem,* proposed a method for ranking tweets instead of users.

Previously, another study based on crowdsourcing on Amazon Mechanical Turk, showed that "interesting" tweets usually contain links with 80% of accuracy (Alonso, Carson et al. 2010). They considered the process not accurate enough so their work also provides an adapted version of HITS algorithm for page rank that considers both user connection and retweet count (Yang, Lee et al. 2012).

2.2 Graph databases

> A graph database is a storage engine which supports a graph data model backed by native graph persistence, with access and query methods for the graph primitives to make querying the stored data pleasant and performant. (Robinson, Webber et al. 2015)

Graph databases deal with the need of leveraging with naturally complex and dynamic relationships between entities. Especially for data of significant size or value, graph databases are one of the best approach for data representation and query. The reason why the scientific community increase the interest in this kind of technology is since graph theory itself is not new. At the beginning, large corporates developed their own proprietary framework to model their business into graph representation but now general – purpose graph databases have been developed for any user who want to experience the technology.

Graph databases greatly fit the need to model the real-world entity-relationship mechanism without the strong preprocessing required by a relational database representation to store data in form of tables.

Relationships assume the same importance of data. They are not anymore only runtime constraints.

In this scenario, users need to understand what should be model as an entity and what as a relationship. Basically, nodes are used for representing entities while relationships are used to express connections between entities and to formalize a semantic content for each entity. Moreover, in graph databases we can specify properties for both entities and relationships even if with a different meaning. In entities, a property represents an entity attribute such as timestamps, ownerships, etc. In relationships properties are used for representing the strength, weight or quality of a relationship.

2.2.1 Differences between Graph databases, RDBMS and NoSQL

Traditional database servers as RDBMS or NoSql usually loose performance when they deal with relationships because they do not store them explicitly as entities.

In RDBMS systems, for example, relationships are extracted through join operations that decreases performance with the data size while NoSql systems have no data structures to model or store relationships, nor query constructs to support data relationships.

Relational databases use to store highly structured data in tables where every column represents a property of the entity and defines which type of value is going to be stored for a certain property. This very rigid data representation forces users to find the most proper way for structuring data.

In relational databases, entities are identified through unique id called primary keys, and cross relationships between different kind of entities are done through join operations.

Normally a primary key is embedded as field of related entities table and it is called foreign key. Joins are computed at query time by matching primary- and foreign – keys after the Cartesian product between tables of related entities had been done.

This operation is either memory and compute intensive with exponential cost.

In case of relationships are many-to-many, a joining table is required and performances are even worst since a Cartesian product between three tables is needed to find desired relationships.

Graph databases are mainly focused on relationships. They allow to explicitly save them giving us the possibility to build complex models that map closely to our domain.

This is possible since every node contains a list of relationships-records that represents its linkage with other entities. This way join operations are not necessary anymore and Neo4j performances in finding relationships are of order of magnitude better than relational databases.

NoSql database normally use a key – value model. Graph database takes the same concept and adapts it in the sense that even properties could be connected.

The main issue in using NoSql databases is that most of them only stores sets of disconnected entities. One way to remedy to this lack is to embed an aggregate's identifier inside the field belonging to another aggregate. Basically, is the same way of using foreign key in relational databases and, even in this case, relationships coming from join operations are prohibitively expensive.

2.2.2 Summary

In this paragraph, we introduced and described graph databases and we mainly focused on the fact that relationships assume the same importance of entities. They

are explicitly saved and not calculated at execution time breaking down performances as in more traditional database systems. For this reason, graph databases answer to the need of modelling connected data as in social networks.

We can now sum up reasons why we choose to use this technology:

- Performances: we are working on trending topic detection on a very changeable world in which results must be available to end-user in milliseconds. This is not possible with traditional database technology since performances drastically decrease when data increases, because of join operations.
- Data Model: reduce the development overhead of finding the best tabular representation into a relational model for entities.
- Schema-Free: a graph solution is more versatile and adaptable to changes in business in which it is working in.
- Reliability: enterprise systems as Neo4j guarantee ACID properties in transactions, high availability, scalability and storage of billions on entities without performances reduction.

2.3 Association Rules Mining

Association rule mining is a systematic approach for finding correlations and frequent patterns among large structured dataset as transactional databases or other data repositories (Kotsiantis, Kanellopoulos 2006a).

The concept was introduced in 1993 by Rakesh Agrawal, Tomasz Imielinski e Arun Swami with the shopping basket analysis. The aim of the research is to find correlations between product bought together (Agrawal, Imieliński et al. 1993b).

The main idea behind this approach is to divided the problem in two subtasks. First of all, the algorithm tries to identify which itemset are frequent within a certain dataset given a specified support. In transactional databases, for example, an itemset is a transaction itself or a subset of it: a list of item that appear together (e.g. Shopping basket).

We could define an itemset as frequent if, given a certain number of transactions and a support threshold s, it appears in at least the $s\%$ of the times out of the total transactions.

Frequent itemsets represent interesting correlations within item in a bigger item space. Once frequent itemsets are discovered, the second subtask is to build association rules among them.

The main idea is to find which rules are "interesting". An association rule is a logic implication of the form $X \Rightarrow Y$ such that X and Y are itemsets and X, Y are disjoint. Such definition is implemented setting a confidence index. A rule $X \Rightarrow Y$ is interesting if, given a total number of transactions T and the itemset $I = X \cup Y$, I exceeds the support and the number of transactions that contains Y out of the number of transactions that contains I exceeds the confidence.

2.3.1 Apriori Algorithm

The *Apriori* algorithm allows to find strong association rules starting from frequent itemsets: the process is based on a prior knowledge of which are the frequent itemsets within the data set.

The way of finding frequent itemset is based on the Apriori property: all non-empty subsets of a frequent itemset must also be frequent. This property belongs to a class of properties called *antimonotonicity* such that if a set does not pass a test, all its superset will fail the same test as well. Apriori takes advantage of this property to easily find biggest itemsets through an iterative approach.

Basically, a two-step process is implemented consisting of a join phase and a prune phase.

Given a set of transactions T, first the algorithm will find all 1-itemsets that satisfies the minimum support.

In the join phase the algorithm will try to find all 2 – itemsets joining all 1 – itemsets found before with each other. Among them, only those which are frequent will be used for finding 3 – itemsets in the same way as before and so on and so forth.

The prune phase ensures the algorithm termination using the Apriori property in the sense that a (k – 1) itemset that is not frequent cannot be a subset of a frequent k – itemset. Therefore, every itemsets that are not frequent are not considered for the next itemset generation step (Agrawal, Srikant 1994).

Now we explore the algorithm steps through an example.

Table 2.1. Example of Transactional Data.

TID	List of item_IDs
T1	I1, I2, I5
T2	I2, I4
T3	I1, I3, I5
T4	I3, I5
T5	I1, I3, I5

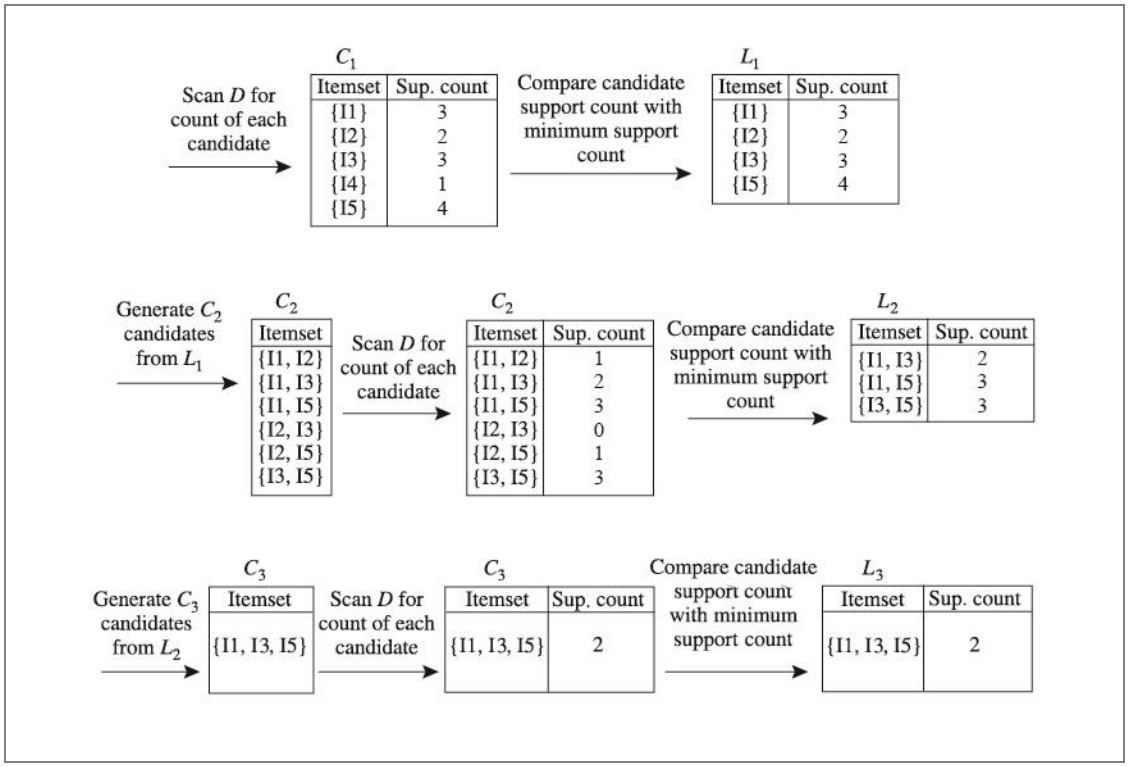

Figure 2.1 Generation of the candidate itemsets and frequent itemsets, where the minimum support count is 2

The figure above show us a transactional database containing four transactions. As we see above, the Apriori algorithm is an iterative approach which starts finding the 1-itemset. A first database scansion is done in order to find all the item that overcome the minimum support (L1). In this example, minimum support is set to 2. After the first scan, the item 4 is deleted since it has a support count equal to 1. To discover the set of 2 – itemsets L2, the algorithm does a L1 join L1 operation to find the candidate set.

After every join phase, the database should be scanned to find the relative support of each itemset found in the candidate set. The process finishes when the next candidate set is empty.

2.3.2 FPGrowth

Another method for frequent pattern mining was introduced by Jiawei Han, Jian Pei e Yiwen Yin and it is called *Frequent Pattern Growth* (Han, Pei et al. 2000). It allows to find frequent itemsets without candidate generation. The idea is to convert the transactional database in a *Frequent Pattern Tree* or *FP-tree*, and then find frequent itemsets scanning it.

The first step of this approach is to find all frequent 1-itemesets as in the Apriori method and to order them with respect to the number of occurrences, starting from the most frequent one. The second step consists of reordering all transaction items follow the same order.

At this point a tree should be created as follow. Starting from an empty radix, all transactions represent a path in which items are linked each other within a chain. Identical subpattern generated by different transaction are not duplicated but a count for each item is updated to keep track of the item support. Given that, the complexity of the tree increases as the uniqueness of each transaction increases.

Finally, all items are saved in a table with a pointer to them to direct access. Multiple occurrences of the same item are linked together within a chain.

In this approach, the support is saved explicitly though the count variable and items with a low support are leaves. Moreover, it is important to notice that, rather than Apriori algorithm, only two database scansions are required.

3 TECHNOLOGY ASSETS

3.1 Twitter API

Twitter makes APIs to access public data, available for developers and third part applications. For our research, we used Streaming APIs [3].

They continuously deliver new responses to REST API, that offer a programmatic access to read and write Twitter data, over a long-lived HTTP connection. Keeping the connection opened, allow us to reduce access latency and it is extremely useful in case we do not have any filter.

For our research, in fact, we only filter tweets by English language.

Furthermore, Twitter offers several streaming endpoints. In our scope, we choose the public streams endpoint, because it is more suitable for data mining process.

Regarding to that, two type of services are provided: a free one that return a small sample of all the public data flowing through Twitter, and a premium one, the *firehose*, that provides the access to all data published on Twitter.

For our research, we use the public one because we are more interested in the methodology that could be eventually extended to a more complete set of data.

[3] https://dev.twitter.com/streaming/overview Retrieved Nov 28, 2016

Limitations of the streaming APIs free access point have been analyzed by (Morstatter, Pfeffer et al. 2013). They compared data retrieved through both freemium and premium access points in the same period and they come out that the freemium one returns at most the 1% of total number of tweets published.

Considering the number of tweets published per day (about 500 M), sample is big enough to make the analysis reasonable but some clarifications are necessaries.

First, Twitter does not provide the method used to return tweets, but results of the analysis show up that, using just a sample of the whole data, introduces a bias in the analysis depending on the way of filtering them.

3.2 Twitter4j

Instead of explicitly querying Twitter API by a HTTP request use Twitter4j, an unofficial Java API to access Twitter services [4].

Even if it is not developed directly by Twitter it is reported in their official documentation as a good way to integrate a Java application with Twitter.

Trough the implementation of the *StreamListener* interface, we can retrieve a tweet whenever it is on the stream as an instance of the *Status* twitter4j class. *Status* object contains all the information about the tweet and provides functions to access them easier then parsing directly the JSON representation returned by Twitter API.

[4] http://twitter4j.org Retrieved Nov 28, 2016

3.3 Graph Database

3.3.1 Neo4j

Neo4j is a native graph database purpose-built to answer to the need of new kind of analysis driven by relationships between data more than by data themselves [5].

In addition to the native graph structure, it offers also a native graph processing, known as *index-free adjacency,* that can evaluate results regarding very huge datasets delivering constant-time performance.

Given considerations above, we decide to use Neo4j because it fits well both the dataset, since tweets do not have a fixed structure, and the need to analyze relationships between them instead of data themselves.

When we use Neo4j, we describe an arbitrary domain as a connected graph of nodes and relationships. In order to explain the structure with all different features we can define for specific entities and relationships, we are going to use an example.

TWO PEOPLE, ALAN AND BOB, ARE FRIENDS AND BOTH LIKE ICE CREAM.

This statement defines two entity labels, *Person* and *Food*, and two relationship labels, IS FRIEND OF and LIKES.

In Neo4J, both nodes and relationships can contain properties. In this example Alan and Bob, are values of a property called *first name* defined in People entity and *ice cream* is a concrete value of name property defined in Food entity. We can also define properties for relationships. In this case, for example, we could specify when Alan and Bob's friendships has been begun.

[5] http://neo4j.com Retrieved Nov 28, 2016

Relationships describe interactions between different nodes. In this case, as we can see from the figure, Alan and Bob are linked through an arrow labeled IS FRIEND OF since they are friends and both are linked with an arrow LIKES to ice cream since they both like it.

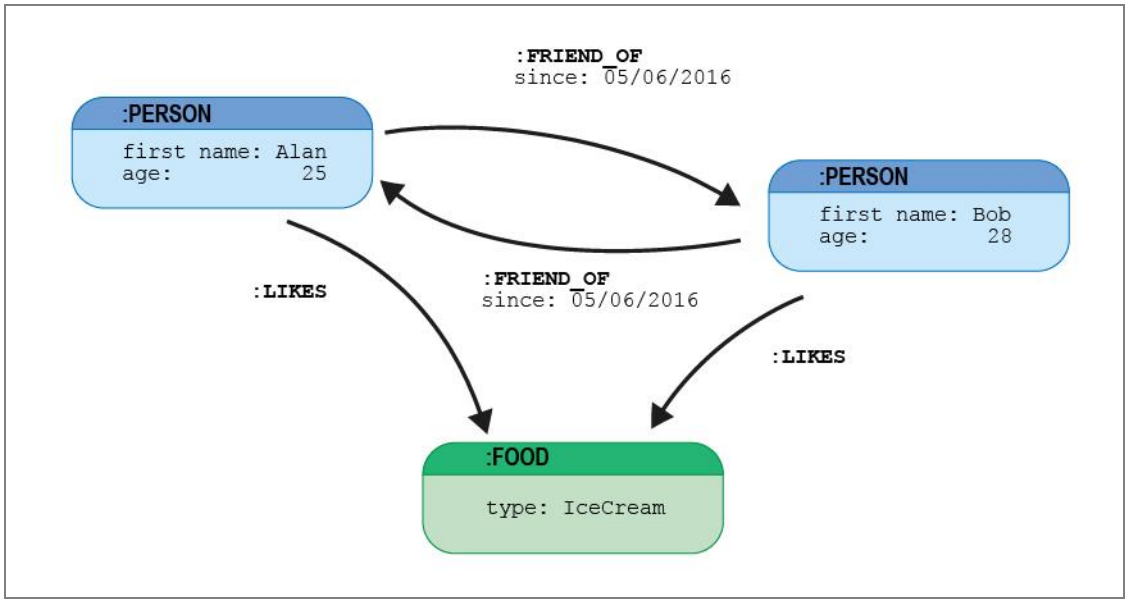

Figure 3.1 Graphic representation of the statement

In Neo4J, every relationship is directional. Since friendship relationship is naturally bidirectional, logically, two arrows are needed. It is not true because we can set up a direction and then, using the query language properly, we can cross relationships in both directions.

In order to characterize our entities and relationship, it is possible to define properties for both of them that allow us to make filters on query results. Properties, also called attributes, belong to the data model.

Graph databases allow to define a very flexible data model, leaving entities as is in the real world. Finally, transactions keep the ACID properties.

3.3.2 Architecture

Neo4j architecture is based on store files accessible through APIs and up to the Cypher Query Language.

Each file contains data for a specific part of the graph. Basically we consider as graph data nodes, relationships and properties for both of them. Three kind of data are stored in different files for better performing in graph traversals.

Nodes are stored in the node store file called *neostore.nodestore.db.* Like most of store files, records have a fixed size.

In particular, nodes are 9 bytes' length record:

- 1 byte: in.use flag, used for checking whenever a node is valid or record could be used for store another value (Another file, *.id,* is used to keep track of nodes that are invalid).
- 2-5 bytes: first relationship connected to the node. Relationships are stored, starting from the first one created, in a linked list.
- 6-9 bytes: first property assigned to the node.

The idea of using fixed dimension for saving node is to enable fast lookup for nodes based on their id. For example, if we want to retrieve information about node id 50, we need to look up at byte 450 within a performance in search of O(1).

Relationships are stored in a file called *neostore.relationshipstore.db.* Like node store file, records have a fixed length of 33 bytes and store both the start and at the end node, a pointer to the relationship type, stored physically in a separate file, and pointers for the relationship chains for the start and end node (as a doubly linked list) since a relationship logically belongs to both nodes and therefore should appear in the list of both nodes' relationships.

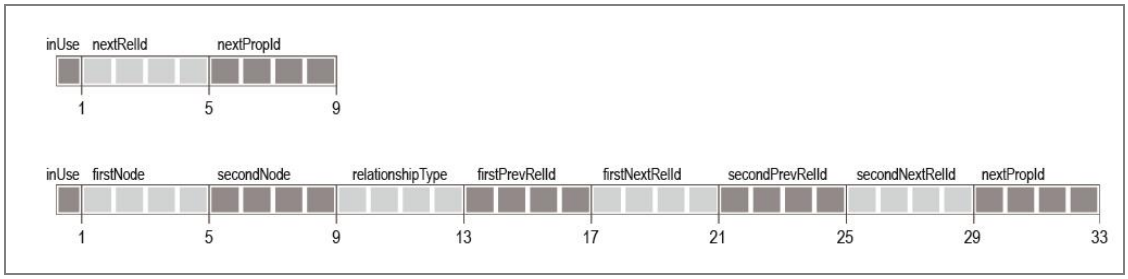

Figure 3.2 Neo4j Node and Relationship Store File Record Structure

Finally, properties are stored in different files as below:

- *neostore.propertystore.db*: each entry contains the property type, a pointer into the property index store file, a pointer to a dynamicStore record, and finally a pointer to the next property.
- *neostore.propertystore.db.index*: each entry specifies which type the property is. Neo4j allow any primitive JVM type, plus strings and arrays of all the previous.
- *neostore.propertystore.db.strings* and *neostore.propertystore.db.arrays*: properties are actually saved depending on property type.

Some kinds of optimizations allow to store values directly in the *neostore.propertystore.db*. This is possible when data could be encoded in a way that fit exactly record size. This strategy reduces I/O operations and improves throughput since only one file as to be looked up.

This file organization to separate graph structure and property values ensure short time for graph traversal.

In order to increase performance, Neo4j provides a two-tired caching architecture to reduce latencies intrinsically related to mechanical mass storage devices. The lowest tier is the filesystem cache that store discrete regions per each store file. The highest one, instead, is all about optimizing for arbitrary read pattern. In fact, it stores nodes, relationships, and properties together for a rapid in memory graph traversal.

In the figure 3.2, we see how the various store files we describe above interact on disk. From one node, we can directly access to the first property and the first

relationship and then, following the doubly linked list until we find the relationship on which we are interesting in. From a relationship, we can read its properties just accessing the singly-linked list structure like for node properties.

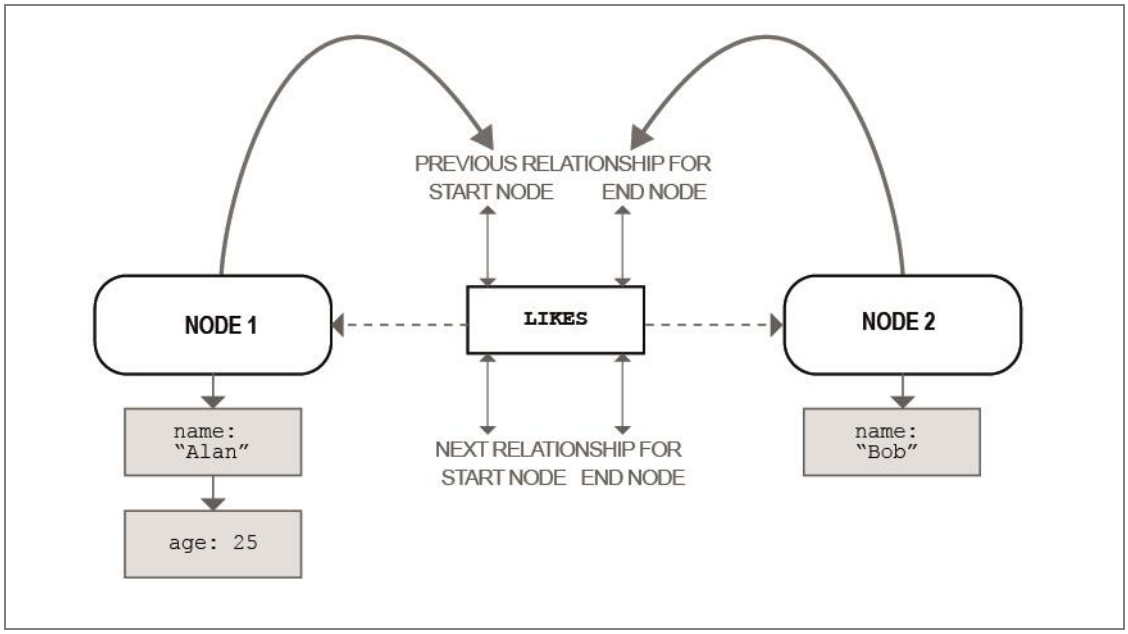

Figure 3.3 How a graph is physically stored in Neo4j

3.3.3 Neo4j APIs

Neo4j provides APIs logically organized as a stack where the more we go ahead the more expressiveness and declarative programming are prioritized and vice versa. That is why on the top of the stack we find the Cypher API that allow us to execute query on the graph in a very easy and declarative way. We will see it in the next session.

- Kernel API: manage user access through transaction events handlers in the kernel[6].

[6] http://docs.neo4j.org/chunked/stable/transactions-events.html Retrieved Nov 28, 2016

- Core API: is an imperative Java API allow users to access graph primitive functions for nodes, relationships and properties through Java language. An interesting characteristic is they are lazily evaluated for reads. This means relationships are only traversed as soon as the API caller can use data being returned. For write operations, the Core API, provides transactions management to guarantee atomicity, consistence, isolation and persistence (ACID properties).
- Traversal API: is a declarative Java API provides a set of constraints to describe which part of the graph we are interesting in. Constraints are expressed in relationships type, directions, type of search (breath versus depth-first), coupled with a user-defined path evaluator which is triggered on each node encountered. User can specify some policies for node inclusion or exclusion during the search phase.

We decide to specify all kind of APIs Neo4j provides to user for completeness even if, in our research, we only use the Core API explicitly since we decide to use Cypher API for graph traversal in a declarative and more understandable way (Robinson, Webber et al. 2015).

3.3.4 Non-functional characteristics

As in relational databases, the evaluation of a database system is not only based on how many transactions it can manage in a second but also in which kind of non-functional characteristics it can provide in order to guarantee data consistency and persistence. Moreover, a database is expected to scale out to provide high availability and scale up for performances (Montag 2013, Robinson, Webber et al. 2015).

Recoverability

Transactions in Neo4j are semantically identical to traditional database transactions. Write consistency is guaranteed through a lock system within a

transactional context. If the transaction fails for any reason, changes are discarded, the lock is released and the graph is restored to the previous consistent state. Concurrent writes are detected by Neo4j and corresponding transactions are serialized in order to prevent deadlocks. Transactions are represented as in-memory objects supported by a lock manager. Lock manager applies write lock to any node or relationship that is going to be created, modified or updated. Moreover, data commitment to the disk is managed through a *Write Ahead Log.* Once a positive response is raised on the *prepare* phase, a commit entry is written to the log and then to the disk. Locks are released only when every change are finally written to the disk.

In case of system failure, Neo4j uses the same transactions log and replays any transactions it finds against the store. Replaying is an idempotent action, so far if Neo4j replays one that has been already committed before the crash, result will be the same and, at the end of the recovery process, store will be consistent. Neo4j also offer a procedure for recovering in case of distributed architecture.

Availability

Being both transactional and recoverable means that Neo4j databases recognize and repair an instance after crashing. This implies automatically the availability of such a system.

Moreover, the possibility to manage with parallel instances, increase availability in doing queries.

A classic deployment methodology is based on a write-master replica that manages all the write operations and some read-slaves that manage read operation. Neo4j also supports writing through slaves. In this scenario, the slave first verifies consistency with respect to the master and then the write is synchronously managed on both instances. It is easy to understand that, even in this case, write operations are not distributed because they still should pass over the master instance. This

method only guarantees an immediate consistency within master and slaves instances.

Scale

Neo4j can store up to 34 billion of nodes, 34 billion of relationship and 68 billion of properties within a single graph. Upper bounds are fixed in order to balance efficiency versus scale (Montag 2013).

As we discussed in previous section, graph databases in general do not loose performances as data increases because join operations are not needed. Queries follow a pattern in which first node is found with `O(1)`, and the traversal in relationships is done following pointers. This insures performance times are more or less constant.

3.3.5 Cypher Query Language

Cypher is an SQL-inspired declarative language for describing patterns in graphs. It allows us to declare what we need to extract without specifying how to do it.

Cypher takes inspiration from SQL – queries structure and its clauses model.

We will now present the most popular clauses we use in our work for reading and updating the graph database.

- MATCH: specify a pattern and to get data from the graph that match it.
- WHERE: is not properly a clause but allow to set pattern constraints for filtering results.
- RETURN: specify which data to return.
- CREATE (DELETE): create (delete) nodes and relationships.
- SET (REMOVE): set (remove) values to properties and add (remove) labels on nodes.
- MERGE: conditional CREATE clause in which a node or a relationship is created only if it is not already exist, otherwise works as a MATCH.

Update queries can be logically divided into two phase: a reading phase and a writing phase. Moreover, Cypher is lazy and pattern specified after the MATCH clause is evaluated only when RETURN clause is called. Update queries lose the laziness since reading operations must finish before writing start instead of waiting the explicit invocation of the RETURN clause. In this case, the RETURN, if specified, will get data just updated.

Another very powerful clause is WITH. In Cypher, it is used to pipe the result from one query to the next one or to explicit separate read-only queries from write-only queries in update statements.

RETURN clause could be used with all query types. This means that, while for a read query it is required, for an updating or a removal query it is not but, at the same time, it could be used. In case of updating, it will return updated data but in case of removal, since removed entities or relationships do not exist anymore, it will return a null pointer.

Now we will see a practical example.

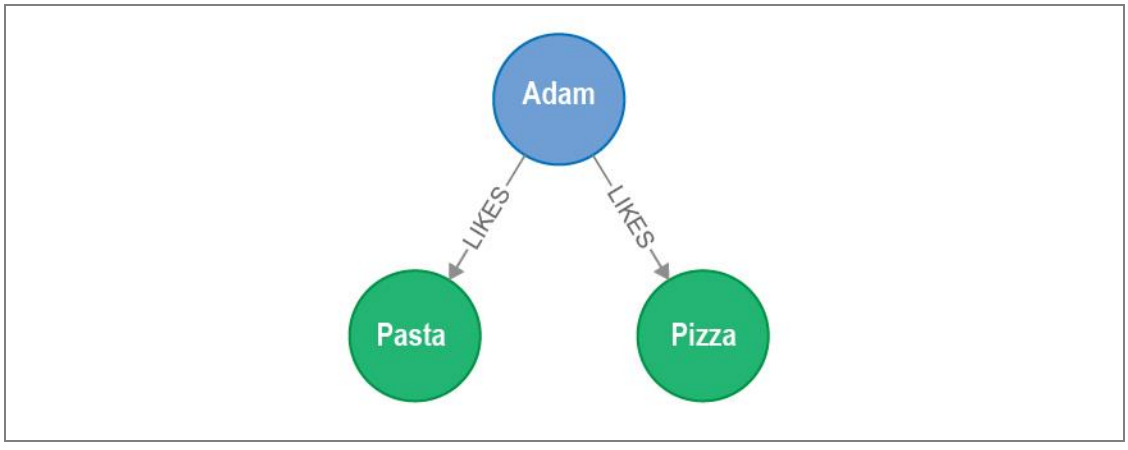

Figure 3.4 Neo4j representation of entities and relationships

In the figure above, first we find a graphic representation of how entities and relationships are logically represented in Neo4j databases. Since it is a graph database, representation is exactly a graph in which entities are nodes and

relationships are arcs. In this example, we can deduce that `Adam,` entity of type *Person* is linked to `Pizza` and `Pasta`, entities of type *Food*, through a *Like* relationship.

Now we focus on how CQL works. For referring to an entity it uses round parenthesis which look like circles. In case of we have more than one type of entity, we also need to specify the type using a colon followed by the type name.

Conventionally is better to use camel case words for both entities and relationships type definition. If we are not interested in a specific entity, we can also use empty round parenthesis.

Since we are doing queries, it is important to understand how we can filter query results and how we can return what we are interested in, as results.

If we want to know what kind of food does `a` like we need to execute the following query:

```
MATCH (:Person {name: 'Alice'}) - [:Like] -> (f:Food)
RETURN f
```

As you can see, we can use nodes and relationships properties for filtering results directly in the pattern specification or after the WHERE clause accessing them through a dot notation.

Specify the node or the relationship type using the column notation is not required if there is not ambiguity in entity types that match the pattern. For example, if Like relationship has only one outgoing node type, the explicit specification can be omitted.

Results are returned as entities of the specified type. The java library provides function to deal with them and eventually access to their properties.

The way of CREATE and DELETE clauses works is straightforward so we will focus on MERGE with the next example.

As we discussed above, MERGE allows to create a node or a relationship only if it does not exist. It works as a sort of a conditional CREATE by a MATCH result.

To make it works a unique property must be specified. It will work as a primary key in relational database.

In the example above, we may be interested in insert a new kind of food if it does not already exist. Instead of doing a MATCH query and then a CREATE, only if the MATCH result is empty we can execute the following:

```
MERGE (f:Food {id: 'id', type: 'fruit'})
RETURN f
```

In this case, the node of type Food should be uniquely identified by the pair id-type. A new Node of type Food will be created with these properties if it does not already exist. These are some of the most used clauses for doing operations on a graph database. For completeness, we refer to the official documentation[7].

3.4 Weka

Weka is an open source software issued under the GNU License written in Java. It was developed at the University of Waikato, in New Zeland and basically it is a collection of machine learning algorithms for data preprocessing and analysis (Frank, Hall et al. 2005).

Being opensource implies that maintainability and function update do not depend on an institution or company since the community itself is in effort to provide them. Weka includes tools for data engineering algorithms for attribute selection, clustering, association rule learning, classification and regression. Moreover, since it is entirely written in Java, it can run on almost any platform even if it loses in performances in comparison with an equivalent developed in C/C++.

Weka has a modular and high scalable implementation. The object-oriented architecture allows new preprocessing tools and algorithm to be added very easily.

[7] http://neo4j.com/docs/developer-manual/current/cypher/ Retrieved Nov 28, 2016

The most important disadvantage is that most of the functionalities is only applicable if all data is in main memory. This clearly imposes a limit on the data size we can process through such tool.

For our research, even if we lose performances because of memory swap, we do not have any unresolvable memory issue.

We use *Weka* implementation of *Apriori Algorithm* for finding association rules within dataset made by tweets.

4 RULES MINING

As we discussed on the previous section of this document, there have been a lot of researches conducted on Twitter data for trending topic detection and events tracking but none of those were used to track different events simultaneously without applying filters in data collection phase.

With our approach, we developed a method for real time clustering of tweets on which we execute the association rules mining analysis to track events in a reasonable time. The clustering process reduces the number of tweets we analyze together, finding which of them are correlated and could generate interesting rules.

Dealing with social networks, with the huge number of feeds published daily and with the fact that we decided to not do any pre-filtering activity, we will demonstrate how the clustering process allows to save time during the execution and computational resources without losing information. Finally, once tweets are divided into clusters, the same analysis could be executed on all of them in parallel with a further reduction in overall execution time.

The main idea for creating clusters is to find which are the most popular hashtags in the dataset and then find which hashtags are correlating to them.

Despite of many researches on this topic[8], our thesis shows that any filtering is needed for classifying tweets. Moreover, our algorithm does the classification in real

[8] cfr. Section 2.1.2

time, explicitly saving a relationship entity between hashtags that appear together when a tweet is pulled from the network, or incrementing the relationship attribute called counter if it already exists.

We will use a graph database to save tweets and relationships between them. Saving relationships explicitly give us a strong competitive advantage with respect to other implementations that use relational databases since we do not need to execute join operations.

Finally, in order to find association rules that represents the final objective of our work, we decided to treat tweets as transactions. We took inspirations from association rule mining in market baskets (Agrawal, Imieliński et al. 1993a). Moreover, we decide to use hashtags as items and tweets containing hashtags as transactions on which applying Weka implementation of the Apriori algorithm.

4.1 Data model

We define two type of entities:

- TWEET in which we saved as attributes:
  - ID: long - the unique ID Twitter assigns to each tweet.
  - Text: string - tweet content.
  - Author: long - the author ID Twitter assigns to each user.
  - IsRetweet: boolean – true if the tweet is a retweet, false in all other cases.
  - IsAnswer: boolean – true if the tweet is an answer to another one, false in all other cases.
  - Hashtags: string – comma separated list of hashtags contained by the tweet.
  - Timestamp: long – creation timestamp.
- HASHTAG in which we saved as attributes:
  - Text: string - the hashtag itself.

Moreover, we defined two type of relationship:

- TAGS: unidirectional relationship between a hashtag and a tweet that contains it
- APPEAR_TOGETHER: conceptually bidirectional relationship between hashtags that appear together at least in one tweet. Within APPER_TOGETHER relationship we keep track of how many times two hashtags appear together through an attribute called *count*.

For our research, we do not use every attribute but we decide to save them for possible further works.

An example of the structure in the figure below.

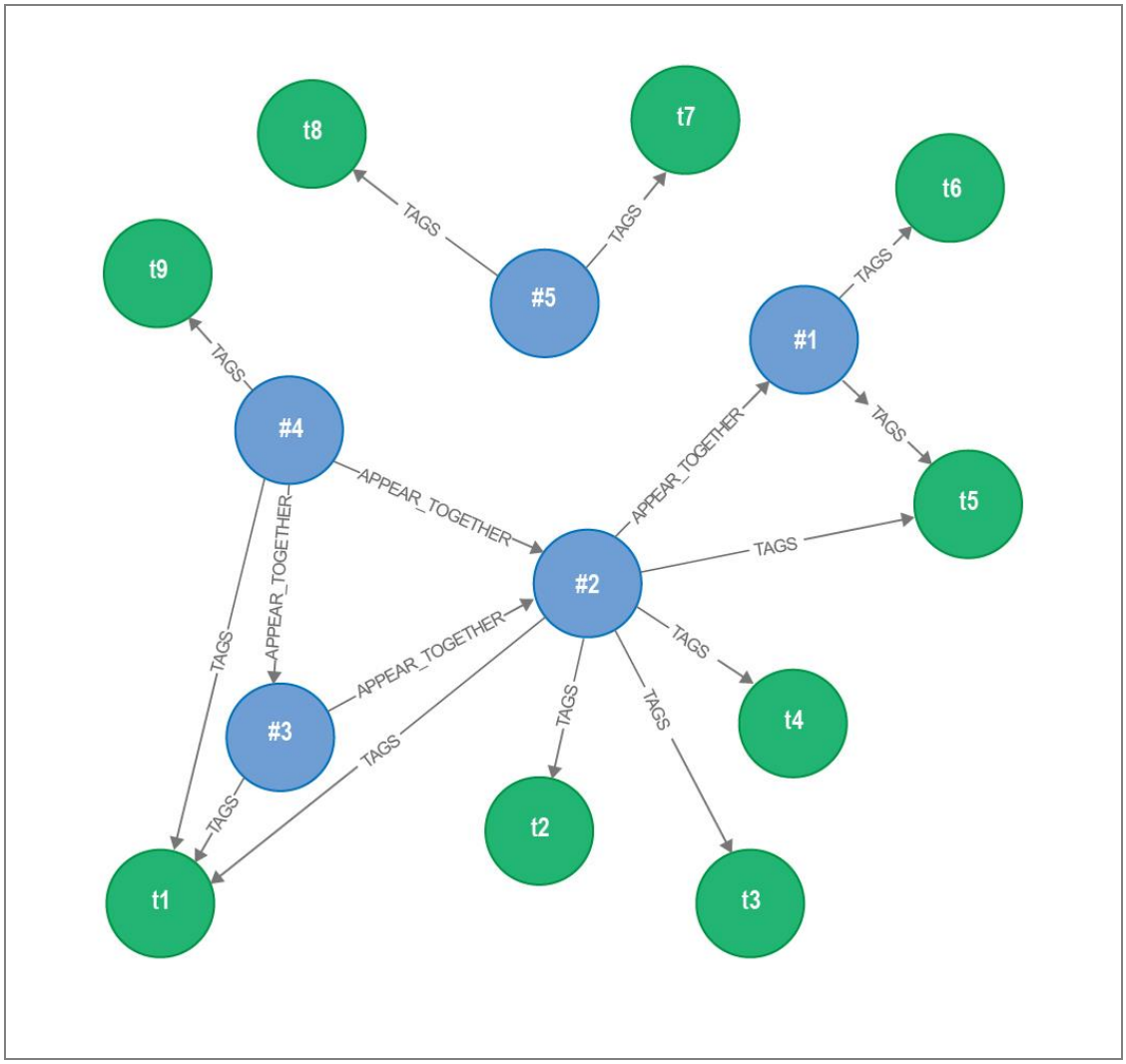

Figure 4.1 Graphic representation of the data model

4.2 Application architecture

4.2.1 Embedded Neo4j

We have already discussed the advantages of using a graph database instead of a more common relational model. The idea of treating relationships as entities, storing them explicitly, goes well with the use of a social network as information source. The "*social graph*" term used by Mark Zuckerberg to describe social networks[9], reinforce the idea of using graph data model in graph databases implementation naturally fits this relationship-centered domain. The absence of join operations guarantees reasonable performances for real time analysis without giving up to ACID properties in transactions, high availability and scalability.

Most databases run on a server accessible through a client library. Neo4j databases can be run in both server and embedded mode. Embedded does not means in memory in fact data is persistent on the disk.

The main characteristic of an embedded instance is that Neo4j runs in the same process as our application. This guarantee low latency since networks overhead are absent. Moreover, with this implementation, we can use the full range of APIs and we have the full control over transactions through java APIs. We can also create and manage named index.

Our choice to use an embedded mode instead of the server implementation, comes from the low latency which guarantees saving time in doing queries plus the opportunity to manage everything at the application level that, since we are developing a prototype only for research aim, is more convenient.

[9] "Facebook Unveils Platform for Developers of Social Applications". Facebook. https://newsroom.fb.com/news/2007/05/facebook-unveils-platform-for-developers-of-social-applications/ Retrieved Nov 28, 2016

The decision to use this kind of implementation, give us some constraints. First, our database instance can run only in a JVM as our java application. Second, it is subject to the garbage collector behavior of the host application which can affect the query time if it pauses for a long time. Last, it makes the host application responsible for database lifecycle.

4.3 Code Structure

We organize our project in three components: one for tweets retrieval, one for tweets analysis and one for data post processing to create the CSV file we use as input in Weka.

Tweet retrieval component is organized in four Java classes:

- TwitterListener: responsible to retrieve tweets via Twitter4j library. It contains the main method of the component and all the authentication keys saved as constant variables.
- GraphDB: manages the Neo4j connection and data persistence into it as we discussed in the precious section.
- NodeType: an enumeration that list all of node type. In our research, it only contains two types: TWEET and HASHTAGS.
- RelType: same as NodeType enumeration but for relationships. In our research, it only contains two types: TAGS and APPEAR_TOGETHER.

TweetAnalyze component is organized in just one class:

- Analyze: contains the software logic to make tweet clusters on which we do the association rule mining through Weka.

Finally, a service component to create the CSV file in post processing, implemented by just one class:

- ExportToFile

4.4 Algorithm workflow

The algorithm can be divided into three main phase that we are going to analyze deeply within this chapter.

The first step is data retrieval in which we extract tweets published in the network and we saved them into the graph.

The second step, is a key point of our research, because doing the clustering we drastically reduce time in finding association rules without losing information. This steps starts specifying one or more hashtags on which we would like to focus on. Once these are defined, all correlated hashtags are extracted, forming the space field in which rules will be calculated. From these hashtags, we then extract all tweets that contains at least one of them. They represent the tweet subset identified, correlated with the topic described by hashtags correlation.

At this point, starting from tweets, another step would be needed to find hashtags contained by each of them. This operation implies a high cost in terms of time since we need to follow all incoming relationships for each tweet that belongs to the cluster. In order to avoid this, we add the list of hashtags also as a tweet property in form of string, made by comma separated hashtags. Moreover, it is exactly what we will save in the CSV file and it will represent the corresponding tweet in association rules mining analysis.

4.4.1 Data retrieval

The first step of our Java application is the information retrieval. What we consider as information, are tweets that, for a certain period, are published on the social network.

As we specified in sections above, for this research, we decide to not do any pre-filtering activities on tweets retrieval except for the language. The choice of filtering on English language is purely technical: give us the opportunity to not consider

languages that is not understandable from our side even if, at the same time, represent huge quantity of information on Twitter network (Eg. Chinese, Arab languages, etc).

We pull Twitter network through Twitter4j API, using Twitter streaming endpoint. The choice of using this endpoint is because we are interested in retrieving information that is flowing on the network in a certain period. We do not interest in doing searches on a certain topic but we only want information on which we can find, by our approach, the most trending topics for that time.

In order to build an access point to Twitter streaming APIs, all we need is a class that implements a *StatusListener*. Twitter4j will then create a thread to consume the stream.

The mechanism used is an event handling process. Twitter4j implements on the *StatusListener* class, the method *onStatus* where we can specify what happens each time we receive a feed. The *onStatus* function is executed every time a tweet is caught from the network, it is a callback function that return a *Status* object [10].

The Status object contains all the information about the tweet that represent and provides all the functions to access information we need.

Every time a tweet is caught, information should be saved in the graph. On the callback execution, a node in the graph of TWEET type is created and relative information is stored. Moreover, every hashtag contained in the feed should be saved as well as a HASHTAG node type, only if it does not exist yet. A relationship of TAGS type should be created to link the hashtag with the current tweet.

Finally, all hashtags within the same tweet should be linked each other with an APPEAR_TOGETHER relationship in a full connected subgraph.

Information coming from the APPEAR TOGETHER relationships is redundant. As we have already discussed, even if relationships in Neo4j are declared as

[10] http://twitter4j.org/javadoc/twitter4j/Status.html Retrieved Nov 28, 2016

monodirectional, they could be seen conceptually bidirectional. Moreover, the Cypher Query Language support the need to cross them in both ways.

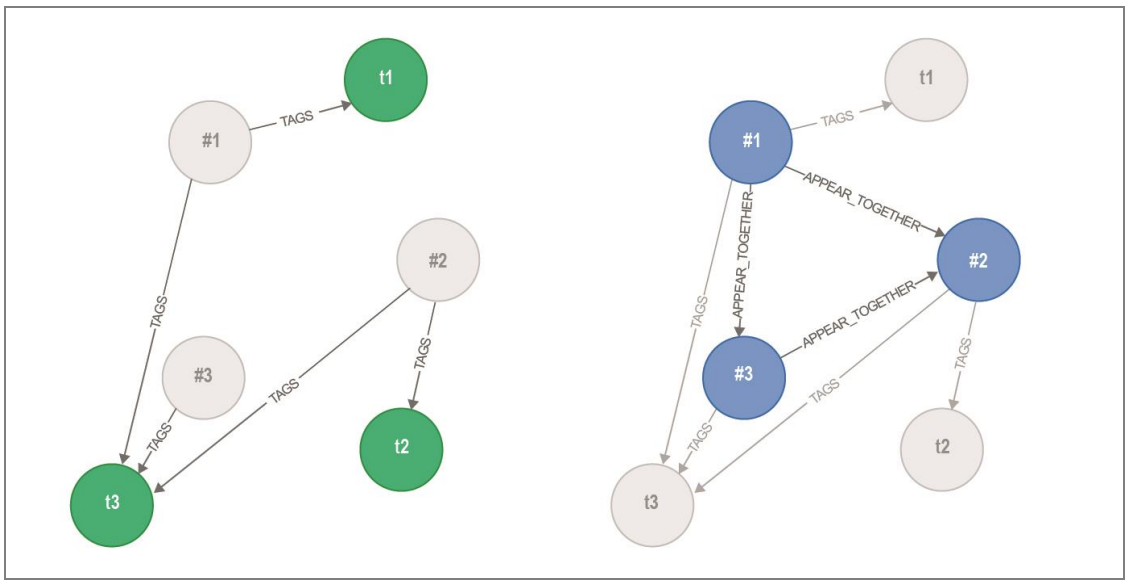

Figure 4.2 Appear_together relationships

Having a look at the figure 4.2, we can see that we could retrieve hashtags that appear together within a tweet starting from the tweet itself. In fact, each tweet is linked by a TAGS relationship to all hashtags contained. This implies that, starting from a hashtag, we can find all hashtags that appear together with it, passing through tweets that tags. The complexity of the process would be very high since we would need a number of queries equals to the number of tweet containing the hashtag on which we start the correlation analysis. It would represent a serious problem in a real-time process that potentially deal with millions of instances.

Instead, the choice of storing APPEAR TOGETHER relationships, even introducing information redundancy, allow us to find out all correlated hashtags just through one query, starting from one of them without depending on the number of instances.

4.4.2 Clustering process

The clustering process begins when hashtags and mutual APPEAR_TOGETHER relationships are stored into the graph. Our algorithm builds cluster in real time when tweets are caught from the network without any additional computation needed.

Since our analysis potentially involves billions of tweets and hashtags, we have specified a method to choose which hashtags, within a cluster, should be considered interesting.

First, once data are collected, our algorithm allow us to investigate most popular hashtags on which extract the relative cluster. Otherwise we can also specify one, making a sort of filtering around an interesting topic represented by a hashtag. This scenario is implemented in case we want to analyze a determinate event eventually using the official hashtag. Once one or more hashtags are chosen, the algorithm extract which hashtags in relative subgraphs are considered interesting.

The main idea of filtering hashtags we do not consider interesting is based on the fact that, reducing the space in which we will look for association rules, we obtain a significant decrease in execution time without losing information, as we will see in the experiment results.

We consider a correlation between two hashtags "interesting", if they appear together a certain number of time in tweets that belong to the sample on which we are doing the analysis. To do this systematically, we need to define a threshold value.

In order to explain the mining of the threshold we are going to use an example. From now on, we will call the hashtag chosen to build the cluster, `pivot`. To build the cluster we only consider hashtags directly connected with the pivot and eventually relationships between them but only if, singularly, all of them reach the threshold value on the direct relationship with the pivot.

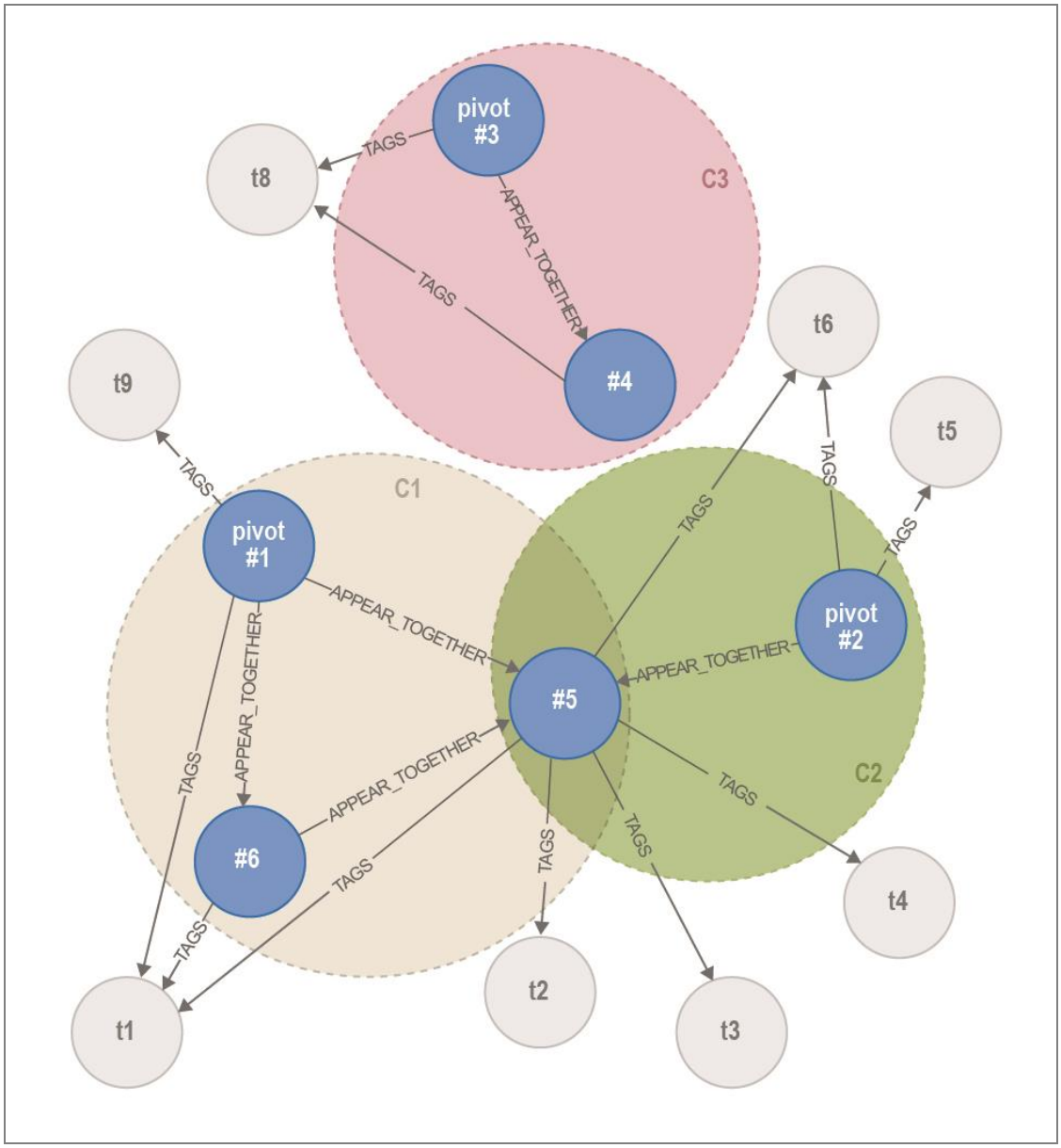

Figure 4.3 Clustering process

To keep track of the number of time two hashtags appear together, we use the count property defined into the APPEAR_TOGETHER relationship.

Consider two hashtags: `a` and `b`, both connected with the pivot and a threshold `s`. They will be part of the cluster if and only if the number of time they appear together with the pivot out of the total number of tweets that contain the pivot

itself, overcomes s. Moreover, if both satisfy the threshold and they are also mutual connected, this relationship is carried into the cluster.

The choice of using a threshold value on the pre-processing phase seems to be useless since the minimum support applied by association rules mining algorithm will exclude all correlated hashtags from generated rules if they do not overcome the threshold.

The reason why we decide to apply a pre-processing filter on hashtags occurrences comes from the need to make the space search smaller, in order to save computational resources.

In fact, experimentally, we notice that, without applying this threshold, we generated a search field made by thousands of hashtags. Thinking about how the Apriori algorithm works, most of them, would disappear automatically in frequent itemsets generation but they could cause the generation of a huge number of candidate sets. The very low minimum support we use to calculate association rules do not help the process in this sense. For example, if there are 10^3 frequent 1-itemsets, the Apriori will need to generate more than 10^5 candidate 2-itemset, causing several Weka crashes for RAM lack.

4.4.3 Tweets as transactions

Our choice to adapt tweets as transactions in a transactional database comes from the fact that this kind of data model fits well the need to find association rules. (Kotsiantis, Kanellopoulos 2006b).

We transform a tweet in a list of item that sum up the content in terms of topics it contains. Scientific community recognizes the semantic value of hashtags in summarizing tweet content (Romero, Meeder et al. 2011) and we exploit it in order to avoid text mining technics to tokenize the tweet.

Items that make a transaction representation of the tweet are hashtags it contains. The graph structure allows us to extract hashtags contained by a tweet but, for

performance reasons, we introduced a redundancy in data, saving hashtags as a property of the tweet.

In this way, every node of TWEET type has the hashtags property: a string with comma separated hashtags text.

4.5 Weka input file

ARFF is the acronym of Attribute-Relation File Format, an ASCII text file that describes a list of instances sharing a set of attributes. It was defined by the same team who develops the Weka suite[11]. The ARFF file is divided into two section: one with header information and one with data information.

The header of the ARFF file contains the name of the relation and the list of the attributes (the column of a virtual table). Data is basically the raw of that table.

In transactional databases, columns represent the attribute space and raws represent transactions. A simple way to represent properly a transaction is to put a placeholder in correspondence to columns that represent an attribute of that transaction.

In our case, attributes are all the hashtags that form the cluster, and transactions are all the tweet that, at least, contain the pivot.

Instead of directly build an ARFF representation of our cluster, Weka allows users to create a CSV file as input of the process.

First raw of the CSV file is a comma separated list of hashtags we are going to consider part of the cluster while all other raw, one for each tweet who contains the pivot, are a comma separated list of placeholders. In particular we use a boolean placeholder representation in which `?` symbol indicates the absence of an attribute and the `t` symbol indicates the presence of that attributes.

[11] http://www.cs.waikato.ac.nz/ml/weka/arff.html Retrieved Nov 28, 2016

Look at the figure below to better understand how the file looks like.

4.6 Summary

As we describe in this chapter, we can divide the process into three main subtasks. The first one is the information retrieval in which we simply retrieve information in form of tweets from Twitter network through Twitter4j API. The result of this phase is a graph representation in which tweets are connected to hashtags entities contained and hashtags that appear together at least one time are fully connected. The graph representation is stored permanently in the native graph database Neo4j. The second phase is the clustering process which already begins when we store data into the graph but that needs a sort of filtering in order to reduce the space of attributes on which we will look for association rules. Cluster is made by using APPEAR_TOGETHER relationships between hashtags. As we see, we defined a correlation rule to make the search space for association rules smaller.
Finally, a post processing phase in which we transform tweets assigned to each cluster in a transactional representation. The output of this phase is a CSV file we will use as input to run for Weka in order to execute the association rule mining analysis through the Apriori algorithm.

5 IMPLEMENTATION AND TEST CASES

In this chapter, we present results of two tests we did to validate our algorithm. We examine results we obtain in terms of association rules found with and without the clustering process. Results show that we drastically increase performances in Weka execution time without loose information. Moreover, rules we found allow us to highlight trending topic and track events around them in real time.

5.1 System architecture

We run all experiments on the same virtual machine hosted by the *Policloud*, the IaaS cloud designed, managed and deployed by Politecnico di Milano in collaboration with IBM and Yahoo[12].

Hardware specifications

The virtual machine is instantiated on a shared hosting and has following hardware specifications:

- CPU: Intel Xeon E3-12xx (Sandy Bridge) series – 8 cores (4 MB Cache, 2.3 Ghz)
- RAM: 31.3 GB

[12] http://policloud.polimi.it/ Retrieved Nov 28, 2016

- Hard Disk: 10 GB

Software specifications

- Operating system: Ubuntu Server 14.04 – Trusty (LTS)
- Java Version: 1.8.0
- Weka Version: 3.8.0
- Neo4j Version: 2.3.3
- Neo4J Java Drivers: 1.0.6

5.2 Test cases

As we discussed in the previous chapter, we run all the algorithm phases, from the most popular hashtags extractions to the association rules mining trough Weka. Conceptually, we do not need any filtering activity except for the tweet language but, to make the analysis reasonable, we decide to extract from the starting sample, the ten most popular hashtags and all other hashtags correlated to them. Then, we execute the association rules mining analysis through Apriori implementation of Weka first using clustering around each cluster and then, putting all hashtags together in the same space. Finally, we compared results in terms of execution time and information retrieval. We show as with our clustering process, we reduce the execution time in finding association rules from minutes to seconds without losing information.

In our test, we define a correlation threshold for hashtags of 2%. In other words, it means we consider two hashtags correlated if they appear together at least the 2% of times the most popular once appears into tweets.

For example, if we consider two hashtags, A and B, with A the most popular of them, we say that they are correlated and belong to the same cluster, if at least the 2% of tweets that contains A, contains also B.

Moreover, we define the support and confidence threshold in association rules mining as follow.

For confidence, we use the value of 0.5% because we noticed that, considering the first 100 rules in support order for every cluster, they all overcome this percentage. For support, we did some experiment in order to find the best threshold to make the analysis reasonable. We notice that we cannot find almost any association rules with a less than 0.001% minimum support. Even if this value could seem too low, it is justified by the fact that we are working on a very huge sample without doing any pre-filtering activity. Values are in line with other researches conducted on the same field (Adedoyin-Olowe 2015). We need to mentioned that, when we run the Apriori algorithm without using clustering, we use the same confidence but a support of 0.0001% since we analyze ten clusters all together that approximately means ten times the number of tweets with respect to only one cluster.

5.2.1 Rio 2016

The first test has been done on tweets retrieved from the 07th of August to the 13th of August 2016.

- Sample dimension: 490971 tweets
- Execution time in association rules mining on clusters built around ten most popular hashtags:
    - MTVHottest – 0 min 1.407 sec
    - PushAwardsLizQueen – 0 min 1.056 sec
    - DolceAmoreOperation1010 – 0 min 1.096 sec
    - VeranoMTV2016 – 0 min 1.400 sec
    - Rio2016 – 0 min 4.021 sec
    - PushAwardsKathNiels – 0 min 1.021 sec
    - ALDUBsaAfrica – 0 min 1.082 sec
    - MUFC – 0 min 1.120 sec

  - Gameinsight – 0 min 1.042 sec
  - USA – 0 min 6.625 sec
- Execution time in association rules mining without using clustering: 2 min 28.89 sec.

In this section, we present and discuss about rules we found in Rio2016 cluster.

Full training set (hashtags) made by 86 attributes:

```
#Archery, #ARG, #ArtisticGymnastics, #Athletics, #AUS, #badminton,
#basketball, #Basketball, #bbcrio2016, #BRA, #BREAKING, #BringOnTheGreat,
#Bronze, #bronze, #CAN, #CHN, #cycling, #CyclingRoad, #CyclingTrack, #DEN,
#DipaKarmakar,      #ESP, #Fencing, #fencing, #FIJ, #Fiji, #FinalFive,
#football, #FRA, #GBR, #GER, #Gold, #GOLD, #gold, #gymnastics, #Hockey,
#Ind, #IND, #IRQ, #ITA, #JAM, #JPN, #Judo, #KheloIndia, #KOR,
#LalitaBabar, #MichaelPhelps, #NGR, #NZL, #Olympic, #OlympicGames,
#olympics, #Olympics, #Olympics2016, #OpeningCeremony, #Phelps, #PHI,
#POR, #Rio, #Rio2016, #RioOlympics, #RioOlympics2016, #rowing, #RSA,
#Rugby7s, #RugbySevens, #Silver, #silver, #SILVER, #SimoneBiles, #SWE,
#Swimming, #swimming, #TableTennis, #TeamCanada, #TeamGB, #TeamUSA,
#Tennis, #tennis, #USA, #USABMNT, #USABWNT, #volleyball, #weightlifting,
#WirfuerD, #yourteam
```

Whit the minimum support of 0.001% and a minimum confidence of 0.5%, 115 rules have been found. For reasons of clarity, we only show the first ten rules, ordered by confidence, and some others that allow to understand the power of our algorithm to highlight events and tracks their outcomes. Other rules have been omitted in this paragraph because they do not give any information because of the absence of semantic values in hashtags or they are redundant with rules presented.

1. yourteam=t 122 ==> Rio2016=t 122   <conf:(1)>
2. Swimming=t BringOnTheGreat=t 46 ==> Rio2016=t 46   <conf:(1)>
3. KheloIndia=t Rio2016=t 28 ==> Hockey=t 28   <conf:(1)>
4. Gold=t ArtisticGymnastics=t 90 ==> USA=t 89   <conf:(0.99)>
5. CHN=t ITA=t 35 ==> USA=t 34   <conf:(0.97)>
6. CHN=t Rio2016=t ITA=t 33 ==> USA=t 32   <conf:(0.97)>
7. CHN=t USA=t 48 ==> Rio2016=t 46   <conf:(0.96)>
8. bronze=t gold=t 36 ==> silver=t 34   <conf:(0.94)>

9. CHN=t ITA=t 35 ==> Rio2016=t 33 <conf:(0.94)>

10. CHN=t USA=t ITA=t 34 ==> Rio2016=t 32 conf:(0.94)

11. Olympics=t NGR=t football=t 29 ==> Rio2016=t 27 conf:(0.93)

16. KheloIndia=t Hockey=t 31 ==> Rio2016=t 28 conf:(0.9)

17. GOLD=t FIJ=t 30 ==> Rio2016=t 27 <conf:(0.9)>

18. NGR=t football=t 38 ==> Rio2016=t 34 conf:(0.89)

25. NGR=t DEN=t 35 ==> Rio2016=t 30 <conf:(0.86)>

43. Rugby7s=t Fiji=t 42 ==> Rio2016=t 31 conf:(0.74)

45. ArtisticGymnastics=t USA=t 123 ==> Gold=t 89

53. Gold=t Rugby7s=t 46 ==> FIJ=t 32

66. Fiji=t Rio2016=t 49 ==> Rugby7s=t 31 conf:(0.63)

77. Gold=t FIJ=t 52 ==> Rugby7s=t 32 <conf:(0.62)>

78. Tennis=t 149 ==> Rio2016=t 91 conf:(0.61)

An interesting information comes from rules 3 and 4. In the first one, we immediately see that one of the most trending argument is related to hockey discipline and the Indian team. During these days, in fact, the men Indian team reached the quarter-finals after thirty-six years. From these rules, we focus on the event but we do not have any suggestion about the outcome. In the second one, instead, we clearly understand that USA wins the gold medal in artistic gymnastic discipline. In this case, we do not just have information about the event itself, but also about the outcomes. In fact, having a look at the training set, we find the hashtag `#SimoneBiles`, the American athlete who won four gold medals in four different artistic gymnastic disciplines exactly in these days. Probably, with a lower minimum confidence index, we would find same rules involving also the SimoneBiles hashtag that it would give us a complete information.

From rules 11, 18 and 25, we can see that something happens that involves Nigerian and Danish football teams. In fact, on the 13th of August, the Nigerian football team won against Denmark and reached the semi-finals.

Finally, from rules 17, 43, 57, 77, we can see that Fiji rugby team won the gold medal. Moreover, it was the first-ever Olympic medal for the pacific island nation.
One more interesting thing is that, even with only a very small training test, we have been able to underline the most important events happened during days in which we made the analysis.

5.2.2 American Elections

The second test has been done before the American election day of the 8th of November. The idea was to see if our method could be used, not only to find interesting topics and to track how they evolve over time, but also to predict event outcomes. In Rio2016 case prediction does not make sense since we deal with a sport event in which result is unpredictable from user's reactions. In fact, this kind of event are not affected by user's behavior.
Moreover, for events in which the outcome is strongly related with user's behavior, like elections, a pre-outcome analysis becomes reasonable.
In this case, we retrieved tweets from the 25th of October to 9th of November.

- Sample dimension: 485.617
- Execution time in association rules mining on clusters built around ten most popular hashtags:
    - EMABiggestFansJustinBieber – 0 min 0.98 sec
    - EMABiggestFansLadyGaga – 0 min 0.96 sec
    - ElectionNight – 0 min 01.46 sec
    - AMAs – 0 min 01.0 sec
    - ThatsMyGirl 0 min 0.90 sec
    - ALDUBBuhayMayAsawa 0 min 0.86 sec
    - ElectionDay – 0 min 02.33 sec
    - 2016MAMA – 0 min 01.66 sec
    - MAGA – 0 min 07.19 sec
    - Gameinsight – 0 min 0.93 sec

- Execution time in association rules mining without using clustering: 2 min 53.42 sec.

Some considerations are needed regarding to the list above. First of all, if we compare it with the first experiment, we notice that `#gameinsight` hashtag is again one of the most popular hashtags. The reason why this happens is due to absence of any correlation with a real event that implies the absence of any temporal implication.

Second, we can find two main events happened during this period: American elections and the MTV Europe Music Awards. The first topic is represented by the official hashtags `#ElectionDay` and `#ElectionNight` moreover with the official hashtag of the Donald Trump campaign `#MAGA`.

Clusters built around ElectionDay and ElectionNight are not disjoint or, even better, the ElectionNight cluster is almost a subset of the ElectionDay cluster. Differences between them are very interesting. Two hashtags represent an evolution of the American election event since ElectionDay was the official hashtag during the voting day and the ElectionNight the official one when election finished and first results had been published.

Rules we found in both cases, in fact, firstly allow us to predict the result highlighting a positive acceptation around Donald Trump and secondly, to confirm the prediction with association rules in which hashtags related with Trump appears in rules that clearly celebrate a winning.

There is a significant difference with respect to the first experiment conduct during the Olympic games. This times we needed to use a lower minimum support and a lower confidence in order to find a reasonable ruleset. The reason has to be found in the way users use hashtags during both events. In Olympic games, even if the attribute space is big enough (86 hashtags), users that were speaking about the same sub-event, used most of the time, the same hashtag set.

In the American election test, instead, apart from official hashtags, users uniformly used all the hashtags of the attribute spaces, strongly influencing the rules support.

ElectionDay

We find a relevant ruleset using a minimum support of 0.0008% and a confidence of 0.4%.

Here below rules we find:

```
 1. Vote=t Vote2016=t 25 ==> ElectionDay=t 25    <conf:(1)>
 2. MAGA=t Election=t 29 ==> ElectionDay=t 28    <conf:(0.97)>
 3. voted=t myvote2016=t 27 ==> ElectionDay=t 26    <conf:(0.96)>
 4. ElectionDay=t Election=t 31 ==> MAGA=t 28    <conf:(0.9)>
 5. TrumpPence16=t TrumpTrain=t 101 ==> MAGA=t 78    <conf:(0.77)>
 6. Voted=t 74 ==> ElectionDay=t 56    <conf:(0.76)>
 7. FoxNews2016=t 192 ==> ElectionNight=t 140    <conf:(0.73)>
 8. USElection2016=t DonaldTrump=t 66 ==> ElectionNight=t 46
<conf:(0.7)>
 9. myvote2016=t 191 ==> ElectionDay=t 110    <conf:(0.58)>
10. ElectionDay=t Vote=t 46 ==> Vote2016=t 25    <conf:(0.54)>
11. Vote2016=t 136 ==> ElectionDay=t 65    <conf:(0.48)>
12. MyVote2016=t 148 ==> ElectionDay=t 69    <conf:(0.47)>
13. voted=t 201 ==> ElectionDay=t 91    <conf:(0.45)>
14. iVoted=t 73 ==> ElectionDay=t 33    <conf:(0.45)>
15. Trump2016=t 803 ==> TrumpTrain=t 340    <conf:(0.42)>
16. TrumpPence16=t DrainTheSwamp=t 69 ==> MAGA=t 29    <conf:(0.42)>
17. ElectionNight=t DonaldTrump=t 114 ==> USElection2016=t 46
<conf:(0.4)>
18. MAGA=t Trump2016=t 67 ==> TrumpTrain=t 27    conf:(0.4)
```

Considering that tweets belong to this cluster were published the day of the elections, most of rules found, comes from statuses of people that had gone to vote and want to share the experience with the community. We can still observe the effectiveness of our research in tracking real life events.

Moreover, this time, we can see how this method could be used to predict event outcomes.

Any of the rule we found has a reference to Hilary Clinton while some of them express the support from Donald Trump and the enthusiasm of his supporters. The presence of the Trump's official hashtags `#MAGA` and some unofficial hashtags but still very viral ones like `#TrumpTrain`, `#TrumpPence` within the simple `#DonaldTrump` attests a strong attention around Trump without any negative meaning. This allow us to make a prediction about the elections outcome.

ElectionNight

Analyzing rules found in this clusters we confirm our prediction with rules that clearly announce Trump as the new president of the United States.

In this case, we use the same support of 0.0008% but a lower confidence index of 0.2%.

Full training set (hashtags) made by 32 attributes:

```
#BREAKING, #trump, #LoveTrumpsHate,    #DonaldTrump, #USElection2016,
#Elections2016, #MAGA, #HesNotMyPresident,   #Election2016,
#PresidentElectTrump, #disappointed, #trumpwins, #CNNElection,   #Brexit,
#ElectionNight, #maga, #MyVote, #ElectionDay, #TrumpPence16, #FoxNews2016,
#ImWithHer, #ElectionFinalThoughts, #myvote2016,    #Trump, #StillWithHer,
#RIPAmerica, #electionnight, #TrumpWins, #MyVote2016, #AmericaIsOverParty,
#PresidentTrump, #electionday
```

Here below rules we found:

```
  1. CNNElection=t 28 ==> ElectionNight=t 24    <conf:(0.86)>
  2. ElectionNight=t myvote2016=t 30 ==> ElectionDay=t 22    <conf:(0.73)>
  3. FoxNews2016=t 192 ==> ElectionNight=t 140    <conf:(0.73)>
  4. DonaldTrump=t USElection2016=t 66 ==> ElectionNight=t 46
<conf:(0.7)>
  5. myvote2016=t 191 ==> ElectionDay=t 110    <conf:(0.58)>
  6. MyVote2016=t 148 ==> ElectionDay=t 69    <conf:(0.47)>
  7. DonaldTrump=t ElectionNight=t 114 ==> USElection2016=t 46
<conf:(0.4)>
```

```
 8. TrumpPence16=t 867 ==> MAGA=t 324    <conf:(0.37)>
 9. AmericaIsOverParty=t 354 ==> ElectionNight=t 124    <conf:(0.35)>
10. USElection2016=t 660 ==> ElectionNight=t 190    <conf:(0.29)>
11. RIPAmerica=t 106 ==> ElectionNight=t 30    <conf:(0.28)>
12. USElection2016=t ElectionNight=t 190 ==> DonaldTrump=t 46
<conf:(0.24)>
13. MyVote2016=t 148 ==> ElectionNight=t 33    <conf:(0.22)>
14. TrumpWins=t 90 ==> ElectionDay=t 20    <conf:(0.22)>
15. TrumpWins=t 90 ==> ElectionNight=t 19    <conf:(0.21)>
16. ElectionFinalThoughts=t 109 ==> ElectionDay=t 22    <conf:(0.2)>
17. ElectionDay=t myvote2016=t 110 ==> ElectionNight=t 22    <conf:(0.2)>
```

As we discussed, ElectionìDay and ElectionNight clusters are overlapped, in fact we can find in both of them some generic rules about the action of going to vote (2,5,6,13,17).

Rules 9 and 16 track the transition toward the end of the elections moreover with incoming outcomes that unequivocally celebrate Trump as new American president (4,7,12,14,15).

Finally, we can also perceive the disappointment about Hilary's supporters coming from rules 11.

All facts we underlined above could be further reinforced playing with minimum support and confidence indexes in order to find ulterior correlations among hashtags. `#PresidentTrump` and `#PresidentElectTrump` testify again the Trump winning while #disappointed, `#HesNotMyPresident`, `#StillWithHer` reinforce Hilary's supporters sentiment of disappointed due to the defeat and their support to the candidate who just lose the election.

5.2.3 MTV Europe Music Awards

In this paragraph, we will discuss about an emblematic case in which our research could not be applied to extract additional information from tweets related to a real life event.

On the 6th of November, the music contest *MTV Europe Music Awards* took place in Rotterdam and it had generated a lot of interests in social networks. The reason is due to the vote mechanism. In fact, to elect the favorite artist for the category *biggest fan,* people had to publish a tweet that contains the official hashtags of the competition, one for each artist in the contest.

This event had a very huge echo in social networks, so that most popular hashtags of the sample we extracted from Twitter feed during these days are two official hashtags assigned to two artists that were competing. In particular, the most popular one is `#EMABiggestFansJustinBieber`, assigned to Justing Bieber that finally won the competition[13].

The interesting thing is that, the relative cluster only contains two hashtags, the one for voting and one related to the *American Music Awards (AMAs*). The Apriori algorithm cannot find any rules unless we use a very low minimum confidence (0.001%). This is due to the fact that only few tweets in the cluster contain both hashtags.

The only two trivial rules found, give us the perception of a correlation between the two events and a forced prediction of Justin Bieber winning also in the AMAs festival but they do not give information in how the current event is evolving over time or how is the reaction of people with regards to it.

[13] http://tv.mtvema.com/vote#cat=biggest-fans Retrieved Nov 28, 2016

5.3 Results comparison

In table 5.1 and 5.2, we put a summary of method execution in both test cases we analyze in the previous section.

In particular, we specified for each cluster, the number of rules calculated, the number of tweet analyzed and the number of attributes of each one. Moreover, we formalize the same data also for the execution without clustering.

In 14 out of 20 clusters, we do not lose any association rules with respect to do the analysis without clustering. In remaining 6 cases, rules that come up in addition are due to the lower minimum support empirically set, since the training set is bigger than the single clusters and because we set a limit of 100 rules for clustering execution and 1000 rules for execution without clustering. In fact, the additional rules finding in the execution without clustering does not contains hashtags belonging to different clusters. This confirm that we are not losing information applying our method.

In fact, even with same support and confidence indexes, we could find almost all rules that seem missing in every cluster that reach the maximum number forced just removing this constraint, but we limit them to make the analysis understandable.

Table 5.1. Test case #1: input and performances

TID	Rules	Execution time	Attributes	Tweets
MTVHottest	100	0,221	14	53953
PushAwardsLizQueen	9	0,074	9	19194
DolceAmoreOperation1010	2	0,068	2	19117
VeranoMTV2016	100	0,2	11	55769
Rio2016	100	2,69	86	23997
PushAwardsKathNiels	5	0,042	7	8283
ALDUBsaAfrica	4	0,049	7	5455
MUFC	26	0,082	24	6195
Gameinsight	42	0,056	10	3992
USA	93	4,659	100	34882
TOTAL	**481**	**8,141**	**270**	**230837**
without_clustering	1000	155,028	219	136314

Table 5.2. Test case #2: input and performances

TID	Rules	Execution time	Attributes	Tweets
EMABiggestFansJustinBieber	2	0,049	2	19202
EMABiggestFansLadyGaga	12	0,055	7	9423
ElectionNight	17	0,348	32	23034
AMAs	1	0,067	6	20156
ThatsMyGirl	0	0,031	2	5179
ALDUBBuhayMayAsawa	0	0,027	2	4777
ElectionDay	30	1,266	54	29813
2016MAMA	100	0,611	70	8789
MAGA	100	5,461	108	36102
Gameinsight	37	0,049	9	3530
TOTAL	**299**	**7,964**	**292**	**160005**
without_clustering	1000	173,42	239	92279

As we can see from the charts below, there is not any correlation between number of rules calculated and execution time needed. In fact, for example, in both `VeranoMTV2016` and `Rio2016` clusters, we generated the same number of rules (100) in 0.2 sec and 2.69 sec respectively.

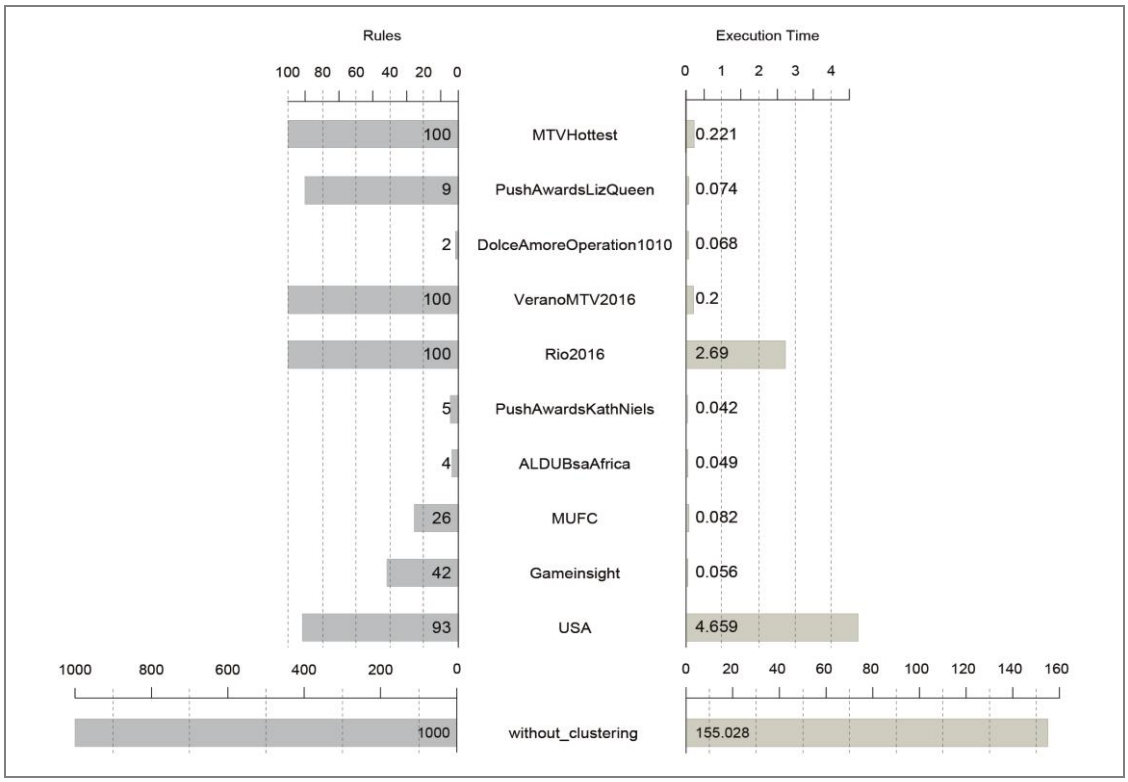

Figure 5.1 Test case #1: Comparison between n. of rules and execution time

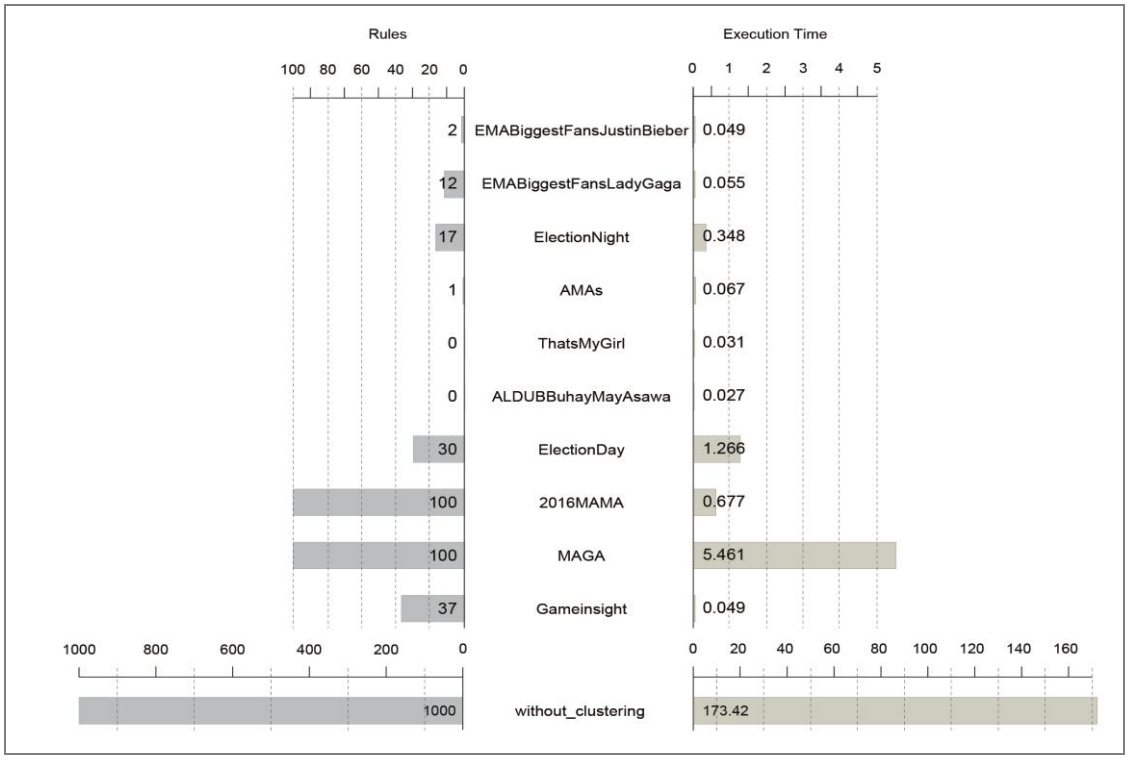

Figure 5.2 Test case #2: Comparison between n. of rules and execution time

The increase in execution time, in fact, is due to the number of attributes on which we execute the Apriori algorithm. With the clustering process, we partition them saving both times and execution resources. The execution time, in particular, represent a big issue in our analysis since the relationship with the number of attributes is more than linear. In fact, in the charts below, it seems to be approximately quadratic.

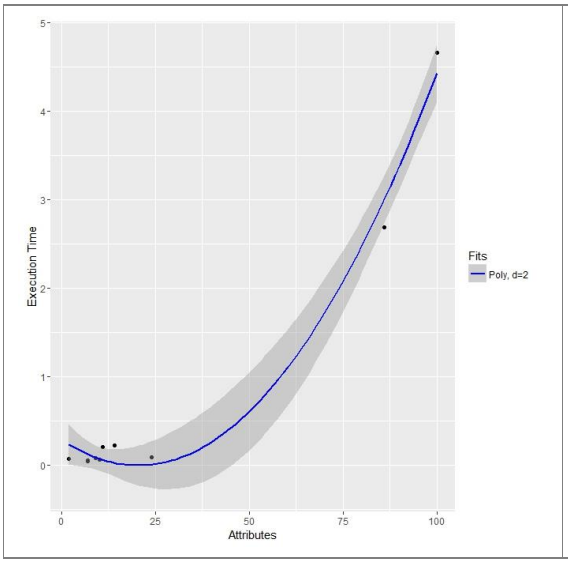

Figure 5.3 Test case #1: Relation between n. of attributes and execution time

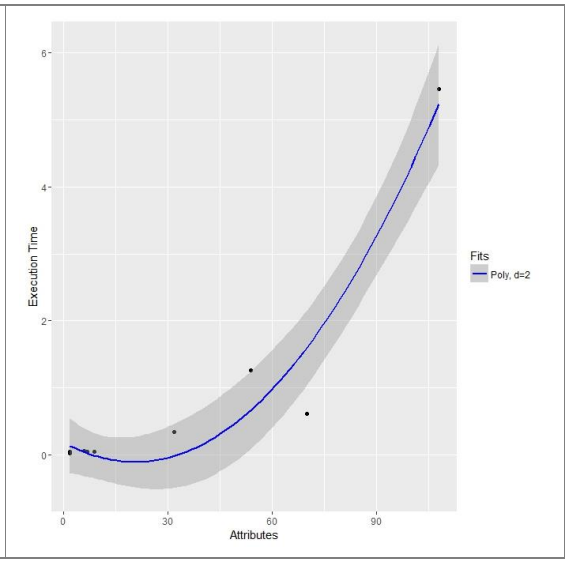

Figure 5.4 Test case #2: Relation between n. of attributes and execution time

If we sum up the execution time spent on every cluster and compare it with time spend in finding rules without clustering, we see that they are two order of magnitude far from each other, while rules found form two comparable itemsets.

6 CONCLUSIONS

The main challenge of this work was to find out a systematic and scalable approach to deal with non-structured data analysis. Our decisions to focus on Twitter data because of its unstructured nature and the huge number of raw data it can provide, suggests to us the use of data mining techniques as a possible way to conduct the analysis.

We show as the direct application of one of the most famous association rule mining algorithm, the Apriori, is not possible without a structuring phase in preprocessing.

At this point we took inspiration from market basket analysis, transforming tweets in itemsets made by hashtags they contain, building a sort of transactional database.

The objective was to highlight, into the Twitter feed, interesting topic and eventual correlations between them, studying the correlation between hashtags. As we deeply discussed, trending topics in social networks are usually related to real-life events and the user's engagement towards them.

Given the consideration above, we deal with some additional constraints. For example, the fact that real-life event detection should be done in real time to be considered valuable. For this reason, we focused our research on finding a way to reduce execution time and computational resources needed.

We supposed that the main problem in executing association rules mining could be the attribute space analyzed each time and not the number of transactions.

Other works on this field apply a prefiltration phase in pulling the network to limit the number of keywords but we only filtered non-english tweets.

To solve this issue, we proposed a clustering process to split hashtags, assigning them to smaller macro-areas before looking for rules. Results obtained confirms our hypothesis, showing as the execution time is in a quadratic relation with respect to the number of attributes.

The correctness of the process is confirmed to the fact that we do not lose information in terms of rules, executing the same analysis without applying clustering on the same dataset.

The test case of American elections proofs that, when the event outcome depends on people behavior we can also predict it.

Finally, we have noted and reported cases in which our approach does not give us useful information on event tracking. If the space search is very small, even if the training set (tweets) is big enough, we cannot find association rules that give us insights on the evolution of a trending topic.

6.1 Future work

Since the beginning, we focused on hashtags because we have studied how scientific community recognizes their semantic value and the use in Twitter. A possible future extension may be to find a preprocessing method to extract semantic keywords from a non-structured content as the Twitter feed. This could extend the same method over other domains in which the content is not classified through keywords.

We proofed how graph databases are the most suitable technology for storing this kind of data. Another possible research objective may be to try to use the mining frequent subgraph techniques instead of frequent itemsets mining before building the association rules.

Given the division in clusters we build, another possible way to improve our works could be to execute association rules mining in parallel on all of them.

We analyzed rules found manually in post processing, trying to extract human understandable information considering how the use of some keywords implies the use of some others to express a concept. In this way, the process of information extraction is not scalable with the increasing in number of rules. Finding a way for classifying them before going further with the analysis could be the key point to make also the information extraction scalable.

BIBLIOGRAPHY

ADEDOYIN-OLOWE, M., 2015. An association rule dynamics and classification approach to event detection and tracking in Twitter.

AGRAWAL, R., IMIELIŃSKI, T. and SWAMI, A., 1993a. Mining association rules between sets of items in large databases, *Acm sigmod record* 1993a, ACM, pp. 207-216.

AGRAWAL, R. and SRIKANT, R., 1994. Fast algorithms for mining association rules, *Proc. 20th int. conf. very large data bases, VLDB* 1994, pp. 487-499.

ALONSO, O., CARSON, C., GERSTER, D., JI, X. and NABAR, S.U., 2010. Detecting uninteresting content in text streams, *SIGIR Crowdsourcing for Search Evaluation Workshop* 2010.

ASUR, S. and HUBERMAN, B.A., 2010. Predicting the future with social media, *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on* 2010, IEEE, pp. 492-499.

BALMIN, A., HRISTIDIS, V. and PAPAKONSTANTINOU, Y., 2004. Objectrank: Authority-based keyword search in databases, *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30* 2004, VLDB Endowment, pp. 564-575.

BECKER, H., NAAMAN, M. and GRAVANO, L., 2011. Beyond trending topics: Real-world event identification on twitter.

CORNEY, D., MARTIN, C. and GÖKER, A., 2014. Spot the ball: Detecting sports events on Twitter. *Advances in information retrieval.* Springer, pp. 449-454.

FENG, W. and WANG, J., 2014. We can learn your# hashtags: Connecting tweets to explicit topics, *2014 IEEE 30th International Conference on Data Engineering* 2014, IEEE, pp. 856-867.

FRANK, E., HALL, M., HOLMES, G., KIRKBY, R., PFAHRINGER, B., WITTEN, I.H. and TRIGG, L., 2005. Weka. *Data Mining and Knowledge Discovery Handbook.* Springer, pp. 1305-1314.

HAN, J., PEI, J. and YIN, Y., 2000. Mining frequent patterns without candidate generation, *ACM Sigmod Record* 2000, ACM, pp. 1-12.

JAVA, A., SONG, X., FININ, T. and TSENG, B., 2007. Why we twitter: understanding microblogging usage and communities, *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis* 2007, ACM, pp. 56-65.

KOTSIANTIS, S. and KANELLOPOULOS, D., 2006a. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering,* 32(1), pp. 71-82.

KWAK, H., LEE, C., PARK, H. and MOON, S., 2010. What is Twitter, a social network or a news media? *Proceedings of the 19th international conference on World wide web* 2010, ACM, pp. 591-600.

MATHIOUDAKIS, M. and KOUDAS, N., 2010. Twittermonitor: trend detection over the twitter stream, *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data* 2010, ACM, pp. 1155-1158.

MONTAG, D., 2013. Understanding neo4j scalability. *White Paper, Neotechnology,*

MOONEY, R.J. and BUNESCU, R., 2005. Mining knowledge from text using information extraction. *ACM SIGKDD explorations newsletter,* 7(1), pp. 3-10.

MORSTATTER, F., PFEFFER, J., LIU, H. and CARLEY, K.M., 2013. Is the sample good enough? comparing data from twitter's streaming api with twitter's firehose. *arXiv preprint arXiv:1306.5204,* .

NAAMAN, M., BOASE, J. and LAI, C., 2010. Is it really about me?: message content in social awareness streams, *Proceedings of the 2010 ACM conference on Computer supported cooperative work* 2010, ACM, pp. 189-192.

ROBINSON, I., WEBBER, J. and EIFREM, E., 2015. *Graph Databases: New Opportunities for Connected Data.* " O'Reilly Media, Inc.".

ROMERO, D.M., MEEDER, B. and KLEINBERG, J., 2011. Differences in the mechanics of information diffusion across topics: idioms, political hashtags, and complex contagion on twitter, *Proceedings of the 20th international conference on World wide web* 2011, ACM, pp. 695-704.

WU, X., ZHU, X., WU, G. and DING, W., 2014. Data mining with big data. *IEEE Transactions on Knowledge and Data Engineering,* 26(1), pp. 97-107.

YAMAGUCHI, Y., TAKAHASHI, T., AMAGASA, T. and KITAGAWA, H., 2010. Turank: Twitter user ranking based on user-tweet graph analysis. *Web Information Systems Engineering–WISE 2010.* Springer, pp. 240-253.

YANG, M., LEE, J., LEE, S. and RIM, H., 2012. Finding interesting posts in twitter based on retweet graph analysis, *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval* 2012, ACM, pp. 1073-1074.