IT 13 079
Examensarbete 30 hp
November 2013

Distributed clustering algorithm for large scale clustering problems

Daniele Bacarella

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Distributed clustering algorithm for large scale clustering problems

Daniele Bacarella

Clustering is a task which has got much attention in data mining. The task of finding subsets of objects sharing some sort of common attributes is applied in various fields such as biology, medicine, business and computer science. A document search engine for instance, takes advantage of the information obtained clustering the document database to return a result with relevant information to the query. Two main factors that make clustering a challenging task are the size of the dataset and the dimensionality of the objects to cluster. Sometimes the character of the object makes it difficult identify its attributes. This is the case of the image clustering. A common approach is comparing two images using their visual features like the colors or shapes they contain. However, sometimes they come along with textual information claiming to be sufficiently descriptive of the content (e.g. tags on web images).
The purpose of this thesis work is to propose a text-based image clustering algorithm through the combined application of two techniques namely Minhash Locality Sensitive Hashing (MinHash LSH) and Frequent itemset Mining.

Handledare: Björn Lyttkens Lindén
Ämnesgranskare: Kjell Orsborn
Examinator: Ivan Christoff
IT 13 079
Tryckt av: Reprocentralen ITC

"L'arte rinnova i popoli e ne rivela la vita.
Vano delle scene il diletto ove non miri a preparar l'avvenire"

Abstract

Clustering is a task which has got much attention in data mining. The task of finding subsets of objects sharing some sort of common attributes is applied in various fields such as biology, medicine, business and computer science. A document search engine for instance, takes advantage of the information obtained clustering the document database to return a result with relevant information to the query. Two main factors that make clustering a challenging task are the size of the dataset and the dimensionality of the objects to cluster. Sometimes the character of the object makes it difficult identify its attributes. This is the case of the image clustering. A common approach is comparing two images using their visual features like the colors or shapes they contain. However, sometimes they come along with textual information claiming to be sufficiently descriptive of the content (e.g. tags on web images).

The purpose of this thesis work is to propose a text-based image clustering algorithm through the combined application of two techniques namely Minhash Locality Sensitive Hashing (MinHash LSH) and Frequent itemset Mining.

Preface

This thesis was conducted at Picsearch AB, a swedish company that provides image search services, under the supervision of Björn Lyttkens Linden.

Thesis reviewer is Associate Professor Kjell Orsborn, Department of Information Technology at Uppsala University.

Contents

Chapter 1

Introduction

During the last decade the size of datasets has increased and with it also the dimensionality of each object stored. Cluster analysis attempts to find natural clusters of similar objects and the complexity of the task depends on many factors that can be the numbers of objects to cluster and their representation in the dataset just to cite some.

Recently, clustering techniques have been applied to: image segmentation [1] , pattern recognition [2], optical character recognition [3], and information retrieval [4], with particular regard to the development of the web. Image clustering, in particular, represent a more challenging task compared to the text document clustering. Image similarity can be obtained by comparing the set of visual feature vectors (e.g. colors, faces, texture) or, in the case of web images for instance, by comparing the set of descriptive keywords collected according to some criteria. The first method probably returns more reliable results in terms of quality but it kills the performance. It is not an option to consider for a large set of images. A trade-off between reliability and performance is represented by the second method. The most simple way to compare two images is to check how many descriptive keywords they have in common. When some of the visual features are available, this knowledge can be exploited in combination with the textual information to obtain a more sophisticated result. The knowledge of visual features such as the presence of a human face in the image can be used to guide the clustering processing in case the textual information would not be sufficient. An example would be separating images of the city of *Paris* from those of the celebrity '*Paris Hilton*'.

Clustering such huge set of images make running even the most common clustering algorithm not trivial. The time complexity explodes over a certain size requiring more powerful machines to complete the task. The standard approach for big data processing is represented by the distributed computation: the original dataset is decomposed in smaller more tractable chunks and assigned to worker units that process them to then merge all the sub-results into one.

Hence it is needed a way to meaningfully split the dataset, a distributed processing framework to process them separately and efficiently and other techniques to refine the clustering.

However, in this thesis the emphasis is on developing an algorithm for image clustering based only on the textual information.

In the following chapters it will be shown how the goals stated above can be achieved with methods such as: Minhash LSH [5][6], Frequent Itemsets Mining [7] and the framework Condor [8].

Organization of the thesis

Chapter 2 provides a theoretical background on the concepts that underlie this thesis work. It starts by introducing the broader concept of clustering to get to a more specific presentation and analysis of the key techniques such as Minhash, LSH and Association rules mining.
Chapter 3 shows the proposed solution architecture along with the motivations. Some implementation details are also provided.
Chapter 4 contains a discussion on the experiments that have been performed on the implemented solution. All the shown results offer an interpretational comment.
The last chapter, Chapter 5, is on future work and conclusion.

1.1 Related work

Two examples of where clustering approaches are used to build some sort of hierarchical structure are Frequent Itemsets and Association Rules [9]. Because of the large number of association rules generated from a dataset, the analysis becomes hard. The authors propose an algorithm to cluster association rules for a user to understand.

In [10] the authors cluster images based on both visual features (in the paper they are referred to as 'signal features') and textual features as using only one is not considered sufficient to successfully disambiguate. However, the textual and visual features (like the dominant color in the image) are used to generate the association rules. Once the set of rules is obtained, this is used to generate a *hypergraph* (a graph in which an edge can connect any number of vertices) which is partitioned. As the hyperedges are weighted, the partitions are the result of a process similar to the minCut problem on graphs. The final clustering hence is represented by the set of visual and text features contained in each cluster.

Another work that inspired this thesis is the *Canopy clustering algorithm* [11]. It is based on the principle that to deal with large dataset, this has to be cheaply partitioned into overlapping subsets (*canopies*) and perform more computationally expensive clustering algorithms on each canopy.

Chapter 2

Theory and Definitions

In this chapter will be given some definitions along with an overview of the theories on which this thesis is based.

2.1 Clustering

The task of clustering is about discovering hidden patterns in a dataset in such a way to find sub-sets of data sharing common properties. Such properties are indeed not well defined and often there is no objective definition of what makes two items similar/close to each other. That is, measuring the distance between two point in a 2-dimensional space it might be easy. But when it comes to deal with high-dimensional data points, the definition of 'similarity/distance' can be interpreted in many different ways, hence be vague and hard to judge.

Clustering differs from *Categorization* in that it does not provides any mean to evaluate the accuracy of the output. In the latter, each item in the dataset comes along a *class label* with which is the possible to evaluate whether it has been assigned to the proper category in the model created or not. Thus, there is no *best criterion* suitable for any application which could imply that it depends on the instance of the problem and our expected result.

2.1.1 Supervised and Unsupervised learning

All the techniques aiming at learning can be grouped in two classes, namely *Supervised* and *Unsupervised* methods (Reinforcement Learning is also another class of problems).

- **Supervised methods** learn a certain function to predict the class an object belongs to. They go through a phase of *training* where the labelled data is used to build so a classifier. This one is later on tested against another set of data (also labelled) to verify its accuracy.
- **Unsupervised methods** instead are not provided with labelled data, hence it is their main problem to find the hidden structure.

2.1.2 Categories

Clustering methods are further divided in several categories, some of them are the following:

- **Density-based algorithms**: datasets are characterized by dense and sparse regions. Clusters are likely to be represented by dense regions as it implies points close to each other.
- **Hierarchical algorithms**: datasets are decomposed in a hierarchical structure according to a criterion. Such structure is built by mean of an agglomerative or divisive approach.
- **Partitioning algorithms**: the whole datasets is split in partitions; afterwards the clustering is evaluated using some metric.

2.1.3 Evaluation

The main goal in clustering is to obtain a set of clusters where the objects in the same cluster are strongly correlated according to a defined measure. Unfortunately, evaluating the goodness of a clustering is never a straightforward procedure and it also need to be chosen to better suit the current case.

In general there are two criterion to evaluate a given clustering: *Internal* and *External.*

Internal criteria makes use of two similarity measures, respectively *intra-cluster similarity* and *inter-cluster similarity.* The first defines how close are the objects in the same cluster, this index is preferably high; the latter defines how far objects in different clusters are from the other, this index is preferably low.

It is worth noting that good values of these indexes does not necessarily imply a good clustering. A clustering might have high intra-cluster similarity within each cluster, yet it does not give us any information as how good is the number of clusters found. For instance, if a natural cluster is divided in many clusters, no matters in how many partitions it is split, the values of the internal criterion will be potentially the same. For this reason, it is preferable to measuring clustering by testing it against a sort of *ground-truth* that would give us some hints about the accuracy. This ground-truth is often produced directly by a human. Below there is an example of *External criteria* [12].

- **Purity**: this simple measure tells us about the level of heterogeneity(purity) of the clusters.
  Given a clustering $C = C_1, .., C_n$ over a set of N objects, each one belonging to either one of the classes in $\Omega = \omega_1, .., \omega_n$, the purity is calculated as follows:
  $$purity(\Omega, C) = \frac{1}{N} \sum_k \max |\omega_j \cap c_k| \tag{2.1}$$
  where it takes values in the interval $[0, 1]$, more specifically the purity is 1 when each cluster contains only elements of the same class.

- **Normalized Mutual Information** (NMI): this measure suits well when the number of clusters is taken into account in calculating of the clustering accuracy. Purity ignores it. In fact, high purity can be obtained just by assigning a single cluster to each single object which it is something to avoid.

  The overall formula of NMI is the ratio of two other formulas:

  the numerator is represented by the *mutual information*
  $$I(\Omega, C) = \sum_k \sum_j \frac{|\omega_j \cap c_k|}{N} * \log \frac{N * |\omega_j \cap c_k|}{|\omega_j|\,|c_k|} \tag{2.2}$$

the denominator is formula of *Entropy*

$$H(\Omega) = -\sum_k \frac{|\omega_k|}{N} * \log \frac{|\omega_k|}{N} \tag{2.3}$$

Therefore:

$$NMI(\Omega, C) = \frac{I(\Omega, C)}{[H(\Omega) + H(C)]/2} \tag{2.4}$$

2.2 Shingling

Given a text document or any other document that can be seen as a set of ordered tokens, the term *shingling* refers to the *set of contiguous sub-sequences of tokens* in it.

2.2.1 k-shingles

A shingling where each sub-sequence is composed by k tokens.

Example 2.2.1. *Given* A = {`All work and no play makes Jack a dull boy`} *a* k=*4-shingling is:*

{`(All,work,and,no), (work,and,no,play), (and,no,play,makes), (no,play,makes,Jack),(play,makes,Jack,a),(makes,Jack,a,dull), (Jack,a,dull,boy)`}

2.3 Jaccard index

Jaccard similarity coefficient[13] (also kwown as Jaccard index) is used to measure the similarity (dissimilarity) of sets.

Given two sets of objects, A and B, the similarity of these two sets can be expressed as the ratio of the cardinality of their *intersection* and the cardinality of their *union*:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.5}$$

In case of A and B being represented as two vectors of n binary attributes, the Jaccard index is expressed with the following equation:

$$J(A, B) = \frac{J_{11}}{J_{10} + J_{01} + J_{11}} \tag{2.6}$$

There are 4 possible combinations of values for each attribute, the total numbers are indeed:

J_{00}: the number of attributes with value 0 in both A and B
J_{01}: the number of attributes with value 0 in A and 1 in B
J_{10}: the number of attributes with value 1 in A and 0 in B
J_{11}: the number of attributes with value 1 in both A and B
where $J_{00} + J_{01} + J_{10} + J_{11} = n$

Since the Jaccard index is based on joint presence, J_{00} is not considered in 2.6.

Using the same principle, the Jaccard index is used to measure the *dissimilarity* of two sets, also referred to as *Jaccard distance*:

$$J_{Distance}(A, B) = 1 - J(A, B) \tag{2.7}$$

2.4 Minhash

The Minhash scheme was introduced by A. Broder in [5]. Given two documents A and B , he formalizes the concept of *"roughly the same"* using the mathematical notion of *resemblance*. Based on the sampling approach, he proved that the resemblance $r(A, B)$ can be computed using a fixed size sample for each document. Furthermore, this technique has been adopted in the AltaVista search engine to clear the search results from near-duplicate documents[14].

To define the problem in terms of sets intersection, each document is considered as a set of tokens. If a document is a bag of strings of characters, it can be represented by the set of its k-shingles (see 2.2.1), such set is computed in time linear in the size of the document $O(|A|)$.

Example 2.4.1. *Given* A = {`the,brown,fox,jumped,over,the,lazy,dog`} *and* k=*3, the set of its k-shingles is:*

{`(the,brown,fox), (brown,fox,jumped), (fox,jumped,over), (jumped,over,the),(over,the,lazy),(the,lazy,dog)`}

After having obtained those shingles, then the resemblance r of two documents A and B is computed as follows:

$$r(A,B) = \frac{|S(A,k) \cap S(B,k)|}{|S(A,k) \cup S(B,k)|} \tag{2.8}$$

where *S(D,k)* represents the set of k-shingles of the document D and $r(A,B) \in [0,1]$. It is easy to see that the formula is the Jaccard index applied to the shingles.

The selection of the size k of the shingles affects both the degree of resemblance of two documents and the sensitivity to tokens permutation. As shown by the author in the paper, be X=`(x,f,i,l,e,s)` and Y=`(f,i,l,e,s,x)`, then the computed *r(X,Y)* is 100% using 1-shingles, 80% using 2-shingles and 75% using 3-shingles. Thus, the size of the shingles is set accordingly to the level of permutation sensitivity of interest.

However, while this way of measuring the resemblance is quite straightforward, the number of shingles generated can be high (especially for large documents) such that it can be prohibitive to store them in main memory.

The second part of the approach used in [5] attempts to estimate the size of the intersection by a *random sampling that can be done independently for each document.*

2.4.1 Estimating the Jaccard index

Computing the resemblance of two documents can be computationally expensive for large inputs, Broder provided a way to estimate the actual value of resemblance by random sampling the input documents. Moreover, as it will be shown later, the size of each sample can be reduced to a fixed value addressing the storing problem above mentioned.

Let Ω be a totally ordered set of k-shingles, where k is fixed. Given a fixed parameter s and a set $W \subseteq \Omega$, $MIN_s(W)$ is defined as:

$$MIN_s(W) = \begin{cases} \text{the set of the smallest } s \text{ elements in } W & |W| \geq s \\ W & \text{otherwise} \end{cases} \tag{2.9}$$

For a set $I \subseteq \mathtt{N}$ define:

$$MOD_m(I) = \text{the set of element in } W \text{ that are 0 mod } m \tag{2.10}$$

Finally the following theorem is presented:

Theorem 1. *Let $g : \Omega \to \mathtt{N}$ be an arbitrary injection, let $\pi : \Omega \to \Omega$ be a permutation of Ω chosen uniformly at random and let M(A)= MIN_s(π(S(A,*k*))) and L(A) = MOD_m(g(π(S(A,w)))). Define M(B) and L(B) analogously.*

Then

$$\frac{|MIN_s(M(A) \cup M(B)) \cap M(A) \cap M(B)|}{|MIN_s(M(A) \cup M(B))|} \tag{2.11}$$

is an unbiased estimate of the resemblance of A and B

Proof.

$$\begin{aligned} MIN_s(M(A) \cup M(B)) &= MIN_s(\pi(S(A,k)) \cup \pi(S(B,k))) \\ &= MIN_s(\pi(S(A,k) \cup S(B,k))) \end{aligned}$$

Let α be smallest element in $\pi(S(A,k) \cup S(B,k))$

$$\begin{aligned} P(\alpha \in M(A) \cap M(B)) &= P(\pi^{-1}(\alpha) \in S(A,k) \cap S(B,k)) \\ &= \frac{|S(A,k) \cap S(B,k)|}{|S(A,k) \cup S(B,k)|} \\ &= r(A,B) \end{aligned}$$

Thus, since it applies to every element in $MIN_s(\pi(S(A,k)) \cup \pi(S(B,k))))$, then the claim 2.11 is proved. $\square$

In other words, this suggests that the resemblance of two documents can be computed/estimated by just considering a small portion of each one. It also have the advantage to require less memory as $M(D)$ has a fixed size.

To further reduce the storage space required, each shingle can be mapped to an *id* of l bits (for instance using g). The random permutation π in this case is applied to the set $\{0, 1, ..., 2^l - 1\}$.

Example 2.4.2. *Let's suppose to break down a text document in a set of 4-shingles containing words. Each word generally comprise of about 5 characters of 1 byte each, that is 20 bytes for each shingle. If each shingle is mapped to an integer of 4 bytes, then the memory required is reduced by a factor of 4.*

Nonetheless, the number of bits used to represent the *id* plays a role on the quality of the estimate: *long representations* ensure a higher chance for an *id* to be unique but require more memory while *shorter representations* degrade the estimate due to large number of collisions.

Once such mapping function $f : \Omega \to \{0, 1, ..., 2^l - 1\}$ is fixed, then the resemblance is formulated as:

$$r_{k,f}(A, B) = \frac{|f(S(A, k)) \cap f(S(B, k))|}{|f(S(A, k)) \cup f(S(B, k))|} \tag{2.12}$$

2.5 LSH: Locality-Sensitive Hashing

Locality-Sensitive Hashing (abbreviated to LSH), introduced by Indyk and Motwani in [6], is a general theory that finds its focus on approximating the Nearest Neighbour task, that is, given a query point q, return p s.t. $\| q - p \| \leq (1+\epsilon)r$, where r is a fixed radius. For high dimensional datasets, time and space requirements turn to be an issue as they grow exponentially in the dimension.

Basically, the main idea [15] is to hash the input points such that the probability of collision is related to their distance. That is, points that are far from each other have fewer chances to be put in the same buckets compared to those that are closer.

2.5.1 Theory

Let S be the set of elements and D a distance function of elements in S. Define *B(q,r)* $= \{p : D(q,p) \geq r\}$ as the set of elements in S within the distance r from p.

The formalization of this concept [6] follows:

Definition 2.5.1. *A family* $H = \{h : S \to U\}$ *is called* (r_1, r_2, p_1, p_2)-sensitive *for D for any q, p,* $p^{'} \in S$

- *if* $p \in B(q, r_1)$ *then* $P_H[h(q) = h(p)] \geq p_1$
- *if* $p \notin B(q, r_2)$ *then* $P_H[h(q) = h(p)] \leq p_2$

Where $p_1 > p_2$ and $r_1 < r_2$ for the locality-sensitive family being useful.

Also, a family of locality-sensitive functions need to meet three conditions [16]:

1. They must be more likely to put together pairs of elements that are close to each other rather than those that are far, see Definition above
2. They must be *statistically independent*, such that the probability of several functions is predictable by the product rule of independent events
3. They must be efficient in two ways:
   - Candidate pairs identified in time much less than the number of pairs
   - They can be combined in such a way to reduce either false positives and negatives

2.5.2 Minhash LSH

It has been shown how efficiently (both in space and time) the similarity of two elements can be estimated. However, for large datasets, even if the minhash signatures are much smaller in size, the number of possible pairs is still high and so the task of finding the pairs with the greatest similarity remains computationally expensive.

In this case LSH can be of help; as it is preferable not to compare all the pairs but the most similar ones, the LSH scheme can be applied to obtain the set of the candidates to be the most similar elements, performing a pairwise check only against those.

One general approach is the *banding technique.* The minhash signatures of size n are divided in b bands of r rows each, where $n = b * r$. Let's consider the example shown in Figure 2.1. The signatures matrix is divided in 4 bands each containing 3 rows. Furthermore for each band, a hash function is applied mapping each column to a large value representing the bucket id. the As it is easy to see, the second and the fourth columns have the same vector ([0,2,1]) thus, they will likely be put in the same bucket. Eventually, for each band a different hash function will be used so that columns with the same values in different band will be put in different buckets.

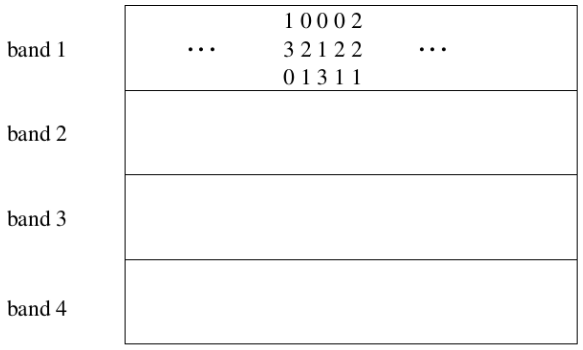

Figure 2.1: Signature matrix - Signatures are divided in 4 band of 3 rows each - (Rajaraman, Anand and Ullman, Jeffrey David) [16]

It is worth noting that the values of the parameters b and r impact on the accuracy.The statement will be soon justified.

Collision analysis of the Banding Technique

Let M be a signatures matrix divided in b bands of r rows each and suppose to be interested in similar documents with a Jaccard index s. because of the theorem 2.11, the probability that signatures agree on one particular row of the signature matrix is equal to s. In the light of this, the probability that two documents become a candidate pair can be computed as follows:

1. The probability that the signatures agree in all rows of one particular band is s^r
2. The probability that the signatures don't agree in all rows of one particular band is $1 - s^r$
3. The probability that the signatures don't agree in all rows of any of the bands is $(1 - s^r)^b$
4. Finally, the probability that signatures agree in all rows of at lest one band is $1 - (1 - s^r)^b$

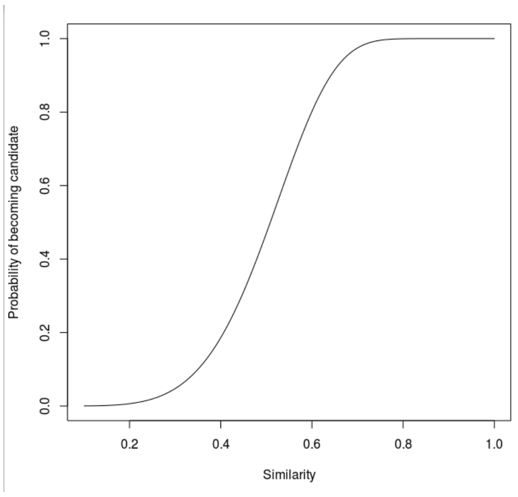

Figure 2.2: S curve b=20 r=5

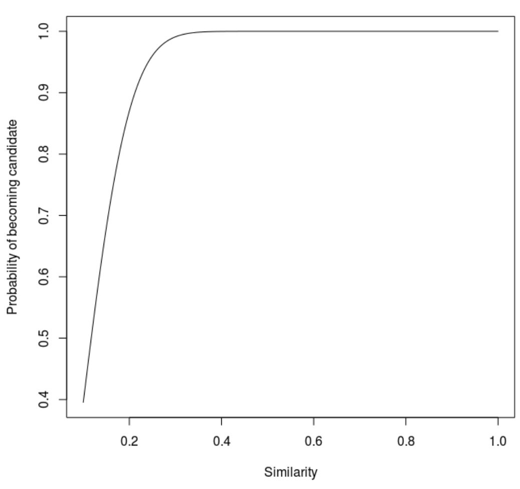

Figure 2.3: S curve b=50 r=2

The plots of the probability function $1 - (1 - s^r)^b$ show us that the parameters b and r change the slope of the function. In Figure 2.2 it is shown a sort of step function where the similarity *threshold* (the point at which the slope becomes steepest) is approximately at 0.55. Figure 2.3 instead is meant to show the fact that the slope can be arbitrarily shifted; in this case the slope is shifted all the way to left so that two signatures with a similarity of 0.1 have roughly 50% chance of becoming candidates and more time than 70% if their similarity is 0.2 .

Fixed b and r, an approximation of the threshold t is $(\frac{1}{b})^{\frac{1}{r}}$.

Given a threshold t, if the attempt is to limit the *false negatives* then b and r are accordingly selected to generate a threshold lower than t. Conversely, to limit the *false positive* and speed up the computation then b and r are selected to generate a higher threshold.

2.6 Association rules mining

The Association rules mining [7] is an *unsupervised learning* method aiming to find particular relations between items over a database of transactions. A typical example of association rules mining is the *Market Basket Analysis* where the set of customer's transactions are analysed to find hidden patterns useful to understand their preferences and so trying to predict their next purchases. For instance, an insight over the database may reveal a strong relation between the purchase of *bread* and of a particular *cheese*; this knowledge can hence be exploited by putting, for instance, the two product close together on the shelf. Association rules mining finds its application in many other domains such as *fraud detection*, *medical diagnosis*, *census data*, *fraud detection*, *Web log mining* along with many others.

The following section provides the formal definitions of the basics concepts in Association Rules mining such as *frequent itemset*, *association rule* along with evaluation metrics, namely *support* and *confidence*.

2.6.1 Concepts definition

Let $I = \{i_1, i_2, ..., i_n\}$ be a set of items and D a set of transaction T where $T \subseteq I$, each one identified by an unique identifier.

Frequent itemsets: Given an itemset $X \subseteq I$, its *support* $s(X)$ is defined as the number of transactions in which it is contained:

$$s(X) = |\{t : X \subseteq t, t \in T\}| \tag{2.13}$$

An itemset X is said to be *frequent* if its support is at or above a fixed *minimum support*:

$$s(X) \geq min\_support \tag{2.14}$$

Example 2.6.1. *Let's suppose to have a set of tagged images of domestic animals:*

Image id	*Tags*
1	*dog, puppy, cute, friend, funny*
2	*outdoor, running, dog*
3	*cute, dog, play*
4	*dog, friend*
5	*cat, funny, couch*
6	*friend, dog, best*
7	*cute, cat, picture*
8	*funny, lol, cat, animals*
9	*silly, cat, funny, pictures*
10	*cute, baby , animals, cat*

Fix a minimum support, ***min_supp****=3, the frequent itemsets are:*

`{dog}, {dog, friend}, {cute}, {cat}, {cat, funny}, {friend}`

A quick insight reveals that about a third of the people posting pictures actually think about ***dogs*** *as their* ***friends*** *and that* ***cats*** *are* ***funny****.*

Association rules: Given two itemsets $X, Y \subseteq I$, an association rule is defined as:

$$X \rightarrow Y \tag{2.15}$$

That is, *every time you find an occurrence of X you also find Y.*

Similarly to frequent itemsets, these rules are mined according to some metrics. The most common way to measure the interestingness of a rule is called *confidence*:

$$Confidence(X \rightarrow Y) = \frac{s(X \cup Y)}{s(X)} \tag{2.16}$$

It is worth to note that the confidence of $X \rightarrow Y$ does not necessary equals the confidence of $Y \rightarrow X$. In fact, while the numerator is the same both for

$Confidence(X \rightarrow Y)$ and $Confidence(Y \rightarrow X)$, the denominator changes as it is the support of the head of the rule.

Example 2.6.2. *Let's compute the confidence of the rule cat → funny and funny → cat*

$$\begin{aligned} \textit{Confidence(cat} \rightarrow \textit{funny)} &= \frac{s(cat \cup funny)}{s(cat)} \\ \textit{Confidence(funny} \rightarrow \textit{cat)} &= \frac{s(cat \cup funny)}{s(funny)} \end{aligned}$$

Since $s(cat, funny) = 3$, $s(cat) = 5$ and $s(funny) = 4$, the confidence of each rule is:

$$\begin{aligned} \textit{Confidence(cat} \rightarrow \textit{funny)} &= \frac{3}{5} = 0.6 \\ \textit{Confidence(funny} \rightarrow \textit{cat)} &= \frac{3}{4} = 0.75 \end{aligned}$$

When it comes to evaluate how interesting a rule can be, a wide set of measures can be chosen such as support, confidence, *lift*, *correlation* or *entropy* just to cite some. It really depends on the instance of the problem as they may provide conflicting information about the interestingness of a rule [17].

2.6.2 Association Rules mining algorithms

The process of generating the association rules comprises of two phases:

- Frequent itemsets mining, according to a specified minimum support.
- Rules mining from frequent itemsets, where low confidence rules are pruned.

Rules generated according to these two phases have both high support and high confidence.

The first phase is the one that requires more effort in the implementation. Given d unique items in I in fact, the total number of possible itemsets is 2^d. In real database the number of unique items is quite high so the task would turn to be computationally prohibitive. Nonetheless, there are data structures and algorithms that considerably reduce the space of the possible frequent itemsets by applying smart methods along with the *anti-monotonicity property* which basically states that all the subsets of a frequent items set are also frequent.

In regards to generating rules, this is more straightforward. Given an itemset X of n items, there are n possible rules that can be generated out of it, namely $X - \{i\} \rightarrow i$ for each i in X.

Algorithms

Algorithms for frequent items set mining are divided in two categories:

candidate-generation-and-test approaches (e.g. Apriori[18] algorithm) , where a set of candidate patterns of length $k + 1$ are generated from a set of frequent patterns of length k ($k \geq 1$). It is not a good fit when there exists a large number of frequent patterns. In general it performs many passes over the dataset to fetch patterns frequencies.

pattern-growth methods (e.g. FP-growth[19], Tree Projection [20], H-mine [21]), an algorithm belonging to this category still use the anti-monotone property but, instead of generate every time a candidates set, it recursively partitions the database according to the frequent patterns found and searches for local frequent patterns to assemble longer ones.

Anyway, independently from the approach adopted, some difficulties can be derived also from the data at hand :

- Main memory consumption: hard to predict, huge space required.
- Real database usually contains data present with several 'patterns' with which we have to deal with
- Large applications requires scalability

The ideal algorithm should efficiently generate all the frequent itemsets and association rules with both great savings of memory and cpu usage. Moreover, it should scale keeping a reasonable performance.
The algorithm that has been chosen for this project is *H-Mine*.

Transaction id	Items	Frequent-item projection
100	c, d, e, f, g, i	c, d, e, g
200	a, c, d, e, m	a, c, d, e
300	a, b, d, e, g, k	a, d, e, k
400	a, c, d, h	a, c, d

Table 2.1: Transactions Database TDB

H-Mine H-Mine[21] is a pattern-growth method and unlike algorithms such as Apriori, it reduces considerably the number of scans over the databases (in particular it performs only two) and use a compact data structure to generate the frequent item sets. Similarly to FP-growth that uses a structure called FP-Tree to obtain a compressed representation of the database, H-mine has a data structure using hyper-links called *H-struct*, designed for fast mining. In addition to H-struct, the authors proposed a new algorithm, *H-mine(Mem)*, for mining frequent patterns for the data sets that can fit into the main memory. Their studies show that H-mine is more space efficient than FP-growth on sparse data sets and also experimental result show that, in many cases, it has a limited and exactly predictable space overhead. Last, to deal with data sets that do not fit in the main memory, they propose *H-mine*, a scalable algorithm that by first partitioning the database, mines each partition in the memory using H-mine(Mem), and then consolidates globally frequent patterns. For dense data sets, H-mine is integrated with FP-growth dynamically by detecting the swapping condition and constructing FP-trees for efficient mining.

General idea of *H-mine(Mem)* Table 2.1 describe our database for this example. Let the minimum support be $min\_sup = 2$. For each transaction d, the third column contains the list of frequent items. All the infrequent items are hence discarded because of the *Apriori* property. So, a first scan over the database is required to get the complete set of frequent items.
Given the ordered list of frequent items (*F-list*), for example by alphabetical order:

```
{a,c,d,e,g}
```

the complete set of frequent patterns can be partitioned into five subsets shown in Fig. 2.4 A second scan is needed to build the H-struct, more specifically to each transaction corresponds a frequent-item projection where the items are sorted according to the *F-list*. Every occurrence of a frequent item i stored in an entry with two fields: an *item-id* and a *hyper-link*.

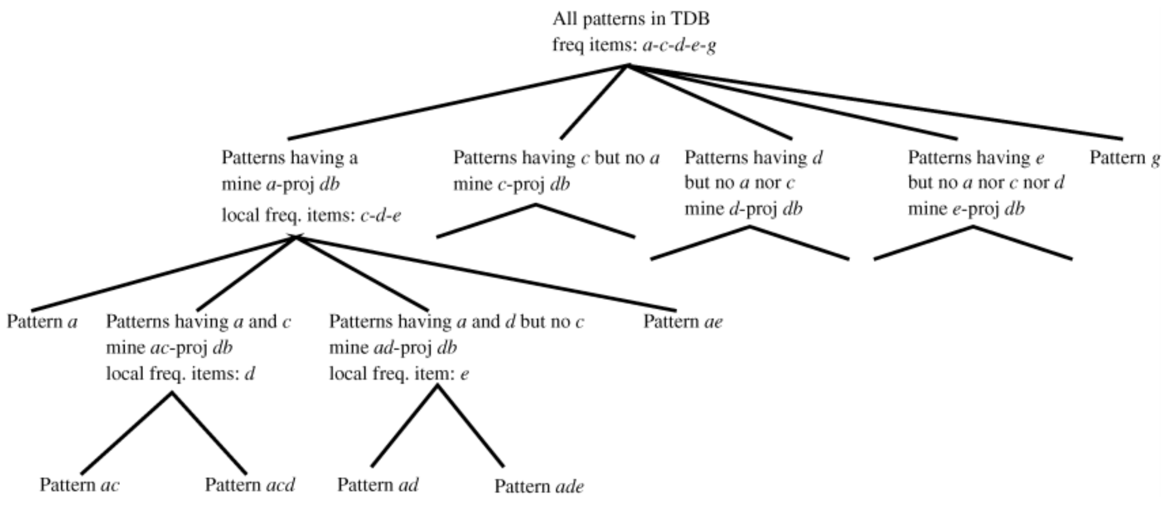

Figure 2.4: Frequent patterns subsets - (Pei, Han ,Lu , Nishio , Tang , Yang 2001 [21])

Apart from the projections, a *header table H* is created, with each frequent entry having three fields: an *item-id*, a *support count*, and a *hyper-link*. When the frequent-item projections are loaded into the memory, those with the same first item (in the order of the *F-list*) are linked together by the hyper-links into a queue, and the entries in the header table *H* act as the heads of the queues. When the *H-struct* is built, the remaining mining can be performed in main memory without referencing any information the original database, see Fig. 2.5

Let's proceed describing one step of the algorithm finding all the frequent patterns containing item *a*. This requires us to search all the frequent items projections containing item *a*. They can be easily obtained by traversing the *a-queue* starting from the head stored in the *header table H*. To mine the *a*-projected database a *a-header table H_a* is created as shown in Fig. 2.6
In H_a , every frequent item, except for *a* itself, has the same three fields as

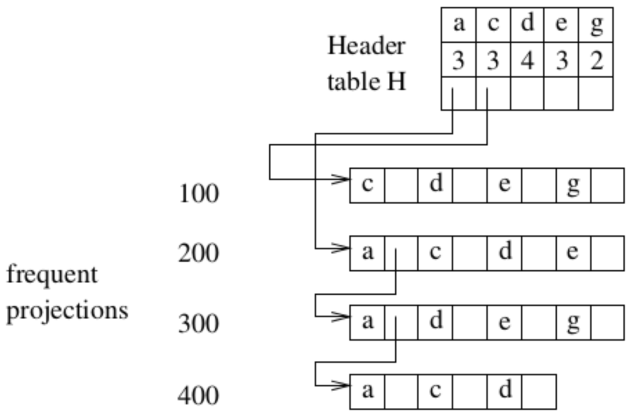

Figure 2.5: H-struct- (Pei, Han ,Lu , Nishio , Tang , Yang 2001 [21])

in H. The support count in H_a records the support of the corresponding item in the a-projected database. As mentioned above,traversing the queue once is sufficient to obtain the support counts for the a-projected database. The locally frequent items are $\{c : 2, d : 3, e : 2\}$, item g ($\{g : 1\}$) is discarded since its support is lower than the minimum required. This scan outputs three frequent patterns: $\{ac : 2, ad : 3, ae : 2\}$ and builds up links for H_a header.

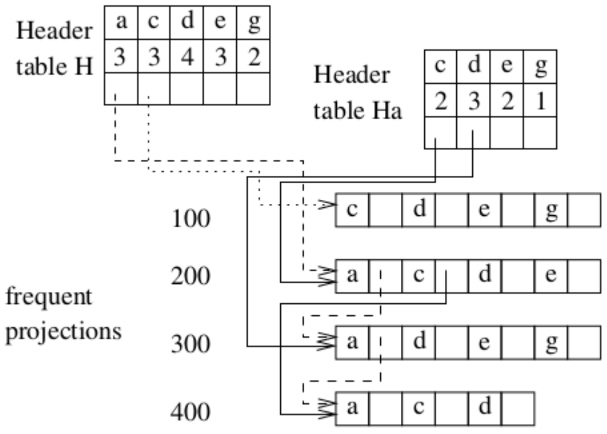

Figure 2.6: Header table H_a - (Pei, Han ,Lu , Nishio , Tang , Yang 2001 [21])

The process continues with the ac-projected database by examining the c-queue in H_a building a H_{ac} header table as shown in Fig. 2.6.
Here only item d is found frequent $\{d : 2\}$ so the output is just $\{acd : 2\}$
The recursion backtracks to find frequent patterns containing a and d but not

c and so on until all the sub trees are visited.

Space requirements An H-struct has an *headertable*. The header table is an array of frequent items in the order of the $F - list$. Each entry has two fields: a *supportcount*, and a *hyperlink* to the *heads* of the *queues* if the frequent item projections. For every transaction H-struct stores its frequent item projections. Besides them, it stores a header table whose size is bounded by the number of frequent items. Therefore, the space requirements is $\theta(\sum_{t \in DB} |freq(t)|)$, where $freq(t)$ is a frequent item projection of a transaction t. Moreover, the number of *headertables* is bounded by the maximal length of a single frequent pattern (not all the header tables need to be generated in a particular moment).

Theorem 2. *(Space complexity) The space complexity of H-mine(Mem) is* $\theta(\sum_{t \in DB} |freq(t)|)$*, where freq(t) is a frequent item projection of a transaction* t *.*

2.7 Hierarchical Frequent Itemsets Clustering

The intuition behind the transactions clustering using frequent itemsets is that eventually, transactions within the same cluster share more itemsets than those in different clusters. In the case of document clustering, where each document (transaction) can be represented as a set of words (items), two documents can be considered belonging to the same cluster when those two share a certain frequent set of terms. In the light of this observation then, the frequent itemsets mining can provide the basics elements of documents clustering.

The method which this thesis is based is about building a hierarchy of itemsets from the set of frequent itemsets mined. In particular, the *Frequent Itemset-based Hierarchical Clustering* (FIHC) [22] attempts to address the problems of document clusterings such as high dimensionality, ease of browsing and meaningful cluster labels. As a matter of fact, document clustering (as clustering in general) is a form of unsupervised learning and as such, it has no mean to clearly measure the similarity of two elements in contrary to document classification where the documents come along with a label attached.

FIHC, as opposed to the article "Frequent term-based text clustering" [23], assign documents to the best cluster among all of the available itemsets that have been found frequent.

It will be here introduced a list of concepts definition along with a brief explanation of the whole procedure:

- *Global frequent itemset*: set of items that appear in a fraction of the dataset in or above a fixed threshold
- *Global frequent item*: item belonging to a Global frequent itemset
- *Global frequent k-itemset*: Global frequent itemset containing k items
- *Cluster frequent (Global frequent item)*: item contained in some minimum fraction of document in the cluster

Constructing Clusters

Contextually to each global frequent itemset found, a node (here also referred to as cluster) labelled with that itemset is created. Documents are assigned to every node that match the label. Thus, initials clusters *overlaps* because a document may contain several global frequent itemsets.

Making Clusters Disjoint In this step documents are left only in the cluster that matches the most according to a certain metric. By doing so, clusters do not overlap any more. For each document j and for each cluster i in which it is contained, the score function adopted by the author is the following:

$$Score(C_i \leftarrow doc_j) = [\sum_x n(x) * cluster\_support(x)] - [\sum_{x'} n(x) * global\_support(x')] \tag{2.17}$$

where x represents an item that is both global frequent in doc_j and cluster frequent in C_i, while x' is global frequent in doc_j but not cluster frequent in C_i; $n(x)$ and $n(x')$ are weighted frequency of x and x' in doc_j. So doc_j is left in the cluster C_i that maximizes the score function.

Building the Cluster Tree

The next logical step is the actual tree building. As the clusters generated in the previous step can be considered as a set of topics and subtopics (e.g. `{nature}`,`{nature, flower}`,`{nature, landscape}`) the tree is constructed based on the similarity of those.

Tree construction In the construction each cluster has exactly one parent node, except the root node. The cluster tree is built in a bottom-up fashion; for each cluster at level k the best parent at level $(k-1)$ is chosen. Furthermore, the candidate parent nodes labels need to be a subset of the label of the current cluster. To choose the best parent among the set of candidates, the same score function 2.18 is used. Let C_i be the current node, all the documents contained in its sub-tree are merged in one and passed as parameter of the score function for every potential parent node. This is easy since the construction is done incrementally in a bottom-up fashion.

Tree pruning The resulting tree can be broad and deep depending on the documents content and the minimum support. After the construction, further changes are applied to the tree to reduce its size. A reason for doing this is that a hypothetical cluster is split in many little clusters leading hence to a poor clustering accuracy.

Whether two clusters C_i and Cj are parent and child or two sibling nodes, it is needed to introduce the notion of *similarity* of two clusters and the notion of *inter-cluster similarity.*

The first is defined with the following:

$$Sim(C_i \leftarrow C_j) = \frac{Score(C_i \leftarrow doc(C_j))}{\sum\limits_{x} n(x) + \sum\limits_{x'} n(x')} + 1 \tag{2.18}$$

while the latter is defined as the geometric mean of the $Sim(C_i \leftarrow C_j)$ and $Sim(C_j \leftarrow C_i)$:

$$Inter\_Sim(C_i \Longleftrightarrow C_j) = [Sim(C_i \leftarrow C_j) * Sim(C_j \leftarrow C_i)]^{\frac{1}{2}} \tag{2.19}$$

Child pruning Child pruning is performed to shorten the tree, by starting from the bottom of the tree going up to the top, stopping at level 2 since level 0 (root node) contains only unclustered documents. For each cluster in each level, if the inter-similarity with its child node is above 1 then the two clusters are merged, and the child node's children become children of the parent node.

Sibling Merging Since the child pruning is only applied to level 2 and below, usually level 1 contains many clusters that should be merged. In the child pruning the number of comparisons is linear in the number of children nodes, whereas the sibling merging is quadratic since the *Inter_Similarity* is calculated for each pair of nodes in the same level. Nonetheless it applies only to level 1 because child pruning has merged similar children into their parent.

Chapter 3

DistHC: Distributed Hierarchical Clustering

In this chapter it will be shown the work realized for this thesis project along with the implementation details.

More specifically in the very next section it will be given a picture of the overall idea followed by the motivations for each decision taken that led to the final proof of concept.

3.1 General idea

When it comes to think on ways to solve a big problem, generally the first idea is for decomposing it in to smaller instances for then attacking them separately. By doing so the hope is to reduce their complexity making them easy to solve. The original solution will be then the composition of the solutions of all the sub problems.

A divide et impera approach thus mitigates the complexity derived by the size of the problem enabling scalability for some extent.

Minhash LSH introduced in Section 2.5.2 seemed to be the best fit to accomplish to this task. As a matter of fact, it only needs the current object to decide in what bucket to put it in. Eventually, the original dataset could be split in X parts, applying an instance of the implementation of Minhash LHS to each partition and then merging the buckets with the same identifier.

The result is a set of buckets, which size can vary from few to very many. As consequence of the original dataset partitioning and/or with the more than reasonable assumption that the buckets are not pure, (that is contains objects not related to each other) a further refinement is applied by mining the set of frequent itemsets from each bucket and consequently building a hierarchical structure based on those.

The output of this phase will be, for each bucket created in phase 1, a tree where each node is labelled with a frequent itemset that it can be considered as a topic. Building and image assignment are explained in Section 2.7.

The last step is simply about merging all the trees by their node's label; in case two tree have a node in common, the set of images are merged and stored in only one tree is updated while the other one after the merging will be discarded.

So far our approach holds, in theory, the following requirements that a clustering algorithm should have:

- **scalability**: by decomposing the original problem in sub-problems processed in a distributed fashion

- **finding clusters with arbitrary shape**: the hierarchical structure built using the frequent itemsets mined changes based on the minimum support.

- **insensitivity to order of input records**: as LSH aims to put similar objects in the same bucket, the order does not matter.

- **high dimensionality**: the vector of keywords for each image in the dataset may be of any size. In fact, Minhash estimates the resemblance of two documents by random sampling.

3.2 Architecture

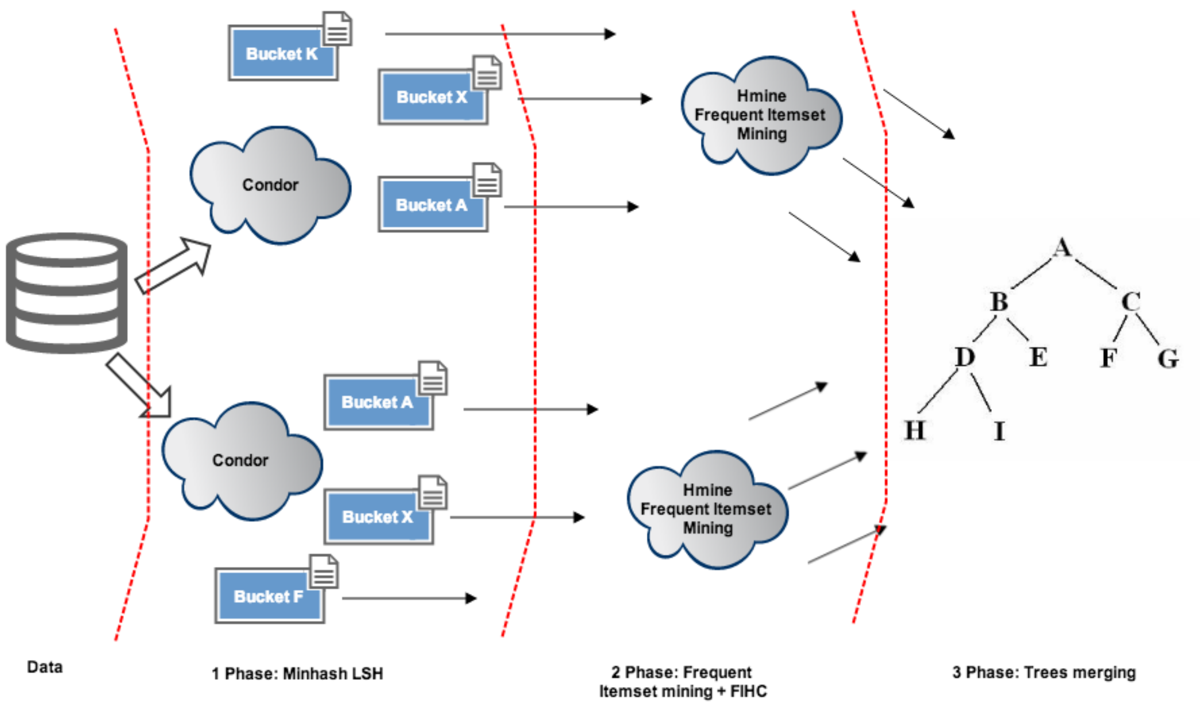

Figure 3.1: DistHC Architecture

Data flow and processing

A dataset entry has the following format:

`<Image ID>  {<Keyword>} +`

All the information contained in the dataset undergo a series of processing and transformations along the application of the algorithm. Processing is concerned mostly on the quantitative and qualitative aspects (e.g. least interesting keywords are removed).

Preprocessing: For each image entry in the dataset, all the keywords attached are checked against a list of stop-words and removed in case of matching. Such words are, using a general definition, those that are very common, bringing little value in describing a concept in a text for example. Examples of stop-words are words such as '*are*', '*for*', '*the*', '*an*' and so on. Moreover, the keywords are kept in memory in their stemmed form as to increase the chance of matching in case a word is present in many forms.

Phase 1: Minhash LSH This phase requires the dataset being entirely processed and hence loaded in main memory. It is here that the entire range of entries is split into an arbitrarily number of non overlapping subsets. The entries within each single subset are used to feed a different instance of the Minhash LSH algorithm being deployed on to a dedicated Condor job. When all the deployed jobs are completed, the output is a set of files containing data entries with the same format as in the original dataset. Nonetheless, the dataset obtained by merging all the generated files differs from the original as many keywords have been eventually removed during the preprocessing step.

Phase 2: Frequent Items set mining, FIHC tree building Similarly to how it has been done in the previous phase, each file is passed as input to an instance of H-mine and FIHC algorithms being executed in yet another dedicated Condor job.

The output of each job is a file containing a representation of the tree's structure derived through the application of the above-cited algorithms to the input file. In particular for each node in tree there is a line formatted as follows:

`<Node's label>  <Parent's label> {List of Image ids assigned}` +

Example 3.2.1.
Input file: `{<100 flower nature>,<200 nature>}`
Output file: `{<nature [root_node] 200>,<flower,nature nature 100>}`

Phase 3: Tree merging In this final phase, all the several trees generated are simply merged by their nodes' labels so that only one will remain at the end of the process.

Example 3.2.2.
Input file 1: `{<nature [root_node] 200>,<flower,nature nature 100>}`

Input file 2: `{<nature [root_node] 456>,<flower,nature nature 123>,`
`<animal [root_node] 222>}`
Output file: `{<nature [root_node] 200,456 >,<flower,nature nature 100,123>,`
`<animal [root_node] 222>}`

Chapter 4

Experiments and Results

All the algorithms in the proof of concept have been implemented in C++.

Experiments have been conducted on a machine with the following characteristics: 7.5 Gb Ram, Processor Intel i7 @3.4GHz x8 with SSD hard drive.

The distributed processing has been realized by deploying jobs on Condor[8], a workload management system for compute-intensive jobs.

4.1 Dataset

4.1.1 Synthetic dataset

The dataset used for the clustering quality measurement is taken from [24]. It is an archive of $25K+$ pictures tagged by the Flickr's users.

For the purpose of the clustering validation task, only a selected part of it has been used. Starting from the original set of images and considering all the keywords before any preprocessing operation, a total 2990 pictures have been selected and judged by a human to form 6 clusters, namely: *dog*, *beach*, *sky*, *cloud*, *flower*, *nature*.

4.1.2 Real dataset

Real dataset has been provided by Picsearch having the following format:

`<Image ID>  {<Keyword>}` +

4.2 Results

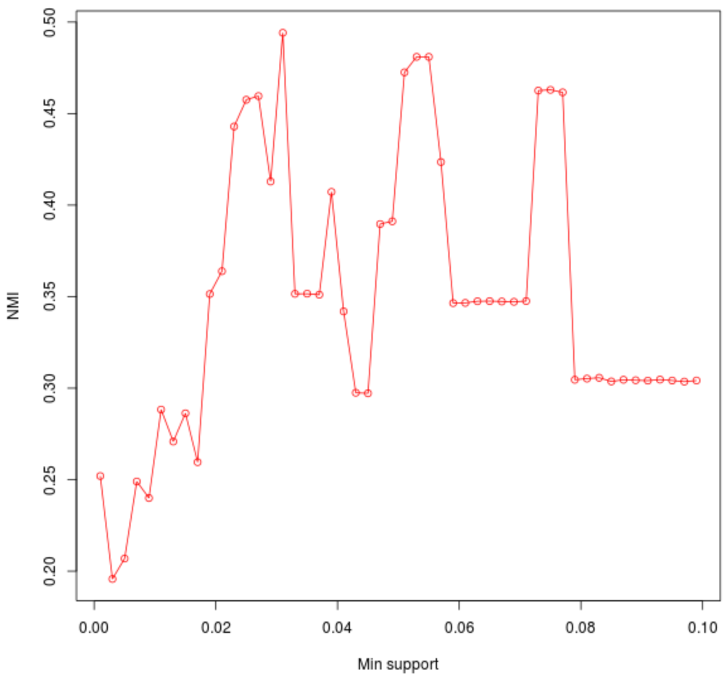

Figure 4.1: Evaluation on the buckets generated after LSH: 1-shingles Band:3, Rows:2

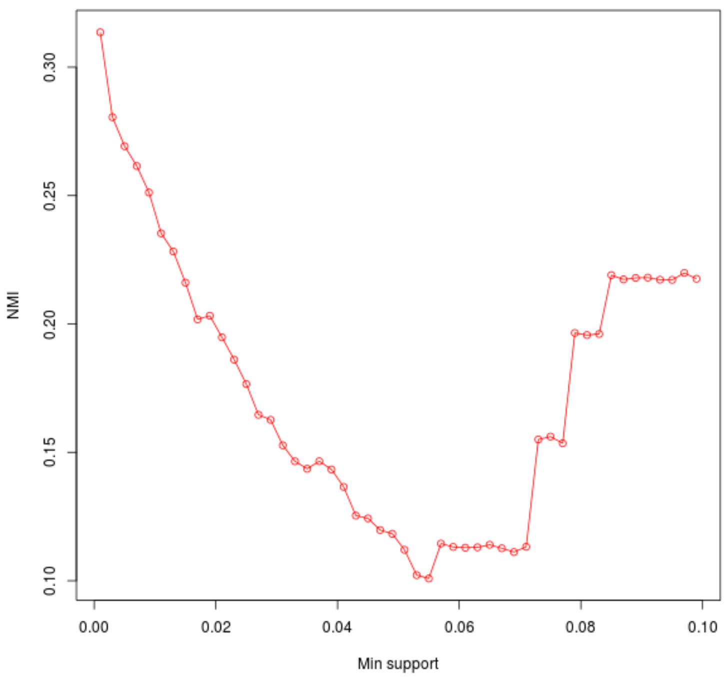

Figure 4.2: Evaluation on the buckets generated after LSH: 3-shingles Band:3, Rows:2

Figures 4.1 and 4.2 show an evaluation of the preliminary clustering represented by the buckets generated during the application of the Minhash LSH algorithm. The study of the clustering quality at an earlier stage can help at showing causes of degradation of performance, towards the choice of a better set of parameters.
At first glance, running the algorithm with 1-shingles gives us better results than using 3-shingles. In particular Figure 4.2 shows how the quality degrades reaching its minimum nearby $Minsupport = 0.05$. From there, the climax starts being ascendant though the values keep being lower than those obtained with 1-shingles, on average.

However, concerning the sole number of buckets generated, a couple of considerations can be made: if the algorithm fails at putting together candidate similar documents into a reasonably limited number of buckets, the second step which is supposed to mine frequent itemsets from each bucket might end up in not having enough data in order to extract useful information. Also, if this occurs, the operation of trees merging at the end of the process might take more than necessary due to the large number of resulting trees.

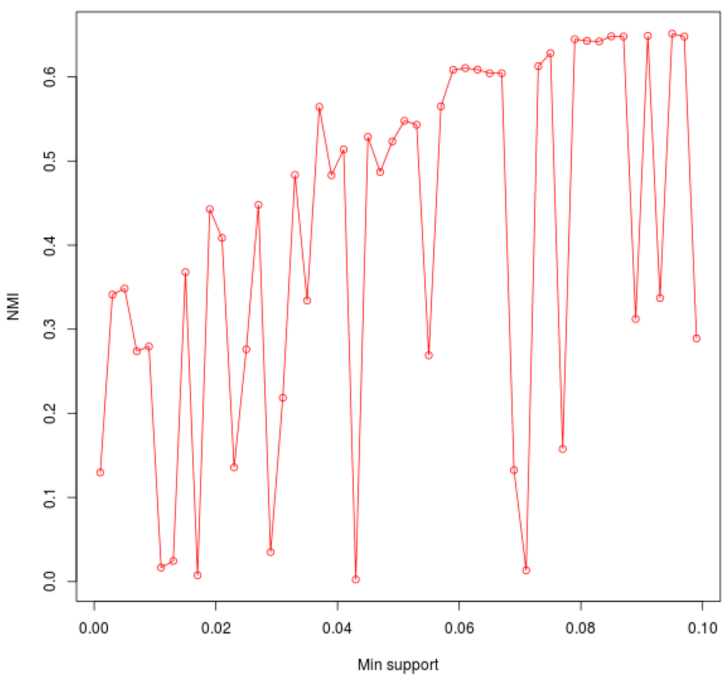

Figure 4.3: Evaluation on the final clustering: 1-shingles Band:3, Rows:2

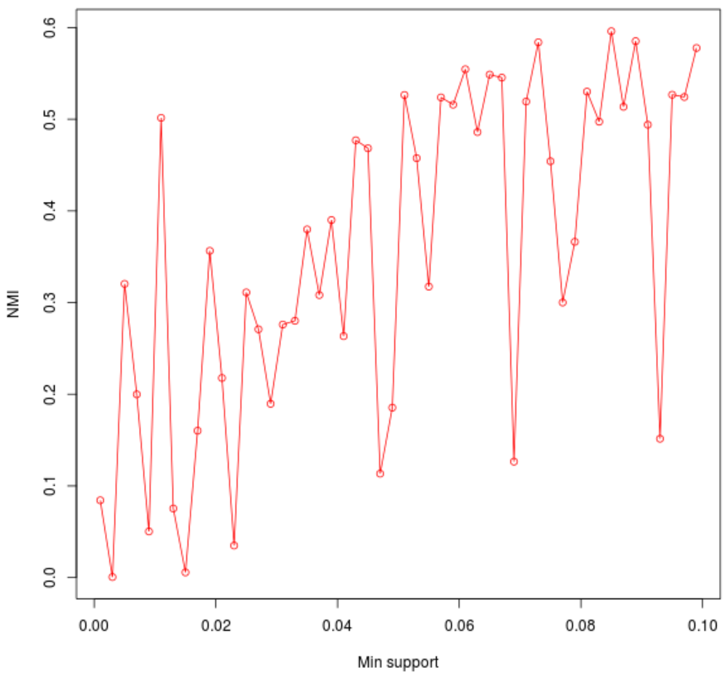

Figure 4.4: Evaluation on the final clustering : 3-shingles Band:3, Rows:2

Figures 4.3 and 4.4 show an improvement tendency on the accuracy. Other than having reached a new optima($NMI > 0.6$ within the range of minsupport 0.08 and 0.10), most of the measurements in the final clustering Fig.4.3 lie over $NMI = 0.4$ while in buckets measurement 4.1 this threshold is near $NMI = 0.3$.

The improvement is more evident between Figure 4.2 and Figure 4.4 with a 3-shingling.

Note: *The remarkable jagginess can be due to a non-deterministic (probably due to a not robust implementation of the Minhash LSH algorithm) content of the generated buckets. As mentioned in the discussion made about Figures 4.1 and 4.2, an excessive segmentation of the candidate similar documents can lead to a bad clustering quality.*

For the sake of completeness, it is here shown a snapshot of the hierarchy built upon the frequent itemsets found(Min support=0.02)

```
-PARENT (ROOT),
    -LEAF abigfav,
    -LEAF anawesomeshot,
    -LEAF anim,
    -PARENT beach,
        -LEAF beach,sand,
        -LEAF beach,sea,
        -LEAF beach,sunset,
        -LEAF beach,water,
    -LEAF beauti,
    -LEAF bird,
    -LEAF black,
    -PARENT blue,
        -PARENT blue,cloud,
            -LEAF blue,cloud,sky,
        -LEAF blue,flower,
        -LEAF blue,sky,
    -LEAF bravo,
    -LEAF build,
    -LEAF california,
    -LEAF canada,
    -LEAF canon,
    -PARENT cielo,
        -LEAF cielo,sky,
    -LEAF citi,
    -LEAF closeup,
    -PARENT cloud,
        -LEAF cloud,explor,
        -LEAF cloud,landscap,
        -LEAF cloud,reflect,
        —PARENT cloud , sky ,
            —LEAF cloud , sky , sunset ,
        —LEAF cloud , sun ,
        —LEAF cloud , sunset ,
        —LEAF cloud , tree ,
        —LEAF cloud , water ,
    —LEAF color ,
    —LEAF diamondclassphotograph ,
    —PARENT dog ,
        —LEAF dog , puppi ,
    —PARENT explor ,
        —LEAF explor , flower ,
        —LEAF explor , interesting ,
        —LEAF explor , natur ,
        —LEAF explor , sky ,
    —PARENT flower ,
        —LEAF flower , garden ,
        —LEAF flower , green ,
        —LEAF flower , macro ,
        —LEAF flower , natur ,
        —LEAF flower , naturesfinest ,
        —LEAF flower , pink ,
        —LEAF flower , plant ,
        —LEAF flower , red ,
        —LEAF flower , spring ,
        —LEAF flower , yellow ,
    —LEAF garden ,
    —LEAF geotag ,
    —LEAF grass ,
    —PARENT green ,
        —LEAF green , natur ,
        —LEAF green , sky ,
    —LEAF hdr ,
    —LEAF i500 ,
    —LEAF impressedbeauti ,
    —LEAF interesting ,
    —LEAF lake ,
    —PARENT landscap ,
        —LEAF landscap , natur ,
        —LEAF landscap , sky ,
        —LEAF landscap , water ,
    —PARENT light ,
        —LEAF light , sky ,
    —PARENT macro ,
        —LEAF macro , natur ,
    —LEAF mountain ,
    —PARENT natur ,
        —LEAF natur , naturesfinest ,
        —LEAF natur , sky ,
        —LEAF natur , tree ,
        —LEAF natur , water ,
    —LEAF naturesfinest ,
    —PARENT night ,
        —LEAF night , sky ,
    —LEAF nikon ,
    —LEAF ocean ,
    —LEAF orang ,
    —LEAF outdoor ,
    —LEAF pink ,
    —LEAF plant ,
    —LEAF portrait ,
    —LEAF puppi ,
    —LEAF red ,
    —PARENT reflect ,
        —LEAF reflect , sky ,
    —LEAF rock,
    —LEAF sand,
    —LEAF sea,
    —PARENT sky,
        —LEAF sky,sun,
        —LEAF sky,sunset,
        —LEAF sky,tree,
        —LEAF sky,water,
    —LEAF snow,
    —LEAF spring,
    —LEAF summer,
    —LEAF sun,
    —LEAF sunset,
    —LEAF supershot,
    —LEAF tree,
    —LEAF water,
    —LEAF white,
    —LEAF winter,
    —LEAF yellow,
```

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Many problems involve clustering a large amount of data and not always it represents a feasible task. Besides the direct problem due to the size of the dataset, each object can be described by a high dimensional vector of attributes which makes the task of finding similar object harder even with small sets. In this thesis project it has been implemented a proof of concept of a distributed clustering algorithm for images with textual features, combining two theories respectively Minhash Locality Sensitive Hashing and a more well known Frequent Itemsets mining. While the first enables us to partition the original dataset in smaller sets, the latter unveil frequent patterns to be used to build a hierarchical topics model.

5.2 Future work

The results show that there is enough room for improvements in the current implementation of the LSH. Also, the performance looks promising which it would suggest to bring the work a little bit further. Particular attention is required in the choice of the parameters.

Real datasets are characterized by the presence of *dirty* keywords, that are keywords that for some reason are the concatenation of two words, or a word and a number or an incomplete word. This is likely to happen when, for instance, keywords are gathered without caring much of the source. So a strong and advanced preprocessing of the keywords added to a smart rearranging inside each document in the dataset may lead to better performance since such documents are compared by random sampling. Also, the estimated similarity can be put in comparison with the actual Jaccard index as further means of performance validation.

Finally, the current implementation might be redesigned into smaller functional modules to be easier deployable on a distributed system.

Bibliography

[1] Pablo Arbelaez, Michael Maire, Charless Fowlkes, and Jitendra Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011.

[2] Wikipedia. Pattern recognition, November, 2013.

[3] Yan Song, Anan Liu, Lin Pang, Shouxun Lin, Yongdong Zhang, and Sheng Tang. A novel image text extraction method based on k-means clustering. In Roger Y. Lee, editor, *ACIS-ICIS*, pages 185–190. IEEE Computer Society, 2008.

[4] Cambridge University Press. Clustering in information retrieval, November, 2013.

[5] Andrei Z. Broder. On the resemblance and containment of documents. In *In Compression and Complexity of Sequences (SEQUENCESâ€™97*, pages 21–29. IEEE Computer Society, 1997.

[6] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. pages 604–613, 1998.

[7] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993.

[8] Htcondor-http://research.cs.wisc.edu/htcondor/, August, 2013.

[9] Dechang Pi, Xiaolin Qin, and Qiang Wang. Fuzzy clustering algorithm based on tree for association rules.

[10] Hassan H. Malik. Clustering web images using association rules, interestingness measures, and hypergraph partitions. In *In: ICWE â€™06: Proceedings of the 6th international conference on Web engineering*, pages 48–55. ACM Press, 2006.

[11] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 169–178, New York, NY, USA, 2000. ACM.

[12] Cambridge University Press. Evaluation of clustering, August, 2013.

[13] Wikipedia. Jaccard index, August, 2013.

[14] Andrei Z. Broder, Moses Charikar, Alan M. Frieze, and Michael Mitzenmacher. Min-wise independent permutations (extended abstract). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 327–336, New York, NY, USA, 1998. ACM.

[15] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[16] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets.* Cambridge University Press, Cambridge, 2012.

[17] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right objective measure for association analysis. *Inf. Syst.*, 29(4):293–313, June 2004.

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

[19] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[20] Ramesh C. Agarwal, Charu C. Aggarwal, and V. V. V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and Distributed Computing*, 61:350–371, 2000.

[21] Jian Pei, Jiawei Han, Hongjun Lu, Shojiro Nishio, Shiwei Tang, and Dongqing Yang. H-mine: Hyper-structure mining of frequent patterns in large databases. pages 441–448, 2001.

[22] Benjamin C.M. Fung, Ke Wang, and Martin Ester. Hierarchical document clustering using frequent itemsets. In *IN PROC. SIAM INTERNATIONAL CONFERENCE ON DATA MINING 2003 (SDM 2003*, 2003.

[23] Florian Beil, Martin Ester, and Xiaowei Xu. Frequent term-based text clustering. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 436–442, New York, NY, USA, 2002. ACM.

[24] M.S. Lew M.J. Huiskes, editor. *The MIR Flickr Retrieval Evaluation*, 2008.