UNIVERSITY OF TRENTO - Italy

Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

Final dissertation

Clustering algorithms and recommender systems analysis.

A comparative Java oriented framework.

Supervisor
MAURIZIO MARCHESE

Student
DAVIDE LISSONI

Co-supervisor
MARCELO RODAS BRITEZ

Academic year 2016/2017

Abstract

Nowadays, organizations are accumulating vast and growing amounts of data in different formats, storing them in huge databases. This data can provide useful information through patterns, associations or relationships. Data mining has emerged as a powerful tool to analyze a huge amount of data in automatic or semi-automatic way, in order to discover as much information and correlations as possible within datasets. A key feature and one of the goals of these systems is to provide, by personalized user recommender systems regards user's interests, recommendations.

Recommender systems usually refers to methods of recommendations based on the information of a single user. However the user's way of thinking, the user's choices and tastes are affected by the user's social life. Consequently new recommender systems approaches are being developed in order to incorporates also user's social parameters as recommender system features. For instance, friend grouping recommendation is a recommender system approach that take into consideration social information by creating groups personalized environments. Shortly, friend grouping refers to groups of people, hidden or not from the user point of view, that has been grouped together, according to some predefined parameters, in order to share the same resources and informations.

Clustering entities according to some entities' features in order to create groups, is a well known problem in computer science. There are many algorithms created in order to perform this operation. However running different clustering algorithms lead to the achievement of different results i.e. different groups. Each clustering algorithms has some predefined evaluations strictly based on the structure of the algorithms. For example the distance mean of the cluster instances from the cluster centroid is used as k-means evaluation, or the similarity means between the cluster instances and its exemplars used in the Affinity propagation algorithm. However these evaluation methodologies are not directly comparable each other, making hard the comparison between different algorithms performances. The key problem we want to study, is how to produce general results, statistics and evaluations usable for every cluster algorithm, in order to simplify the comparison of the cluster results and in order to understand which algorithm produces the best groups according to the problem requirements.

The solution proposed in this thesis is a Java-oriented framework, able to run different clustering algorithms, recommender systems and analysis of the outputs. The framework shall facilitate the development of cluster algorithms and produces statistics about the created clusters, so we can compare in a simpler way the different algorithms. More precisely, it contains methodologies designed to simplify the creation and execution of clustering algorithms on a dataset, having as final output statistics about the results achieved. Furthermore, it is able to compare the results with a given ground truth dataset, returning different statistics and evaluations based on the comparison.

The framework has been designed to provide to developers and data analysts pattern analysis of cluster algorithms and recommender systems, thus facilitating their choice on which algorithm will fit better their requirements.

To test the framework we have been made testing on the domain of social groups and group recommendations based on users' preferences. The final part of this thesis presents some results of the use of the proposed framework to show how to use it in order to make some algorithms comparisons on given datasets.

Acknowledgements

I would first like to thank my thesis advisor professor Maurizio Marchese of the Department of Information Engineering and Computer Science at University of Trento. He gave me the opportunity to work on a research of my interest and he helped me in structure and write correctly the thesis.

I would also like to acknowledge the PhD student Marcelo Rodas Britez of the Department of Information Engineering and Computer Science at University of Trento as the second reader of this thesis, and I am gratefully indebted to him for his advices and for all the time he spent in helping me to develop the research project and the writing of this thesis. I really enjoyed these months spent working with him.

A very special gratitude goes out to Daniele Dellagiacoma and Franco Avi, my friends of studies and life, who accompanied and helped me in all these five university years. We had very good time together and your good company has made these years unforgettable.

Last but not least, I must express my very profound gratitude to my parents and to my friends for providing me with unfailing support and continuous encouragement throughout my years of study. This accomplishment would not have been possible without them.

Thank you.

Davide Lissoni.

Contents

1 Introduction

The amount of information and data published and made available on the Internet increase exponentially day by day. An IBM Marketing Cloud study report that the 90% of data has been created since 2016 and, in the last year, the number of Internet users rose by around 300 million [8]. Customization of contents and recommendations of objects used in order to adapt data to users are getting therefore increasingly indispensable. Approaches applied to perform this information filtering are commonly known as recommender systems.

Recommender systems are information filtering systems that deal with the problem of information overload by filtering vital information fragment out of large amount of dynamically generated information according to user's preferences, interest, or observed behavior about item [10]. Basically recommender systems create users' personalized environments.

Recommender systems approaches are used by almost every big company, especially in the e-commerce area and therefore they are constantly evolving. New social information start to be considered as recommender features like the influence of the society on users.

The typical recommender systems recommend items basing their recommendations strictly on users' feature such as users' likes, preferences, friends, objects they bought or they searched and so on and so forth. But also environment in which users live, friends' features, advertising seen, news red, life quality and lifestyle, are all factors that indirectly affect the users' way of thinking and therefore their choices. Moreover, people operate in groups not individually, they like to share news, talk in groups and do group activities. For example, it is much more pleasant to exchange opinions among friends on a news that everyone knows than talking about an event that only a single person has seen. Furthermore some studies mentioned in [11] proved that items, events and activities are used/performed at least as often by groups as by individuals. Groups recommender systems try then to fill the gaps of personalized recommender system brought by the lack of social parameters by creating a batch personalized environment. In addition, grouping recommendation can also bring benefits to some personalized recommender systems problems, brought to the fast growth of data made available on the Internet, such as algorithms running time and resources required for their execution.

In the digital world, social medias are the firsts that made available group creation methodologies. The firsts methodologies created allow users to manually create groups. However manual group creation is a time-consuming operation that soon has led to the need to develop unsupervised or semi-unsupervised group-building approaches. More information about this topic are reported in the chapter 2 of the paper.

In computer science , grouping, better known as clustering, is one of the most important unsupervised learning methodologies. Clustering algorithms are born in order to group together a set of unlabeled data, based on a predefined similarity measure. Basically the main goal is to create clusters (groups) in which their instances are similar each other. Cluster algorithms are analyzed in more details in this paper in the section 3.2.

Plenty of studies have been made on methodologies used to group people, movies, items or whatever together. Researchers, developers and data analysts are spoiled for choice about the algorithm to use. Nowadays is not required to be an expert developer or to have a through knowledge of machine learning algorithms in order to build a clustering system. There exists in fact many external libraries that help to implement in an easy and fast way such algorithms. The real researcher key problem is the cluster algorithms choice, and its calibration, strictly connected to cluster analysis. Other research reported that clustering algorithms choice are often based on heuristics and justified by practical performance considerations rather than by formally proven guarantees [17]. Often the choice of the clustering algorithm is also based on results already obtained in works or researches concerning similar domains and requirements. The problem occurs also talking about grouping people where the aims and

methodologies for which people are grouped are very many and independent of each other. Moreover a user entity usually contains a huge number of dependent and independent features data that cause a loss of mathematical meaning to any similarity formula used to compare users.

Analyzing social groups once they have been tested on the "real world" is not a problem. Evaluations could be based on users' likes, on users' questionnaires, on how much the group is used by its members and so on and so forth. Anyway test algorithms in the long future require a great availability in terms of time and resources that is not always taken for granted during research projects.

A priori evaluation about new groups created by a clustering algorithm and a comparison between different clustering algorithms, can then lead social researchers to a faster choice of the algorithms to use in their domain, giving also an idea about their achievable results. The framework exposed in this thesis was created in order to specifically help the evaluation of social groups on a given dataset. The framework is build in order to create applications in charge to run different clustering algorithms and to evaluate the groups created by taking into consideration data made available during the running process. During its implementation, different statistics and quality measures for groups were defined. Statistics and quality measures were designed to be as general as possible. This means that they are the same regardless of the algorithm to which they refer, simplifying in this way the comparison between the results obtained by different algorithms. Statistics take care about the groups quality, the groups size and finally the runtime routine parameters. In more details the framework returns an evaluation for each single cluster created, the evaluation of the algorithm results in general, the algorithm execution performance and an evaluation based on the comparison of the clusters generated with a ground truth dataset. More technical details about the statistics created are reported in this paper in the section 4.4.

Usually frameworks define the structure of the program allowing developers to implement their own version of the project. Anyway in order to test it, analyze the results and as implementation example, three different cluster algorithms were implemented and analyzed in the first version of the present work:

- K-means;
- Fuzzy K-means;
- Affinity propagation.

These algorithms has been tested on datasets containing users, their preferences respect to some items and users' groups. The results reported are based on tests made on three different datasets coming from different ground truths. Thanks to the framework we have been able to define the best similarity measure to use in order to cluster users through user's preferences, the best algorithms settings and the strengths and weaknesses of the algorithms developed related to our topic.

The foundations of the framework structure are based on Java patterns. The framework functionalities are extensible, meaning that new algorithm and statistics can be easily added.

Additionally, the framework provide structures and implementations regards some collaborative filtering recommender systems, used in order to recommend the groups created by the execution of the clustering algorithms of the framework, and their related evaluation methodologies. In particular the recommender system implemented are:

- User-user collaborative filtering recommender system;
- Item-item collaborative filtering recommender system;
- SVD collaborative filtering recommender system based on matrix factorization.

In order to evaluate the recommender systems, we defines a quality measure of the recommendations based on the user-group rank we compute in order to recommend groups to users. In-fact the user-group rank on which we based the recommendations it is generated by us and then it is not present in any ground truth dataset. Furthermore, with the recommender systems we don't recommend existing groups but the groups generated by the framework through the clustering algorithms. Then, in order to evaluate recommendations using a ground truth database, there is the need to define

an equality formula between the groups generated and the ground truth groups present in the dataset. Until now this formula has not been defined, however future works of the frameworks will take into considerations the evaluation of the recommender systems using some ground truth data.

Personal recommendation systems and grouping recommendation systems are often treated and analyzed independently. We think instead that, talking about social matters, recommender system topic is closely linked to clustering. Recommender systems were then added for the sake of completeness and convenience of the use of the framework, avoiding possible and common developers' problems brought by the import and use of many different external dependencies in the same project. Furthermore, it was thought that a possible combination of users clustering and collaborative filtering recommender systems may produce very interesting results.

This project has born as an initial contribution to a wider social recommendation research project. The research is about recommender systems based on both users' preferences and personalities traits. This research purpose is to use users' personality traits as users' similarity parameters for collaborative filtering recommender system techniques. The study could also lead to a new hybrid recommender system that will take into consideration user's psychological aspects.

This thesis document is organized as follows. The first chapters focus on an overall view about existing clustering algorithms and recommender systems. Then we enter into the detail of recommendations used for social purpose such as clustering people basing on their preferences and group recommendations. Following, the proposed framework is introduced, analyzing the problem for which it was created and presenting the proposed solution, by analyzing and motivating the algorithms and the statistics used in the project development. After that, we move into a technical explanation about the development and structure of the framework. Finally the paper reports and analyzes the results achieved on the tests, highlighting conclusion about our specific topic, and proposes possible implementation improvements and future works.

2 State of the art

The analysis of the state of the art took us to analyze some techniques of evaluation of clustering algorithms, and how existing Java libraries provide the evaluation of the clustering algorithms and recommender systems as well.

This thesis is focused on the comparison between clustering algorithms, so, only the most general evaluation measures are mentioned in this chapter. These general evaluation measures are the ones that can be used for all clusters regardless of the algorithm that generated them. Moreover also all the evaluation measures based on clusters tested in the real world are not covered in the thesis since our topic is based on an a-priori clusters evaluation.

A technique used to evaluate cluster algorithms consist in the comparison of the algorithms results with a ground truth dataset. Basically the more the algorithm results are similar to the ground truth dataset the more the algorithm results are good. We refer in this document to the ground truth dataset also as external criterion. Actually there are various way to define cluster similarity. [20] reports similarity measures based on the distances between cluster centroid using different distance metrics. The study explained in the reference [24] instead calculates clusters similarity by using the Jaccard similarity measure between cluster instances pairs. Another classical way to calculate clusters similarity is to count how many instances two clusters have in common.

The document [26] instead proposed some evaluation based on so-called external criterion, without using similarity measures. Basically clusters are evaluated on how well clusters matches with a predefined class or parameter. The first evaluation measure is the purity($purity(W,C) = \frac{1}{N}\sum_1^k max_j|w_k \cap c_j|$ where $W = \{w_1, w_2, ...w_k\}$ clusters and $C = \{c_1, c_2, ...c_j\}$ classes) which describe how many cluster entities classes differ from the major class present in the cluster. Other interesting formulas mentioned in the document are Precision, Recall and F-measure. Precision ($Pre = \frac{TruePositive}{TruePositive+FalsePositive}$) , Recall ($Rec = \frac{TruePositive}{TruePositive+FalseNegative}$) and F-measure ($F_\beta = \frac{(1+\beta^2)(Precision*Recall)}{\beta^2 Precision+Recall}$) are evaluation measure used in order to evaluate classification supervised learning algorithms. The formulas are built on the top of the so called confusion matrix. Confusion matrix is a matrix where each row represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa)[28]. Since clustering algorithms are unsupervised learning algorithms, in order to generate a confusion matrix there is the need to use ground truth groups as corrected prediction. Ground truth is then used to define the "predefined classes"(groups), while the clusters generated represent the "actual classes". The confusion matrix are created through the comparison between the predefined cluster instances and the actual cluster ones. An example of calculation of precision recall and F measure on clusters, proposed in this page [26] is computed by comparing the classification of each clusters instances pair of the cluster generated with each ground truth group instances pair. The paper in-fact reports that, in order to evaluate the clustering algorithm results, precision, recall, and F-measure were calculated over pairs of points. For each pair of points that share at least one cluster in the overlapping clustering results, these measures try to estimate whether the prediction of this pair as being in the same cluster was correct with respect to the underlying true categories in the data. Precision is calculated as the fraction of pairs correctly put in the same cluster, recall is the fraction of actual pairs that were identified, and F-measure is the harmonic mean of precision and recall [26].

The paper [12] instead has presented a new criteria in order to compare two clusterings of a data set, that is derived from information theoretic principles. The criteria, called variation of information (VI), measures the amount of information lost and gained in changing from one cluster to another one[12]. Furthermore VI define a metric (or distance) on clusterings.

There are already various existing free and open source machine learning Java projects, focused on data mining algorithms, created in order to help people during the implementations of such algorithms and their evaluation. Apache Mahout [1], for example, is a free Apache Software Foundation platform

that makes available different scalable machine learning algorithms focused primarily in the areas of collaborative filtering built on the top of Apache Hadoop. Another famous Java framework is WEKA. WEKA is a collection of machine learning algorithms for data mining tasks [2]. Since our topic is more focused on data mining algorithm evaluations, we want to mention only the approaches used by these two frameworks in order to evaluate their algorithms. Both WEKA and Apache Mahout contains implementations of some evaluation measures used in order to evaluate their algorithms.

Regarding clustering evaluation Mahout does not provide any type of cluster quality evaluation while WEKA make available three different ways in order to measure the quality of a cluster:

1. The percentage of instances contained in each cluster;
2. WEKA can evaluate clusterings on separate test data if the cluster representation is probabilistic (e.g. for Expectation Maximization);
3. Classes to clusters evaluation. In this mode WEKA first ignores the class attribute and generates the clustering. Then during the test phase it assigns classes to the clusters, based on the majority value of the class attribute within each cluster. Then it computes the classification error, based on this assignment and also shows the corresponding confusion matrix.

Regarding recommender systems evaluations instead Mahout provides a way to evaluate the accuracy of the recommendations comparing the prediction with the ground truth data, while WEKA provides multiple evaluation measures based on the confusion matrix.

Despite the many features that these frameworks can offer, the main goal of our framework was to help the developers in the evaluation and comparison of clusters and recommendations. In fact, our framework aims to enrich the already existing functionalities by adding some cluster and recommendations evaluation and comparison methodologies focused on helping the user in choosing the clustering algorithm to use. We in fact build our framework on the top of Apache Mahout.

3 Main project components

The project reported in this thesis is a Java framework used to define a structure pattern for clustering algorithms, recommender systems and their evaluations as described in the figure 3.1. Furthermore implementation of some algorithms were added in the research in order to verify the meaning of the evaluations created and the framework usability. This chapter makes an overall view about the technologies utilized i.e. frameworks, cluster algorithms and recommender systems.

3.1 Frameworks

Frameworks are defined as reusable semi-finished architectures for various application domain [18]. Higher productivity and shorter time-to-market for the development of object-oriented applications are the major framework's goals. These goals are achieved through design and code reuse.

In order to create a good framework, software architecture and design are the most relevant framework properties to take into consideration. Design and software architecture structures are the results of good knowledges and in-depth analysis about a well understood domain. Well designed frameworks allow future framework users to avoid the analysis of their domain in detail, so as to be able to devote more time and attention to the development of the application rather than to low-level code.

Design patterns are usually incorporated in the framework structure in order to simplify the use of the framework components and also the extensibility of the framework itself. Since we implemented a Java-framework, architecture and patterns reported are object oriented and in particular Java-oriented.

As a general rule, an application, to be addressed as framework should respect some principles:

- Extensibility: the framework should be extensible. This means that an user can add functionalities to the frameworks without changing the existing code;
- Inversion of control: the framework maintain the control of the application life-cycle;
- Interface and class segregation: the framework should separate different functionalities into different interfaces and different entities in different classes;
- Dependency inversion: high-level framework components shall perform their functions using lower-level framework components, through the interfaces exposed by the latter.

3.2 Clustering algorithms

The framework idea, it has been an evolution of an initial research problem. The problem was to understand which clustering algorithm, used in order to group people, would have produced better results in our research domain.

Clustering is an unsupervised and unlabeled machine learning problem. Different clustering algorithms could work very differently, but the final purpose is common i.e. group similar objects in clusters.

Nowadays exists over 100 published different clustering algorithms [27], but they can be syntactically merged in major categories depending on their behavior. A common differentiation of clustering algorithms is made on the properties of the clusters generated by the algorithms. In particular we can define two main cluster methodologies subgroups:

- Hard clustering: in hard clustering a data-point or belongs entirely to a cluster or does not. This means that clusters can not overlaps;

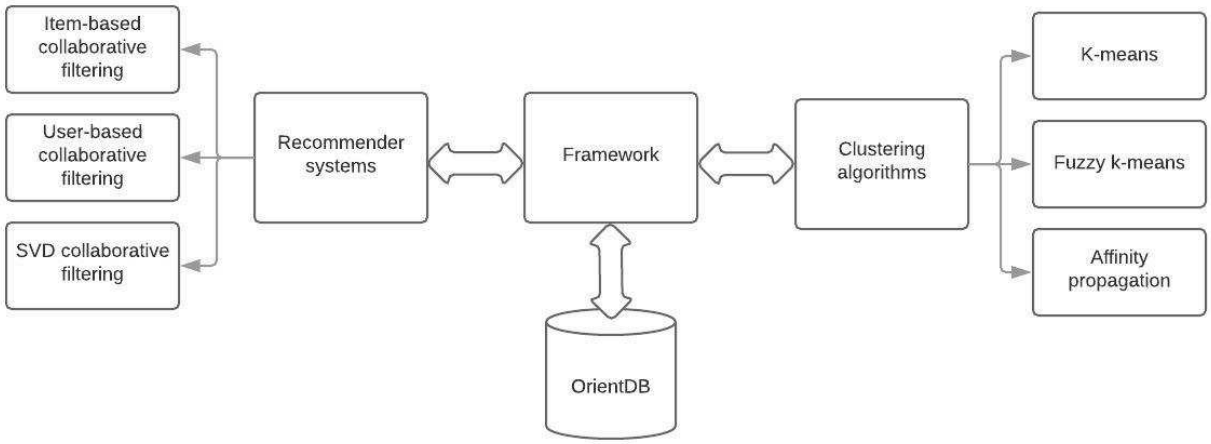


Figure 3.1: Diagram that shows the interaction between the project components.

- Soft clustering: in soft clustering each data-point belongs to every cluster with a certain probability. This means that the output of this clustering methodology does not consist in a set of well defined groups but rather to a set of probabilities. A probability describe how much a data-point is consistent respect to a certain group.

Other clustering algorithms subdivisions are made on the basis of the comparison metrics used among data objects. Data-point distances is the most common and intuitive comparison methodology used for clustering, however also density and probabilistic methodologies are pretty popular in clustering algorithms. Since, in social domains, is very difficult having knowledge about the probability distribution of the dataset, we decided to focus the research on cluster algorithms that use distance measures methodologies. With distances measures methodology we refer to clustering algorithms that rely on the main idea that nearby data-points (objects projects in a feature space) are more related than further ones. Therefore the similarity measure used in order to group instances must be a distance measure. Centroids and connectivity models are two typical examples of such algorithms.

3.2.1 Centroid models

Briefly, in centroid models, clusters are represented by centroids (a cluster center point). The inclusion of a data-point in a cluster depends on the distance between the data-point and the cluster centroid.

Usually, in centroid models algorithms, the number of clusters desired is defined by the algorithm user before its execution. In this cases we talk about k-means clustering techniques.

K-means assign to k number of clusters the object instances to be grouped basing on the distance between data-points and the clusters centroid. Clusters centroids and the related clusters instances are defined by running iteratively (until convergence) two main steps:

1. Assign each data-point to the cluster which have the nearest centroid;
2. Recalculate the clusters centroids. A centroid for a cluster is the point situated in the middle of the cluster instances.

Distance measure can be defined by developers but the metric must satisfy the fundamental distance properties i.e. reflexivity, symmetry and triangle inequality;

The soft clustering extension of the k-means algorithm is called fuzzy k-means (or Fuzzy C-means). The steps executed by the fuzzy k-means in order to create clusters and assign them instances, follow the same logic of the k-means but in a soft-clustering view:

1. Compute the probability of a point belong to a cluster for every ¡point,cluster¿ pair;
2. Recompute the cluster centers using above probability membership values of points to clusters [4].

3.2.2 Connectivity models

Connectivity models calculate distance between each pair of data-points and define clusters basing on a maximal distance used to connect parts. At different distances, different clusters will form. So, these algorithms do not provide a single partitioning of the data set, but instead provide an extensive hierarchy of clusters that merge with each other at certain distance [6]. Connectivity model algorithms does not require, therefore, to define a number of clusters before their execution. A typical connectivity model example is the Affinity Propagation algorithm.

Affinity propagation is an algorithm based on the concept of "message passing" between data-points. Cluster centroids are called exemplars and correspond to existing data-points instances. There is one exemplar for each cluster and correspond to the most representative cluster data-point instance for the cluster in question. Shortly, the message passing schema is based on two matrices:

- One matrix contains, for each data-point, how well-suited an instance is to serve as exemplar for another instance, relative to all the other candidate exemplars;
- The other matrix represents, for each data-point, how "appropriate" it would be for the instance to pick all the others instances as exemplar [6].

The affinity propagation algorithm is explained in details in the reference[6].

3.3 Recommender systems

The second part of the research was based on recommender systems and in particular groups recommendations analysis. Actually, recommender systems analysis was an additional topic of the initial clusters analysis research proposed. The main goal of the group recommender systems is to recommend to users the groups created by the framework clustering algorithms executed. More details about the motivations of the insertion of recommender system in the framework are present in this paper in the section 4.5.

Recommender systems, in data mining, are information filtering methodologies used in order to perform recommendations of items to users. Actually a recommendation is the consequence of a user's preference prediction. Therefore, the main goal of a recommender system is to predict users' preferences, and then, if a preference is positive, it will recommend to the user in question, the item to which the preference refers.

Recommender systems are commonly designed on the top of two approaches:

- Content based filtering approach;
- Collaborative filtering approach.

3.3.1 Content based filtering

In content based filtering, recommendations are based on the users' histories. Basically this approach rely on what users like or not. Content based filtering are structured using two main entities: item and user. The entity user contains the user's items' preferences specifications, while items contains a set of tags that are used in order to create correlations between different items. The item recommended to an user will be the one that has the greatest match between users' preferences and items tags.

The limited number of feature required in order to perform recommendations, the capacity of easily provide explanation about recommendation motivations, and the high capability of fit the users' tastes, are some of the reasons that make content based approaches so popular.

One of the problem of content based recommender system is the difficulty for some items to be explained, and then to categorize them using features (tags). Furthermore content based systems don't take into consideration other users' choices, decreasing therefore the number of possible items-preferences correlations.

3.3.2 Collaborative filtering

Collaborative filtering is another recommender system approach based on similarity. Collaborative filtering stores a huge amount of users' information and calculates similarities using the users' data collected. Recommendations are made on the base of the similarity calculated.

The vague nature of the definition of collaborative filtering recommender systems has allowed to insert different approaches in the same category. Common division of collaborative filtering approaches are:

- User-based approach: this collaborative filtering approach in short calculates similarity between users and then recommend to an user items that similar users likes;
- Item-based approach: this approach calculate instead similarity between items. Recommendation will be made based on the most similar items of the items that the user has expressed preference for.

Usually, item-based methodologies outperform the user-based ones. This is because items features are more meaningful than users' ones. Items are more explainable in features and then their comparison makes more sense.

Problems of collaborative filtering are due to the lack of data to be processed. This is because collaborative filtering approach works on users' reviews and, the most users, don't release ratings on items they bought/saw/used. Major problems of collaborative filtering approaches are in-fact the sparsity of the data and the first rater problem (an item never reviewed will not be recommended). A possible solution to collaborative filtering and content based problems could be the combinations of these two approaches.

3.3.3 Hybrid methods

The combination of collaborative filtering and content based approaches are called hybrid methods. There exist a huge number of possible combinations between collaborative filtering and content based systems but the most utilized are:

- Implement more than one recommender and then combine predictions in some ways, for example through a linear model;
- Add content-based methods to collaborative filtering or vice versa.

4 Framework development

This chapter exposes how the framework has been implemented, explaining also the in detail the technologies used and motivating the choices we made. Although, this chapter analyzes every part of the framework, it will more focused on the clusters evaluation topic by highlighting the quality measures and statistic measures used in order to evaluate groups. The explanation of our choices is always based on the assumption that we are talking about users groups and groups recommendation since this was the domain of the research.

4.1 Database structure

With the growth of data available in the Internet, data mining and big data are more and more correlated. Data mining nowadays must also deal with an huge amount of data.

Relational databases were not designed to cope with the scale and agility challenges that face modern application [13]. Its rigid structure and the considerable cost in terms of time of one of the most common SQL operation, the JOIN, have triggered the usage of different database solutions called No-SQL. No-SQL is a generalization of different database technologies that are not structured in relational databases and therefore do not use SQL as query language.

Graph databases are no-SQL technologies that use a directed graph structure schema i.e. labeled nodes and directed edges. Nodes represent entities, while edges describe nodes relationships. Nodes and edges can contain attributes and also meta-data (information concerning other nodes or edges).

A dataset graph structure allows to perform an efficient, constant-time operation and let to traverse a big amount of connections per second per core [22].

Since, for our research purpose, we had to handle users social data, our database choice has been oriented on graph structure databases and in particular OrientDB. OrientDB is the first Multi-Model Open Source NoSQL DBMS that combines the power of graphs with documents, key/value, reactive, object-oriented and geo-spatial models into one scalable, high-performance operational database [14]. Our OrientDB choice is motivated not only by its graph structure, but also by the presence of Java API and a SQL-likely query language. However, since the framework is extensible, other databases could be used by simply implementing in the framework additional connections and DAOs classes.

OrientDB avoid join operations by using "links". A link is a meta-data attribute containing another dataset components. Thanks to links, it is possible to move into the structure of the database. The database structure designed for the framework is minimal and simple. During the domain analysis, we tried to define and use only the indispensable fields used in order to perform clustering and recommendations algorithms, simplifying in this way the intuitiveness of the framework and its possibility of extension. The structure resulted is shown in the graph 4.1. Tables 4.1 and 4.2 explain instead role and details about dataset node, edges and their attributes.

The application use the database in order to get all the input data (users and tags) and to save the algorithm results. The import of the data starts during the cluster algorithm initialization while the insertion of the results starts once the algorithm has generated all the clusters. The database life-cycle for a correct insertion of the results should works as follow:

1. Create a new cycle for every cluster generated;
2. For every user belonging to a cluster, create a user-cycle relationship through the creation of an Is-Member edge.

Since statistics and comparison data are generated during the framework runtime process, we decided not to save the data generated into the dataset, avoiding in this way to overload the dataset

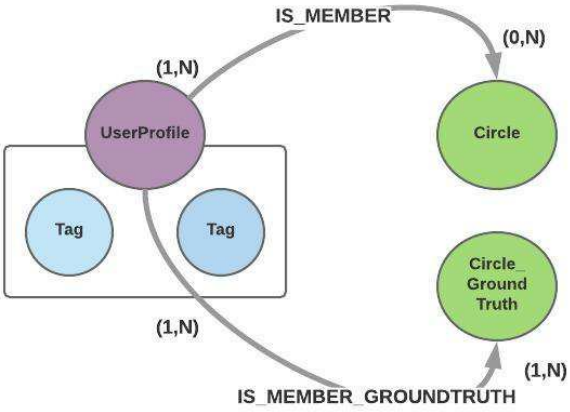


Figure 4.1: Structure of the dataset used in the framework. As it can be notes an UserProfile node could contains more Tags nodes. Circle and UserProfile nodes are connected through Is-Member edges. When the connection exist it means that a certain UserProfile is member of a certain Circle. Regarding Circle-GroundTruth the logic is the same.

Table 4.1: Database nodes used in the framework.

Node	**Attributes**
UserProfile: the user entity.	• *rid (String)*: user identification (UNIQUE); • *screenname (String)*: user nominative; • *tags (Linkset)*: link to Tag (meta-data).
Tag: tags are the user's features on which the clustering algorithm will calculate the similarity between users.	• *rid (String)*: Tag identification (UNIQUE); • *label(String)*:tag nominative.
Circle: entity in which the clusters generated by the algorithm are saved.	• *rid (String)*: Circle identification (UNIQUE); • *tags(Linkset)*: link to Tags (meta-data). This list attribute contains the five tags more present in users belonging to the cluster.
Circle-GroundTruth: identical to Circle, this entity is designed in order to save the groups coming from the ground truth.	• *rid (String)*: Circle identification (UNIQUE); • *tags(Linkset)*: link to Tags (meta-data).

Table 4.2: Database edges used in the framework.

Edge	**Attributes**
Is-Member: user-circle relation. If this edge is present, this means that the user node is an instance of the circle node in question.	• *rid (String)*: edge identification (UNIQUE); • *in (Link)*: Link to Circle; • *out (Link)*: Link to UserProfile; • *timestamp (Datetime)*: field used in order to store when the relation has been created. • *rank (Double)*: attribute that define how much the user is affine with the cluster. Computation, formula and explanation of the Rank are present in the section 4.5.
Is-Member-GroundTruth: this edge is about relationship between UserProfile and Circle-GroundTruth. It defines the users members of a circle from the ground truth dataset. The structure is identical to Is-Member.	• *rid (String)*: edge identification (UNIQUE); • *in (Link)*: Link to Circle-GroundTruth; • *out (Link)*: Link to UserProfile;

and to reduce therefore the speed of the application. Once the clusters have been saved into the dataset, it is possible to run statistics and comparison multiple times without re-execute the clustering algorithm.

In order to evaluate the clusters by using a ground truth, there is the need to upload ground truth users, cycles(groups) and their relationships into the dataset. However, the framework do not provide methods used to handle the upload of the data into the dataset. This is because databases are never equal, they come from different domain, with different format and extensions. We concluded that the implementations in the frameworks of some methods used in order to upload ground truth data into the dataset would have been trivial.

Moreover, the database doesn't present any structure designed to manage groups recommendations persistence. In this stage of the project, in-fact, we took care about just the implementation and evaluation of the groups recommendations. Further implementation of the framework will include the ability of management of the recommendations and their persistence.

The basic structure of the framework database is available in the framework resources as Schema.osql. The structure is in a osql format specifically created for OrientDB dataset. By the import of the schema, OrientDB will create automatically the appropriate vertices and edges.

4.2 Structure and pattern used

Through the framework we define a specific application structure and behavior. However, the interactions between classes reported in this chapter could be modified and extended by framework users.

Before introducing the structure of the framework we want to list some design pattern notions, making in this way the explanation of the chosen structure easier.

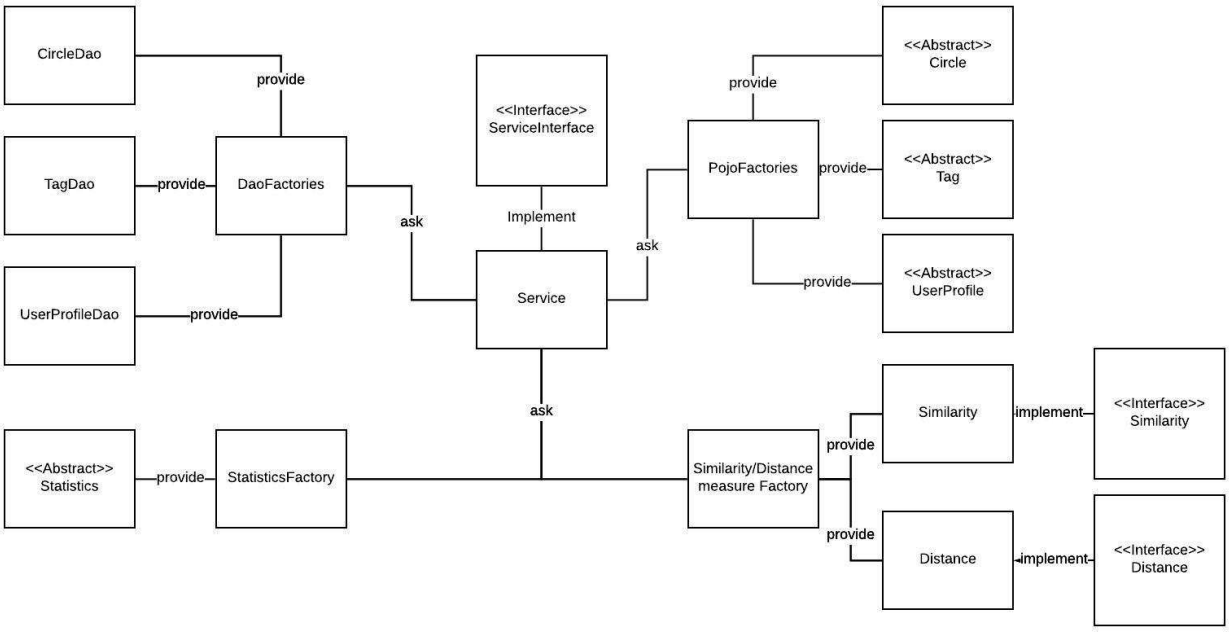


Figure 4.2: The diagram shows the structure of the service in charge to run the clustering algorithm. In particular this diagram is focused on the design patterns used and their interactions.

4.2.1 Design patterns

Design patterns are structure design practices used by object-oriented software developers to provide general and reusable solutions to common design problems. Design patterns make sure that the components of an application become reusable and extensible, thus preventing developers from wasting time writing low level code multiple times. Furthermore they provide an universal structure, common to all developers aimed at avoiding problems such as "spaghetti code", the misunderstanding of the application logic and of the components roles.

One of the goals of our framework was to build a structure that can be used for the implementation and evaluation of most clustering algorithms and recommender systems and, at the same time, build a structure that is extensible and adaptable for more high level implementations. Object oriented and in particular Java pattern allowed us to meet all these requirements. In particular the pattern used are:

- Data access object (DAO): DAOs are classes with relative methods that are used to isolate the database entity access through query. Basically DAOs are used in order to separate the code used for managing the persistence of a certain entity;
- Factory: Factories are used to encapsulate instantiation. Factory is an inversion of control technique with the purpose of not allow to developers the freedom to instantiate particular framework classes. The developer ask the factory for a class instantiation and the factory will pass the instance desired.

The diagram 4.2 reports the design patterns and their interaction used in the framework service in charge to run a clustering algorithm as example. However all the frameworks components utilize the framework patterns in the same way.

The documentation is one of the most important pattern of an application project. Documentation allows application users to understand the application logic and components roles. The framework documentation has been written by using JavaDoc comments. JavaDoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format [25]. Classes, interfaces and public methods of the framework has been properly commented. Classes and interfaces comments include a general

explanations of their functionalities. Regarding public methods instead, each comment contains an explanation of the function, its life-cycle, the arguments required and the output returned.

4.2.2 Abstract classes and interfaces

Framework plain old Java object (POJO) classes are all defined as abstract classes. In doing so, we defined just general and minimal entity super-classes 4.3. It will then be the task of the developer to implement the desired subclasses in detail. Abstract classes are used in the framework as entity POJO super classes, while, we defined the structure of the business logic classes through interfaces.

Each POJO should be assisted by a DAO class used in order to interact with the dataset.

Abstract class	**Explanation of the class and its role**
UserProfile	This is the POJO class relative to the UserProfile database node. The fields contained into this class reflect the UserProfile node attributes. The methods contained into this class are getter and setter methods. Furthermore has been defined the toString() method used in order to print out the users' specifications.
Tag	POJO class for the database Tag node. The fields contained into this class reflect the Tag dataset node attributes. Methods contained in this class are the usual functions for a Java oriented class i.e. get and set, toString() and equals() functions. During the framework explanation we refer to tags also by using the name preferences.
Circle	POJO class that reflect the dataset Circle node. This class is used to store the clusters generated by the framework clustering algorithms. Fields contained in this class are: • *String rid*; • *String name*; • *List (UserProfile) users*; • *int numUsers*; • *CircleStatistics circleStatistic*; CircleStatistics is a class used to generate and store statistics about the circle. More details about the class are shown in the section 4.4. In addition to the classical methods (get, set and toString()), the class contains the following methods: • *initializeCircle (void)*: this method is in charge to initialize the circle specifications by calling function used to generate the circle name and circle statistics; • *generateName (String)*: method used to generate the name of the circle. The name of the circle is defined as the concatenation of the names of the two most frequent tags in the circle. With frequency of a tag we mean how many circle's users have the tag in question.

Comparison	This abstract class does not refer to any node or edge in the database. The class has been designed in order to store data about the comparison between a circle generated by the algorithm and a group coming from the ground truth. In particular the comparison between the circles are made on circle users and preferences. This class contains methods used in order to calculate percentage of correctness of a circle respect to a ground truth group. More detailed specifications about the comparison are present in the section 4.4.
ComparisonPairwiseService	This class defines the service structure used in order to calculate the circle evaluation measures chosen: Precision,Recall and F-measure. Since these measures are based on circle pairs of users, the code used to calculate them is the same regardless of the cluster algorithm used. That's allowed us to implement some business logic methods also in the abstract class. In particular this class contains methods in order to calculate the confusion matrix on circles users and circles preferences as well. Hence, this class stores Precision, Recall, F-measure evaluation measures and positive and negative predictions about circles users and circles tags.
Statistics	This abstract class is used in order to generate statistics about the results of the clustering algorithm and its execution specifications. Such as Comparison class, this class does not interact with a database entity. Statistics include the circles sizes and circles preferences specifications. Statistics specification are explained more in details in the section 4.4. This class contains all the fields useful for statistics and their getter and setter methods. Furthermore the class contains all the methods used to calculate the statistics defined.
Recommendation	This abstract class is a POJO class used by the framework in order to contain the groups recommendations generated by the recommender systems. In particular the fields contained in this class are: • user (UserProfile): the user for whom the recommendation is intended; • circle (Circle): the group recommended; • value (double): the value of the recommendations (usually is the predicted rating).

Table 4.3: Abstract classes present in the framework.

Interfaces presented in the table 4.4 are used in order to define business logic functions of the framework components such as services and comparison classes.

All the methods declared in the interfaces were designed in order to maintain a linear application behavior. However, developers are free to implements their application logic in the way they think more appropriate for their requirements.

Interface	**Explanation of the interface and its role**

ServiceInterface	All the classes in charge to execute a clustering algorithm must implement this interface. In particular the interface contains the following methods: • *preProcessing (void)*: data initializer method; • *postProcessing (void)*: method used in order to refine the cluster output; • *process(void)*: method used to orderly call all the other methods; • *saveCircles (void)*; • *printStatistics (void)*: method used in order to print out all the statistics generated; • *deletePrecedentResults (void)*: method designed in order to delete circles and their relative connection of past algorithm execution.
ComparisonServiceInterface	This interface define the structure for comparison services. Comparison services are in charge to process the comparison between between the clusters generated by a framework algorithm and the ground truth groups. The interface define the following methods: • *run (void)*: method used in order to call the interface methods neatly; • *preProcessing (void)*; • *postProcessing (void)*; • *process(void)*; • *printComparison (void)*:
Similarity/Distance	Developers can define the metric used by the clustering algorithms in order to compare users as input argument of clustering algorithms. The only requirement in order to implement the own similarity/distance version is that, the class must implement this interface. The interface contain the method: double compare(v1[],v2[]) which is in charge to compare two vectors and return a numerical value.
CircleDAO	Since the framework should help developers to implement clustering algorithm and compare the results, "Circle", is the most important framework entity. That's why we define some circle management constraint through this interface. This interface is a DAO interface, then, the methods implemented refer to database operations. In particular in this interface were defined functions used to perform the most common database operation such as: set (insert), get(select) and delete. In addition, methods in order to manage circle users members were inserted in this interface.

RecommendationMahoutService	This interface designs the structure for recommendation systems services made available by Mahout. The methods proposed are: • *preProcessing (void)*: used to create the data model. The data model basically consist in the user-item matrix on which the recommendations are based. The matrix should be created in a format that satisfy the Mahout recommendation system requirements; • *buildRecommender (Recommender)*: this method build the Mahout Recommender object. In the initialization of the recommender, the developer must define the recommendation system to use and its specification arguments. The most commons specifications are the similarity measure to use and the Data model; • *evaluateRecommendations (void)*: method used in order to start the recommender system evaluation based on the recommendations generated by the system; • *printRecommendations (void)*: method used to print out all the recommendations generated by the system; • *runRecommendation(void)*: method used to call the service methods in the correct sequence.
UserCircleRankDAO	This DAO interface refers to the connection between Users and Circles. It is used to manage the persistence of the Is-Member database edge. This DAO is used by recommender services in order to create the user-item (score) matrix on which the recommendations are based. In particular in our case the matrix consist in a UserProfile-Circle (rank) matrix.
RecommenderEvaluation	This interface define the structure for the classes used in order to evaluate the recommender systems. The design of this interface maintain the structure used for all the services implemented in the framework. The predefined methods are: • *preProcessing(void)*; • *process(void)*; • *print (void)*; • *run (void)*;

Table 4.4: Interfaces present in the framework.

4.2.3 Clustering algorithms

Despite the difference between different clustering techniques, through the framework we structured a common clustering algorithms behavior. We defined three main operations to execute for clusters generation and evaluation:

- Running a clustering algorithm (figure 4.3);
- Generate statistics about the clusters created (figure 4.4) and general statistics about the algorithm results (figure 4.5);

- Compare clusters generated with ground truth data (life-cycle described by pseudo-code in Algorithm 1)
- Perform the pairwise comparison of the clusters generated with ground truth groups (figure 4.6).

Data: Clusters generated, Clusters from ground truth
Result: all the comparison processed, algorithm statistics derived form the comparisons.

```
1  initialization;
2  foreach pair(Cluster generated, cluster ground truth) do
3  |  calculate for the cluster generated the instances precision respect to the ground truth
   |   cluster;
4  |  calculate for the cluster generated the tags precision respect to the ground truth cluster;
5  |  save results in the Comparison(cluster generated,cluster ground truth).java class; print
   |   Comparison(cluster generated,cluster ground truth).toString();
6  end
7  new List(Comparison) bestComparisons();
8  foreach Cluster generated do
9  |  select max(Comparison.get(cluster generated);;     // the comparison for the cluster
   |   generated in question which have highest precisions values.
10 |  bestComparison.add(max(Comparison.get(cluster generated));
11 end
12 algorithm-instances-precision= AVG(instance precision in bestComparison);
13 algorithm-tags-precision= AVG(tag precision in bestComparison);
```

Algorithm 1: Pseudo code that shows the life-cycle of the comparison between the cluster generated and the groups from the ground truth. More details about formulas and the technique used are present in the section 4.4

4.2.4 Recommender systems

All the recommender systems available in the framework were built on the top of the Apache Mahout framework. Structure and behavior of the recommender systems reflect therefore the Mahout recommender systems implementations. Operations on recommender systems for which the framework is structured are:

- Build and execute the recommender system;
- Evaluate the recommender system.

The structure of the recommender system service in the framework follow the same logic used in the framework for the service used for clustering algorithm execution. The recommender system service as already introduced in last section has been subdivided in three main parts i.e. prePocessing, buildRecommender and runRecommendation and its life-cycle should works as follow:

1. Collect the data from the dataset and set the environment for the recommender system in the preProcessing() service method;
2. Set the recommender desired and initialize it in the buildRecommender() service function;
3. Start the recommender system in the runRecommendation() service method.

Since recommendations are based on simple data features, usually, recommender systems tend to maintain the original format of the data without modify them. Furthermore, the output of a recommender systems consist in a set of recommendations that are easy to parse, extract and print. Post-processing() and print() function therefore, were not defined as mandatory methods in the recommender system service structure.

Evaluation of the recommender system in the framework is defined as the ability of the system to make correct recommendations. The literature on recommender systems distinguishes typically

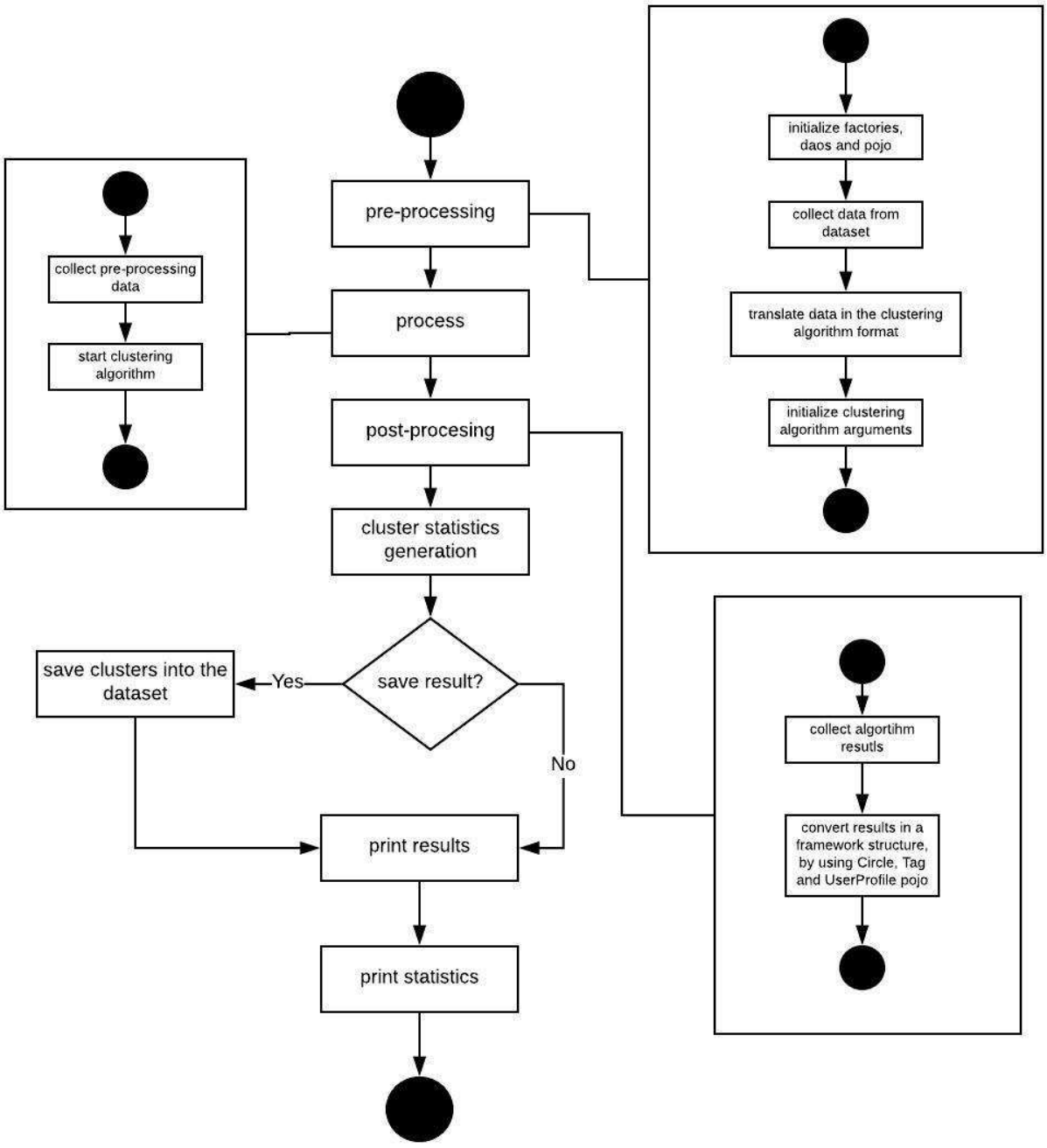


Figure 4.3: Activity diagram that explains how clustering algorithms implementation, on the top of the framework, should be structured. The diagram highlights the various processes that the service executes in order to run a clustering algorithm. However this is just a general life-cycle of the service since further details are dependent to the clustering algorithms technique used.

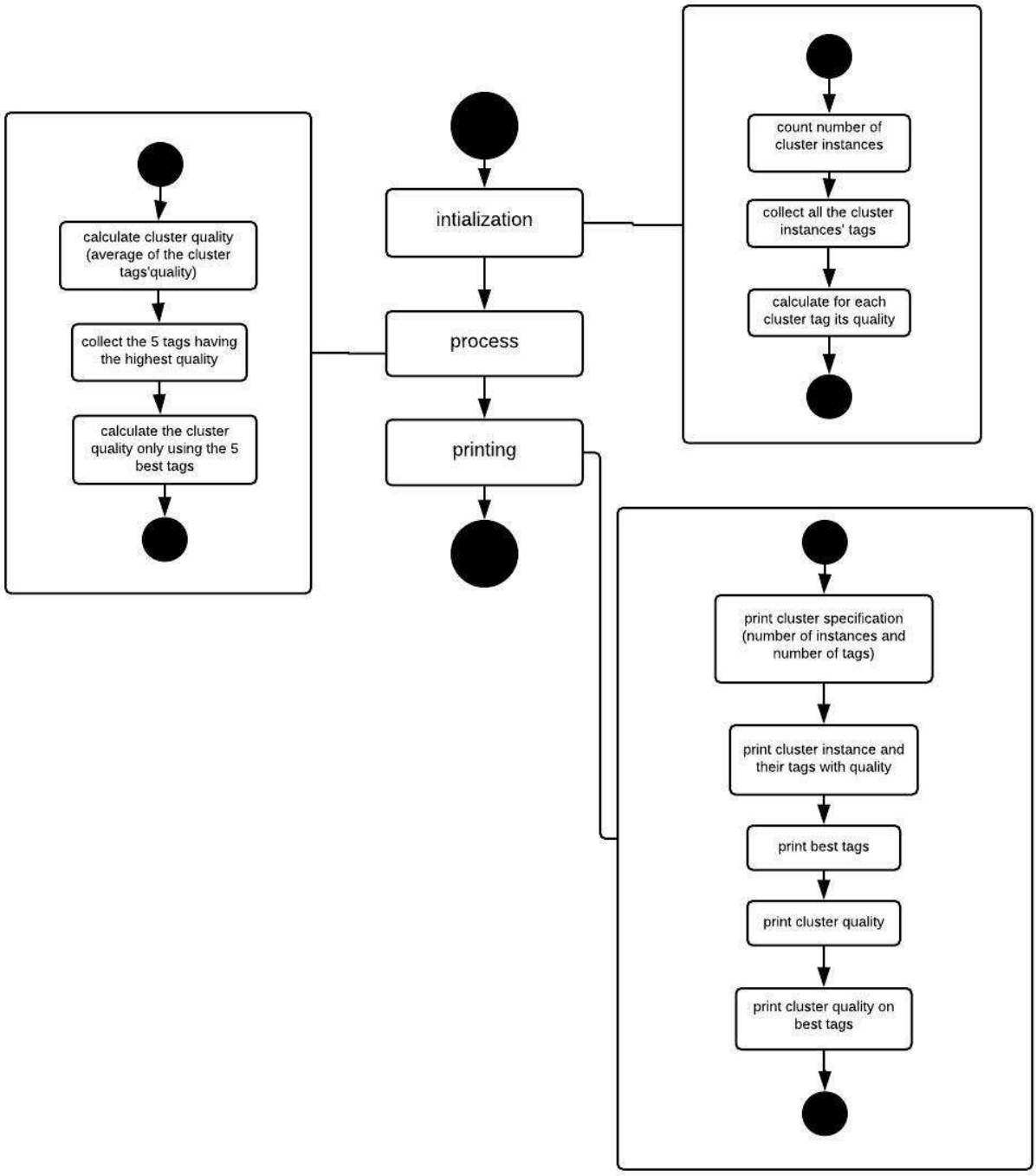


Figure 4.4: Activity diagram that shows the life-cycle of the framework used in order to generate statistics about a single cluster. This life cycle is called for every cluster generated in the post-processing of the clustering service class. More details about the evaluation measure used are present in the section 4.4

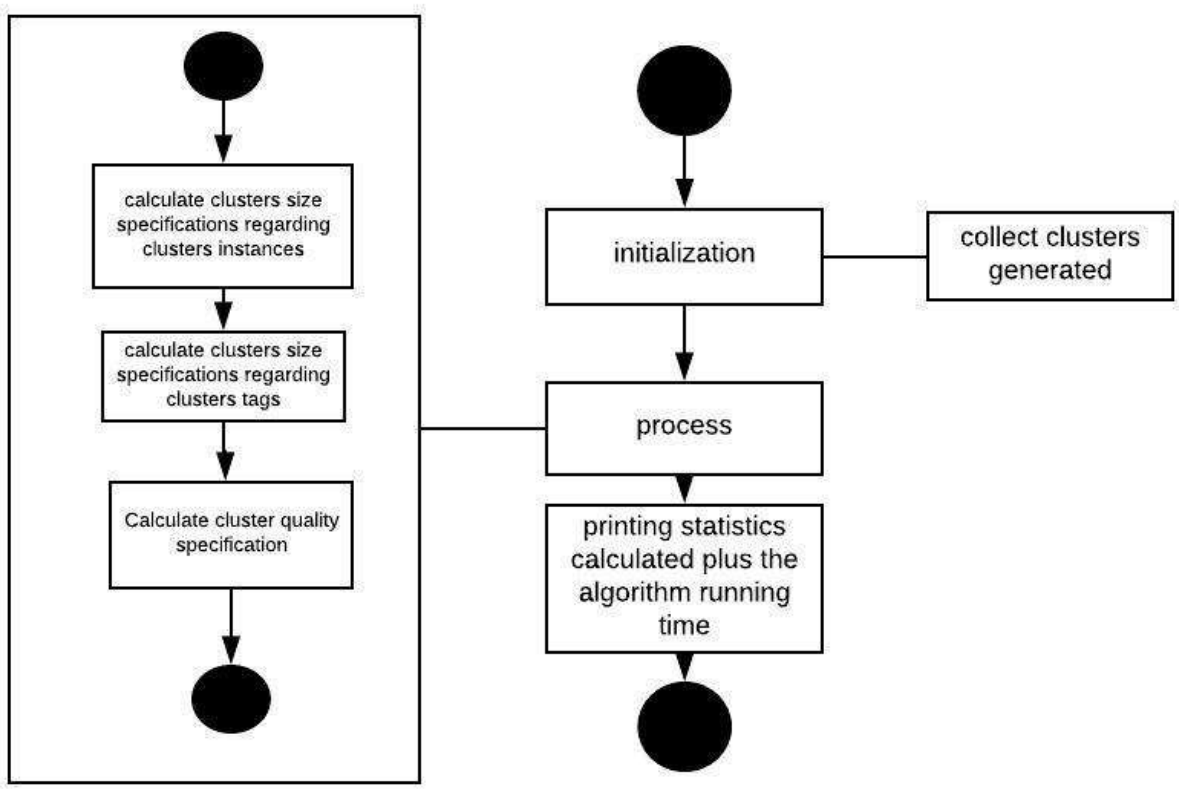


Figure 4.5: Activity diagram that shows the life-cycle of the framework used in order to generate statistics about the clustering algorithm execution and results. This life-cycle is called after the post-processing service method, when all the clusters have been generated and properly translated in the user-friendly format required. More details about statistics used are present in the section 4.4

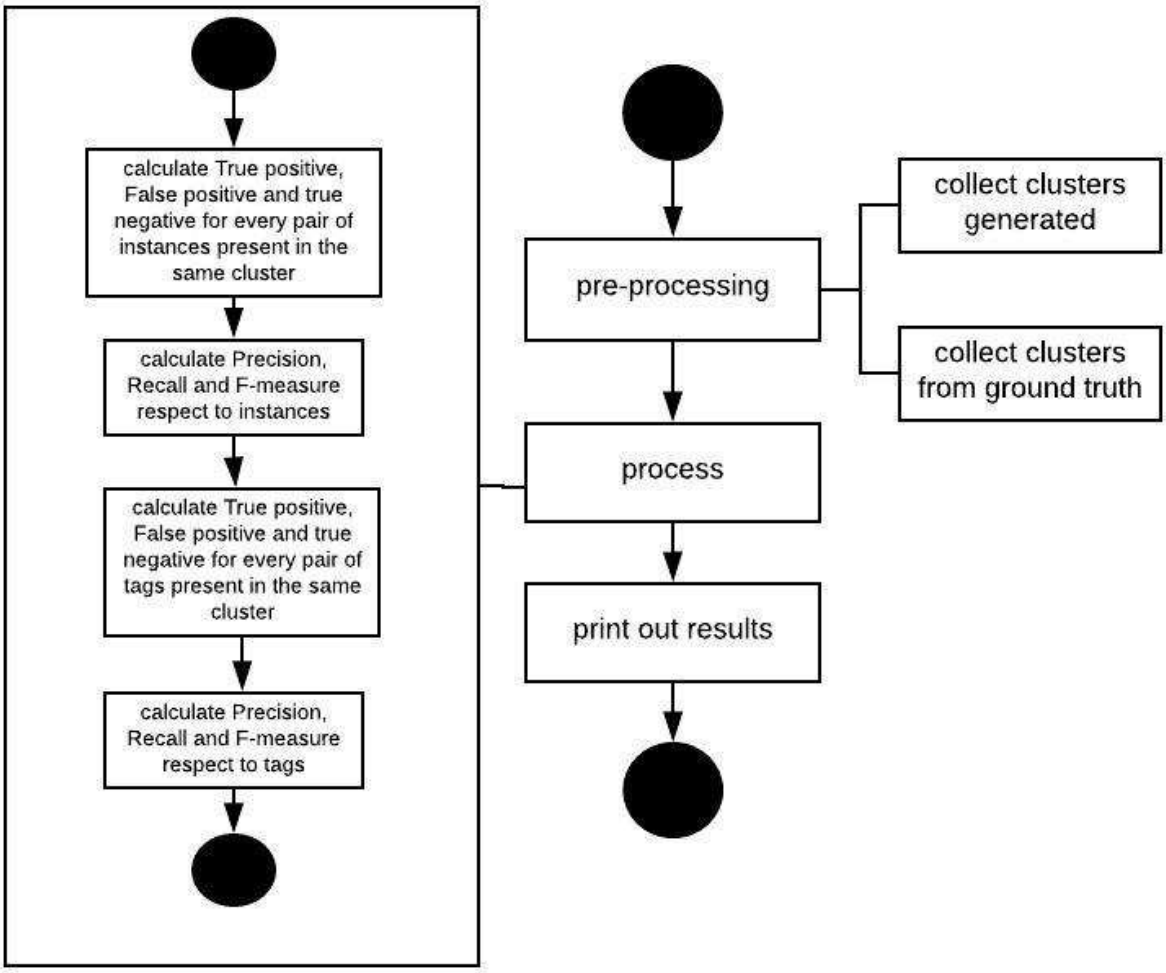


Figure 4.6: Activity diagram that shows the life-cycle of the class in charge to run the pairwise comparison evaluation. This life-cycle is called after the post-processing service method, when all the clusters have been generated and properly translated in the user-friendly format required. More details about statistics used are present in the section 4.4

between two broad categories of measuring recommendation accuracy: rating prediction, often quantified in terms of the root mean square error (RMSE), and ranking, measured in terms of metrics like precision and recall, among others [21].

Since, through the recommender systems implemented, we are recommending groups generated by the framework and not groups that actually exist in a ground truth, the correctness of a recommendation is not so easy to define. We in fact established and proposed a recommendation quality measure, explained in section 4.6, that we used in order to evaluate the recommender systems based on the groups recommendations they generate. The EvaluationRecommendations.java class is in charge to evaluate the recommender system desired. The structure used for this class follows the design used for every service of the framework and its life-cycle works as follow:

1. preProcessing(): is in charge to collect the recommendations made by the system in question;
2. process(): generate the statistics about the recommendations and their evaluation (more details on how we evaluate recommendations and the statistics generated are described in section 4.6);
3. print(): print out the statistics and evaluations genrated.

4.3 Clustering algorithms implementations

The research purpose was to analyze and evaluate social users groups. The framework is born as evolution and extension of this purpose. Hence, some cluster algorithms implementation have been implemented by using the framework structure in order to make progress in the primal research goal. In particular we provide implementation of three different cluster algorithms i.e. k-means, fuzzy k-means and affinity propagation. The general architecture of every clustering algorithm implemented follow the ServiceInterface interface and therefore the four interface steps:

1. Pre-processing: in this step, information are adapted as required by the algorithm in order to be processed;
2. Execution of the clustering algorithm;
3. Post-processing: consist in the inverse process of the pre-processing step. That is to collect the data resulting from the algorithm execution and translate them in a "user-friendly format";
4. Save clusters: step in charge to save the circles created into the dataset;

This chapter exposes the algorithms implemented by giving an initial overall view of the algorithm logic for then analyze its implementation through the framework.

4.3.1 K-means implementation

First of all, before running the algorithm, in the k-means, users must specify the number of groups they want to generate, then k-means will group data-points accordingly. Defining the best k is a common problem in centroid models cluster algorithms. The correct choice of k is often ambiguous. This is because the choice of the number of clusters, strictly depends on the ratio between clusters dimension and clusters quality the users want to achieve. In-fact, bigger is the cluster, lower will be its quality. However exist methodologies used in order to try to find the best k according to the user requirements. For the ones interested in this topic [23] [16] illustrate some methodologies.

For our k-means implementation, the number of clusters (k) has been defined as the division of the total number of users by a defined ideal cluster size.

The initialization of the k-means, consist in the translation of the users as data-point in a feature space. Each data-point (in our case user) in fact, to be clustered, has to be represented as a point in a "n" dimensional space, where n represent the number of possible users' features (tags in our case).

The algorithm starts by randomly choose k points (in the n dimensional space) as initial centroids and assign each data-point to the nearest centroid. The distance is calculated through a metric predefined. The default distance measure is the Euclidean distance but also other metrics are provided by the framework.

Once all the data-points have been assigned to a cluster, the algorithm re-generate cluster centroids by calculating the center point between cluster data-points instances. The metric used to calculate the center point is the same one used in order to assign data-points to clusters. Finally the k means re-assign every data-point to the nearest centroid. K-means runs these two last steps until convergence or, until a predefined number of iterations has been reached up.

There exist several Java libraries that allows to perform k means algorithm in a semi-automatic way. We decided to use the Apache Mahout framework. Apache Mahout(TM) is a distributed linear algebra framework and mathematically expressive Scala DSL designed to let mathematicians, statisticians, and data scientists quickly implement their own algorithms [1]. We decided to use Mahout because it provides a map-reduce K-means implementation. Thanks to map reduce it is possible in fact to execute the algorithm also on huge dataset in a distributed system. Furthermore Mahout provide pretty simple and intuitive framework Java structures and documentations.

As mentioned, users, in order to be processed by the algorithm should be translated in a n dimensional numerical array. In our particular research, feature data are user's tags, and the array resulting from this transformation represent an user. User's tags are treated as user's preferences, that an user may have or not have. The dimension of the user's array is the number of tags present in the dataset and each element of the user's array correspond to a tag. If the user in question has the tag (preference), the value of the array element corresponding to the particular tag will be 1, 0 otherwise.

The pre-processing method is the one in charge to execute this operation. It collects the data from the dataset and prepares them for the k means execution. Furthermore the pre-processing data execute operations in order to pass the input arguments required by Mahout. Arguments are the delta-convergence, the classification threshold and the distance measure. Delta convergence value defines when to declare the convergence and then when to stop the algorithm. In particular the algorithms will stop when, from an iteration to another one, all the cluster centroids have not moved more than the delta-convergence value. Classification threshold value defines instead when a data-points must create a new cluster or will become part of an already existing one. Basically, if the distance from a data-point to the nearest centroid is greater than the threshold, then, the object will create a new cluster.

The k-means service pass then the pre-processing output data to the process method. The process method calls the Mahout function "runKmeans (void)" that starts the algorithm. Once finished the execution, the service passes the process output to the post-processing function.

The post-processing step is in charge to write back the cluster generated by the algorithm in a user-friendly format. The runKmeans() output in-fact should be translated in order to have more understandable output data.

The save circle step instead saves through the appropriate DAO classes the clusters generated and their instances (users) into the dataset.

The last part of the service consist in the generation of the statistics regards the algorithm execution (execution time and arguments setting) and the cluster generated (clusters size and quality specifications). The generateStatistics() method is the one in charge to call the functions that generate them. More details about statistics are explained in the next section 4.4.

Function used to print all the results is also defined in the service.

4.3.2 Fuzzy K-means implementations

Fuzzy k-means is the soft clustering extension of the k-means algorithm. The major difference between k-means and fuzzy k-means is that, in the fuzzy k-means each data-point can belong to more than one cluster with certain probability. Since users can belongs to more than one group at the same time and for different motivation, fuzzy k-means seems to us an algorithm that potentially fits better our domain with respect to k means. In fact users usually belong to more than one group.

Design, structure and implementation of the fuzzy k means are pretty the same of the k-means ones. The only difference is the required "fuzzyness" input argument, a positive value which control the extent of sharing among fuzzy clusters.

4.3.3 Affinity propagation

The last cluster algorithm we coded and tested on the top of our framework is the affinity propagation algorithm. Affinity propagation is an agglomerative hierarchical clustering technique.

We chose affinity propagation essentially for three main reasons:

- In order to compare results on two completely different clustering techniques (research purpose);
- Cluster centroids (exemplars) in the affinity propagation are data-points (users). In the k means implementations instead, centroids are just points in a certain vector space. This can be useful in order to find the most representative users;
- Affinity propagation do not require a number of clusters to be determined before the algorithm execution. Avoiding in this way the "initial k number" problem.

Affinity propagations starts by the creation of a similarity matrix between data-points to be grouped. This means that, in affinity propagation, data-points(users) are compared each other using a similarity measure and not a distance metric as in the k-means. The similarity matrix is an $M\ x\ M$ dimensional matrix where M represents the number of data-points. Each row and column element correspond to a data-point. The cells contain the similarity between the row data-point and the column data-point in question. The main diagonal of the similarity matrix represents the object preferences, a value that define how likely a particular data-points is to become an exemplar.

The algorithm, once initialized the similarity matrix, executes the following operation until convergence:

1. Calculate responsibilities: responsibility r(i, k) reflects the accumulated evidence for how well-suited point k is to serve as the exemplar for point i, taking into account other potential exemplars for point i. Responsibility is sent from data point i to candidate exemplar point k ($responsibility(i,k) = similarity(i,k) - max_{k' \neq k} \{availability(i,k') + similarity(i,k')\}$);
2. Calculate availabilities: availability a(i, k) reflects the accumulated evidence for how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from other points that point k should be an exemplar. Availability is sent from candidate exemplar point k to point i ($availability(i,k) = min(0, responsibility(k,k) + \sum_{i' \notin (i,k)} max(0, responsibility(i',k)))$).

The stopping criteria are the convergence achievement or the execution of a maximum number of iterations predefined.

We implemented the affinity propagation algorithm by using the APRO Java library. APRO is a Java implementation of the affinity Propagation clustering algorithm. It is efficiently parallelized for use on multi-core processors and NUMA architectures (using "libnuma" native library), offering also simple API [9].

The service class we used in order to implement Affinity propagation is the same one utilized for k-means and fuzzy k-means implementations even if the functionalities of the service methods are very different from the one previously exposed. This demonstrates the flexibility of the framework structure.

First of all, the algorithm parameters that require to be set up before the algorithm execution are:

- Maximum number of iterations;
- Similarity measure to use;
- Data-points preferences value (if there are some).

Data points (users) are translated in numerical vectors through the "one shot" approach regard data-point features(tags), as explained in the k-means description. The pre-processing service function is the method in charge to perform this job. Furthermore the pre-processing initialize and populates the similarity matrix calculating similarity between users. Finally the pre-processing method creates the builder for the APRO library by passing it all the necessary data.

Once the pre-processing finished its execution, the process method starts. The process method calls, through the builder created, the APRO run() function. The run() function requires, as input argument, only the number of maximum iterations.

The APRO run() function returns as output users and exemplars. Each output user is represented by an integer number, where the value of the number correspond to the position of the initial similarity matrix created in the pre-processing.

The post processing method is in charge to create the clusters by using the APRO output and translating the output in a user-friendly format.

The saveCircle() and generateStatistics() methods are the last business logic functions called in the service and their functionalities and implementation are the same mentioned in the k-means algorithm explanation.

4.4 Clustering analysis implementation

The most relevant part of the framework and the main reason for which the framework has been implemented is the analysis of the clustering algorithm results. The evaluation of a clustering algorithm is a very personal and variable topic that strictly depends on the user requirements. An algorithm could be evaluated on its execution performance (execution time, resources required, scalability, performance on big dataset etc.) or on the clusters generated. Execution performance are more or less standard measures, easy to decide and calculate. Evaluations based on the clusters generated by the algorithm are instead more complicated to define. This is due mostly to the vague definition of a "good cluster". First of all cluster size and cluster quality are strictly dependent each others. In fact is easy to demonstrate that, by reducing a cluster keeping the same center, the cluster in question will increase its quality. Moreover, also the quality of the cluster is an ambiguous measure. Cluster quality could refer to different parameters for example the distance between the cluster instances and the cluster centroids, the similarity between cluster instances etc. Furthermore, to define the quality of a group, it is necessary to take into account according to which parameters the data-points have been grouped. In fact, often, the data-point are grouped only on the basis of just a few data-point features. In these cases it would therefore be useless to measure the quality of the cluster taking into consideration all the cluster instances features.

There are two main ways in order to evaluate clusters i.e. by using internal or external criterion. For internal criterion we mean parameters used in the evaluation of the cluster quality that are derived from the clusters theme self, hence obtained without introduce external factors. With external criterion instead we mean parameters that comes from a ground truth. In this case, the evaluation of the cluster quality is based on a comparison between the cluster generated and the real ground truth group.

In the framework we decided to return a general and thorough evaluation of the cluster algorithm by including: execution performance of the algorithm, cluster evaluation based on internal criterion and cluster evaluation based on external criterion as well. So that the user of the framework has the possibility of having an overall view of the results obtained through the algorithm.

4.4.1 Execution performance

The variation of the execution time and the resource utilized with the change of the amount of data handled by the algorithms are important evaluation factors especially for applications that have to generate clusters on demand.

In the framework we implemented a special class "ExecutionTime" which is used in order to calculate the time required for each different service operation to be executed. Furthermore it returns also the total application execution time. We decided in-fact to split the execution time for each different service step. This is because different steps, such as the generation of the cluster statistics or the store of the results into the dataset are operations that are not mandatory for a right execution of the clustering algorithm itself. Hence, the performance of the algorithm on these operations, may not affect the user's evaluation of the algorithm.

The calculation of the memory used by the program is another runtime execution performance evaluation often taken into account by developers. In Java is possible to know the current memory used by the program via java.lang.Runtime.getRuntime(). Actually this method returns the total

memory available in the Java virtual machine and the currently unused memory. The memory used by the program can be obtained by calculating the difference between total memory and free memory. However this is just an estimation of the memory used by the program since other parameters such as heap memory and garbage collector actually use the Java virtual machine memory. Furthermore, the algorithms implemented, being scalable, tends to use all the memory available, especially when the algorithms have to handle and manage a big amount of data. In addition, memory performance is a runtime parameters, hence, there is the need to use an always active thread in order to store the memory data. Having an always active thread is a bad programming practice, especially for application that require a great computer effort. Furthermore monitor systems are present on every operating system. So, we decided to not implement in the framework a methodology used to calculate the memory used by the application.

4.4.2 Cluster evaluation on internal criterion

As already mentioned, cluster quality and cluster size are two parameters that are strictly correlated. Therefore, clusters evaluation strictly depends on the user requirements. Hence, it's impossible to automatically evaluate the clusters generated just by using internal criterion. The algorithm user in fact should make a final review of the clusters in order to verify how much the algorithm results satisfy his/her requirements.

In the framework we add three different analysis of the clusters i.e. cluster size, cluster tags and cluster quality specifications.

Regards clusters instance size specifications, we calculate and print the size (number of instances) of each cluster and then we report, the biggest cluster, the smallest one and the average size. Furthermore we calculate the size distribution. The size distribution consist, for each different cluster size reported, in the percentage of the number of clusters having the size in question.

Clusters tags size specifications follow the same logic of the clusters instances size specifications. We basically calculate the cluster size regards cluster features (tags) instead that cluster instances. Cluster tags are all the feature of the cluster instances for which the data-points (users) have been grouped. For each cluster generated, we calculate and report the cluster tag size (the number of different tags present in the cluster). Then we calculate the cluster containing more different tags, the one that contains the smallest number of tags and the average.

As evaluation of the clusters quality, we defined a measure that could be significant independently of the clusters algorithm that generated the clusters and the type of the cluster evaluated. We have indeed considered important to maintain a unique quality measure, in order to simplify the comparison between the results deriving from different algorithms. We decided to calculate the quality of a cluster basing the quality measure on the cluster instances' features (tags) for which instances have been grouped. We define the quality of a tag in the cluster as the percentage of the number of cluster instances that has the tag ($TagQualityInCluster = \frac{NumClusterInstancesWithTheTag}{NumClusterInstances} * 100$). The whole cluster quality is defined as the average of the cluster tags qualities. We report the cluster quality for each cluster and, as in the other evaluation explained, we highlight the cluster having the highest quality, the lowest one, the qualities average and the qualities distribution.

Furthermore, during our tests, we discovered that, if the data-points to cluster contain lot of tags, cluster algorithms tends to group the data-points on the basis of only the most frequent tags. In this cases the quality of the clusters will be always low. This is because the time frequency of tags in a cluster created is high only for the tags used as grouping criterion, but, we calculate the cluster quality by using all the tags present in the cluster. So we add to cluster statistics, the possibility to calculate the cluster quality by taking into account only the "n" most frequent tags in the cluster. Where "n" shall be defined by the framework user.

The statistics of each cluster (cluster size and cluster quality) are calculated during the clustering algorithm execution as soon as the cluster is generated. The class in charge to calculate these statistics is called CircleStatistics and is stored as Circle field. The statistics that concern all the cluster generated such as maximum, minimum and average calculations are instead calculated in the Statistics abstract class. This class should be instantiated and called once the clustering algorithm finished its executions.

4.4.3 Cluster evaluation on external criterion

In short, this cluster quality evaluation consist in the comparison between groups coming from a ground truth and the clusters generated by the algorithm. The more a cluster generated is consistent with a group coming from a ground truth the more its quality will be high.

First of all, it's a task of the framework users to choose and make available the ground truth data. Indeed, given the immense difference between groups of various types, coming from different environments, it would have been useless and senseless to have predefined ground truth datasets into the framework.

We implemented two different comparison evaluations between the clusters generated and ground truth groups:

- Comparison of cluster instances and tags;
- Pairwise instances and pairwise tags comparison.

The comparison of clusters instances and tags methodology basically evaluates the clustering algorithm by performing a comparison of the clusters generated with the groups of the ground truth. Obviously, the cluster algorithm that produced the clusters to compare had to be executed on the same ground truth object data in order to make the comparison meaningful. The comparison between a cluster generated and a ground truth group is made by counting how many instances and tags they have in common. In particular for each pair (cluster generated,cluster ground truth) we calculate:

- Percentage of the precision of the cluster instances ($InstancesPrecision = \frac{NumInstancesPresentInBothCircles}{NumInstancesInClusterGenerated} * 100$);
- Percentage of the precision of the cluster tags ($TagsPrecision = \frac{NumTagsPresentInBothCircles}{NumTagsInClusterGenerated} * 100$);
- Users missing: users that are present in the cluster generated but are not present in the ground truth group;
- Tags missing.

As it can be noted the precision percentage represent how much a certain cluster generated is included in a ground truth group. This is because we found that is pretty improbable to generate groups identical to the ground truth ones, especially talking about users' groups in a social domain. Then, we decided that, if a cluster created is a subgroup of a ground truth group, this cluster is 100% correct. The Comparison.java abstract class is the class in charge to perform this comparison. In particular the Comparison class store the comparison data about a single pair (cluster generate, ground truth group).

The class in charge to perform the overall algorithm comparison of cluster instances and tags is the ComparisonService.java class. This Java class must implement the ComparisonServiceInterface and its methods should work as follow:

1. Pre-processing(): collects all the clusters generated and all the ground truth groups;
2. Process(): computes and store the comparison data for each (cluster generated, group ground truth) pair through the Comparison.java instance class;
3. Post-processing(): once finished the comparison between every possible cluster pairs, the post-processing define for every cluster generated the most similar ground truth group. The most similar ground truth group for a cluster generated is the one that has the highest precisions percentage. Final evaluation are then calculated by taking into account for each cluster generated only the statistics about its similar ground truth group. Furthermore the post-processing method calculates the averages of percentage precision on cluster instances and tags;
4. PrintComparisons(): prints out all the evaluation measures calculated.

An alternative of the clustering comparison reported above, is a pairwise instances and tags comparison. The evaluation of the cluster algorithm in the pairwise comparison is made by calculating the ability of the algorithm to classify pairs of instances or tags correctly. A pair of instances is classified as correct when both the elements of the pair present in a ground truth group are also present in a cluster generated. We say then that the algorithm produce 100% correct results, when each instances pair contained in all the ground truth groups are also contained in the clusters generated. Regarding tags instead, the comparison logic is the same but based on cluster tags pairs.

In pairwise comparison we evaluate the algorithm using precision, recall and f-measure evaluation measures. These evaluation measures are all based on confusion matrix. Hence, true positive (TP), false positive (FP) and false negative (FN) predictions about cluster instances and clusters tags pairs must be defined:

- True positive prediction: when an instances/tags cluster pair is present in both cluster generated and in at least one ground truth group;
- False positive prediction: when an instances/tags pair is present in the cluster generated but is not included in any ground truth group;
- False negative prediction: when an instances/tags pair is present in the ground truth group but is not included in any cluster generated.

Precision ($Pre = \frac{TruePositive}{TruePositive+FalsePositive}$),Recall ($Rec = \frac{TruePositive}{TruePositive+FalseNegative}$) and F-measure ($F_\beta = \frac{(1+\beta^2)(Precision*Recall)}{\beta^2 Precision+Recall}$) are then computed.

ComparisonPairwiseService.java abstract class is the framework class in charge to perform the pairwise comparison. The class implement the ComparisonServiceInterface and its methods should work as follow:

1. Pre-processing(): collects all the clusters generated and all the ground truth groups;
2. Process(): generates the confusion matrices about cluster instances and tags and then calculate the evaluation measures;
3. PrintComparations(): prints out all the specifications calculated.

4.5 Group recommender systems implementations

In hard clustering algorithms, as already mentioned, an instance (user) could be included in at most one cluster. Moreover soft clustering algorithm tends to include data-points in more than one cluster through clusters overlapping. This means that a data-point usually will be included just in similar clusters. Furthermore, a re-execution of the same clustering algorithm, after updating only a small amount of data, will produce the same results of the last execution. Talking about application used for users' groups creation and groups recommendations, this behavior could bring applications in a deadlock. In fact, after some cluster algorithms executions, no more groups would be created or updated, meaning that users will not receive new group proposals.

In order to avoid this problem we decided to include in the framework three different group recommendation systems which are in charge to recommend to users the groups created by the clustering algorithms present in the framework. In this way we had the possibility to include another data mining topic (recommender systems) maintaining at the same time the focus of the framework on clusters.

In the framework we implemented three different recommendation system approaches i.e. user based, item-based and SVD collaborative filtering techniques. All the recommended systems operate on the same users-circles (rank) matrix. Rank defines the affinity ratio between the user and the circle in question and its calculated in the clustering algorithm service when the clusters created are saved into the dataset. The rank is a ratio between the user's tags and the circle tags($rank = \frac{TotUserTags-UsersTagsMissing}{5}$ where UsersTagsMissing is the number of users tags that are not equals to the best 5 cluster tags).

All the recommendation systems have been implemented on the top of the Apache Mahout framework. Mahout recommender systems operate on a matrix of user-item ratings. In our case, items are groups and the rating is the rank.

4.5.1 User based approach

In the user-based approach the algorithm predict item-users rating by combining the ratings of other users that are similar to the user in question. The items (groups) recommended to an users are the ones for which have been predicted the highest rating.

Similarity between users is based on their ratings. The more two users rated the same items in the same way the more the two users are similar. The similarity measures used can be chosen by developers.

Since this is a Mahout recommender system implementation, the structure strictly follow the Mahout requirements.

First of all, Mahout ask to build a org.apache.mahout.cf.taste.recommender.Recommender object. The Recommender is the object in charge to perform the recommendations. The object should contains:

- Similarity measure: the similarity measure that the algorithm will used in order to calculate similarity between users. The similarity measure must implement the org.apache.mahout.cf.taste.similarity.UserSimilarity class;
- User-user technique: for user-user recommendation systems Mahout only provides nearest neighbor approaches. There are multiple implementations of nearest neighbor made available by Mahout. The only constraint in order to choose an approach is that it must implement the org.apache.mahout.cf.taste.impl.neighborhood class;
- Data model: org.apache.mahout.cf.taste.impl.model.GenericDataModel object is used to contain and manage the data. Data model should contains the data used in order to create the user-item (score) matrix i.e. groups, users and rank. Mahout, once collected the data, generates the matrix automatically. Data could be collected at running time easily from a csv file or from a database depending on how comfortable the developer is. In our case, we get the data from the framework orientDB dataset since we work on data generated by the framework cluster algorithms.

The creation and instantiation of the Recommender object is performed in the framework in the preProcessing() recommendation service method.

Once instantiated the Recommender object, in order to generate the recommendations, there is the need to call the recommend(user, number of recommendation) Recommender method. This method will return a list of RecommendedItem for the user passed as argument. A RecommendedItem object contains the item recommended and the value of the recommendation for the user in question. In the case that there are not recommendations for a determined user, the list will return as a null object.

By parsing the (RecommendedItem) list is possible therefore to obtain the recommendations made for the user required.

In the framework we call a Recommended.recommend method for every user present in the dataset so as to obtain recommendations for every user.

4.5.2 Item based approach

In the item-based approach the prediction of an item-user rating is based on items similarity. In particular to predict the rating of a certain item "I" by an user, the algorithm make a weighted average of the user's rating of items similar to "I". The weight used is the similarity between "I" and the item in question. The similarity between items is based on the ratings they received by users. The more the two item have been rated similar by the same users the more their similarity is high. As in the user based the similarity measure can be chosen by developers.

The item-based recommender system implementation has the same structure and behavior of the user-based one. The only thing that change is the builder instantiation. In fact, in the item based approach, the similarity must implement org.apache.mahout.cf.taste.similarity.ItemSimilarity class.

Furthermore also the recommendation techniques are different. In our case we used the GenericItemBasedRecommender approach.

4.5.3 Singular value decomposition (SVD) item based approach

SVD is a latent factor based recommender system technique. SVD approach works in the same way as the item-item collaborative filtering approach but, instead to operate on the whole user-item ratings matrix, it operates on the user-item ratings matrix transposed in a factor space (factorization matrix). The matrix factorization is in charge to reduce the number of variable used by merging some original matrix variables that vary together. The element that is created through the merge of more matrix variables is called factor. This technique allows to discover possible correlations between features, creating also in this way a less sparse matrix. The matrix is reduced through matrix factorization and in particular by using the singular value decomposition. For more details about this technique please refer to the paper [7].

The real problem of matrix factorization is to find the right factor space of the matrix, that is the number of factors that will produce better results in the recommender system. In-fact in the factorization matrix each factor explains a percent of the total variance of the original matrix. Factors that do not explain much variance might not be worth including in the final model. So, ideally, the factorization matrix should contains the factors that explain the most part of the total original matrix variance.

The technique used in order to find the right number of factors is called factor analysis. Factor analysis is a technique that allows to highlight the existence of one or more factors, not measurable directly, within a set of directly observable variables.

In the framework we implemented factor analysis using the Principal component analysis (PCA) statistical procedure. PCA is a mathematical procedure that transforms a number of (possibly) correlated variables into a (smaller) number of uncorrelated variables called principal components. The steps we used in order to perform PCA are the following:

1. Collect the data and create the user-group rank matrix;
2. Calculate the group-group correlation matrix. As correlation measure has been used the Pearson correlation. The Pearson correlation is calculated by comparing the user's ranks respect the group in question;
3. Compute the eigenvalues of the correlation matrix. The eigenvalue for a given factor measures the variance in all the variables which is accounted for by that factor;
4. Sort the factors in a descendant order respect to their eigenvalue ratio and get neatly, starting from the first one, the factors until the factors collected will have explained the most part of the total matrix variance. This is because the ratio of eigenvalues is the ratio of explanatory importance of the factors with respect to the variables. If a factor has a low eigenvalue, then it is contributing little to the explanation of variances in the variables and may be ignored as redundant with more important factors.

In the framework we implemented the factor analysis in the FactorAnalysis.java class. The class will take in input a matrix and will return the number of factors to use in the SVD recommendation. The class has been implemented in order to improve the quality of the recommendation system and also in order to avoid that the framework users will lose time in randomly finding the number of factors to use in the recommender system.

Framework users, in order to execute the analysis, have only to specify the percentage of the variance of the matrix they want to explain with the factors. The framework gets automatically the data from the OrientDB dataset and will return the number of factors that explain the percentage of variance of the original matrix required.

The SVD recommender system works in the same way of the item-based and user-based approaches. Then the framework procedure and structure is identical to them. The only thing that change is the instantiation of the Recommender object. In-fact, in order to perform the SVD Mahout recommender

system the Recommender must be instantiate as org.apache.mahout.cf.taste.impl.recommender.SVDRecommender object. Its constructor requires as arguments the data model, the number of factors to use and the maximum number of iterations to perform.

For more technical details about Mahout recommender systems please refer to the Mahout documentation.

4.6 Groups recommender systems analysis implementation

Recommender systems, as already mentioned, are used in the framework in order to recommend to the cluster instances (users in our case) other clusters generated by the framework clustering algorithms.

Evaluations of the recommender systems are based on their ability to generate right recommendations. Usually, in order to evaluate the recommender systems, the systems are executed in a labeled training set. In particular the ordinary steps followed in order to evaluate a recommender systems are:

1. Create a new dataset by removing randomly, from the original labeled dataset, some users preferences about items;
2. Executing the recommender systems and generate recommendations on the new dataset just created;
3. Compare the recommendations generated with the original labeled dataset (if an item recommended by the system to a certain user it's an item that actually the user in question likes in the original dataset, then the recommendation is good).

There are two main ways in order to evaluate a recommender system: rating predictions or ranking.

The rating prediction is based on the comparison between the predicted values of the recommender system recommendations with the ground truth users' rating of the items. This methodology is usable, therefore, only when recommender systems are executed on dataset which provide to users the possibility to rate items. The formula typically used in order to evaluate recommender systems through rating prediction is the root mean square error ($RMSE(X,Y) = \frac{\sum_{i=0}^{n}(x_i - y_i)^2}{n}$ where X is the vectors containing the predictions while Y stays for the vector composed by the ground truth values).

Ranking instead, is a recommender system evaluation methodology based on the creation of the confusion matrix and its attached formulas (precision, recall, accuracy etc.). Also this evaluation is therefore computed through the comparison between the recommendations generated with the ground truth data.

The problem we found during the creation of the evaluation system for the recommender systems implemented in the framework is that, by recommending groups that the framework generates, we have not ground truth data on which to compare the recommendations generated. Then we had to define a recommendation quality measure in order to evaluate the recommender systems.

As recommendation quality measure we simply decided to use the user-group rank explained in the 4.5. In particular for every recommendation produced by the system the framework calculates the rank between the user that received the recommendation and the group recommended. Higher is the rank, higher will be the recommendation value. Then, in order to evaluate the system, the formula used is the average of the ranks respect to every recommendation generated.

This recommendation quality measure used is also coherent with the way the systems produce recommendations. In fact all the recommended systems implemented operate on the same users-circles ranks matrix 4.5.

The functions used by the framework in order to compute evaluations and generations of the statistics about recommender systems are implemented in the EvaluationRecommendations.java class. This class is called by the recommender service once the recommender system finished its execution. Statistics calculated includes:

- Execution time of the recommender system;
- Number of recommendations generated;

- Number of users that receive and do not recommendations;
- Average of the number of groups recommended per user.

Evaluations of the system instead are based on the recommendations quality measure and in particular the EvaluationRecommendations.java class reports:

- The average of the recommendations ranks (value used to evaluate the system);
- The recommendation which achieved the highest rank value;
- The recommendation which achieved the lowest rank value.

4.7 How to extend framework functionalities

The framework structure has been constructed in such a way that, interested developers, can easily incorporate new features. With extension we mean possible implementation and upgrade of the framework aimed at improving and extending the usefulness of the framework itself. The creation of applications and therefore the customization of the framework through the creation of subclasses that extend the framework classes will not be mentioned in this paper. This is because customization of the framework is an intuitive routine framework operation. However, It is worthwhile to mention that, the framework, thanks to the large use of abstract classes, is easily customizable for various user's interests.

The framework extension allowed are:

- Insertion of new cluster algorithms;
- Insertion of new recommender systems;
- Insertion of new cluster statistics;
- Insertion of new similarity measures;
- Insertion of new cluster evaluation techniques;
- Insertion of new cluster comparison methodologies;
- Insertion of new recommender system evaluation techniques.

To extend the framework, the only constraints required are:

- New customizable classes should be implemented as abstract classes. This technique allows to framework users an easy application customization;
- Each service, DAO, or model class should implements its appropriate interface. The interface implementation is necessary to maintain a consistent framework structure and logic. If the class in question has a completely new logic and structure, a new interface must be defined;
- The instantiation of each class should be reachable by the appropriate factory class. Therefore, factories are modifiable. New factories can also be implemented in order to return new instances groups;
- Classes and methods inserted, should be commented by using proper JavaDoc comments;
- In order to create distance measures classes usable by Mahout algorithms, the class must implement org.apache.mahout.common.distance.DistanceMeasure Mahout interface;

5 Tests and results

The framework was implemented with the purpose of creating a Java structure which helps developers evaluating clustering algorithms and recommender systems in order to simplify the framework user algorithms choice based on which algorithm works better in the framework users domain. The evaluations of the algorithms and the relative choice of one algorithm rather than another one are possible only if the results of the executions of different algorithms are comparable each other. Then the main goal of the framework was to generate comparable statistics about algorithms that could give to the framework users an idea about the results produced by different algorithms. So, tests of the framework were mainly executed in order to verify if the generated algorithms statistics are coherent with the clusters created, comparable each others and meaningful.

In order to test the framework we performed different tests of the clustering algorithms and recommender systems implemented on the top of it. We also tried to draw some conclusions about the algorithms behavior based on the statistics generated by the framework, even if the goal was just to verify the comparability of the results.

The data, observations and conclusions reported in this chapter always refer to a social domain. So we will always refer to groups of people and their recommendation. In particular with the test executed we tried to:

- Evaluate precision and sensibility of the cluster algorithms and recommender systems evaluations defined;
- Evaluate the clustering algorithms implemented in our domain;
- Evaluate the cluster recommender systems implemented in our domain;

5.1 Domain of the algorithms

Group creation is a topic very used by social media in order to speed up the share of content among more than one user at time. Mailing list or Whatsapp groups are common and popular examples. Also on-line social network such as Facebook, Google plus and Twitter built the possibility for users, to manually create groups. For the ones interested on how existing groups methodologies work, this paper explains and analyzes some [5]. One problem that arise on user manual group creation is the user time required due to the big amount of parameters to be set. Social media association such as Facebook and Google moved therefore on automatic grouping methodologies called respectively Facebook Smart List and Google Circles. These grouping methodologies offers a pre-build user's list based on common user characteristics, for example friends or features. Other technologies such as ReGroup proposed methodologies that help people to create groups on-demand. Regroup, relying on social networks people interactions, learning probabilistic model in order to suggest new group members or group characteristics [3]. Social groups have also started to be used for recommendation systems purposes. The research explained in this paper, for example, [29] uses social groups in order to weight the user's review in recommender systems. They differentiate user's review importance relying on how much an users trust another user respect to the domain of the recommendation. A user trust another user in a determined domain if they are in the same group regards the domain in question.

Group creation used for the aim of recommendations purpose is a relatively new topic. Traditional recommender systems and in particular collaborative filtering approaches predict users' interests by mining users' rating history data. Group recommendations start to have first results on media and tourist attraction [11], "activities" that people like to do in groups. An example is Polylens [15], a MovieLens recommender system extension about collaborative filtering item recommender system that perform recommendations to groups of users.

The success of group recommender systems is strictly correlated with the quality of the groups about at least one feature of the item domain to recommend. Distances between cluster members, density of the data space, statistical distributions are just some of the possible parameters utilized in order to evaluate the quality of clusters. All these performance measures, are valid clusters evaluation measures but are dependent to the clustering algorithm used. This makes very difficult the comparison between clustering algorithm performances and their results. The contribution of the framework is to define a technique able to evaluate clustering algorithm and recommender system results independently of the algorithms used. Then have been defined also some quality measures used in order to evaluate all the algorithms implemented on the top of it. The tests executed are therefore focused on the demonstration of the comparability of the algorithms evaluations and the quality measures used.

5.2 Datasets

Tests were performed on three different datasets in order to avoid results dependent to the dataset choice. Since the datasets structures and distributions were completely different, different results has been achieved during this test set. In particular the dataset we used were called:

- Acanto.
- VK.
- Meetup.

5.2.1 Acanto dataset

The first dataset used is a sub-part of a bigger dataset called Acanto. Acanto is a randomly-automatically generated dataset containing random data. The dataset were generated for the "ACANTO - A Cyberphysical social Network using robot friends" project. This project is about "a robotic friend (the FriWalk) that sup-ports the user in the execution of daily activities that require physical exercise and an intelligent system that recommends activities that a senior user perceives as compelling and rewarding" [19]. The dataset we used is an extraction of the original Acanto dataset, and, in particular the data extracted are (senior) users, preferences and their correlation. Acanto dataset does not contain users' groups, so it was not possible to evaluate clusters by using the external criterion framework methodology since there was no appropriate ground truth on which compare the clusters generated by the algorithms. This dataset has been mostly used in the early stage of the research in order to verify the correctness of the algorithms implemented. The tests executed on this dataset were also used in order to understand the most correct arguments values of the various cluster algorithms to use, in order to properly calibrate the execution of the algorithms and their results. The distribution of the dataset is shown in the table 5.1.

Table 5.1: Acanto dataset data distribution.

Number of users	31820
Number of tags	142
Average of the number of tags per user	4,42

5.2.2 VK dataset

The database called "VK" contains data collected from the Russian social network VKontakt. The data were collected through the Java API made available by the social network. We respectively collected users, tags and groups. In particular, the groups collected correspond exactly to VK's groups. Tags instead correspond to VK's pages where users can subscribe to (follow). We collected the VK data by following these steps:

1. Pick randomly a user;
2. Save the user specifications;

3. Collect and save as tags the pages that the user follow;
4. Collect and save the user's group;
5. For every user member of the user's groups repeat these steps.

The distribution of the data collected are exposed in the table 5.2.

Table 5.2: VK dataset data distribution.

Number of users	5576
Number of tags	34631
Average of the number of tags per user	10,73
Number of groups	7
Average groups size	796

As it can be noted in the table the number of tags is very high compared to the number of groups and to the number of users as well. This is because, on social networks, users tend to follow a big number of pages. Furthermore, the user from whom we started collecting data is a "VK popular and active user". The activeness and popularity of an user in the social network in-fact increase the probability of having more pages and groups related to the user in question. Another important observation about VK dataset is the size of its groups. In fact we collected groups containing a big amount of users and consequently a big amount of tags.

5.2.3 Meetup dataset

Last ground truth data were collected from Meetup. Meetup is an event based social networking that facilitates hosting events in various localities around the world. Users are subscribed on Meetup mainly to organize or participate to events. In order to get events recommendations, users must specify their preferences (tags). Furthermore Meetup users can create groups or subscribe to existing ones. So we had the possibility to collect, users, tags and groups as well. The data have been collected through the Meetup API.

Actually we created two different datasets using Meetup data. The first Meetup dataset has been created by uploading random Meetup users, their tags and their groups. The 5.3 shows the distribution of this dataset.

Table 5.3: Meetup dataset data distribution.

Number of users	2111
Number of tags	4340
Average of the number of tags per user	26,01
Number of groups	3767
Average groups size	5

The second Meetup dataset, called "Meetup-older", has been created by updating only the Meetup users belonging to the "60+ happy hour" group. We then uploaded their tags and the other groups to which these users are subscribed. Distribution of the dataset is listed in the table 5.4. We created this dataset in order to check if clustering algorithms and recommender systems executions outperforms other executions by running them on users that are correlated in some ways. In our case by example, we could assume that the users present in this dataset are all registered in the same group, interested in the same group's tags, and also that they have a common age group (60+). We started in fact from the supposition that the execution of clustering algorithms on similar users would have produced higher quality results (for quality we always refer to the cluster quality measures defined in this paper).

5.3 Algorithms arguments calibration tests

In this part of the thesis we use very often the terms "cluster quality" or "quality of the cluster" . With these term we refer to the quality of the cluster measured by using internal criterion explained

Table 5.4: Meetup-older dataset data distribution.

Number of users	489
Number of tags	1248
Average of the number of tags per user	22,99
Number of groups	942
Average groups size	5

in this thesis in the section 4.4.

First tests were performed in order to understand how to calibrate the clustering algorithms implemented in our particular social domain. By calibrate we mean define which algorithms' arguments values to use depending on which are the results the framework users want to achieve.

5.3.1 K-means and Fuzzy K-means

The arguments that must be defined before the execution of one of these two algorithms are:

- Threshold;
- Estimated circle size;
- Delta convergence;
- Distance measure.

Threshold value defines when to assign an object (data-point) to a cluster. In general if the distance to an unclustered data-point and its nearest cluster center is less than the threshold, then the data point will be assigned to the cluster in question. Since we had no idea about the distance distribution of our data points, we were not able to define a proper set of tests focused on the threshold value.

The first tests were made on the estimated circle size value. There is a direct correlation between the estimated circle size and the initial k since we calculate the initial k by using the $\frac{NumberOfDataPoint}{EstimatedCircleSize}$ formula. The higher are the estimated circle size the lower will be the number of initial ks. We performed this test set by using the Acanto dataset, and, the best results have been achieved for an estimated circle size value equal to 25.

Other tests were performed in order to understand the delta convergence arguments role and how its variation compromises the results of the algorithms. Delta convergence defines when to declare the convergence of the clusters and then when to stop the algorithm. In particular the algorithms will stop when, from an iteration to another one, all the cluster centers have not moved more than the delta convergence value. Intuitively by decreasing the delta convergence value the execution time of the algorithm and the quality of the results should be higher by producing more iterations. The best results were achieved with delta-convergence=0,1 and table 5.5 shows the more representative tests results.

Table 5.5: This table describes the test results achieved by changing the delta value argument. The quality distribution group of columns contain the percentage of clusters having a certain cluster quality. Quality average for size distribution list the averages of the cluster tag qualities of the clusters having a certain size.

Delta	**Quality distribution**				**Quality average for size distribution**							**AVG**
	<=70%	**80%**	**90%**	**100%**	**1**	**2-5**	**6-10**	**11-20**	**21-50**	**51-75**	**>75**	
1	0,40%	30,06%	53,67%	15,87%	100%	98%	90,75%	83,65%	81,61%	81,09%	80,78%	84,39%
0,1	0%	24,74%	63,27%	11,97%	0%	97,64%	89,44%	84,50%	82,76%	83,24%	81,72%	84,14%

It can be noted that using a delta-convergence=0,1 the quality of the clusters improved, avoiding also clusters containing only one user. The decrement of the number of cluster having 100% of quality is due to the absence of clusters containing a single person.

The last test set, regarding K-means algorithm arguments, is focused on the distance measure used. Distance measure are used by the algorithm in order to calculate the distance between the data-point to cluster. During these test we used four different distance measures i.e:

- Euclidean distance ($d(X,Y) = \sqrt{\sum_{i=0}^{n}(x_i - y_i)^2}$);
- Pearson correlation distance ($d(X,Y) = 1 - \frac{\sum_{i=0}^{n}(x_i-\overline{x})(y_i-\overline{y})}{\sqrt{\sum_{i=0}^{n}(x_i-\overline{x})^2}\sqrt{\sum_{i=0}^{n}(y_i-\overline{y})^2}}$);
- Jaccard similarity distance ($d(X,Y) = 1 - \frac{|X \cap Y|}{|X \cup Y|}$);
- Cosine similarity distance ($d(X,Y) = 1 - \frac{\sum_{i=0}^{n} x_i y_i}{\sqrt{\sum_{i=0}^{n} x_i^2}\sqrt{\sum_{i=0}^{n} y_i^2}}$).

Initial tests have been executed on the Acanto dataset and the table 5.6 report the results.

Table 5.6: This table describes the test results achieved by changing the distance measure. The quality distribution group of columns contains the percentage of clusters having a certain cluster quality. Quality average for size distribution list the average of the cluster tag qualities of the clusters having a certain size.

Metric	Quality distribution				Quality average for size distribution							AVG
	<=70%	80%	90%	100%	1	2-5	6-10	11-20	21-50	51-75	>75	
Euclidean distance	0,40%	30,06%	53,67%	15,87%	100%	98%	90,75%	83,65%	81,61%	81,09%	80,78%	84,39%
Cosine distance	0,39%	28,27%	57,66%	13,66%	100%	98,46%	89,39%	84,12%	82,29%	81,12%	81,01%	83,70%
Jaccard distance	0,77%	29,93%	55,75%	13,55%	100%	93,66%	88,40%	83,44%	82,04%	80,19%	76,05%	83,40%
Pearson distance	0,16%	31,02%	58,32%	10,58%	100%	97,68%	89,91%	83,16%	83,94%	83,05%	81,23%	83,17%

In general, cosine similarity is the one that produce higher average quality. By taking into account in the analysis of the results also the cluster size distributions, Pearson correlation distance was the metric that reached the best quality on clusters of magnitude greater than 20 users. Jaccard similarity, instead, is the one that produced the worst results in term of quality of the clusters, but should be take into account that it is also the one that creates bigger clusters in term of number of users.

The same test were then executed by using the available ground truth databases (VK, Meetup and Meetup-older) in order to verify the results achieved previously. In these tests Pearson correlation has been detected to be the metric more able to produce clusters whose size is more around the average of the "cluster estimated size" requested by the user in the framework. Furthermore it also significantly reduces the number of clusters containing a single user. Graphs in figure 5.1 report the clusters size distributions produced by the execution of the K-means algorithms using different distance measures.

The reduction of the number of small clusters (with small we mean clusters whose size is between 1 to 10 instances) involved the reduction of the average quality of the clusters produced by the algorithm. In-fact, in this test set, Pearson correlation distance is the metric that produced the worst clusters' quality results in term of quality of the clusters' tags (graph in figure 5.2 shows the results of the clusters tags qualities achieved). Moreover, also the comparison of cluster users and cluster tags with the ground truth data seems to be pretty affected by the cluster size. In-fact, as showed in the table 5.7, the distances measures that produced smaller clusters are also the ones that achieved higher results. By performing the pairwise comparison evaluation of the same results, Jaccard and Pearson are the distance measures that achieved the highest pairwise evaluations (table 5.8 shows the specifications). However, the high recall value produced by the Jaccard measure tests is affected by the slow number of false negative predictions. A false negative prediction reflect a pair of users or tags that is not present in the cluster generated but actually is present in the ground truth database. So, higher is the size of the clusters generated by the algorithm, lower is the probability of having a false negative prediction. Also in pairwise comparison, therefore, the size distribution of the cluster must be taken into consideration for a good evaluation of the algorithms results.

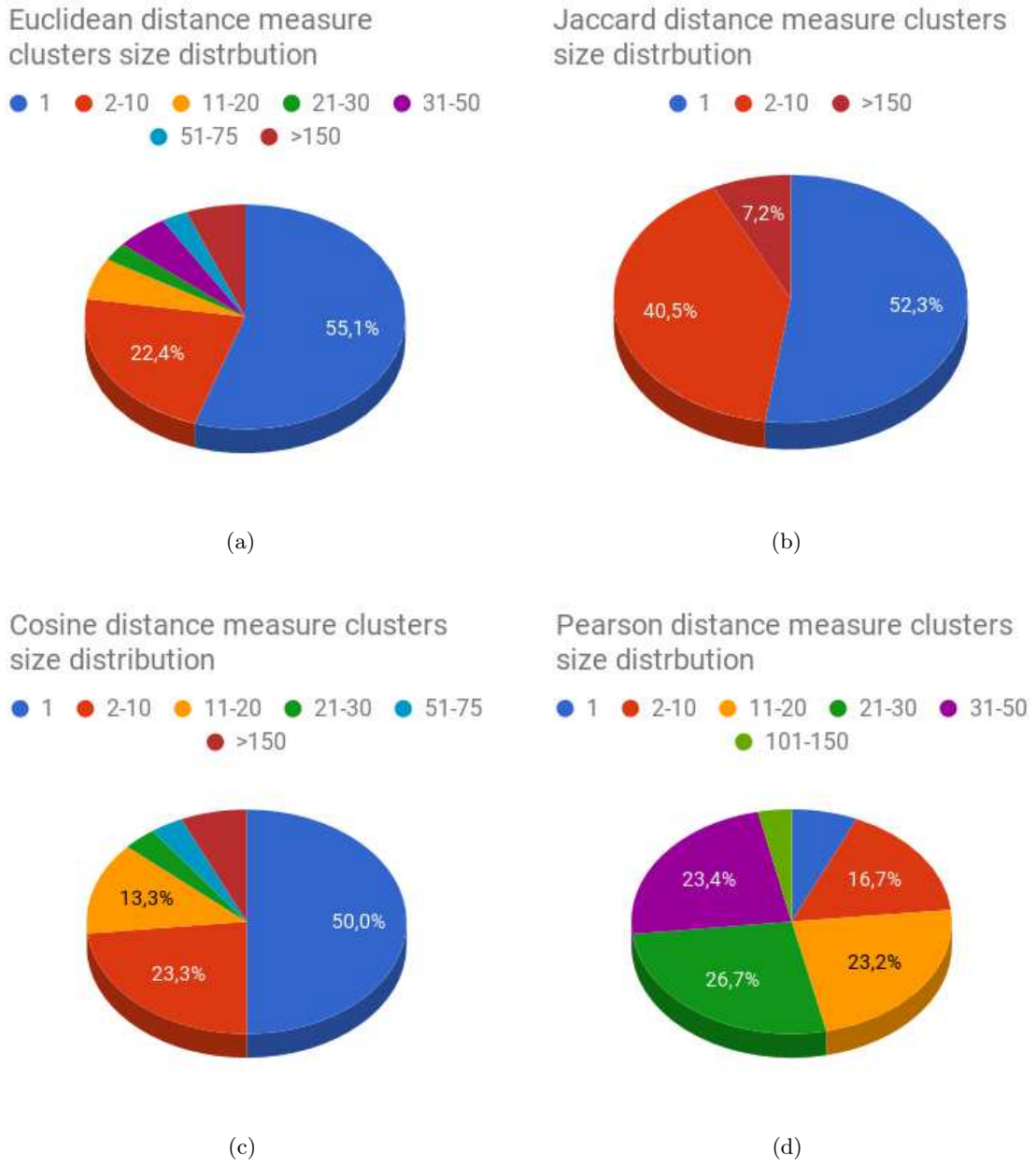


Figure 5.1: Size distributions produced by running the k-means algorithm on VK, Meetup and Meetup-older databases. The legends of the graphs indicated the size of the clusters. Since the estimated circle size required during these tests was "25", it can be noted how Pearson correlation distance is the metric that produced more accurate results, in term of clusters sizes, by creating groups whose size is more around the value required, avoiding also small clusters which, in our domain, were not good to have. Jaccard similarity distance measure is the one that produced more uncontrollable results in term of size. In fact it produced a huge quantity of clusters composed by just a few instances, while the other clusters generated are instead very big.

Table 5.7: This table reports the evaluation of the clusters generated obtained by using external criterion of the clusters generated by K-means algorithm using different distance metrics. The tests are the same described in the section 5.3 and figures 5.1 5.2. In particular the clusters user correctness describe the percentage of users of the "cluster generated" which are also present in a ground truth group. The clusters tags correctness percentage instead describe the percentage of tags of the "cluster generated" which are also included in a ground truth group. More details about the formulas used are present in the section 4.4.

Metric	**Cluster users' correctness**			**Cluster tags' correctness**		
	AVG	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**
Euclidean distance	79%	100%	26,87%	97%	100%	90%
Pearson distance	52,28%	85,71%	24%	94,20%	99,76%	88,78%
Cosine distance	80,72%	100%	33,33%	96,84%	100%	85,90%
Jaccard distance	91,28%	100%	44,26%	98,60%	100%	82,16%

Table 5.8: This table reports the evaluation obtained by using external criterion of the clusters generated by the k-means algorithm using different distance metrics. The tests are the same described in the section 5.3 and figures 5.1 5.2. In particular this table refer to clusters evaluation made through users and tags pairwise comparison. More details about the formulas used are present in the section 4.4.

Metric	**Pairwise users comparison**			**Pairwise tags comparison**		
	Precision	**Recall**	**F1-measure**	**Precision**	**Recall**	**F1-Measure**
Euclidean distance	47,48%	9,09%	15,27%	98,49%	16,39%	28,10%
Pearson distance	62,24%	4,13%	7,76%	99,22%	15,28%	26,49%
Cosine distance	45,80%	16,44%	24,19%	97,02%	23,20%	37,45%
Jaccard distance	52,11%	48,94%	50,48%	96,64%	18,13%	30,43%

5.3.2 Affinity propagation

The arguments required to be set in order to execute the Affinity propagation algorithm are the maximum number of iterations, the preferences values if there are some and the similarity measure to use.

Affinity propagation execution stops when the algorithm reach the convergence or when it has been executed the number of iterations pre-defined. The behavior of the algorithm by changing the maximum number of iterations is simple and intuitive. Furthermore, all the affinity propagation algorithm executions we performed during tests have reached the convergence. We decided therefore to avoid further tests focused on setting the number of iterations because we reputed it as useless.

Preferences define for every data-point how likely a particular data-point is to become an exemplar. A preference is a positive or negative double value. Higher is the preference value for a certain data-point, higher is the probability that this data-point becomes an exemplar. In our particular domain, we did not have any knowledge about users importance. Therefore it was impossible for us to determine, a-priori, the preference values to assign to users. We decided then to maintain the same preference value for every user. However we observed that by changing the value the results changes. We in-fact were not been able to find a static preferences value, since the results change according to the dataset used. We had in-fact to run multiple time the algorithms on each dataset in order to calibrate the preferences values accordingly.

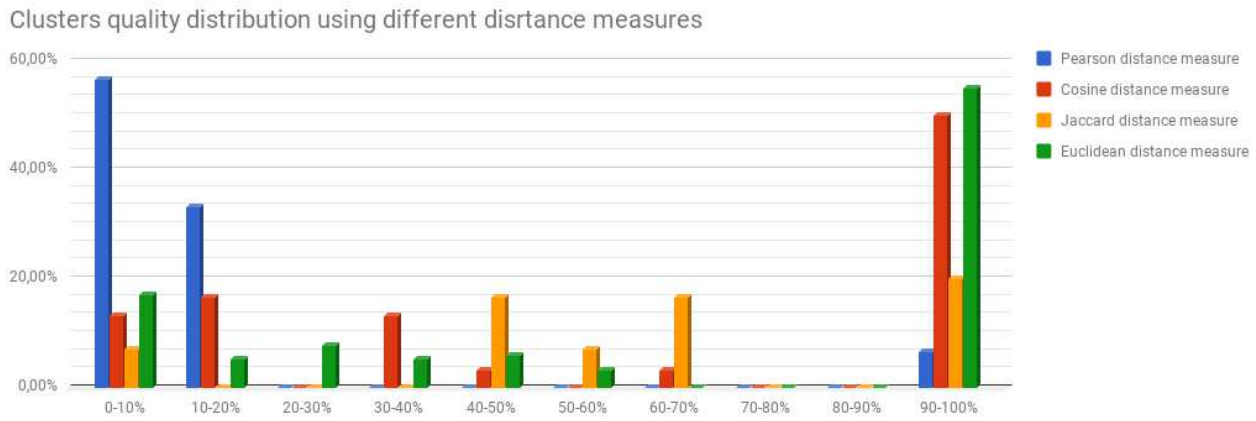


Figure 5.2: The graph shows the clusters tags qualities distribution achieved in the same tests reported in the pie charts in figure 5.1. The X axis represents the quality of the clusters, Y axis corresponds instead to the percentage of the number of the clusters generated. Note that the clusters which report 100% of quality are all clusters containing a single instance (user). By comparing the pie charts and this column graph it can be noted how size and quality of the graph are correlated. Bigger are the clusters lower are the probabilities for the clusters to have obtain good qualities. This is because, usually, higher is the number of clusters instances, higher will be the number of cluster tags.

5.4 Clustering algorithms comparison

In this test set we try to compare the results coming from the execution of the three different clustering algorithms implemented in the framework. The first observation we could do is about the algorithms execution time. Affinity propagation seems to be the algorithm which require more time in order to finish its execution. However, execution time could be affected by other factors, for example the different implementation. We were able then just to really compare the execution time of K-means and Fuzzy K-means algorithms, which have similar implementations. On average, fuzzy K-means algorithms resulted to be slower than K-means, however the difference of the execution time reported are not so relevant.

Another observation, valid for all the three clustering algorithms, observed through the tests, was that, execution time is more influenced by the number of users' tags (instances feature) reset in the dataset than the number of users (instances).

By comparing the results achieved through the execution of the three clustering algorithms in question on the same dataset and, by taking into account only the execution that produced more or less the same clusters size distribution, fuzzy k-means resulted to be the algorithm that produced the best results in terms of clusters quality (the graph 5.3 shows the cluster quality specification differentiated by the clustering algorithm used). This means that fuzzy K-means manages to cluster data points taking into account a greater number of data features than the other algorithms.

However, by evaluating the same tests results by using the external criteria comparison (tables 5.9 and 5.10 show the results), the fuzzy K-means algorithm is the one which has achieved more unsatisfactory results, while, k-means and Affinity propagation evaluations are more or less equal.

5.5 Databases comparison

Since the datasets used for testing the framework have difference data distributions, different results were achieved by running the same test on different datasets. This is because the domain from which the databases derive is different, then the meaning and the use of tags and groups are different. In Meetup, for example, tags are predefined preferences that an user must choose in order to get recommendation about groups or events. In VK, instead, we collected as tags VK pages that users follow. The distribution of users and tags in VK is in-fact more messy and then, by running the clustering algorithms on this dataset, we expected to achieve more random and lower quality results.

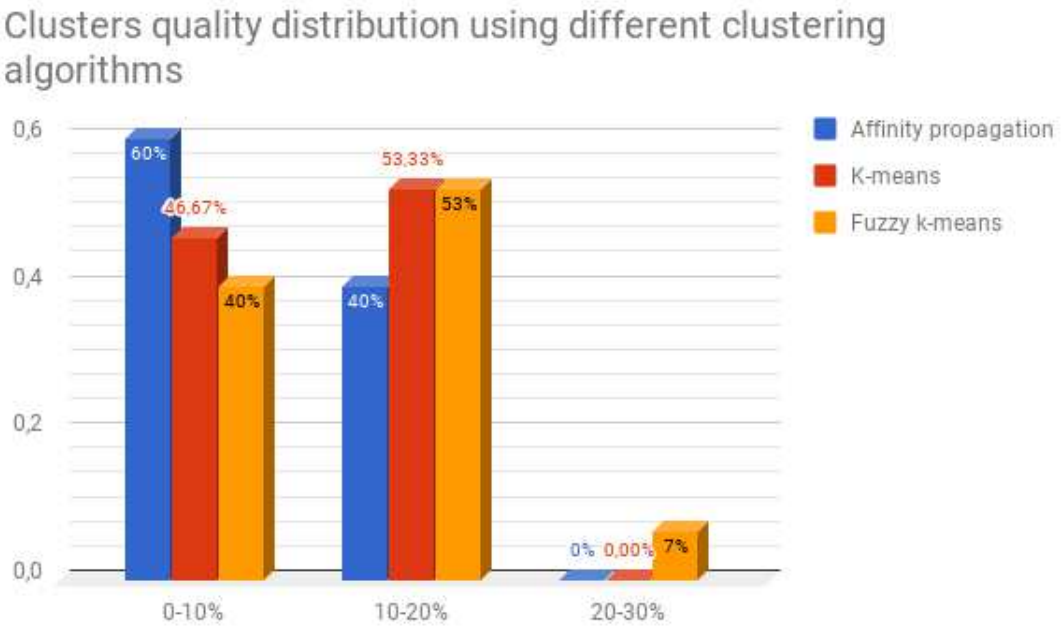


Figure 5.3: The graph shows the cluster quality distribution resulted by running affinity propagation, k-means and fuzzy k-means algorithms on the same dataset (for clusters quality we mean the cluster quality measured by using internal criterion). The X axis represents the quality of the clusters, Y axis corresponds instead to the percentage of the number of the cluster generated. Since the cluster generated has more or less the same size distribution the quality of the algorithm is comparable.

Meetup and Meetup-older, instead, have proportionally the same data distribution. However, since the Meetup-older dataset is composed by users coming from a common group (meaning that all the users have at least a certain number of preferences in common), our initial guess was that, by running clustering algorithms on this dataset, we would have obtained higher evaluations.

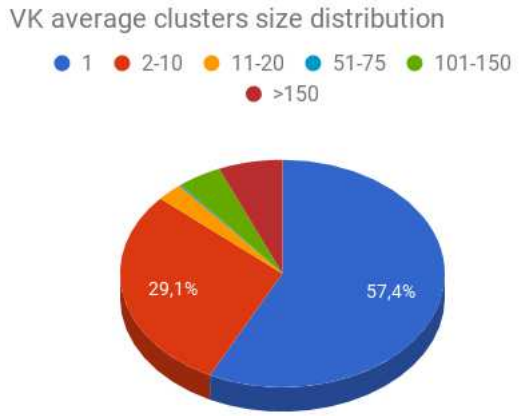
In order to validate our guess, we ran every framework clustering algorithm implemented on the three dataset available. The results reported were generate by averaging, for each dataset, the results produced by the clustering algorithms. The graphs in figure 5.1 confirmed our guess respect to cluster size distribution. As expected, the executions made on the VK dataset produced a lot of small clusters. Almost the 90 % of the cluster generated contains in-fact less than 10 users. Size distributions of the cluster obtained through the executions of the algorithms on Meetup and Meetup-older are instead similar. However it could be noted that the execution of the algorithms on the Meetup-older produce clusters whose size distribution present a slight decrease of the number of small clusters opposed to an increase of the number of clusters with size between 25 and 50 users.

The clusters quality reported by these tests confirm the correlation between the size of the clusters and their quality. In fact, by analyzing the graph 5.5, the tests performed on VK seems to have produced higher quality clusters. However, by taking into account also their size we could say that, the dataset on which were achieved more satisfactory results by evaluating clusters using internal criterion, is the Meetup-older dataset.

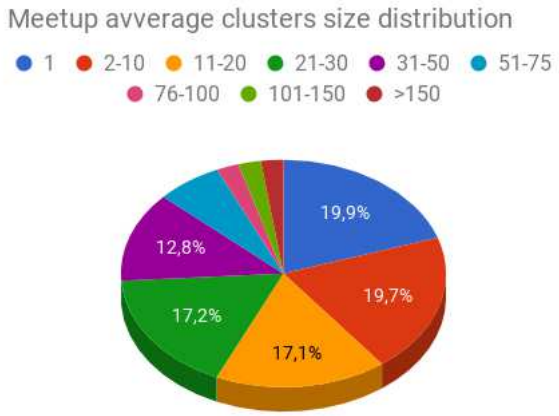
Precision, Recall F measure and comparison calculated on cluster users and cluster tags generated by these tests confirmed our initial assumptions again, by reporting highest evaluation values on the execution performed using the Meetep-older dataset 5.11.

5.6 Groups recommender systems tests

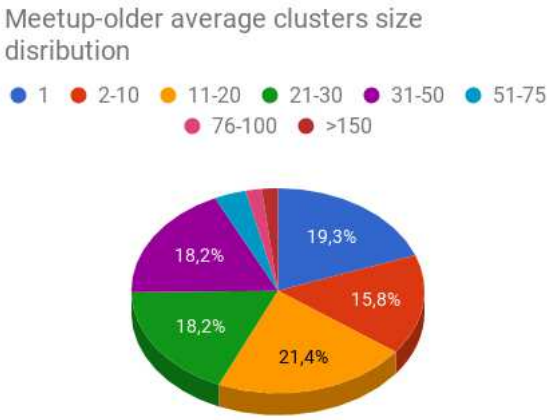
The last test set executed on the framework was focused on the evaluation of the recommender systems implemented (user-based, item-based and SVD mahout collaborative filtering Apache Mahout recommender systems). First, we executed some tests in order to register the behavior of the systems



(a)



(b)



(c)

Figure 5.4: Size distributions produced by averaging, for every dataset, the test performed by using the three clustering algorithm implemented in the framework.

Table 5.9: This table reports the evaluation made by using external criterion of the clusters generated by the three algorithms implemented on the top of the framework using the Meetup-older database. Note that the size distribution of the cluster generated by the clustering algorithms are more or less the same, thus making it possible the comparison of their quality. In particular clusters users' correctness percentage describe the percentage of users of the cluster generated which are also present in a ground truth group. The class tags' correctness percentage instead describe the percentage of the tags of the cluster generated which are also present in a ground truth group. More details about the formulas used are present in the section 4.4.

Algorithm	**Cluster users' correctness**			**Cluster tags' correctness**		
	AVG	**MIN**	**MAX**	**AVG**	**MIN**	**MAX**
Affinity propagation	56,45%	79,17%	13,64%	95%	99,36%	89,44%
K-means	52,28%	85,71%	24%	94,20%	99,76%	88,78%
Fuzzy K-means	38,50%	100%	12,50%	60%	100%	49,92%

Table 5.10: This table reports the evaluation made by using external criterion of the clusters generated by the three algorithms implemented on the top of the framework using the Meetup-older database. Note that the size distribution of the cluster generated by the clustering algorithms are more or less the same, thus making it possible to compare their quality. In particular this table refer to clusters evaluation made through users and tags pairwise comparison. More details about the formulas used are present in the section 4.4.

Algorithm	**Pairwise users comparison**			**Pairwise tags comparison**		
	Precision	**Recall**	**F1-measure**	**Precision**	**Recall**	**F1-Measure**
Affinity propagation	62,04%	3,82%	7,21%	99,38%	17,22%	29,35%
K-means	62,24%	4,13%	7,76%	99,22%	15,28%	26,49%
Fuzzy K-means	19,50%	0,01%	0,19%	80,02%	0,70%	1,39%

in our particular domain. Other tests were then executed in order to verify if the quality measures of the recommendations defined in the framework could be used as recommender system evaluation. Moreover, since the framework is born in order to help developer in their algorithms choice, we wanted also to verify if the comparison of the quality measures of the systems, achieved by different systems executions, could help framework users to draw some conclusions helpful for their final recommender system choice.

The recommender systems in the framework are used in order to recommend to users the groups generated by the framework clustering algorithms.

An important observation before introducing the results obtained in the tests is that, since the evaluation of the recommender systems is the average of the recommendation quality measure of the recommendations generated by the system (the rank) 4.6, the quality value of the recommendations is strictly correlated to the clusters generated previously by the framework clustering algorithms. Hence, it was not possible for us to evaluate a recommender system individually. This means that, during these tests we were not able to say if a recommender systems works good or not, but just which one outperforms the others in particular domain's conditions.

5.6.1 Similarity measures comparison

First recommender systems tests were focused on the recommendations quality produced by using different similarity measures. In the Apache Mahout recommender systems implemented, similarity measures are used in order to get the users(for user based collaborative filtering) or items (for item based AND SVD collaborative filtering) nearest neighbor on which the recommendations will be based for a certain user.

During this test set we used the following similarity measures:

- Euclidean distance similarity;
- Pearson correlation similarity;

Table 5.11: This table reports the pairwise evaluation using external criterion of the clusters generated by the three algorithms implemented. Each row of the table reports the averages of the values achieved by performing k-means, fuzzy k-means and affinity propagation algorithm executed on a certain dataset.

Database	Pairwise users comparison			Pairwise tags comparison		
	Precision	Recall	F1-measure	Precision	Recall	F1-Measure
VK	10,13%	0,81%	1,50%	85,43%	1,35%	2,66%
Meetup	14,22%	0,38%	0,75%	95,23%	1,99%	3,90%
Meetup-older	54,76%	5,84%	10,30%	99,01%	16,19%	27,82%

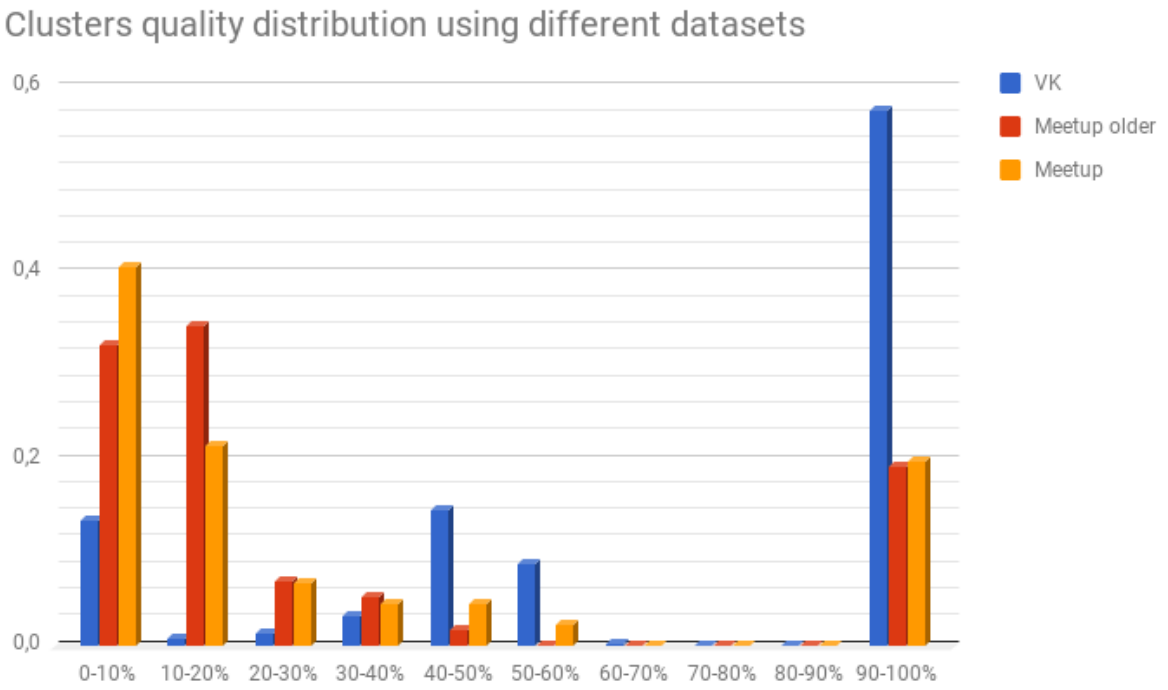


Figure 5.5: The graph shows the clusters quality distribution produced by averaging, for every dataset, the test performed by using the three clustering algorithm implemented in the framework. For cluster quality we mean the cluster quality measured by using internal criterion. The X axis represents the quality of the clusters, Y axis corresponds instead to the percentage of the number of the clusters generated.

- Log likelihood similarity;
- Cosine similarity;

The results achieved are summarized in the table 5.12. These results show the inability for all the recommender systems to perform acceptable results using cosine similarity and person correlation measures. Besides obtaining an average rank of 0, recommender systems that use cosine and Pearson correlation similarity measures are struggling to produce recommendations. In fact, the average number of total recommendations generated using these similarity measures is 10 for more than 300 users.

Euclidean distance and log likelihood similarity instead produced exactly the same results. Furthermore the number of recommendations produced by the recommender systems using these measures matches the number of recommendations required in input to the user by the system.

5.6.2 Dataset comparison

Other tests were executed in order to compare the results of the recommendations obtained using different datasets. These tests were performed in order to verify our initial assumption, that is, the

Table 5.12: Results achieved by running the recommender systems using different similarity measures. The values reported were calculated by averaging the recommender system evaluations 4.6 obtained by executing the recommender systems on the VK, Meetup and Meetup-older databases.

Similarity	User based	Item based	SVD
Euclidean distance	0,065	0,060	0,044
Pearson correlation	0	0	0
Log likelihood	0,065	0,060	0,044
Cosine	0	0	0

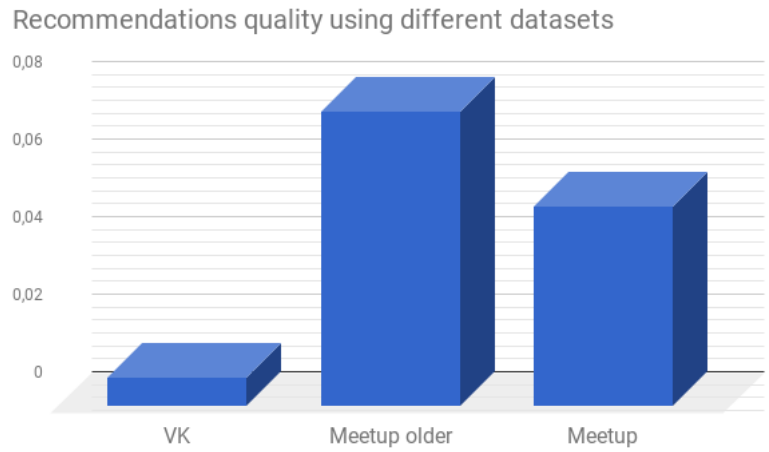


Figure 5.6: The graph shows the recommendations quality obtained by averaging respect to each dataset of all the recommendations qualities obtained

higher the quality of the clusters generated by the clustering algorithms, the greater the evaluation of the recommendation systems will be. The results reported in the graph 5.6 confirm this assumption. In fact, the best evaluation for both clustering algorithm and recommender systems have been reached by executing the systems by using the Meetup-older database. The worst evaluations were obtained instead, by using the VK dataset.

5.6.3 Recommender systems comparison

Last tests were executed in order to highlight the difference of the results achieved by the execution of different recommender systems. The results of these tests have been obtained by averaging, for each recommender system, the quality value achieved in all the tests executed. The graph 5.7 shows that, the quality of the recommendations produced by the three systems is more or less the same. This result can be explained by the slight difference in implementation of the recommender systems proposed by Mahout. However the user-based collaborative filtering recommender systems seems to outperform the others.

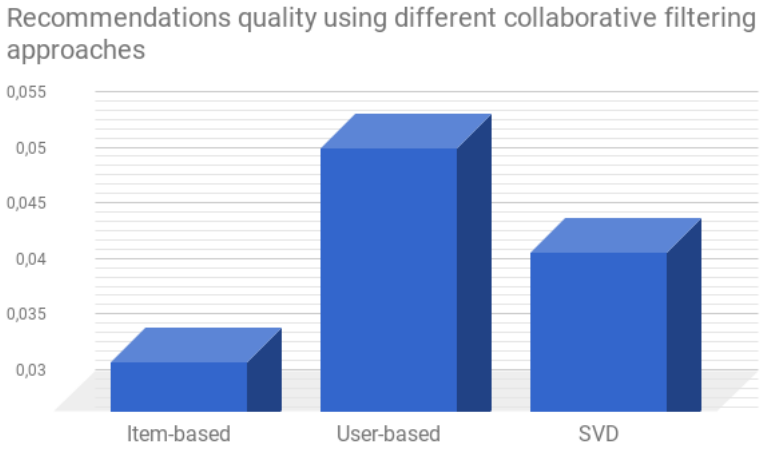


Figure 5.7: The graph shows the recommendations quality obtained by averaging respect to each recommender system approach used of all the recommendations qualities obtained

6 Conclusions and future works

The framework structure was designed in order to provide to developers the possibility to easily add new functionalities. This structure should lead the project to greater development and implementation of clustering algorithms and recommendation systems.

We have implemented an extensible Java framework with the aim of giving to its users the opportunity to implement, evaluate and compare different clustering algorithms and recommender systems. The frameworks were specifically designed in order to execute data mining algorithms on users' data, however, by implementing small changes, the framework is usable also in other domains.

The tests performed on the algorithms implemented have highlighted the ability of the framework to produce comparable quality measures and algorithms evaluations. Moreover the framework structure gives us the possibility to execute, evaluate and compare the available algorithms, also by quickly and easily changing their execution parameters. Thanks to the framework, in fact, we were able to draw some conclusions about different algorithms behavior and their results in our research domain. As demonstrated in the test chapter, we could compare and draw some conclusion about the results obtained by:

- Running different clustering algorithms using the same settings;
- Running the same clustering algorithm using different distance measures;
- Running the same clustering algorithm using different setting of the algorithms input attributes;
- Running the same clustering algorithm using different datasets;
- Running different recommender systems using the same settings;
- Running the same recommender system using different similarity measures;
- Running the same recommender system using different setting of the systems input attributes;
- Running the same recommender system using different datasets.

Since the analysis of the results obtained in the testing part of the project led us to some conclusions about the algorithms behaviors, we can conclude by saying that the quality measures and evaluations implemented in the frameworks are valid, usable and meaningful for the comparison of different clustering algorithms and recommender systems.

The tests results reported in this paper were focused on the demonstration of the comparability of the algorithms evaluations. Then, for the moment, we leave to the researcher choice the interpretation of the results and future choice of the best algorithms in our research domain. However, eventually, if we could reconcile a group of measurements expected for the algorithms in our research domain, we could define them as parameters for choosing the best algorithms.

Major future works of the framework will consist in the implementation of other quality measures and comparable evaluations.

In particular we would like to improve the evaluation of the clustering algorithms by providing, to the framework users, the possibility to project the ground truth groups in a feature space. This would pave the way to the implementation of more evaluation measures based on the comparison between the cluster generated and the ground truth groups such as the distances between the ground truth centroids and the cluster generated centroids, the percentage of overlaps between the ground truth groups and the clusters generated etc.

Another important feature to add is the possibility to evaluate the recommender systems using the ground truth data. Since recommender systems, are used by the framework in order to recommend to users the groups generated by the frameworks and not existing groups, in order to evaluate the recommendations by using ground truth data, there is the need to define a measurement of equality between the clusters generated and the ground truth groups. Or better, define when a group generated can be consider equal to a ground truth group.

In this way will be possible to treat the clusters generated as ground truth data. Once defined the equality measure between groups, it will be possible to calculate recommender systems evaluations such as RMSE, precision, recall and accuracy.

An example of measurement of equality could be to define an equality threshold for the percentage of common users between a cluster generated and the ground truth group. If the percentage of common users is higher than the threshold the two clusters are treated as equal.

Other future work can include the implementation of new clustering algorithms and recommender systems or the improvement of the already existing ones. Possible improvement could consist in the addition of new distance and similarity measures, or, the possibility to take into consideration more parameters in addition to tags during clustering.

Bibliography

[1] Apache mahout. http://mahout.apache.org/. Accessed: 2018-03-13.

[2] Weka 3: Data mining software in java. https://www.cs.waikato.ac.nz/ml/weka/. Accessed: 2018-03-13.

[3] Saleema Amershi, James Fogarty, and Daniel Weld. Regroup: Interactive machine learning for on-demand group creation in social networks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 21–30. ACM, 2012.

[4] Apache Mahout documentation. Fuzzy k-means. https://mahout.apache.org/users/clustering/fuzzy-k-means.html. Accessed: 2018-03-13.

[5] Motahhare Eslami, Amirhossein Aleyasen, Roshanak Zilouchian Moghaddam, and Karrie Karahalios. Friend grouping algorithms for online social networks: Preference, bias, and implications. In *International Conference on Social Informatics*, pages 34–49. Springer, 2014.

[6] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *science*, 315(5814):972–976, 2007.

[7] SongJie Gong, HongWu Ye, and YaE Dai. Combining singular value decomposition and item-based recommender in collaborative filtering. In *Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on*, pages 769–772. IEEE, 2009.

[8] IBM. 10 key marketing trends for 2017. https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN, 2017. Accessed: 2018-03-13.

[9] Lovro Ilijašić. Java affinity propagation library // parallelized. https://github.com/lovro-i/apro. Accessed: 2018-03-13.

[10] FO Isinkaye, YO Folajimi, and BA Ojokoh. Recommendation systems: Principles, methods and evaluation. *Egyptian Informatics Journal*, 16(3):261–273, 2015.

[11] Anthony Jameson and Barry Smyth. Recommendation to groups. In *The adaptive web*, pages 596–627. Springer, 2007.

[12] Marina Meilă. Comparing clusterings—an information based distance. *Journal of multivariate analysis*, 98(5):873–895, 2007.

[13] MongoDB. Nosql database explained. https://www.mongodb.com/nosql-explained. Accessed: 2018-03-13.

[14] OrientDB. Why a multi-model database? https://orientdb.com/multi-model-database/. Accessed: 2018-03-13.

[15] Mark O'connor, Dan Cosley, Joseph A Konstan, and John Riedl. Polylens: a recommender system for groups of users. In *ECSCW 2001*, pages 199–218. Springer, 2001.

[16] Dan Pelleg, Andrew W Moore, et al. X-means: Extending k-means with efficient estimation of the number of clusters. In *Icml*, volume 1, pages 727–734, 2000.

[17] Samuel D Pimentel. Choosing a clustering: an a posteriori method for social networks. *Journal of Social Structure*, 15:1, 2014.

[18] Wolfgang Pree. Framework development and reuse support. *Burnett et al.[5]*, 1995.

[19] European Unions Horizon 2020 Research and Innovation Programmer. Acanto - a cyberphysical social network using robot friends. `http://www.ict-acanto.eu/`. Accessed: 2018-03-13.

[20] Julien Soler, Fabien Tencé, Laurent Gaubert, and Cédric Buche. Data clustering and similarity. In *FLAIRS Conference*, 2013.

[21] Harald Steck. Evaluation of recommendations: rating-prediction and ranking. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 213–220. ACM, 2013.

[22] Poornapragna Malamandi Suresh. Database as a graph store, 2016.

[23] Electronic Statistics Textbook. Finding the right number of clusters in k-means and em clustering: v-fold cross-validation. Technical report, Technical report, 2010. 6, 2010.

[24] Guadalupe J Torres, Ram B Basnet, Andrew H Sung, Srinivas Mukkamala, and Bernardete M Ribeiro. A similarity measure for clustering and its applications. *Int J Electr Comput Syst Eng*, 3(3):164–170, 2009.

[25] TutorialsPoint. Java- documentation comments. `https://www.tutorialspoint.com/java/java_documentation.htm`. Accessed: 2018-03-13.

[26] Cambridge university. Evaluation of clustering. `https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html`, 2008. Accessed: 2018-03-13.

[27] Wikipedia. Cluster analysis. `https://en.wikipedia.org/wiki/Cluster_analysis`. Accessed: 2018-03-13.

[28] Wikipedia. Confusion matrix. `https://en.wikipedia.org/wiki/Confusion_matrix`. Accessed: 2018-03-13.

[29] Xiwang Yang, Harald Steck, and Yong Liu. Circle-based recommendation in online social networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1267–1275. ACM, 2012.