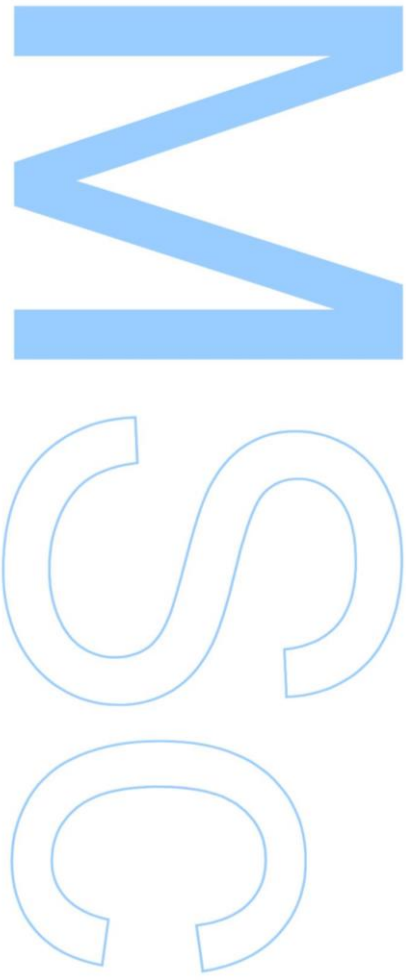
# Online ensembles of local recommendation models



Lúcia Raquel Brandão Moreira
Mestrado em Ciência de Dados (Data Science)
Departamento de Ciências e de Computadores
2021

**Orientador**
João Vinagre, Professor Auxiliar Convidado, FCUP

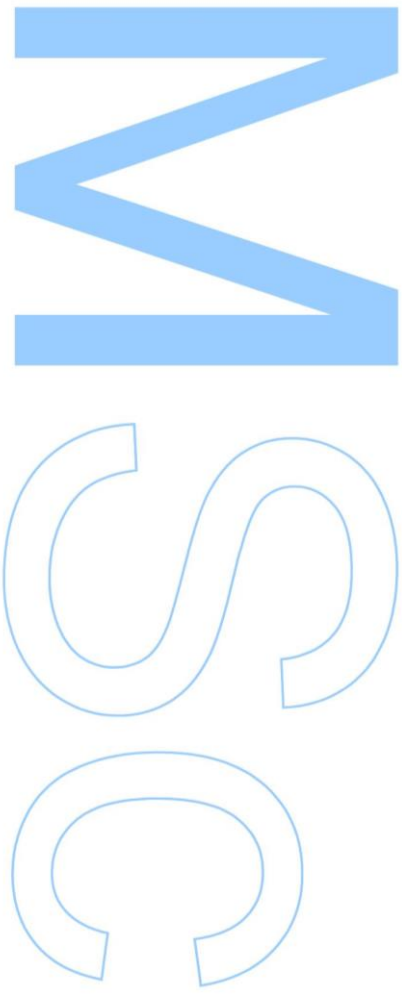
**Coorientador**
Alípio Jorge, Professor Associado, FCUP

U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____



## Abstract

The major goal of a recommender system is to get accurate predictions for the active user. However, recommendation problems may have localized phenomena that are hard to combine in a single global model what may lead to poor accuracy. On the other hand, local models are able to better capture local phenomena among subsets of users and combining such local models could increase recommender predictions. In this thesis, it is proposed a user clustering approach connected to an incremental ensemble recommender learner in order to evaluate the enhancement on recommendation accuracy by combining improved local models.

For that, batch clustering is considered for user segmentation using the user-item positive feedback matrix. A wide range of clustering methods is surveyed; a density-based and a non-negative matrix factorization clustering algorithms providing fuzzy clustering are the most suitable for this application among the few readily available clustering methods providing soft membership probability. Fuzzy clustering offers access to the probabilities of a user belong to different clusters once users might have simultaneous affinities for a wide range of items. Additionally, an ensemble algorithm based on bagging is used where the bootstrap nodes should contain controllable overlapping examples among the different nodes.

Five datasets suitable to recommendation problems are analysed for clustering and then used for performing a recommendation analysis. The concatenation of the user clustering with the ensemble recommender model presents advantages in terms of average recall@20 under some circumstances for 4 of the 5 datasets considered in relation to the state-of-art recommender model alternatives. For the ML1M dataset, using as threshold 40 % of the highest soft cluster memberships (related to the overlapping degree) improves accuracy when compared to the user-based ensemble recommender for all considered nodes. For a 50 % threshold, the same improved accuracy is observed up to 14 nodes. For the LFM-50U dataset, recall@20 for the local ensemble is systematically higher than all the other ensemble algorithms except under some circumstances at lower and higher number of nodes, for all thresholds. Regarding the YHM-6UK dataset, at the lowest number of nodes considered the local ensemble outperforms all the other ensemble models for all thresholds. Finally, for the PLC-playlist dataset local ensemble models perform better at lower numbers of bootstrap nodes (4 and 6) except for the 40 % threshold.

Keywords: recommendation systems, incremental algorithms, clustering, ensemble models, bagging, local models

## Resumo

O objetivo principal de um sistema de recomendação é obter previsões precisas para um determinado utilizador. No entanto, os problemas de recomendação têm vários fenómenos localizados que são difíceis de combinar em um único modelo global. Os modelos locais são capazes de capturar tais fenómenos entre vários subconjuntos de utilizadores e ao combinar estes tais modelos pode-se melhorar a previsão. Nesta tese é proposto o agrupamento de utilizadores seguido de um sistema incremental de recomendação composto, com a finalidade de avaliar a melhoria da previsão de recomendação através da combinação de modelos locais.

Para isso, um agrupamento em *batch* foi considerado para a segmentação dos utilizadores usando a matriz de *feedback* positivo utilizador-item. Uma ampla gama de métodos de agrupamento foi considerada; um algoritmo de agrupamento baseado em densidades e o método de factorização de matrizes não negativas com capacidade de efetuar uma abordagem difusa (*fuzzy*) mostraram-se os mais adequados para a presente aplicação entre os poucos métodos difusos prontos a utilizar disponíveis. O agrupamento com uma abordagem difusa permite o acesso às probabilidades de um utilizador poder pertencer a vários subgrupos porque este pode ter afinidades simultâneas para uma ampla gama de itens. Além disso, um algoritmo de recomendação composto baseado em *bagging* foi utilizado e os nós devem conter exemplos sobrepostos por isso este grau de sobreposição também deverá ser controlável.

Cinco conjuntos de dados, adequados a sistemas de recomendação, foram considerados para a análise de agrupamento seguida de um modelo de recomendação. A concatenação da partição dos utilizadores com o modelo de recomendação composto apresentou vantagens em termos de recall@20 dentro de algumas circunstâncias para 4 dos 5 conjuntos de dados em relação ao estado da arte. Para o conjunto de dados ML1M e com 40 % das maiores probabilidades de pertencer a um grupo (grau de sobreposição) há uma melhoria da previsão quando comparado com o modelo composto com uma distribuição dos utilizadores menos criteriosa. Para um valor de 50 %, esta mesma melhoria é observada até 14 nós. Para o conjunto LFM-50U, o recall@20 para o modelo local composto é sempre maior do que os outros modelos compostos exceto para os maiores e menores números de nós utilizados. Relativamente ao conjunto YHM-6UK, para um número de nós igual a 4 o modelo local composto tem um melhor desempenho do que todos os outros modelos compostos. Finalmente para o conjunto PLC-playlist, o modelo composto local é globalmente melhor para um múmero de nós igual a 4 e 6.

*Palavras-chave*: sistemas de recomendação, algoritmos incrementais, algoritmos de partição, sistemas compostos, *bagging*, modelos locais

# Index

## Caption to figures

## Caption to tables

# 1. Introduction

More than 2 decades ago, the Netflix prize[1] created a buzz around recommender systems. The contest aimed to improve its recommender system by releasing a training dataset with 100 million ratings for half million anonymous customers and their respective ratings on more than 17 000 movies. Participating teams submitted the predicted ratings for a test set with 3 million ratings. Netflix calculated the root-mean-square error (RMSE) based on the hold-out truth. The first team that could improve the Netflix algorithm's RMSE performance by 10 percent or more would win a $1 million prize. Until that point, publicly available datasets for recommender systems, and particularly collaborative filtering research, were orders of magnitude smaller [1]. The release of such dataset and the competition's appeal encouraged a surge of activity in the field. According to the contest website by that time, more than 48 000 teams from 182 different countries had downloaded the data [1].

Other sources of information for such recommender systems may come from users generating events like the purchase of a product, rating a product, creating a bookmark or clicking on a specific item. Independently of the area of application or the type of information used, it is a major goal to get accurate predictions. For instance, for a subscription service like Netflix, good recommendations are key to customer loyalty while for online stores better recommendations directly increase the revenue [2].

## 1.1 Motivation

This thesis involves the use of collaborative filtering on recommender systems, particularly addressing an ensemble of local models such as bagging to improve prediction accuracy. Recommendation problems are known to have several local phenomena that are hard to combine in a single global model. However, local sub-models are able to capture these local phenomena among user subsets.

Yet today, most recommendation techniques are executed in a batch mode using the whole data stored in a database. However, new users and items are frequently entering the recommendation system and if the new input data is not used to update the model, the original recommendation model built in the batch mode may become obsolete, causing poor recommendations. Because of this, intensive research has been made for having fully online incremental recommender systems [3-9].

[1] https://www.netflixprize.com/

In [4], Vinagre et al. present a fast incremental collaborative filtering algorithm well adjusted for online learning that is also highly competitive in terms of accuracy and at least one order of magnitude faster than its alternatives. The same team extended such model to an online bagging approach, pioneering on incremental ensemble learning for recommendation [8]. It was also observed that attributing user-item pairs according to the user to the same bootstrap nodes, i.e. splitting data across bootstrap nodes by user, yielded significantly better results when compared to a fully random user node allocation - usual in bootstrap ensemble models - due to a possible completeness of user profiles in all bootstrap nodes. This approach has pointed that a clustering approach applied to users and previous to recommendation could improve further the incremental ensemble results. Previous works have also shown that local based models could hold promises for improved recommender systems [10-12]. In this thesis we propose a user clustering approach integrated with an online ensemble recommender learner in order to improve recommendation prediction accuracy by combining local models.

## 1.2 Research goals

The target of this thesis is to build a local-based bagged recommendation algorithm suitable for data streams, where user feedback is continuously generated and processed. For that, any end-user should only input the number of bootstrap nodes required - by interplaying between higher accuracy and recommendation time to deliver an online recommendation to the active customer- while the recommender algorithm performs incremental user clustering fully integrated with the incremental local ensemble model.

The focus of this project is to use bagging for the ensemble algorithm, to learn and combine local models for recommendation problems and the main goals are:

- To review the state-of-the-art on ensemble recommender systems and local models for recommendation;
- To develop an ensemble method that combines local models, and compare it with alternatives;
- To carry out empirical studies on synthetic and real-world datasets.

## 1.3 Contributions to state-of-the-art advancements

The innovations from this research study are:

- Contribute to a novel online ensemble learning technique for recommendation;
- Contribute on the development of an algorithm able to capture and adapt to local phenomena;

- Provide the first guesses for the most probable range on cluster/bootstrap nodes when using several real world datasets and previous relevant knowledge that would provide improved accuracies on a fully online local-based bagged recommender algorithm.
- Suggest the most suitable clustering type of algorithms to be integrated in the local-based recommender model.

## 1.4 Thesis outline

This thesis is divided into 4 chapters besides this one. **Chapter 2** introduces the reader to the theme by revolving around the background knowledge and the state-of-the-art approaches previously reported in the open literature. **Chapter 3** details clustering algorithms, methods and datasets used during this research study. **Chapter 4** addresses the results obtained and discusses the main outcomes from those results. Finally, **Chapter 5** involves the main conclusions and prospects possible future work.

# 2. Background and state-of-the-art

This chapter is divided into two parts: one addresses the background subjects related to this work while the other reviews the state-of-the-art on ensemble recommender systems and local models for recommendation.

## 2.1.1 Background

### 2.1.1.1 Recommender systems

Data available in the Web is unique in many ways, e.g.: it is huge, distributed and uncoordinated by nature – being rich of varied data types -, it has enabled a great diversity of applications - such as e-commerce, user collaboration, and social network analysis - and etc. According to Aggarwal, the two primary Web data types used by mining algorithms for returning insights are [13]:

1. *Web content information* corresponding to Web documents and the links created by users where understandings can be mined from: a) *Document data* extracted from the pages on the Web and b) *Linkage data* where the Web is viewed as a massive graph used for either searching the Web itself or for defining similarities between nodes;
2. *Web usage data* corresponding to patterns of user activity such as: a) *Web transactions, ratings, and user feedback* - the buying behaviour and/or ratings are leveraged to infer preferences of users and feedback in the form of textual reviews that can be denoted as opinions. b) *Web logs* where browsing behaviour can be used to make inferences about user activity.

This way and in synchronisation with the respective data type, Aggarwal states that data mining final applications can be either content- or usage-centric [13]:

1. *Content-centric applications* are: a) *data mining applications* used by Web portals for organizing pages; b) *Web crawling and resource discovery* used for knowledge discovery and its proper storage in order to make inferences; c) *Web search* used to discover highly relevant documents in response to a user-specified set of keywords; d) *Web linkage mining* for the discovery of logical representations of the Web structure such as social and information networks.
2. *Usage-centric applications* leverages user activity to make inferences, such as: a) ***Recommender systems*** where preference information is used to make recommendations to other like-minded users and b) *Web log analysis* is useful for Web site owners to determine

relevant patterns of user browsing such as anomalous patterns, user interests, and optimal Web site design.

According to that, recommender systems algorithms apply statistical and knowledge discovery techniques to the problem of making product recommendations based on previously recorded data. Such recorded data are usually user–item pairs with utility values associated with them. Thus, for $n$ users and $d$ items, this results in an $n \times d$ matrix of utility values, usually mentioned as utility matrix [13]. The utility value for a user-item pair can correspond to either the buying behaviour or the ratings of the user for the item and thus impact the choice of the recommendation algorithm [13]:

1. *Positive preferences only*: the utility matrix contains only positive preferences (binary matrix), such as specification of a "like" option on a social networking site, the browsing of an item at an online site, or the buying of a specified quantity of an item.
2. *Positive and negative preferences (ratings):* the ratings-based utility provides a way for users to express negative preferences for items.

Besides the utility matrix, recommendations can be improved with content from the user or item representations or by the determination of similar groups of items and users, accordingly [13]:

1. *Content-based recommendations* leverage feature-based descriptions such as the text of the item description or explicitly specified interests in user profile. Otherwise, user profile can be inferred from their buying or browsing behaviour.
2. ***Collaborative filtering*** leverages user preferences in the form of ratings or positive preferences for the benefit of all users. The key assumption is that users who showed common interests on some items in the past will tend to have similar preferences towards other items in the future. There are two types of collaborative filtering: *user-based algorithms* and *item-based algorithms*. User-based algorithms predict a user's preference for an item based on past behaviours of similar users towards that item. On the other hand, item-based algorithms predict a user's preference for an item based on past behaviours of that user toward similar items. Specifically, the utility matrix is used to determine either relevant users for specific items, or relevant items for specific users in the recommendation process. Thus, a key intermediate step is the determination of similar groups of items and users. The patterns in these peer groups provide the collaborative knowledge needed in the recommendation process. Collaborative filtering approach can be built using statistical techniques (e.g. user-to-user correlation), data mining techniques (e.g. clustering, classification, association rules) or linear algebra techniques (e.g. matrix factorization) [14].

#### 2.1.1.2 Collaborative filtering

Collaborative filtering can be regarded as a missing-value estimation or matrix completion problem, in which an incomplete $n \times d$ utility matrix is identified, and trying to "guess" the missing values is the main goal [13]. Collaborative filtering systems can be generally classified according to two main approaches [13, 14]:

a) in the *memory-based* approach a subset of users similar to the active user is chosen and a combination of their ratings or positive preferences is used to predict unseen items for the active user. This approach makes predictions based on the entire history (stored in memory) of users and may also be called neighbourhood-based, similarity-based or correlation-based;

b) the m*odel-based* approach fits a model based on splitting ratings or positive preferences (called training dataset), tests its performance on another split of data (called test dataset), and once the model performs well on the test dataset, it is used for making recommendations by extracting the best "guesses". Model-based algorithms map users and items for instance to latent vectors in the same space and predict users' preferences for items based on the inner products of those vectors. Most latent factor models apply ***matrix factorization*** to discover the hidden features between items and users from the raw data of customers' ratings (or positive preferences) by factorizing the user-item (utility) matrix into lower rank user factor and item factor matrices, thus representing both the users and the items in a common latent space. In a nutshell, matrix factorization finds a new basis in the original data space where the dataset is projected. Then this new dataset is simply the linear projection of the original data onto the new basis directions [15].

#### 2.1.1.3 Matrix factorization for recommendation

Model-based techniques can take into consideration multiple attributes to group similar users together to build a model that may comprise classification, dimensionality reduction, graph-theoretic and other techniques. Building and updating a model is a time-consuming process but model-based techniques have lower computational complexity as compared to memory-based systems [14].

Matrix factorization is probably the most used model-based collaborative filtering technique greatly popularized also by the Netflix prize [16]. Similar to singular value decomposition (SVD) from dimensionality reduction linear algebra, matrix factorization decomposes the user-item record matrix into the multiplication of two lower-rank matrices, in

which the latent factors of users and items can be stored. Thus, matrix factorization can discover items with similar contents as well as other implicit features.

A *n* x *d* matrix **R** is decomposed into the user latent factor matrix **P** and the item latent factor matrix **Q**. The factor vectors include the attributes of items - such as music types, movie genre, tags in the twitter messages - and the user's attributes - such as a particular preference for a certain movie genre. In this model, some ratings (or positive feedbacks) are known while others are not. Based on known ratings, the goal is to factorize **R**, under the constraint that $\widehat{\mathbf{R}}$ = **PQ** should approximate **R** well. After **P** and **Q** are randomly initialized, in general, the minimization of a $L_2$-regularized cost function is achieved by Stochastic Gradient Descent (SGD) or Alternating Least Squares (ALS) [17]. A more detailed description on the minimization algorithm can be found e.g. on [5].

#### 2.1.1.4 Ensemble methods on recommender systems

The Netflix contest winners and other top teams have used ensemble learning to increase accuracy prediction on recommender systems [16]. Bagging learning algorithms combine multiple learned base models with the aim of improving generalization performance by reducing variance, which is especially useful with unstable algorithms that are very sensitive to small changes in the data [8].

Particularly, bagging is a quite simple ensemble method that is based on training some amount of copies of a model on slightly different training sets. Each training set is produced by random sampling with replacement – bootstrapping -, i.e. some examples will not be present whereas some others will occur multiple times. Efron and Tibshirani [18] proved that there is 2/3 examples overlap between each training set. Training several models with slightly different versions of the data allows focusing on learning the main concepts—because they are probably present in most samples—while giving less importance to noisy phenomena that may distort learning [8]. However, one drawback of bagging and ensemble methods in general is their increased training time.

Bagging applications are mostly in batch mode for recommender applications, i.e., the entire training set is available at front and, and in some cases, only requires random access to the data. However, Oza and Russel [19] back in 2001 pioneered on a general purpose incremental ensemble algorithm suitable for data streams opening the doors for wide spread and quite fast implementation of online bagging. In their suitable for online learning bagging algorithm the number of copies of a particular example to be present in each of the training sets follows a simple Poisson(1) distribution for a high number of online arriving examples what is

very common in data streams. They observed that the difference between the accuracies of the batch and online ensemble algorithms was mainly related to the differences between the accuracies of the batch and online base model learning algorithms. The work of Vinagre et al. in 2018 [8] built on top of Oza and Russel's work established an online bagging application adapted to incremental recommender systems for the first time.

#### 2.1.1.5 Clustering methods

Clustering is an unsupervised learning problem and the goal is to partition a set of unlabelled objects into clusters where the objects falling into the same cluster have more similarities than others. Clustering methods can be divided into several categories, herein it will be addressed the clustering methods used in this work: Partitioning Clustering, Hierarchical Clustering, Density-based Clustering, and clustering via Non-Negative Matrix Factorization. An effective clustering algorithm performs clustering accurately, using limited input from the user while the only constrains should be the memory, processing and power limitations of the embedded system/processing unit.

Clustering algorithms can be yet divided in to hard (or crisp) and fuzzy (or soft) clustering methods. Fuzzy algorithms can handle imprecise information and un-sharpness in cluster boundaries. In recommender systems, such algorithm allows for gradualness of preference profiles and can effectively capture partial matching of users' preferences across multiple clusters. A user can belong to multiple clusters and the strength of belongingness is determined by the probability (membership) of belonging to a cluster. This is a particularly relevant feature in ensemble recommender systems using bagging in order to distribute the users for the different bootstrap nodes according to the probability of the user belonging to that node. On the other hand, on hard clustering, the boundary of clustering is rigid, and an object can only strictly belong to a single cluster.

Batch-mode clustering algorithms have been studied and employed as data analysis tools for decades now, but clustering large amounts of data takes a long time as new unlabelled datasets which will not fit in memory are now available. To perform batch clustering on such datasets, either sub-sampling is required to fit the data in memory or the time will be greatly affected by disk accesses making clustering an unattractive choice for data analysis. Data stream algorithms have more recently evolved to tackle continuous flow of very large data that occurs in real-time with quick response requirements where multiple access is almost impossible and storage is restricted as opposed to batch traditional forms of data - invariable and static.

2.1.1.5.1 Partitioning Clustering

Partitioning algorithms construct a partition of a set of objects into *k* clusters created from a distance-based objective function where the distance between the data point and a randomly selected centroid is minimized (e.g. the sum of squares distances to the centroid representative). One example on batch and crisp algorithm is *k-means* being the most widely used clustering algorithm [20]. By minimizing a squared error function only round-shape clusters are obtained so it cannot detect non-spherical clusters. The input parameter *k* is fixed and must be given in advance. Moreover, the intrinsic limitation to choose the initial random *k*-centroids affects the clustering results once randomly selected centroids may lead to inaccurate results and increased expense of training time.

*Fuzzy c-means* is a probabilistic fuzzy version of the crisp *k-means* and effectively partitions tractable size datasets but requires significantly more time to cluster large datasets and takes an exorbitant amount of time on data that do not fit in main memory. A streaming version of the algorithm has been developed to solve such issues by dividing the data set into chunks and clustering them separately and a weighed version of the objective function is used in the final clustering [21-24]. For each chunk of data, the algorithm finds the *k* centroids where the sum of distances from data points to their closest centroid is minimal. Only the *k* centroids (representing the clustering results) are retained, with the corresponding *k* cluster features. In the following iterations, the buffer is initialized with the *k*-centroids, found in previous iteration, weighted by the *k* cluster features, and new incoming data points from the stream. Despite these advances, the standard Euclidean distance used in fuzzy *c-means* may suffer from dimensionality effects even to a larger extent than in the crisp case [25].

2.1.1.5.2 Hierarchical Clustering

Hierarchical clustering produces a nested series of crisp partitions from batches of data in two modes: agglomerative and divisive. Agglomerative clustering uses a bottom-up approach where all data points are assumed to be different clusters and successively the most alike pair of clusters (according to a predefined distance measure) is merged (until only one cluster is obtained) to form a cluster hierarchy [14]. On the other hand, divisive clustering is a top-down approach where all data points are assumed to belong to one single cluster and subsequently split into smaller clusters until some stopping criteria is met [14]. The clusters are organized in a hierarchical manner, and the resulting hierarchical structure is known as dendrogram.

The advantage of hierarchical clustering is that unlike *k-means* specifying the number of clusters is not required. The desired number of clusters can be obtained by cutting the tree at a

selected level. However, it is difficult to interpret the number of clusters by the dendrogram and decide the level where to perform the cut in the tree (for finding the clusters) [14].

Stream versions of agglomerative hierarchical clustering algorithms are BIRCH and CluStream for hard clustering [20]. These are based on micro-clustering algorithms that divide the clustering process into two phases: the first phase is online and summarizes the data stream in local models (micro-clusters - a cluster feature that is a compact representation of a set of points) and the second phase (usually off-line) generates a global more refined cluster model from the micro-clusters (efficiently using only the summary statistics) usually based on batch *k-means* or other methods. BIRCH uses the cluster feature for the summarizing statistics together with a data compress system by building a hierarchical height-balanced tree data structure – each clustering feature corresponds to a cluster that is hierarchically organized in a tree [20]. The cluster feature has three components: the number of data objects and two n-dimensional arrays with the linear sum of the data objects and the sum of squared data objects, all the three components are used to calculate cluster feature mean (centroid) and its radius (or diameter). When a new object arrives, it descends the tree from the root to the leaves by choosing in each non-leaf node its closest cluster feature entry (closeness is defined by the Euclidean distance between new objects and the centroids of cluster feature entries in non-leaf nodes). Arriving at a leaf node, the closest entry is tested to verify whether the leaf node can absorb the new object (and the cluster feature vector is updated) or a new entry leaf node should be created (if so, at this point the new entry only contains this particular new object) [20]. Every leaf node contains a maximum number of entries and only can have a maximum user-defined diameter (or radius) that defines if a new object is absorbed or not by a cluster feature vector and determines the size of the tree - higher diameters lead to smaller trees [20]. In the off-line step only the leaf nodes are further clustered by e.g. *k-means*. Similar to BIRCH, CluStream algorithm is also divided in two components: one on-line and another off-line. Micro-clusters generated on-line are locally kept, having statistical information of data in the cluster feature but here temporal information is also added and only the most recent micro-clusters are kept in memory [20]. One drawback is that BIRCH and CluStream may present limitations in the presence of non-spherical clusters [20]. Also, because of Euclidean distance used may suffer from dimensionality effects as well.

ODAC that stands for Online Divisive-Agglomerative Clustering is a crisp hierarchical procedure for incrementally cluster streaming time series by monitoring composite correlations, i.e., groups of highly correlated pairs of streams among multiple time series [26]. It constructs a tree-like hierarchy of clusters of streams, using a top-down strategy based on the correlation

between streams. The leaves are the resulting clusters with each leaf grouping a set of time series. This algorithm addresses the issue that most of the work in incremental clustering of data streams has been concentrated on example clustering rather than variable clustering [26]. Clustering variables (e.g. time series) might be a very useful tool for some applications, such as sensor networks, social networks, electrical power demand, stock market, etc. The goal of a clustering system for multiple time series is to find a partition of streams, where streams in the same cluster tend to be more alike than streams in different clusters.

2.1.1.5.3 Density-based Clustering

Density-based algorithms are based on connectivity between regions and density functions, i.e. clusters are highly dense regions separated by sparse regions. This type of algorithms finds clusters of arbitrary shapes without the need to specify previously the number of clusters, pioneered by DBSCAN [20]. Density-based approaches are also efficient algorithms to remove noise points/anomaly from the dataset. However, one drawback is that it requires determining pairwise distances between all data points, leading to a higher time complexity thus limiting application on larger datasets [9].

DBSCAN is a crisp batch algorithm that creates clusters based on optimizing the concept of density to determine the most connected dense areas. Dense areas are defined by the number of data points (or instances) lying inside a pre-specified radius. Given the definition of data point neighbourhood (those points at distance smaller than the specified radius threshold when a distance function is given), a data point is considered to be in a dense area if its neighbourhood contains at least a minimal number of points, and the other data points in its neighbourhood are considered to be in the same cluster [27]. According to that, data points are classified as core points and border points. Core points are those data points with at least a minimum number of neighbourhood points (prior specified) with a distance less than or equal to the user pre-specified radius while borders points (outliers) do not reach simultaneously both minimum values. One drawback and extended to all density-based algorithm is that the merge of clusters which are connected via a very narrow density region may occur [14]. Besides, there is no general method to get the best parameters which are adaptive to different application domains and choosing the correct values of parameters is based on a hit-and-trial strategy [14]. DBSCAN also is not able to detect clusters in data of varying density [27].

HDBSCAN is an evolution of DBSCAN supporting hierarchical clustering (i.e. recognizes clusters inside clusters) and recognizing clusters with different densities besides one less parameter input (the radius distance threshold) [27]. HDBSCAN computes the minimum

spanning tree of a complete reachability graph having data items as nodes and their reachability distance as weights; the hierarchical clustering is obtained from such tree by removing all edges in order of decreasing weight [28]. The great advantage of HDBSCAN for this work is the available possibility of performing fuzzy clustering besides hard clustering by providing a soft membership vector (only for some distance functions) interpretable as a probability of a point being a member of a particular cluster. However, this algorithm is not incremental and has a quadratic computational complexity when using distance functions for which no accelerated indexing exists [27] such as Jaccard distance. Further improving on top of HDBSCAN, an incremental algorithm called FISHDBC was developed by maintaining an approximate version of the minimum spanning tree and updating it incrementally, at a low cost, as new data arrive [27], however the soft clustering option in the incremental version is not offered.

2.1.1.5.4 Non-Negative Matrix Factorization Clustering

As matrix factorization, Nonnegative Matrix Factorization (NMF) is also a dimension reduction method that seeks to find a lower rank approximation of the data matrix but the factors that give the lower rank approximation are all nonnegative, for a set of observations in the form of a matrix with only nonnegative elements as well. NMF has received much attention due to its straightforward interpretability for applications, i.e., we can explain each observation by additive linear combination of nonnegative basis vectors. Greater attention has been given to NMF for its application to data clustering [15, 29-31]. The clustering capabilities of such linear projection method arise from how well the new basis projected directions follow the clouds (or clusters) of data in the original data space. In real data applications, clusters are distributed according to irregular directions, so if unrealistic constraints (such as orthogonality) are imposed on the matrix factorization problem a significant loss of information may occur which may worsen the clustering results. The only constrain on NMF is the non-negativity, so such method has capabilities for finding the real directions in a data set that is distributed into a high dimensional space and the better the new subspace axis the better the data is discriminated and then more appropriated for clustering.

There are theoretical results which show the relation between NMF and the classical *k-means* clustering because both methods optimize the same objective function, and more particularly NMF is closely approximated to soft *k-means* [32] when the parameter that controls fuzziness in fuzzy *k-means* algorithm equals 2. When NMF is seen as a clustering tool the lower rank basis matrix corresponds to the cluster centroids while the coefficient matrix is the cluster membership indicator. NMF is a naturally batch method but some incremental approaches have

been proposed, e.g. [30, 31] however increasing issues may rise for clustering problems targeting recommender systems. The main bottleneck is that the utility matrix dimension (user-item matrix) is not known a priori in an incremental approach while e.g. NMF-based incremental algorithms [30, 31] targeted stream data with fixed dimensionality.

### 2.1.2 State-of-the-art

Most recommender models demonstrated quite good performances for the top-*n* recommendation problem, however they are usually based on a single model for all users. A single model may not be enough to capture the preferences of a set of users. In particular, a single model may not detect diverse or opposing preferences existing in user subsets. Local models built separately for each subset of users may then be used to represent fine-grained patterns. This way, the idea of estimating multiple localized models has been proposed mostly targeting an ensemble recommender approach [10-12]. A pioneering work in 1999 involved clustering the utility matrix item-wise with different algorithms followed by estimating separate local models but each of the models were used for the respective independent clusters with neighbourhood-based collaborative filtering. i.e. one local model is built for each cluster [33]. They have obtained mixed results on improving accuracy, however, clustering resulted in more accurate predictions than a random partitioning [33].

In 2008, Koren proposed a combined model utilizing both global and local components, which estimates every user-item rating as a combination of a global latent space model and local neighbourhood interactions [34].

Later, a co-clustering method on both users and items was developed for estimating a separate local model for each cluster with the rationale that is more natural to make preference predictions for a user via similar subgroups than the entire user-item matrix. Following they applied different collaborative filtering methods; including an item-based neighbourhood method [35]. Authors observed an improved predicted value for a user-item pair with the prediction coming from the model from the cluster with the largest weight for that user.

Weston et al. [36] also used latent space to model the user's latent tastes and item's latent vectors. The prediction for each user and item is obtained by computing the maximum possible score after multiplying each of the user latent vectors to the item one. This way, allowing users to have different sets of latent factors, instead of a global one.

Komkhao et al. [4] developed an incremental collaborative filtering algorithm based on the Mahalanobis distance. The algorithm has two phases: the learning phase, in which models of similar users are constructed incrementally, and the prediction phase, in which interested

users are clustered by measuring their similarity to existing clusters in a model. In order to handle the confusion on decision making about overlapping clusters, fuzzy sets are employed, and the degree of membership to them is expressed by the Mahalanobis radial basis function. Experimental results demonstrate that the proposed algorithm leads to improved prediction accuracy and prevents the scalability problem in recommendation systems.

Lee et al. [37, 38] proposed a method based on the idea that the user-item matrix is locally low-rank. They cluster user-item pairs based on a function that measures distances between pairs of users and items and then a local low-rank model is estimated for every cluster. The prediction is computed as a combination of weighted local models. Authors have claimed that the combination of a mixture of local low-rank models outperformed many of the currently used state of-the-art recommendation systems.

Siddiqui et al. [39] studied the problem of recommendations in the context of multi-relational stream mining where the algorithm learns and adapts to the user preferences, as these preferences evolve over time. Their algorithm first separates users based on their historic data into clusters and then employs collaborative filtering to recommend new items to the users based on their clustering structure. They use multi-relational stream clustering built upon their methodology for clustering a stream of complex objects, and extend it to deliver insights on user similarity on the basis of their past transactions. User preferences are reflected in the ratings that users give to items that are accumulated into user profiles, then stream clustering is used to build the profiles and adjust them to change. Authors claim that this approach gives a more elaborate notion of similarity: instead of a static similarity between users, they have used a dynamic similarity between user profiles.

A batch approach combining a global item-item model and a personalization factor for each user leveraging the interplay between the global and the local information was proposed by Christakopoulou and Karypis [10]. They focus on item-item models by previously clustering user subsets and following their recommender model computes the prediction scores as a user-specific ensemble combination of the predictions derived by a global and local models outperforming competing top-*n* recommendation approaches. Later, they extended their approach but instead of item-item models they have used a latent space models both for the global and multiple local models estimated for every user [11].

Vinagre et al. pioneered in 2018 on incremental ensemble models for improving recommender prediction accuracy [8]. Of particular importance for this work, they have addressed two incremental approaches using bagging: instance-based bagging and user-based bagging [7]. In the instance-based bagging, *n* models on *n* bootstrap nodes are learned, where

the number of copies of a particular user-pair to be present in each of the bootstrap nodes follows a Poisson(1) distribution (see 2.3.1.4- Ensemble methods on recommender systems), i.e. data is distributed blindly across the *n* bootstrap nodes. In practice, Poisson(1) is the number of times each user's examples is used to update each sub-model in the ensemble. As a result of a blind distribution, each individual user profile is different in each of the different nodes. The final list of item recommendations for a user is sorted based on an average score of all nodes.

In order to address the question if the predictive ability of the models is affected by the artificially modified user profiles the user-based bagging model is considered. In this case, instead of distributing user-items pairs independently, data is distributed locally according to the user. This means that all user-item pairs from the same user will be assigned to the same local bootstrap node the same Poisson(1) number of times. This way, Poisson(1) is then drawn only once for each user in each bootstrap node and all subsequent user-item pairs of the same user in that bootstrap node are used exactly the same Poisson(1) times as the first drawn for training that node. The authors address that such approach guarantees the completeness of user profiles in the local bootstrap nodes.

Authors have observed that with all datasets tested, bagging improves recall scores at all cut-offs, especially with a number of nodes higher than 30 and all the best results are obtained with 64 nodes [8] with improvements of up to 55 % over the not-ensemble model version. Moreover, they have also perceived that bagging yielding higher relative improvements over the not-ensemble incremental model may suggest that improvements of the predictive ability are mostly obtained after the top few recommended items. In its turn, the user-based bagging model achieves considerably better results than both the not-ensemble and instance-based bagging model variants, except for one of the datasets, indicating that a user-based local approach is thus beneficial, since it does not break apart interactions from the same user across the several independent learning nodes.

An incremental approach for recommendation systems based on item-based local models was proposed by Al-Ghossein et al. based on the knowledge that dynamic local models have the ability to capture diverse preferences among user subsets [12]. Authors maintain one global model for all users and several local models built separately for each subset of users. Their approach automatically detects the drift of preferences that leads a user to adopt a behaviour closer to the users of another subset, and adjusts the models accordingly.

In 2019, the possibility of obtaining latent user and item feature vectors from user/item clusters was investigated by Khawar and Zhang for collaborative filtering [40]. The user preference dimension is the tendency to consume a subset of items while items associated with

the same preferences tend to be related. Their algorithm first partitions items into disjoint item clusters. Following for each of the item clusters they partition users into two taste groups - like and dislike those items, such that the more items in an item cluster a user consumed in the past the more likely the user is going to be assigned to the first group. This way, they obtain different local user taste groups that can overlap because users can have multiple tastes. In the third step, the algorithm map users and items to local latent feature vectors based on the two groups such that the vector for a user tells user preferences while the vector for an item indicates item's popularity among several users. By using such integrated clustering method, authors claim that more meaningful latent factors are obtained and hence recommendations are easier to explain.

# 3. Methods and algorithms

This chapter details the clustering methodologies and the four recommender algorithms used in this work besides introducing the evaluation method to calculate the performance of the recommender algorithms, finally the description of the five datasets used is provided.

## 3.1 Clustering methodologies

User clustering performed previously to the ensemble recommendation is addressed in order to infer if there is an improvement on prediction accuracy by using improved local models. For that, preliminary steps involved a batch clustering analysis using different clustering algorithms. This way, a user-item binary matrix was built for each of the datasets and such utility matrix was used for a clustering analysis that precedes the incremental bagging recommendation algorithm.

Table 1 summarizes all the clustering algorithms considered in this work and in parallel this task also involved the survey of the most suitable incremental clustering algorithms for an online incremental recommender system.

## 3.2 Recommender algorithms

The accuracy obtained by the local ensemble models will be compared with previously developed incremental recommender systems versions [8] in order to infer the possible advantages of the strategy considered in this work. Following, a short description of the algorithms used for comparison as well as the present algorithm is addressed.

### 3.2.1 Baseline version

Baseline algorithm [4] is a simple online matrix factorization method developed by Vinagre et al. that uses Stochastic Gradient Descent to solve Eq. (1) [5]:

$$\min_{P,Q} \sum_{(u,i) \in D} (1 - p_u q_i^T)^2 + \lambda(||p_u||^2 + ||q_i||^2) \quad (1)$$

The algorithm continuously updates factor matrices ***P*** and ***Q***, correcting the model to fit to the incoming user-item pairs - See section 2.3.1.3 Matrix factorization for recommendation- where the regularization term $\lambda$ penalizes overly complex models that tend to overfit. The baseline algorithm pseudocode is shown below [5]:

*Table 1 - Experimentally tested clustering algorithms.*

| Name | Type | Incremental | Fuzzy | Access to soft cluster probabilities | Based on pair-wise distances | Suitable for incremental clustering on incremental recommendation systems | Outcome from trial for this problem |
|---|---|---|---|---|---|---|---|
| BIRCH [1] | Hierarchical + Distance based to centroid | Yes | No | No | No | No | One huge big cluster and other 30 few users clusters. Due to the high dimensionally, all the clusters probably lay very near to each other being impossible to distinguish. |
| Hierarchical Clustering [2] | Hierarchical | No | No | No | Yes, cosine | No | More suitable than distance to centroid methods but no soft probabilities obtained |
| CluSTREAM [3] | Hierarchical + Distance based to centroid | Yes | No | No | No | No | Same outcome as for BIRCH |
| ODAC [4] | Hierarchical time series clustering | Yes | No | No | Only correlation | No | Not suitable to this stream problem, it is a time series clustering algorithm |
| NMF [5] | Matrix factorization | No | No | Yes | Out of scope | No | Used for the strategy in this work |
| Fuzzy c-means [6] | Distance to predefined number of centroids | No | Yes | Yes | No | No | All soft clusters showed the same probabilities as random attribution – no clustering power probably because of the high dimensionality as observed for the other distance to centroid based algorithms |

1 https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html
2 https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering
3 https://github.com/narjes23/Clustream-algorithm
4 https://github.com/rodrigoejcm/odac
5 https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html
6 https://pypi.org/project/fuzzy-c-means/

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| FISHDBC [7] | Density/ hierarchical (based on HDBSCAN) | Yes | Yes | No | Yes | No | Not suitable to this stream problem, because it does not apply to an incremental learning of a utility matrix |
| DBSCAN [8] | Density-based | No | No | No | Yes, cosine | No | Possibly good clustering properties of the utility matrix |
| HDBSCAN [9] | Density/ hierarchical (based on DBSCAN) | No | Yes | Yes | Yes, jaccard | No | Partially used for the evaluation of the strategy in this thesis |
| Size Constrained Clustering [10] | Distance to predefined number of centroids but clusters size can have a min and max limit | No | No | No | No | No | Limiting min and max cluster size could avoid the formation of one very big cluster as happened with all the distance to centroid based algorithms. Cluster sizes should be similar for the current application. No soft probabilities, no further evaluations. |

7 https://github.com/matteodellamico/flexible-clustering
8 https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
9 https://hdbscan.readthedocs.io/en/latest/index.html
10 https://github.com/jingw2/size_constrained_clustering

*Baseline Algorithm:*

**Data**: stream $D = \{(u, i)_1, (u, i)_2, \ldots\}$
**input**: latent features $z$, iterations *iter*, regularization $\lambda$, learn rate $\eta$
**output**: factor matrices **P** and **Q**
**for** $(u, i) \in D$ **do**
    **if** $u \notin$ Rows(**P**) **then**
        $p_u \leftarrow$ Vector (size : $z$)
        $p_u \sim N(0, 0.1)$
    **if** $i \notin$ Rows(**Q**) **then**
        $q_i \leftarrow$ Vector(size : $z$)
        $q_i \sim N(0, 0.1)$
    **for** $n \leftarrow 1$ **to** *iter* **do**
        $err_{ui} \leftarrow 1 - p_u q^T_i$
        $p_u \leftarrow p_u + \eta(err_{ui} q_i - \lambda p_u)$
        $q_i \leftarrow q_i + \eta(err_{ui} p_u - \lambda q_i)$

### 3.2.2 Instance-based bagging version

An online bagging approach was applied to the baseline algorithm [4] by initializing $M$ sub-models (or bootstrap nodes) and then using $K$ = Poisson(1) (See section 2.3.1.4 Ensemble methods on recommender systems) to train new examples $K$ times by redrawing $K$ for each node. The instance-based bagging algorithm pseudocode is shown below [8]:

*Instance-based Bagging Algorithm:*

**Data**: stream $D = \{(u, i)_1, (u, i)_2, \ldots\}$
**input**: latent features $z$, iterations *iter*, regularization $\lambda$, learn rate $\eta$, bootstrap nodes M
**output**: factor matrices $\mathbf{P}^m$ and $\mathbf{Q}^m$
**for** $(u, i) \in D$ **do**
    **for** $m \leftarrow 1$ *to* $M$ **do**
        $K \sim$ Poisson(1)
        if $K > 0$ **then**
            **for** $I \leftarrow 1$ to K **do**
                **if** $u \notin$ Rows($\mathbf{P}^m$) **then**
                    $p^m_u \leftarrow$ Vector (size : $z$)
                    $p^m_u \sim N(0, 0.1)$
                **if** $i \notin$ Rows($\mathbf{Q}^m$) **then**
                    $q^m_i \leftarrow$ Vector(size : $z$)
                    $q^m_i \sim N(0, 0.1)$
                **for** $n \leftarrow 1$ **to** *iter* **do**
                    $err_{ui} \leftarrow 1 - p^m_u (q^m_i)^T$
                    $p^m_u \leftarrow p^m_u + \eta(err_{ui} q^m_i - \lambda p^m_u)$
                    $q^m_i \leftarrow q^m_i + \eta(err_{ui} p^m_u - \lambda q^m_i)$

### 3.2.3 User-based bagging version

Vinagre et al. [8] have shown that a localized user-based approach on top of the bagging algorithm is beneficial, since it does not break apart interactions from the same user across several independent learning nodes. If a user *u* is in a given node, then all of its interactions (u, i) are fed to this node - potentially more than once. The user-based bagging algorithm pseudocode is shown below [8]:

*User-based Bagging Algorithm:*

**Data**: stream $D = \{(u, i)_1, (u, i)_2, \ldots\}$
**input**: latent features *z*, iterations *iter*, regularization $\lambda$, learn rate $\eta$, bootstrap nodes M
**output**: factor matrices $\mathbf{P}^m$ and $\mathbf{Q}^m$
**for** $(u, i) \in D$ **do**
  **for** $m \leftarrow 1$ *to* $M$ **do**
    if $k_u^m$ == NULL **then**
      $k_u^m \sim Poisson(1)$
    if $k_u^m > 0$ **then**
      **if** $u \notin$ Rows($\mathbf{P}^m$) **then**
        $p^m{}_u \leftarrow$ Vector (size : $z$)
        $p^m{}_u \sim N(0, 0.1)$
      **if** $i \notin$ Rows($\mathbf{Q}^m$) **then**
        $q^m{}_i \leftarrow$ Vector(size : $z$)
        $q^m{}_i \sim N(0, 0.1)$
      **for** $n \leftarrow 1$ **to** *iter* **do**
        $\text{err}_{ui} \leftarrow 1 - p^m{}_u \, (q^m{}_i)^T$
        $p^m{}_u \leftarrow p^m{}_u + \eta(\text{err}_{ui} q^m{}_i - \lambda p^m{}_u)$
        $q^m{}_i \leftarrow q^m{}_i + \eta(\text{err}_{ui} p^m{}_u - \lambda q^m{}_i)$

### 3.2.4 Cluster-based bagging version

The cluster-based algorithm further addresses the advantages of the user-based bagging approach but instead of considering a Poisson(1) user node allocation distribution, it uses user-clustering results obtained from the user-item final matrix to distribute the users in the bootstrap local models. Additional to the algorithm above, this one has as input the membership probabilities of each user belong to a cluster. Based on that, a user belongs to a node/cluster or not if a certain threshold (*t*) is observed. In this work, different threshold methodologies are also considered. The cluster-based bagging algorithm pseudocode is shown below:

*Cluster-based Bagging Algorithm:*

**Data**: stream $D = \{(u, i)_1, (u, i)_2, \ldots\}$

**input**: latent features $z$, iterations *iter*, regularization $\lambda$, learn rate $\eta$, bootstrap nodes M, user fuzzy cluster probabilities $C$ with dimensions (number of users, number of nodes), threshold $t$

**output**: factor matrices $\mathbf{P}^m$ and $\mathbf{Q}^m$

**for** $(u, i) \in D$ **do**

  **for** $m \leftarrow 1$ *to* $M$ **do**

    **if** $C^m{}_u \in$ Top $t$ x $M$ (or $t$ x $1/M$) highest probabilities for $u$ to belong to any node **then**

      **if** $u \notin$ Rows($\mathbf{P}^m$) **then**

        $p^m{}_u \leftarrow$ Vector (size : $z$)

        $p^m{}_u \sim N(0, 0.1)$

      **if** $i \notin$ Rows($\mathbf{Q}^m$) **then**

        $q^m{}_i \leftarrow$ Vector(size : $z$)

        $q^m{}_i \sim N(0, 0.1)$

      **for** $n \leftarrow 1$ **to** *iter* **do**

        $\text{err}_{ui} \leftarrow 1 - p^m{}_u \, (q^m{}_i)^\top$

        $p^m{}_u \leftarrow p^m{}_u + \eta(\text{err}_{ui} q^m{}_i - \lambda p^m{}_u)$

        $q^m{}_i \leftarrow q^m{}_i + \eta(\text{err}_{ui} p^m{}_u - \lambda q^m{}_i)$

Hyperparameters for the baseline models of each of the datasets were previously determined [8] and were kept the same for all the subsequent models herein described.

## 3.3 Evaluation

The evaluation process is based on predictive ability using Recall@20 on a prequential evaluation method [5, 8]. Recall is the proportion of all relevant items in a system that lies within a list retrieved by that system. Using a cut-off of 20, this is truncated to the first 20 elements. Recall yields 1 if item $i$ is within the 20 first recommended items, and 0 otherwise. An averaging recall from all the nodes is calculated at the end of the experiment.

Prequential evaluation is an alternative to the holdout method in the evaluation of recommendation algorithm [5, 8]. The prequential process considers that for each incoming ($u$, $i$) pair a prediction with the current model is performed and the score of that prediction is performed by matching it to the actual observation. Then, the model is updated with the observation and advances to the following ($u$, $i$) pair (

*Figure* 1 – *Prequential evaluation process* [8].).

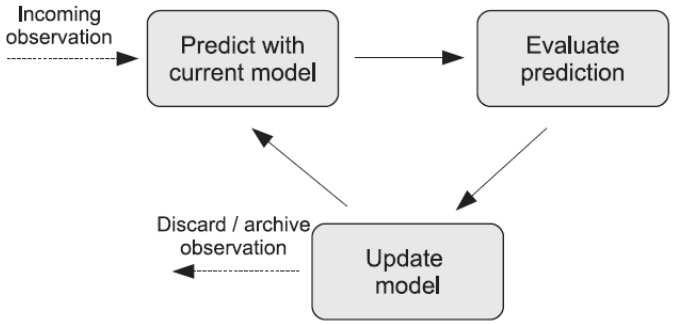



*Figure 1 – Prequential evaluation process [8].*

## 3.4 Datasets

In this work, five real-world positive-only preference datasets were considered, according to Table 2, with different proportions of items and users. All datasets consist of a chronologically ordered set of pairs in the form < user; item > in order to simulate a streaming environment in the recommender models.

PLC-STR[1] and LFM-50U [2] consist of music listening events obtained from two distinct sources, where each tuple corresponds to a music track being played by a user. LFM-50U is a subset consisting of a random sample of 50 users taken from the Last.fm [3] dataset. Unique occurrences of < user; item > pairs in both sets were removed, since these probably do not reflect a positive interaction. Moreover, PLC-STR and LFM-50U contain repeated events in music listening, because users listen to their favourite music tracks more than once. Most systems do not recommend items to a user that already knows them, except in very specific applications, such as automatic playlist generation. Repeated events are also considered to update the models, in order to better reflect a real world scenario but not used for evaluation.

PLC-playlist [4] consists of a timestamped log of music track additions to personal playlists. Both PLC-STR and PLC-playlist are extracted from the Palco Principal [1] website, a Portuguese social network dedicated to non-mainstream music enthusiasts and artists. ML1M is a well-

1 http://www.palcoprincipal.com/
2 http://ocelma.net/MusicRecommendationDataset [Jan 2013]
3 http://last.fm/
4 https://rdm.inesctec.pt/dataset/cs-2017-003,file: playlisted tracks.tsv
5 http://www.grouplens.org/data [Jan 2013]

known dataset consisting of timestamped movie ratings in a 1 to 5 rating scale from MovieLens-1M [5], to use this dataset as positive-only feedback, movie ratings below the maximum rating 5 were excluded [8]. Finally, for YHM-6KU, 6000 users were randomly sampled from the Yahoo! Music dataset [6]. YHM-6KU is also obtained from a ratings dataset and to use it as positive-only data as well, only ratings of 80 or above in the YHM-6KU dataset were retained [8].

*Table 2 - Dataset description.*

| Dataset | Events | Users | Items | Sparsity |
|---|---|---|---|---|
| PLC-STR | 588 851 | 7 580 | 30 092 | 99.74 % |
| LFM-50U | 1 121 520 | 50 | 159 208 | 85.91 % |
| PLC-playlist | 111 942 | 10 392 | 26 117 | 99.96 % |
| ML1M | 226 310 | 6 014 | 3 232 | 98.84 % |
| YHM-6UK | 476 886 | 6 000 | 127 448 | 99.94 % |

## 3.5 Experimental set-up

The algorithms and prequential evaluation code in all the bagging recommender models are implemented in Python 3.7 from an adaption of the baseline model implemented in Python 3.7 code which was originally implemented in a MyMediaLite Gantner, Rendle, Freudenthaler, & Schmidt-Thieme, 2011 code version [8]. This task showed to be quite time consuming though during this thesis.

[6] https://webscope.sandbox.yahoo.com/catalog.php?datatype=r [Jan 2013]

# 4. Results and discussion

This chapter revolves around the results of the cluster analysis for the four types of clustering methods used and addresses the recommendation results for two approaches considered for distributing the users with overlapping through the local nodes.

## 4.1 Clustering analysis

Table 2 shows the 10 different algorithms considered in this research work, three main reflexions had to be pondered before selecting those clustering algorithms:

*i)* A batch clustering approach should be initially considered by using the user-item positive feedback matrix, as an exploratory methodology to infer the feasibility of the local model approach, and before applying an incremental clustering version integrated with the online recommender models;

*ii)* The clustering algorithm should perform a fuzzy-based approach, in order to have access to the probabilities of a user belong to several different clusters. This was considered not only because indeed users might have simultaneous affinities for a wide range of items but also because an ensemble bagging algorithm was chosen as the method to be explored under the different ensemble approaches available. Accordingly, as the core concept behind a bagging algorithm is that the bootstrap nodes indeed contain overlapping examples among the different nodes, such core concept should be maintained as well.

*iii)* Clustering algorithms with code openly available in Python programming language should be targeted.

After those three starting conditions defined, a wide range of clustering methods should be considered ranging from hierarchical clustering, passing through density-based approaches and also considering distance-based approaches. However, due to the high dimensionality of the user-item matrix, a probable limitation of distance-based approaches was already anticipated and that clustering methods with pair-wise calculations between all users could embrace the most probable suitable algorithm to the present problem.

From the survey analysis on clustering algorithms, it became clear that openly available fuzzy algorithms with Python code are not widely abundant. Several clustering algorithms with a core fuzzy approach algorithm are indeed found in the open literature but either with no open access to suitable code or fuzzy clustering is only an intermediary internal step of the algorithm

and the label obtained after clustering points only to hard labels [9]. This actually was a severe limitation as will be further discussed.

#### 4.1.1 Distance to centroid clustering algorithms

Initial algorithm screening departed then from the well-known distance-based fuzzy clustering algorithm: *fuzzy c-means* [24]. Actually this algorithm indeed outputs probability cluster memberships, it works on both batch and incremental approaches and it allows controlling previously the number of target clusters.

Controlling the number of output clusters in advance is very advantageous because bagging algorithms have as an input figure the number of bootstrap nodes on the ensemble model. Moreover, the accuracy of the bagging algorithms in this research work are indeed evaluated as function of a previous properly defined number of bootstrap nodes. This also removes complexity on the input parameters in any possibly final commercial recommender system based on such algorithm.

Figure 2 shows the user probability distribution for the utility matrix based on PLC-playlist dataset for four previously defined clusters for one randomly selected cluster (fuzzy parameter ($m$) = 2, maximum number of iterations = 150 and stopping criteria = $10^{-5}$ were used). It is clear that the discrimination capacity of the clustering algorithm is null in this cluster with an equal distribution of ¼ for all the users in the dataset for that cluster while the other 3 clusters also show a plot with exactly the same profile (data not shown). The same approach was considered for a total number of clusters of 32 and same dataset, and additionally for a utility matrix based on the PLC-STR dataset with 5 predefined number of clusters: in all cases a similar result is obtained, all users have the same $1/n$ probability of belonging to a particular cluster. So, *fuzzy c-means* is simply uniformly distributing the users into the predefined number of clusters thus not presenting any relevant cluster ability. This result may point to the strong limitation of distance-based clustering algorithms on datasets with a very high dimensionality: more than 30 000 and 26 000 items on the PLC-STR and PLC-playlist datasets, respectively.

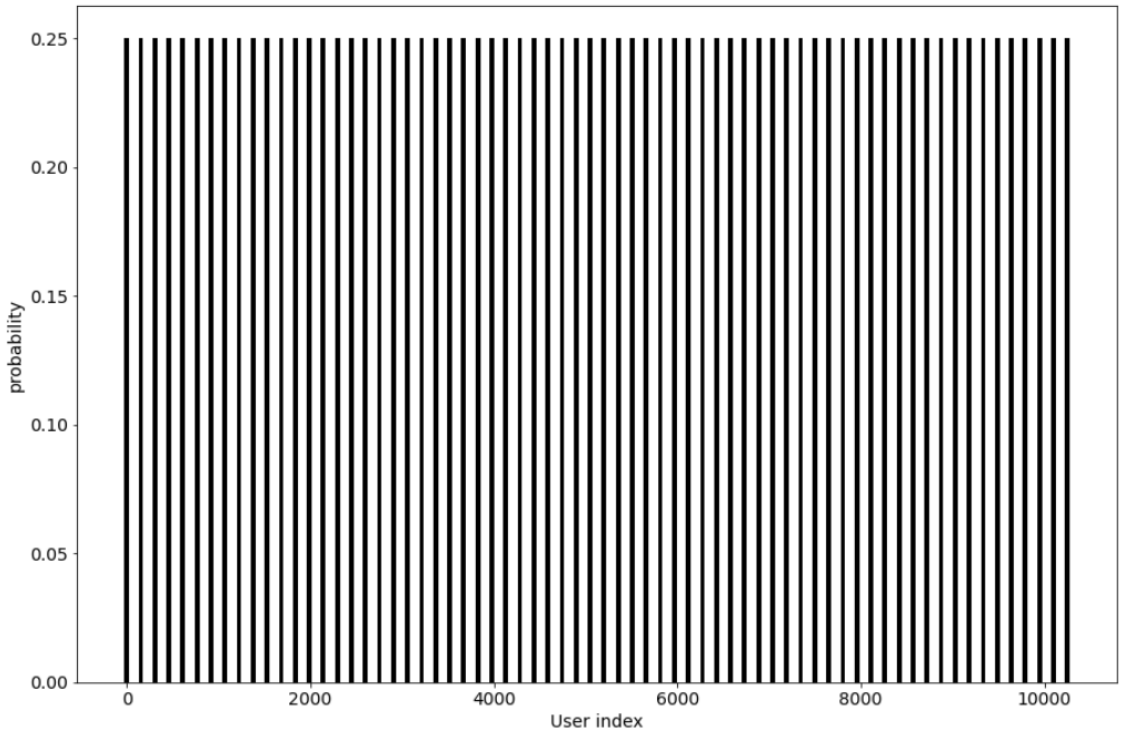



*Figure 2 - User probability distribution in one of the clusters for the PLC-playlist based utility matrix using fuzzy c-means and n = 4.*

In order to be clear on the probable lack of feasibility of a distance-based clustering method for addressing this problem, the *Size Constrained Clustering* is used (Table 1). This procedure is a hard batch clustering algorithm that considers the minimization function the Euclidean distance to a predefined number of centroids as well but clusters size can have a predefined minimum and maximum limit of objects in each cluster. The idea was to infer if limiting the minimum and maximum cluster size could improve distance-based clustering performances and thus avoid the formation of a single big cluster obtained with the previous distance to centroid based algorithm. Moreover, it was anticipated that cluster sizes with a similar number of objects for the current application would be ideal due to the behind core bagging concept. This way besides clarifying the feasibility of distance-based methods, the evaluation of a constrained cluster size method was also important to evaluate. Testing was considered on the PLC-playlist user-item matrix using 32 clusters and minimum and maximum sizes of 35 and 1000, respectively.

Figure 3 shows the user count distribution across the 32 clusters. One can see that most of the cluster sizes are exactly 35 or 1000. This might indicate that probably the user distribution among the different clusters was based on uniform distribution -but limited by the cluster size constrains- than based on the object to centroid distance. In such case distances to centroids for

each object probably are all very similar - laying all on the same quite comparable hypersphere radius. This way, distance to centroid based cluster algorithms limitation seems continue to be relevant, moreover any further evaluations were not considered because unfortunately soft membership probabilities are not an option in this hard clustering method.

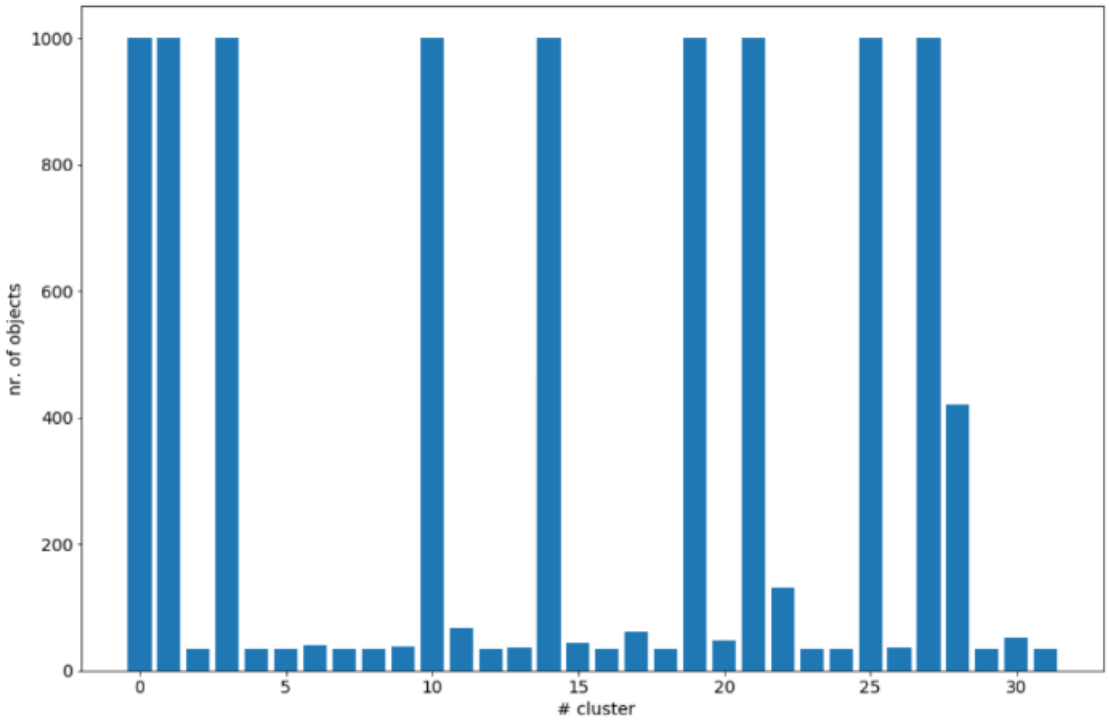



*Figure 3 - Size Constrained Clustering users' distribution count.*

4.1.2 Hierarchical-based clustering algorithms

The next type of clustering approach considered involved 4 different agglomerative hierarchical clustering algorithms: the batch version from the scikit-learn clustering module and 3 stream-based versions: BIRCH, CluStream and ODAC (Table 1). All of them are based on hard-clustering algorithms and in some of them there is the possibility of predefining the final number of clusters. The goal here was to infer if such type of clustering method would be a better fit to a utility matrix based type of data and evaluate if the users' distribution through the different clusters would be more spread than the typical profile observed in the distance-based clustering algorithms.

For the BIRCH algorithm, the input parameters on the algorithm were the maximum number of subclusters in a node =300, and the maximum diameter of a subcluster = 1.6 with no previously defined number of clusters - so the second more refined clustering was not performed and the final cluster centroids are then returned (see section 2.3.1.5.2 - Hierarchical

Clustering). Under these conditions, 35 user clusters were obtained for the PLC-playlist binary user-item matrix where one huge cluster with more than 10 000 objects and other with 18 objects are obtained while all the others are in the range from 1 to 3 objects (Figure 4).

Such result also points to a wide range of users being considered quite similar (the merge of the recently arriving user to the most alike subcluster is again based on the Euclidean distance of the new arriving object to the subcluster centroid where merging is being tested). This way, users are most probably consistently considered similar to each other due to the dimensionality curse. Moreover, if the clusters are not spherical in nature, it does not execute well because it uses the notion of radius or diameter to regulate the borderline of the cluster/group. Despite expending sometime on fine tuning parameters for better understand the algorithm and to better adapt it to the data, this time was not highly extended because both the soft probabilities not being provided and the use of the Euclidean distance based similarity measure are quite limiting.

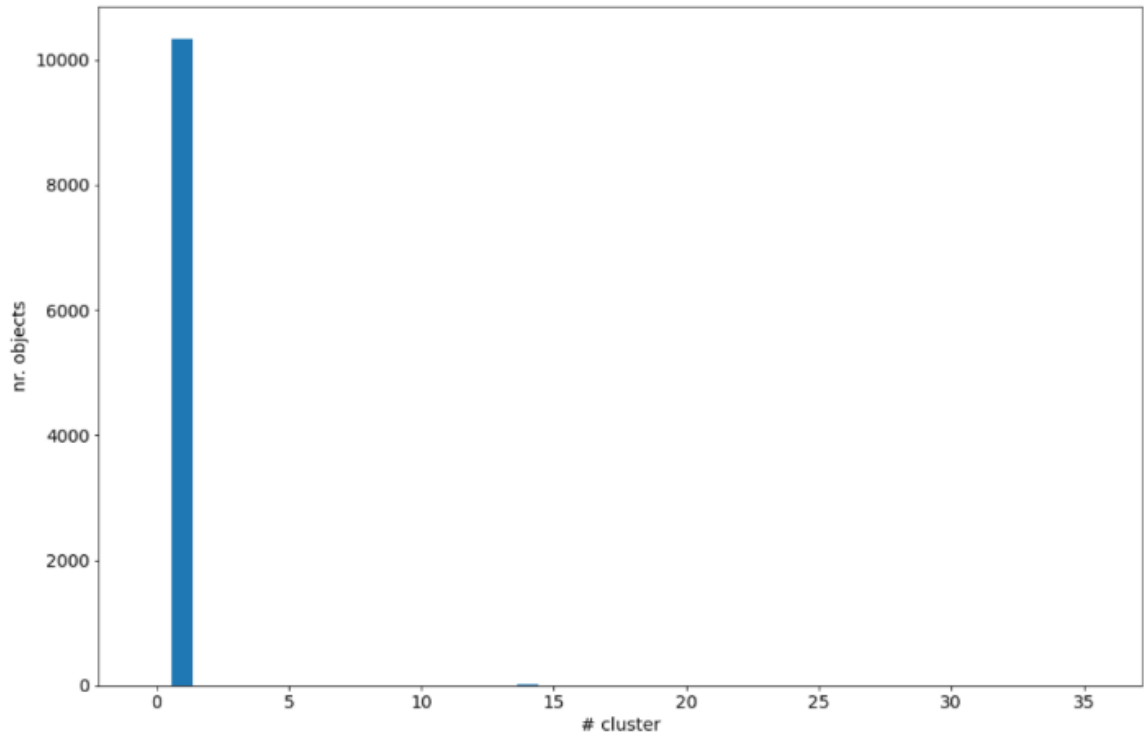



*Figure 4 - BIRCH cluster size distribution for the PLC-playlist utility matrix.*

The CluStream algorithm was only briefly considered in this study with most of the conclusions similar to the BIRCH algorithm. For a predefined number of clusters of 32, user clustering the user-item matrix from the PLC-playlist data returned nearly the same conclusions

and objects distribution throughout the clusters very similar to the BIRCH algorithm once both methods are quite similar on its core concepts.

Regarding the batch hierarchical agglomerative clustering algorithm, a measure based on the pair-wise cosine distance and a three linkage types were used to build the cluster dendograms for the PLC-playlist utility matrix. The cut-offs at a level of 8 on the dendogram are presented on Figures 5, 6 and 7. One can see that even a pair wise distance based clustering algorithm also fails to cluster the users, by grouping nearly all the users on the same big cluster independent of the linkage used. This shows that very high dimension sparse datasets are very hard to cluster what is the typical type of datasets observed in recommender problems. Also maybe indicating that a crispy clustering in this type of data is actually a not good approach. Indeed, two other types of soft clustering algorithms considered have successfully provided cluster membership probabilities (see below) and such information is indeed very relevant in order to infer what are the most probable clusters for a same user belong thus allowing a more informed decision on the way to group the users.

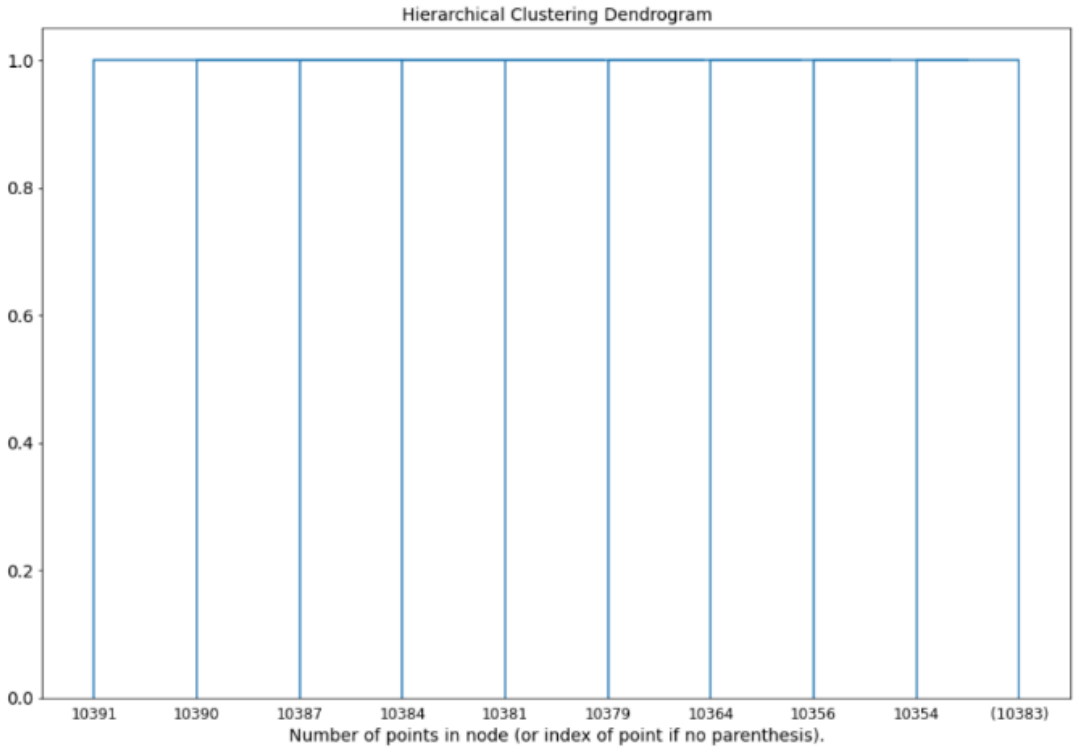



*Figure 5 – PLC-playlist dataset clustering based on agglomerative clustering for a cosine distance with single linkage.*

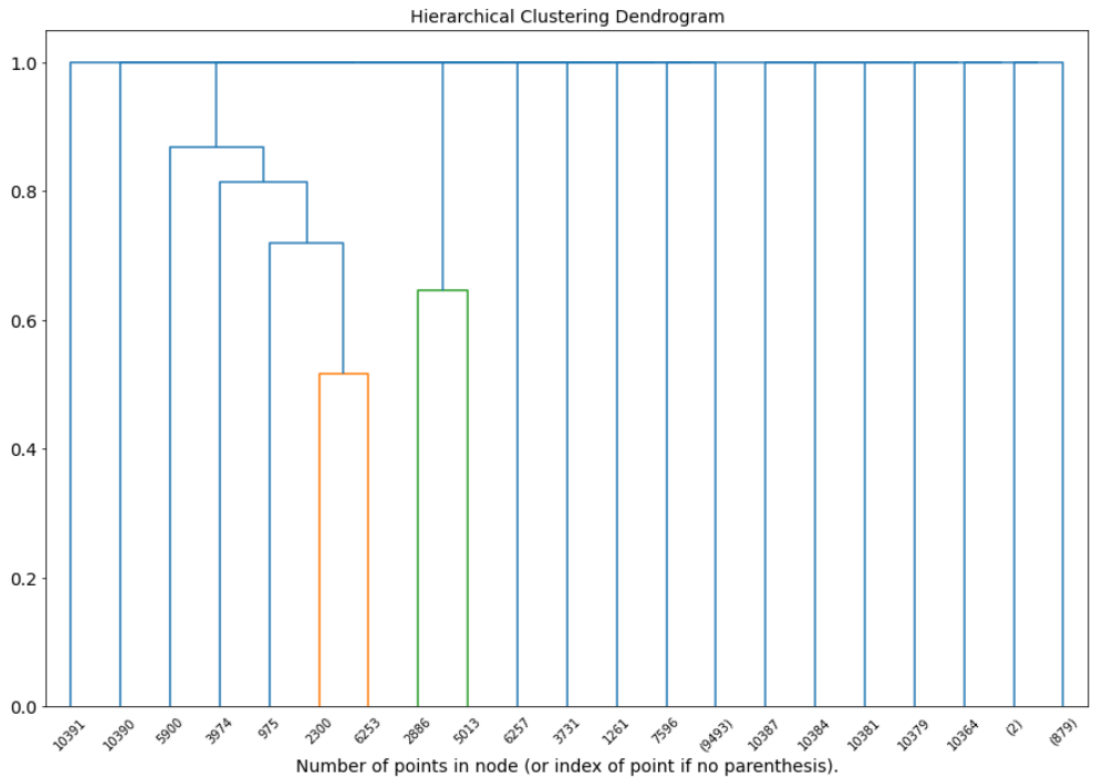



*Figure 6 - PLC-playlist dataset clustering based on agglomerative clustering for a cosine distance with average linkage.*

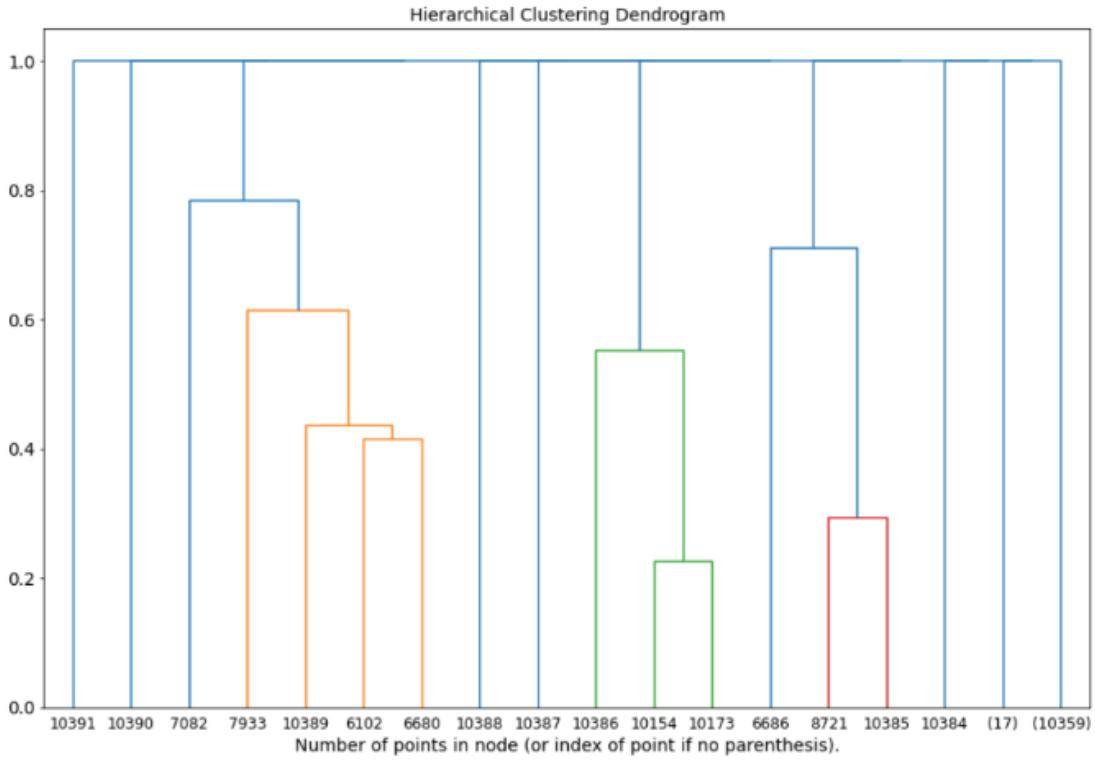



*Figure 7 - PLC-playlist dataset clustering based on agglomerative clustering for a cosine distance with complete linkage.*

Finally, ODAC algorithm is a time series incremental clustering algorithm based on a pair-wise correlation measure. In this case, each of the users in the user-item matrix is treated as a pseudo time series - obviously it is not a time series, and each single item in the utility matrix is the binary streaming data values of each 'time series'. This algorithm showed to be very intensive time consuming and the experiments were aborted because it is also a hard clustering algorithm.

#### 4.1.3 Density-based clustering algorithms

Three different density-based algorithms were experimentally considered: DBSCAN, HDBSCAN and FISHDBC (Table 1). DBSCAN and HDBSCAN perform batch clustering while FISHDBC is incremental. Yet, HDBSCAN allows a soft clustering option while the two others only offer hard clustering.

HDBSCAN showed to be the first soft clustering algorithm presenting a result different from all the same probability for each user belonging to each of the clusters and then having a result different from the *fuzzy c-means* algorithm presented above. So, some quite amount of time in this work was used for exploring such algorithm. Initial thoughts on a density-based algorithm were considered due to the poor results from the other algorithms; besides the utility matrix for the PLC-playlist data set (the data set most explored in the pre-screening clustering analysis) could be indicating that some strong outliers could be present in this utility matrix and on top of that non-spherical shape clusters could also be possible. Because DBSCAN is the standard density-based algorithm the candidate started exploring from that and after some research work the HDBSCAN algorithm was also quickly spotted.

DBSCAN results for a cosine distance smaller than a diameter threshold = 0.5 and the neighbourhood contains at least a minimal number of points = 15 indicated that 26 clusters could be obtained for the PLC-playlist dataset and indeed some objects were considered outliers. It also became quite clear that the two input parameters strongly affect the output number of clusters. This way, the radius threshold being no longer a required user input and the option of soft clustering drove research attention to HDBSCAN.

Table 3 shows the influence of the minimal number of points in the neighbourhood on both the total number of clusters obtained and the number of user outliers in the PLC-playlist dataset for a Jaccard distance measure. One can see that as the minimal threshold number of neighbourhood points increases the number of clusters consistently decreases (more points in the neighbourhood have to be observed in order to a cluster be created) while the number of

outliers reaches a constant value for 118 outlier users from 35 minimum points onwards thus indicating that in the PLC-playlist dataset 118 users are consistently considered as outliers.

*Table 3 - Minimal number of points influencing both the final number of clusters and user outliers for the HDBSCAN algorithm with the PLC-playlist utility matrix.*

| $N_{min}$ | # clusters | # user outliers |
|---|---|---|
| 5 | 508 | 579 |
| 10 | 181 | 288 |
| 35 | 31 | 118 |
| 50 | 23 | 118 |
| 60 | 10 | 118 |
| 70 | 6 | 118 |

Figure 8 shows the soft cluster membership probabilities obtained with the HDBSCAN clustering algorithm for the PLC-playlist dataset for 3 random clusters out of the 6 total clusters. The slashed black line indicates a uniform cluster distribution probability (1/*n*). One can see that different clusters point to different probabilities for the same user with some probabilities either higher or lower than the uniform probability. This result is quite different from the result observed on Figure 2 where *fuzzy c-means* simply outputted a uniform cluster attribution.

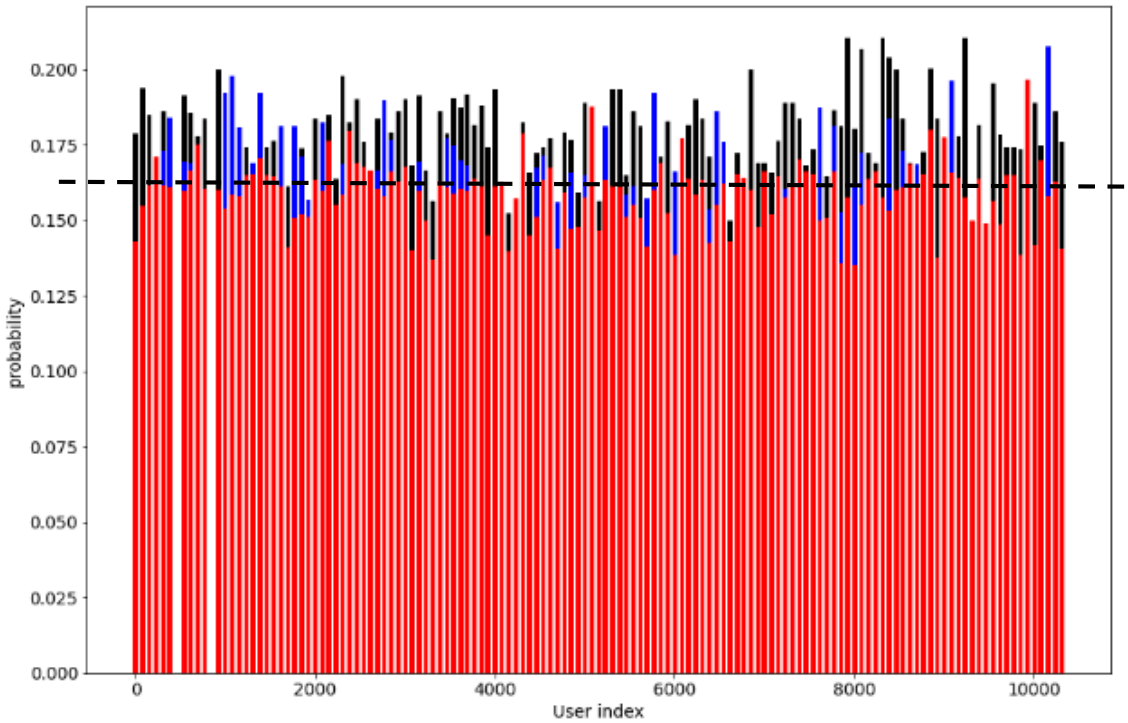



*Figure 8 - HDBSCAN soft probabilities from sampled users for 3 random clusters from a total of 6 clusters for PLC-playlist dataset (each colour refers to one different random cluster).*

Figure 9 shows the equivalent result as before but now for the PLC-STR dataset (minimal number of points in the neighbourhood = 70) where same conclusions are withdrawn: users present different soft memberships for the clusters.

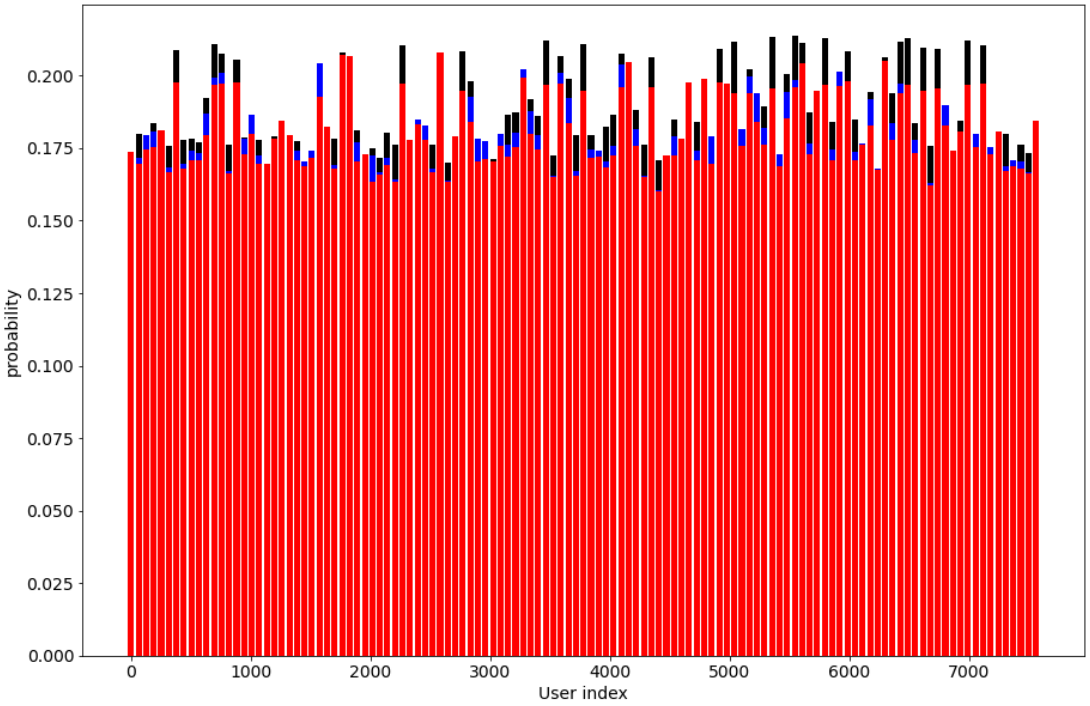



*Figure 9 - HDBSCAN soft probabilities from sampled users for 3 random clusters from a total of 5 clusters for PLC-STR dataset (each colour refers to one different random cluster)..*

From this analysis one can conclude that HDBSCAN apparently might be used for testing local models on the bagged recommender algorithm. One major drawback is that previously defining the number of clusters is not possible and for each dataset; one should define the minimal number of points in the neighbourhood that will then influence the final number of clusters obtained. Moreover, the clustering algorithm is quite time consuming. Despite of that, several experiments were performed with the integration of clustering analysis and the bagging recommender algorithm (see Section 4.2 – Recommender models, below). These first experiments indeed indicated that this approach is advantageous for an improved recommender accuracy. Following such result, an incremental algorithm for density-based clustering was then also started to be screened. The FISHDBC [27] is an incremental clustering version of the density-based HDBSCAN openly available in Python code but unfortunately the soft membership probabilities are not implemented in such incremental version so no much longer time was dedicated to this algorithm, besides such as HDBSCAN is quite time consuming for obtaining a clustering result. An incremental density-based clustering algorithm and with a

fuzzy approach [9] can be found very recently in the open literature but they use the distance-based *fuzzy c-means* approach to assign the incoming points to the local micro-clusters and then a crisp valley seeking algorithm is used to determine the final clusters. So, no soft membership probabilities return as an outcome from such algorithm. Other incremental density-based clustering algorithms are available, such as DenStream and D-Stream [20] but up to present knowledge no fuzzy algorithm is used neither soft probabilities are provided.

#### 4.1.4 Non-negative matrix factorization clustering algorithms

Inspired by the matrix factorization model approach used in the recommender models, the search for a suitable clustering algorithm where soft membership probabilities could be returned and with a better control of the output number of clusters lead this research work to NMF. NMF is known to have cluster capabilities and background theory indeed proves that NMF minimizes the same objective function as *k-means* when orthogonality is imposed, but, if this constraint is not imposed, NMF might become equivalent to soft *k-means,* but with an improved clustering capacity [15]. This way, in this research the orthogonality constraint was not imposed. Moreover, in NMF the low-rank of the fitted matrices is the number of final clusters and it is an input parameter of the algorithm. The output of the model is then the cluster centroids matrix and the coefficient matrix is the membership indicator. The coefficient matrix obtained is scaled between 0 and 1 (min-max normalization) so that the equivalent to soft membership probabilities are obtained. Additionally, clustering time is approximately 60 times faster than using HDBSCAN.

Figure 10 shows the normalized soft probabilities for 3 random clusters out of 6 and sampled index users from NMF clustering algorithm for PLC-STR dataset. One can see that some users are indeed distributed across different clusters while for other users the probability of belonging to the other different two clusters is null (although non null probabilities might be observed for these users in the other not represented 3 clusters). In its turn, Figure 11 shows the equivalent results but for the ML1M dataset while pre-defining 10 clusters (sof probabilities from only 5 random clusters are shown).

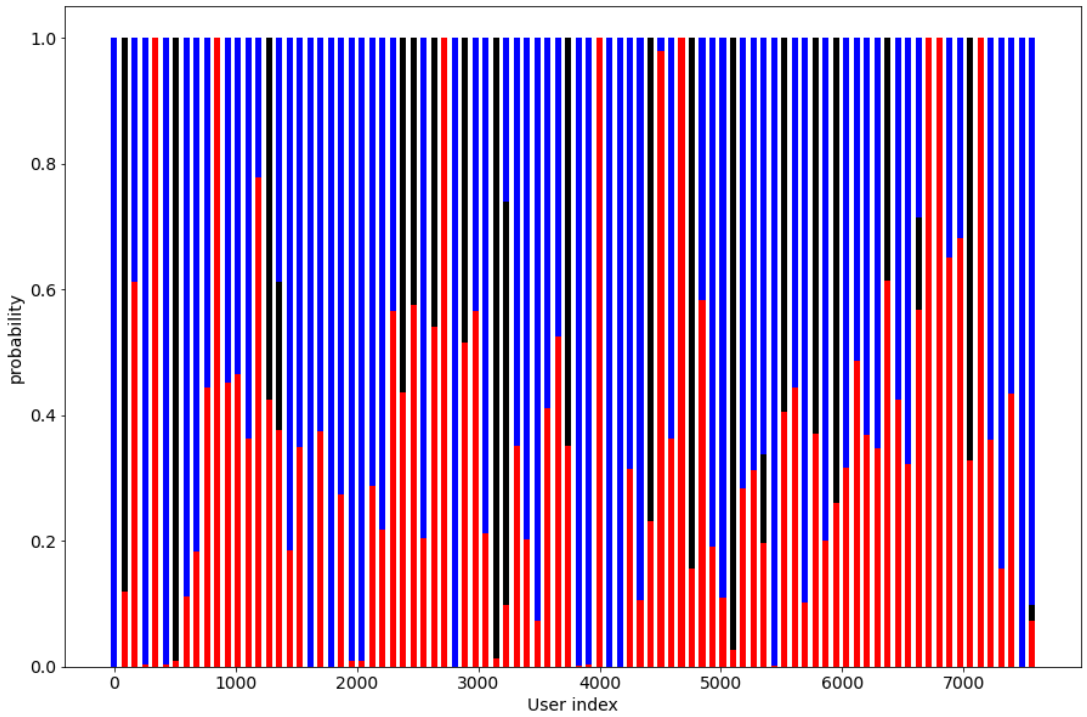



*Figure 10 - NMF soft probabilities from sampled users for 3 random clusters from a total of 6 clusters for PLC-STR dataset (each colour refers to one different random cluster).*

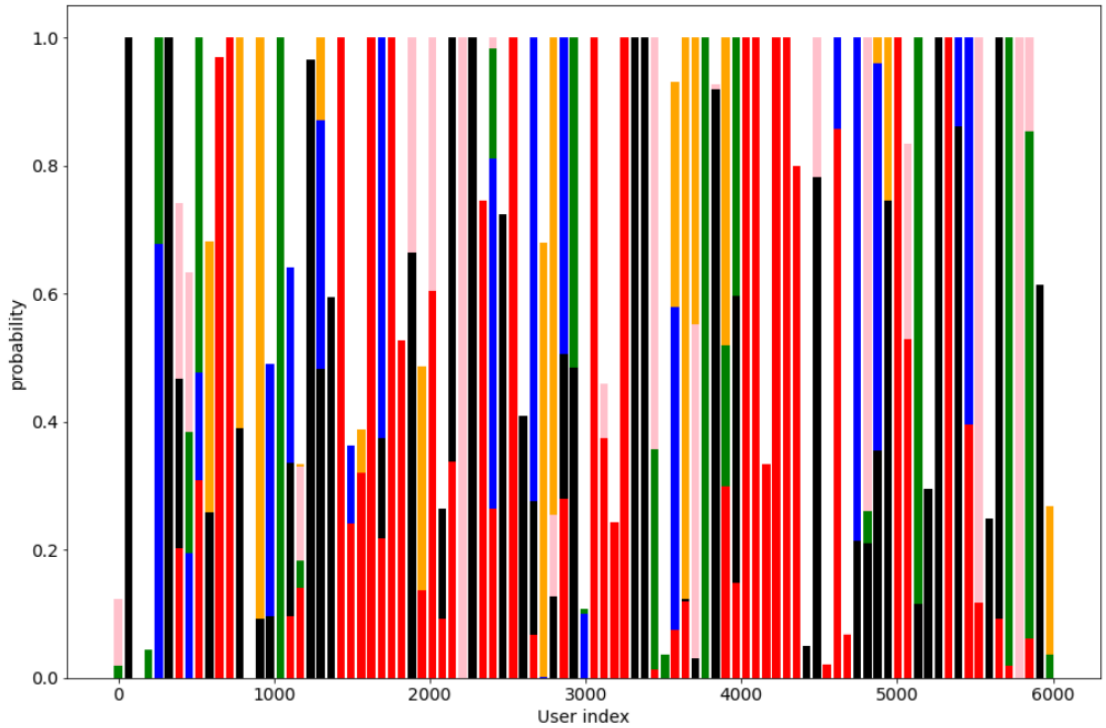



*Figure 11 - NMF soft probabilities from sampled users for 5 random clusters from a total of 10 clusters for ML1M dataset (each colour refers to one different random cluster).*

## 4.2 Recommender models

According to the previous section, two clustering batch algorithms were successfully obtained for providing soft cluster membership probabilities for different users from several datasets. These two clusterting algorithms will be used to evaluate if such improved local based recommender algorithms provide indeed better predictions for the users when considering different datasets. For that, the final user-item matrix for each dataset was built and after performing batch user clustering on such utility matrix, the membership probabilities for each user in each cluster are obtained. These probabilities will be used to distribute the users with overlapping through the different local nodes of the bagging recommender algorithm.

Two approaches were considered for distributing the users with overlapping through the local nodes: *i*) based on the soft cluster membership probability comparison with a uniform cluster probability atribution (an uniform probability equals the inverse of the total number of clusters and *ii*) based on the top-*n* highest soft cluster membership probability values.

### 4.2.1 Comparison to the uniform distribution probability

Different thresholds were here considered ranging for instance from 85 % to 100 % of the uniform probability value when soft clustering is performed with HDBSCAN. A user in considered to be present in a particular cluster if, for instance, the soft membership probability for the user in a particular cluster is higher than 95 % of the uniform cluster probability. The different thresholds considered and the final number of clusters will be shown to affect strongly the overlapping of users in the different clusters. The average overlapping percentage after threshold definition was then also considered as a parameter of study on the recommender model´s accuracy.

Figure 12 shows the number of users distributed through each cluster if the soft membership probability of a user belong to a cluster is higher than a uniform cluster distribution probability. The soft cluster memberships were obtained from a HDBSCAN-based clustering algorithm for the PLC-playlist dataset with a Jaccard distance used in the threshold diameter and the neighbourhood contains at least a minimal number of points = 60. In this case, the number of clusters (*n*) obtained is 10 and the uniform cluster distribution probability is =1/*n* = 1/10 = 0.1. From user counts in each cluster the average overlapping percentage was calculated. Figure 12 shows that the number of users is each cluster is not very uniformly distributed once one cluster contains significantly more users than other clusters and an average overlapping percentage of 18.8 % was then calculated for this case, thus indicating that the overlapping percentage is not very high.

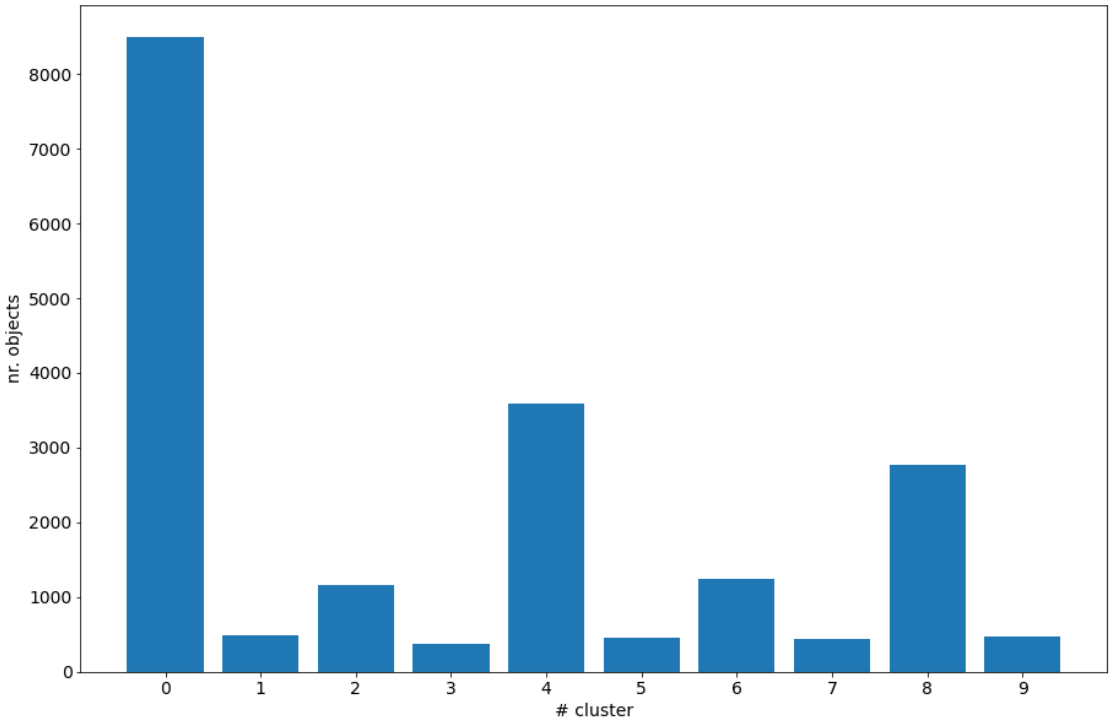



*Figure 12 – Total count of users in each of the 10 clusters obtained by using HDBSCAN on the utility matrix from the PLC-playlist dataset and allowing user overlapping based on comparison to a uniform cluster attribution.*

Figure 13 shows the equivalent plot of the results above but now for a threshold of 97.5 %, i.e 97.5 % of the value of the uniform cluster distribution probability is =97.5% x 1/*n* = 97.5% x 1/10 = 0.0975. One can see that the number of users in each cluster is more uniformly distributed indicating that a higher user overlapping through the 10 clusters is highly probable. Indeed, the average overlapping percentage calculated for these conditions is 58.5 %.

Table 4 shows the average overlapping percentages obtained for each of the thresholds considered for the PLC-playlist and the PLC-STR datasets and for a different final numbers of clusters. It was observed that lower fractions (low thresholds) of the uniform cluster distribution probability and a higher number of clusters favour a higher average overlapping of the users in the different clusters for both the datasets considered.

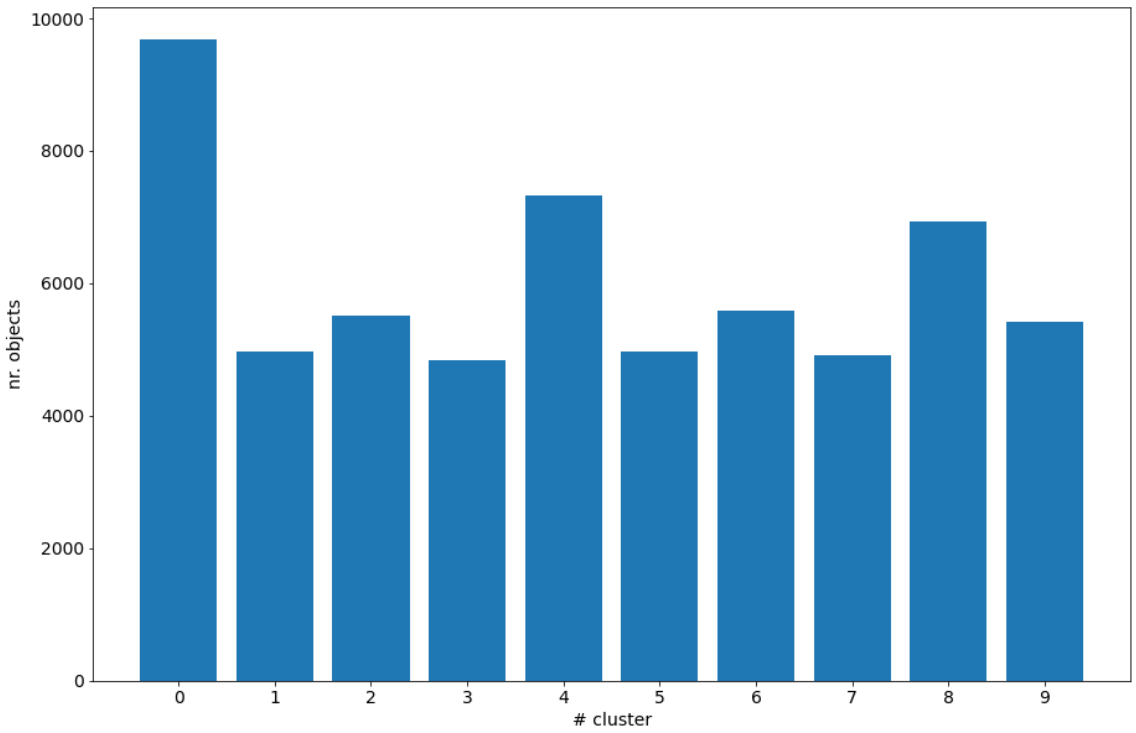



*Figure 13 - Total count of users in each of the 10 clusters obtained by using HDBSCAN on the utility matrix from the PLC-playlist dataset and allowing user overlapping based on comparison to 97.5 % of the uniform cluster attribution probability value.*

*Table 4 – Average user overlapping in the clusters from PLC-playlist and PLC-STR datasets (some experimental conditions were not considered).*

| ***Dataset*** | ***Threshold / %*** | ***Overlapping / %*** | | |
|---|---|---|---|---|
| | | ***n* = 6** | ***n* = 10** | ***n* = 23** |
| PLC- playlist | 100.0 | 21.4 | 18.8 | 17.5 |
| | 99.0 | - | - | 46.7 |
| | 98.5 | - | - | 61.7 |
| | 98.0 | - | - | 71.5 |
| | 97.5 | - | 58.5 | 79.0 |
| | 97.0 | - | 65.4 | - |
| | 95.0 | 64.0 | 83.5 | 95.2 |
| | 90.0 | 80.8 | 98.0 | 98.5 |
| | 85.0 | - | 98.6 | - |
| ***Dataset*** | ***Threshold / %*** | ***n* = 5** | ***n* = 7** | ***n* = 11** |
| PLC- STR | 100.0 | 11.1 | 14.7 | 19.8 |
| | 97.0 | - | - | - |
| | 97.5 | - | 45.2 | - |
| | 97.0 | - | | 63.6 |
| | 95.0 | 36.0 | 64.0 | 78.0 |
| | 92.5 | 41.1 | 80.7 | - |
| | 90.0 | 49.5 | - | - |
| | 87.5 | 63.1 | - | - |
| | 85.0 | 79.9 | - | - |

Tables 5 and 6 show the first results of the accuracy (recall@20) from the local ensemble recommender model (cluster-bagging recommender algorithm) considering the PLC-playlist and PLC-STR datasets and comparing with the base-, instance bagging- and user-bagging-recommender models at different number of nodes. One can see that the cluster-based bagging recommender algorithm performs quite poorly on both datasets with recall@20 values consistently bellow the user-based bagging algorithm for different number of clusters and different threshold values (average user overlapping).

*Table 5 - Recall@20 for the PLC-playlist dataset recommender models at different numbers of nodes.*

| ***# of nodes*** | ***Base*** | ***Instance-based*** | ***User-based*** | ***Cluster-based*** *Overlapping/ % (threshold/ %)* | | | |
|---|---|---|---|---|---|---|---|
| - | 0.248 | - | - | - | | | |
| 6 | | | | *64 (95)* | | *81 (90)* | |
| | - | 0.248 | 0.268 | 0.173 | | 0.229 | |
| 10 | | | | *65 (97)* | *83 (95)* | *98 (90)* | *98.5 (85)* |
| | - | 0.272 | 0.287 | 0.241 | 0.250 | 0.276 | 0.277 |
| 23 | | | | *62 (98.5)* | *95 (95)* | | *98.5 (90)* |
| | - | 0.290 | 0.307 | 0.250 | 0.294 | | 0.284 |

*Table 6 - Recall@20 for the PLC-STR dataset recommender models at different numbers of nodes.*

| ***# of nodes / n*** | ***Base*** | ***Instance-based*** | ***User-based*** | ***Cluster-based*** *Overlapping/ % (threshold/ %)* |
|---|---|---|---|---|
| - | 0.326 | - | - | - |
| 5 | | | | *63 (87.5)* |
| | - | 0.302 | 0.347 | 0.317 |
| 7 | | | | *64 (95)* |
| | - | 0.321 | 0.363 | 0.202 |
| 11 | | | | *63 (97)* |
| | - | 0.342 | 0.381 | 0.208 |

### 4.2.2 Top-*n* highest soft cluster membership

The second method used to allow user overlapping on the different clusters in based on selecting the top highest cluster membership probabilities from a user and allow the user belong only to those clusters where such criteria is met. As before, different thresholds were here considered for selecting the top probabilities ranging from 10 % to 90 % of the number of clusters.

Table 7 shows the accuracy (recall@20) comparison for the different algorithms when considering different number of nodes and threshold values in the cluster-based algorithm for the PLC-playlist and PLC-STR datasets. One can see that accuracy for the cluster-based bagging recommender algorithm mostly increases with number of nodes and average user cluster overlapping percentage (threshold). Moreover, mainly for a lower number of clusters and medium thresholds values, the cluster-based algorithm performs better (bold highlighted) than the user-based algorithm, for both datasets.

*Table 7 – Recall@20 for different algorithms and thresholds for the PLC-playlist and PLC-STR datasets. For the cluster-based algorithm between brackets is the average user overlapping percentage (some experimental conditions were not considered).*

| ***Algorithm*** | **PLC-playlist** | | | | **PLC-STR** | | | |
|---|---|---|---|---|---|---|---|---|
| Base | 0.248 | | | | 0.326 | | | |
| | ***n* = 5** | ***n* = 6** | ***n* = 10** | ***n* = 23** | ***n* = 3** | ***n* = 5** | ***n* = 7** | ***n* = 11** |
| Instance- | 0.235 | 0.248 | 0.272 | 0.290 | 0.272 | 0.302 | 0.321 | 0.342 |
| User- | 0.250 | 0.268 | 0.287 | 0.307 | 0.302 | 0.347 | 0.363 | 0.381 |
| Cluster- | **PLC-playlist** | | | | **PLC-STR** | | | |
| *Thresh/ %* | ***n* = 5** | ***n* = 6** | ***n* = 10** | ***n* = 23** | ***n* = 3** | ***n* = 5** | ***n* = 7** | ***n* = 11** |
| 10 | - | - | - | - | - | - | - | 0.013 (9) |
| 20 | - | - | - | - | - | - | - | 0.193 (18) |
| 30 | - | - | - | - | - | - | - | 0.316 (27) |
| 40 | - | - | - | - | - | - | - | 0.364 (36) |
| 50 | - | **0.270 (49)** | 0.277 (49) | 0.290 (52) | - | 0.298 (40) | 0.354 (57) | 0.377 (54) |
| 66 | **0.272 (60)** | **0.279 (66)** | 0.280 (69) | 0.285 (65) | **0.319 (60)** | **0.354 (60)** | 0.355 (71) | 0.368 (64) |
| 75 | - | - | - | - | - | - | - | 0.361 (73) |

These preliminary results have motivated the research towards a soft clustering algorithm that allows a precise control of the final number of clusters for better understanding

both the influence of user cluster overlapping and the number of clusters on the cluster-based bagged recommender model. For that, NMF clustering algorithm was considered when using as input parameter the number of clusters ranging from 4 to 32. For the local bagging recommender model, besides inputting the number of clusters/nodes, threshold values in the middle range from 40 % to 70 % were used. This study was also extended to 3 additional datasets: ML1M, LFM-50U and YHM-6UK.

Figures 14 to 18 show that the local-based ensemble model indeed outperforms the best performing user-based ensemble recommender under some circumstances. Major improvements seems to be observed for the datasets with lowest sparsity, indicating that most probably stronger feedback preferences for each user might improve clustering performance and thus improve the local-based ensemble model. The two datasets with lower sparsity rates (Table 2) are ML1M and LFM-50U. LFM-50 has the lower number of users while the ML1M dataset has the lowest number of items from the datasets considered. This particularity of those datasets may allow an improved clustering performance of the NMF algorithm.

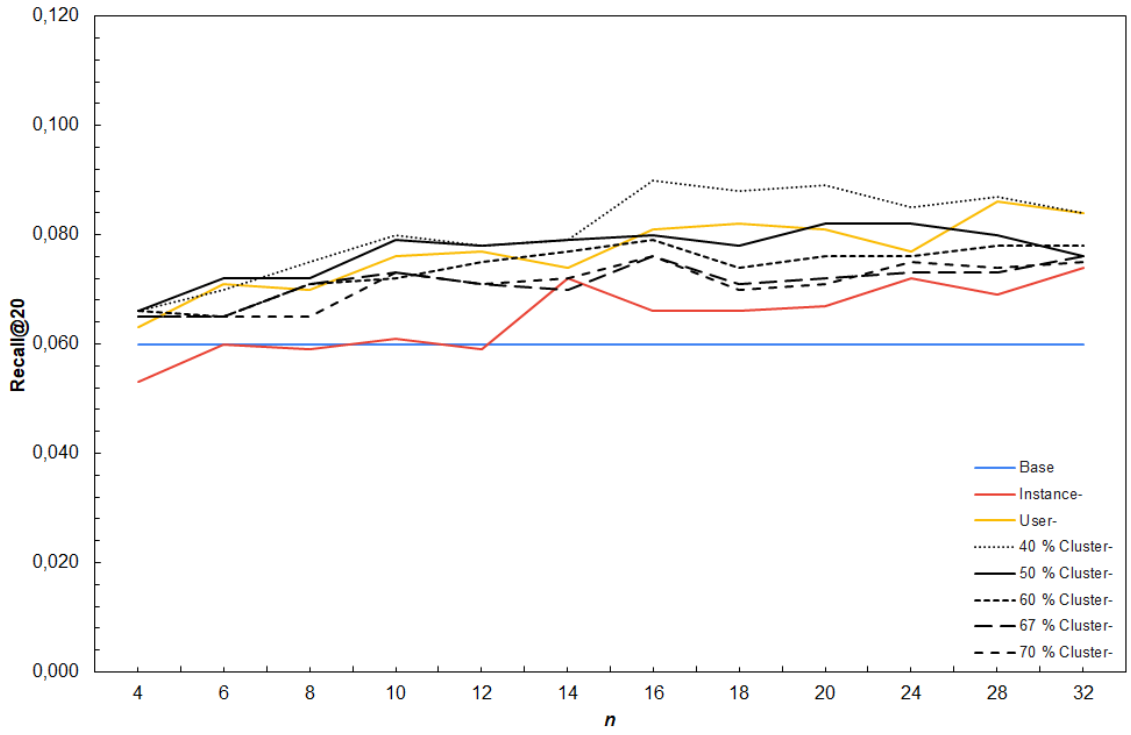



*Figure 14 – Average recall@20 for the ML1M dataset recommender models.*

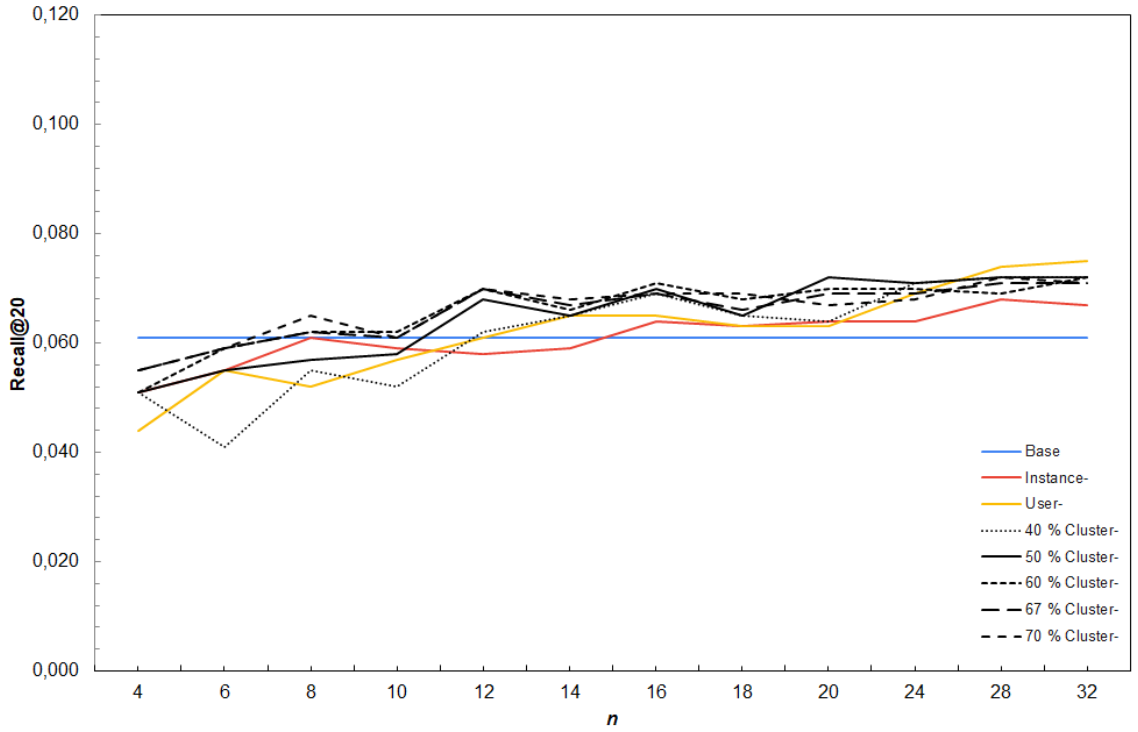



*Figure 15 - Average recall@20 for the LFM-50U dataset recommender models.*

For the ML1M dataset (Figure 14), the usage of a 40 % threshold improves accuracy when compared to the user-based ensemble recommender. For the 50 % threshold an improved accuracy is observed up to 14 nodes and afterwards model accuracy behaves intermittently. For the other thresholds, local ensemble performance is placed between the performances for the instance-based and user-based bagged algorithms.

For the LFM-50U dataset (Figure 15), recall@20 for the local ensemble is systematically higher than all the other ensemble algorithms compared except under some circumstances at lower and higher number of nodes, for all thresholds. Additionally, at lower bootstrap nodes all the ensemble algorithms perform worse than the base algorithm.

Regarding the YHM-6UK dataset (Figure 16), local ensemble nearly matches the performance of the user-based ensemble and it presents a better performance than the instance-based bagged algorithm for all nodes. At the lowest number of nodes considered the local ensemble outperforms all the other ensemble models for all thresholds.

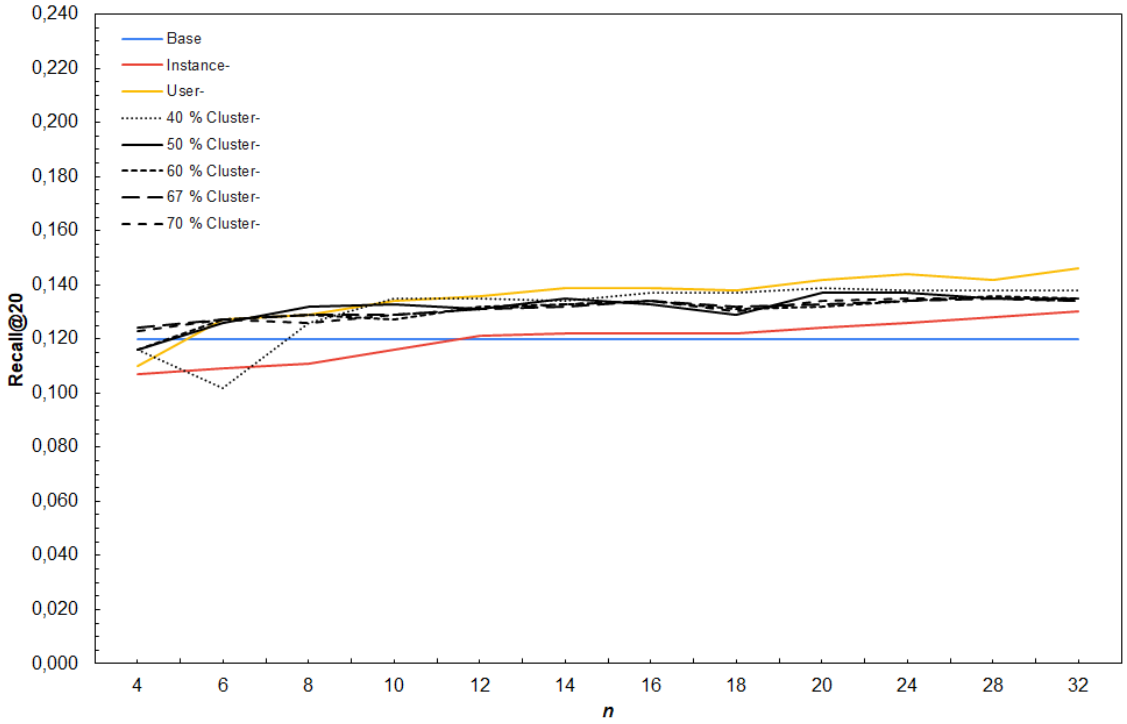



*Figure 16 - Average recall@20 for the YHM-6UK dataset recommender models.*

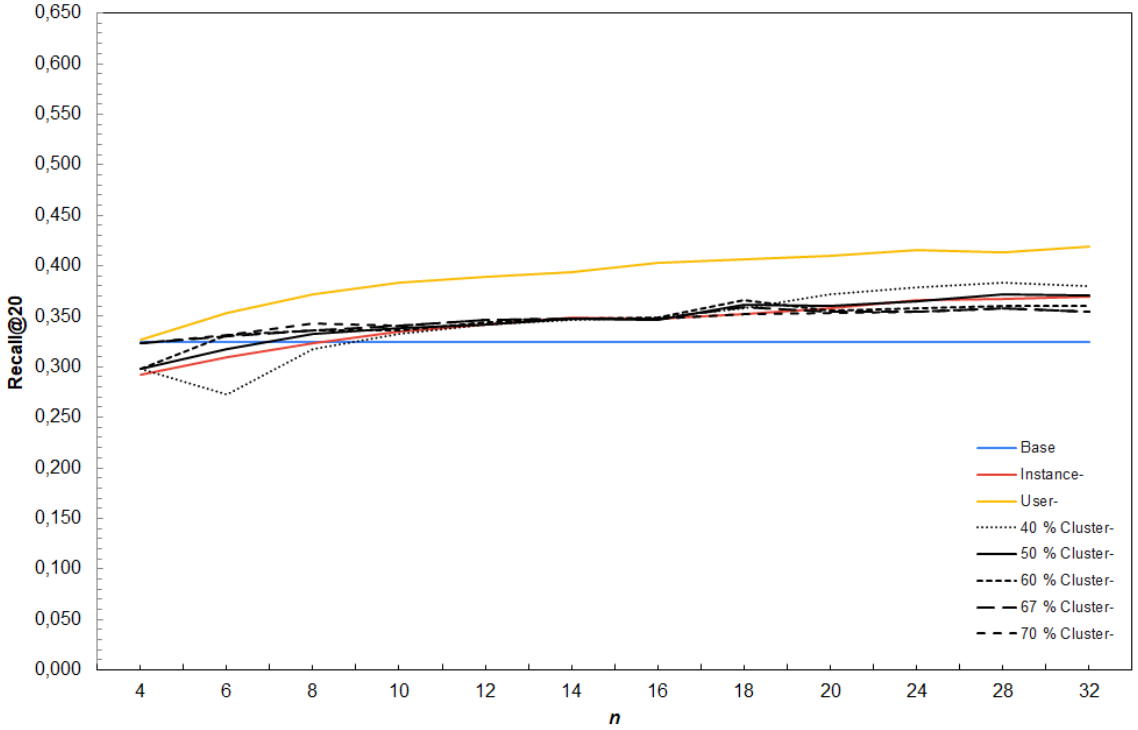



*Figure 17 - Average recall@20 for the PLC-STR dataset recommender models.*

Finally, for the PLC-STR dataset (Figure 17), performances of all the local ensemble models are always bellow the user-based ensemble benchmark and very similar to the instance-based bagged algotithm. On the other hand, for the PLC-playlist (Figure 18), local ensemble models perform better at lower numbers of bootstrap nodes (4 and 6) except for the 40 % threshold and afterwards all the ensemble models' performances are similar and outperform the base algorithm.

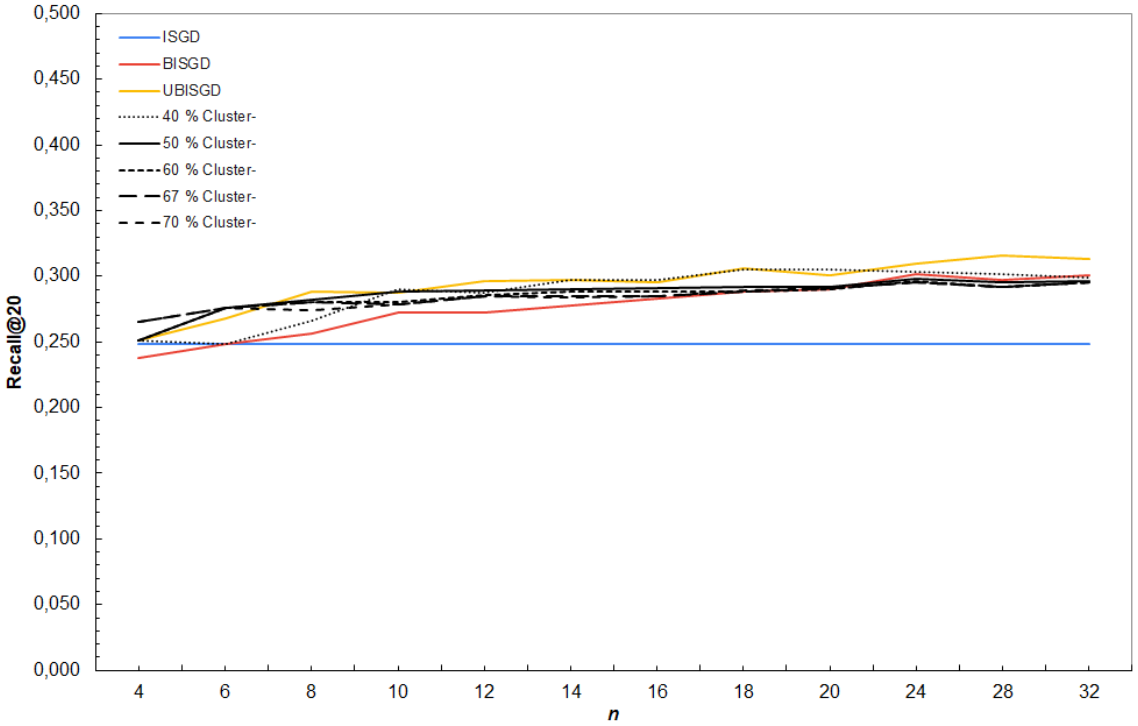



*Figure 18 - Average recall@20 for the PLC-playlist dataset recommender models.*

# 5. Conclusions and future directions

## 5.1 Conclusions

It is known that there is a vast number of clustering algorithms currently available; navigating through this subject was a very deep training experience for the candidate but some narrowing conditions had to be set. One very relevant condition was the use of a fuzzy approach clustering algorithm that showed to be a quite limiting one indeed. Despite of that two of the three fuzzy algorithms experimentally tested successfully returned membership probabilities different from a simple uniform distribution. Moreover, some crisp algorithms were considered in order to infer if a certain type of clustering algorithm could be significantly outstanding from the other types of algorithms.

This way, from the hard clustering algorithms, distance- and hierarchical- based algorithms failed to discriminate different groups of users with a single cluster being mainly obtained. The density-based algorithm DBSCAN indeed provided the first clustering method which outcome was different from a big single cluster and it was also able to detect outliers in the testing dataset considered. This result lead to deepen the search on density-based algorithms and an improved version of the DBSCAN that also included cluster membership probability became focus of research. One disadvantage of HDBSCAN besides time consuming is that the number of final clusters is not possible to be predefined, what in this particular research is actually required.

Results from the soft clustering with HDBSCAN were then concatenated with the ensemble bagged recommender model and promising initial results were obtained, however the type of criterion used to overlap users through the different local nodes became critical. When such overlapping user distribution through the nodes is based on a proportion of the uniform distribution probability, the cluster-based bagging algorithm did not present improved results. However, when the comparison is based on the top-*n* highest soft cluster membership probability values where *n* is based on a proportion of the number of clusters, results showed to be promising on some specific conditions, namely at lower numbers of cluster groups for two different datasets and midle thresholds (proportions).

In order to study further such preliminary results and dedicating some time to a more detailed background understanding of the matrix factorization approach used in the incremental recommender algorithm, it became clear that NMF could also provide soft cluster membership probabilities under certain conditions and on top of that, the final number of clusters could be previously defined. This algorithm then allowed a stronger analysis under different conditions of the local ensemble recommender model. Results observed showed that local-based ensemble

model indeed outperforms the best performing user-based ensemble recommender. However, such improvement is not observed across all total number of nodes on the ensemble considered, neither for all the datasets tested. Major improvements are observed for the datasets with lowest sparsity, indicating that most probably stronger feedback preferences for each user might improve clustering performance and thus improve the local-based ensemble model. The two datasets with lower sparsity rates are ML1M and LFM-50U. LFM-50 has the lower number of users while the ML1M dataset has the lowest number of items from the datasets considered. This particularity of those datasets may have allowed an improved clustering performance of the NMF algorithm.

## 5.2. Future directions

In this work, an incremental version of the NMF clustering algorithm suitably integrated with the incremental ensemble recommender algorithm would have been pursued if not deadline constrains. Nevertheless, this is a quite exciting next step for the future of this research line. Very few research papers indeed address incremental MF on recommender systems that could be adapted for NMF addressing this local-based recommender model problem [7, 17], however as no open code is available, the target algorithm strategies will have to be developed nearly from scratch and because of that it can be quite time consuming. The present work has focused more on proving the concept of this research line instead of starting to developing a new incremental clustering NMF algorithm without further evidence that such approach would indeed be relevant. Moreover, an incomplete algorithm could face a high probability of being delivered at deadline.

Furthermore, we have made preliminary experiments - not herein presented - when developing an informal incremental clustering algorithm based on matrix factorization but not yet mathematically supported. These very preliminary results also show evidence to instigate further research on the development of a suitable coded and mathematically supported incremental NMF algorithm appropriate to be integrated with the available ensemble recommender.

Another interesting suggestion for future work could be further deeply infer the numerical methods behind NMF and *fuzzy c-means* and try to better understand both numerically and theoretically, for namely addressing why *fuzzy c –means* and NMF converge to different results even though both algorithms minimize the same objective function. This way an already available incremental fuzzy clustering algorithm would only require an adaptation to

a recommendation problem. If so two different clustering algorithms would be available to be integrated with the ensemble model.

# 6. References

1. Koren, Y., R. Bell, and C. Volinsky, *Matrix Factorization Techniques for Recommender Systems.* Computer, 2009. **42**(8): p. 30-37.
2. Gomez-Uribe, C.A. and N. Hunt, *The Netflix Recommender System: Algorithms, Business Value, and Innovation.* ACM Transactions on Management Information Systems, 2016 **6**(4, Article No.: 13): p. pp 1–19.
3. Miranda, C. and A.M. Jorge, *Item-Based and User-Based Incremental Collaborative Filtering for Web Recommendations.* Progress in Artificial Intelligence, Proceedings, 2009. **5816**: p. 673-+.
4. Komkhao, M., et al., *Incremental collaborative filtering based on Mahalanobis distance and fuzzy membership for recommender systems.* International Journal of General Systems, 2013. **42**(1): p. 41-66.
5. Vinagre, J., A.M. Jorge, and J. Gama, *Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback.* User Modeling, Adaptation, and Personalization, Umap 2014, 2014. **8538**: p. 459-470.
6. Matuszyk, P., et al., *Forgetting Methods for Incremental Matrix Factorization in Recommender Systems.* 30th Annual Acm Symposium on Applied Computing, Vols I and Ii, 2015: p. 947-953.
7. Yu, T., et al., *Incremental Learning for Matrix Factorization in Recommender Systems.* 2016 Ieee International Conference on Big Data (Big Data), 2016: p. 1056-1063.
8. Vinagre, J., A.M. Jorge, and J. Gama, *Online bagging for recommender systems.* Expert Systems, 2018. **35**(4).
9. Laohakiat, S. and V. Sa-ing, *An incremental density-based clustering framework using fuzzy local clustering.* Information Sciences, 2021. **547**: p. 404-426.
10. Christakopoulou, E. and G. Karypis, *Local Item-Item Models for Top-N Recommendation.* Proceedings of the 10th Acm Conference on Recommender Systems (Recsys'16), 2016: p. 67-74.
11. Christakopoulou, E. and G. Karypis, *Local Latent Space Models for Top-N Recommendation.* Kdd'18: Proceedings of the 24th Acm Sigkdd International Conference on Knowledge Discovery & Data Mining, 2018: p. 1235-1243.
12. Al-Ghossein, M., T. Abdessalem, and A. Barré, *Dynamic local models for online recommendation*, in *Workshop on Online Recommender Systems and User Modeling*. 2018, IW3C2: Lyon, France.

13. Aggarwal, C.C., *Data Mining: The Textbook*. 2015: Springer.
14. Singh, M., *Scalability and sparsity issues in recommender datasets: a survey.* Knowledge and Information Systems, 2020. **62**(1): p. 1-43.
15. Lazar, C. and A. Doncescu, *Non Negative Matrix Factorization Clustering Capabilities; Application on Multivariate Image Segmentation.* Cisis: 2009 International Conference on Complex, Intelligent and Software Intensive Systems, Vols 1 and 2, 2009: p. 924-+.
16. Jahrer, M., A. Tosher, and R. Legenstein, *Combining predictions for accurate recommender systems*, in *KDD´10*. 2010, ACM: Washington, DC, USA.
17. Lin, C.Y., L.C. Wang, and K.H. Tsai, *Hybrid Real-Time Matrix Factorization for Implicit Feedback Recommendation System.* Ieee Access, 2018. **6**: p. 21369-21380.
18. Efron, B. and R. Tibshirani, *Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy.* Statist. Sci., 1986. **1**(1): p. 54-75.
19. Oza, N.C. and S.J. Russel, *Online bagging and boosting*, in *Proceedings of the Conference on Artificial Intelligence and Statistics*. 2001, Morgan Kaufmann: Key West, FL, USA. p. 105-112.
20. Silva, J.A., et al., *Data Stream Clustering: A Survey.* Acm Computing Surveys, 2013. **46**(1).
21. Hore, P., L. Hall, and D. Goldgof, *A cluster ensemble framework for large data sets.* 2006 Ieee International Conference on Systems, Man, and Cybernetics, Vols 1-6, Proceedings, 2006: p. 3342-+.
22. Hore, P., L.O. Hall, and D.B. Goldgof, *Single pass fuzzy c means.* 2007 Ieee International Conference on Fuzzy Systems, Vols 1-4, 2007: p. 240-246.
23. Hore, P., L.O. Hall, and D.B. Goldgof, *A fuzzy c means variant for clustering evolving data streams.* 2007 Ieee International Conference on Systems, Man and Cybernetics, Vols 1-8, 2007: p. 802-807.
24. Hore, P., et al., *Online fuzzy c means.* 2008 Annual Meeting of the North American Fuzzy Information Processing Society, Vols 1 and 2, 2008: p. 173-177.
25. Abdullatif, A., F. Masulli, and S. Rovetta, *Clustering of nonstationary data streams: A survey of fuzzy partitional methods.* Wiley Interdisciplinary Reviews-Data Mining and Knowledge Discovery, 2018. **8**(4).
26. Rodrigues, P.P., J. Gama, and J.P. Pedroso, *Hierarchical Clustering of Time-Series Data Streams.* IEEE Transactions on Knowledge and Data Engineering 2008. **20**(5): p. 615 - 627.

27. Dell'Amico, M., *FISHDBC: Flexible, Incremental, Scalable, Hierarchical Density-Based Clustering for Arbitrary Data and Distance.* arXiv.org - Cornell University, 2019. **arXiv:1910.07283v1 [cs.LG]**: p. 1-10.

28. McInnes, L., J. Healy, and S. Astels. *The hdbscan Clustering Library*. 2016 [cited 2021 Jun, 2021]; Available from: https://hdbscan.readthedocs.io/en/latest/.

29. Li, T. and C. Ding, *The relationships among various nonnegative matrix factorization methods for clustering.* Icdm 2006: Sixth International Conference on Data Mining, Proceedings, 2006: p. 362-371.

30. Bucak, S.S. and B. Gunsel, *Incremental Clustering via Nonnegative Matrix Factorization.* 19th International Conference on Pattern Recognition, Vols 1-6, 2008: p. 640-643.

31. Bucak, S.S. and B. Gunsel, *Incremental subspace learning via non-negative matrix factorization.* Pattern Recognition, 2009. **42**(5): p. 788-797.

32. Ding, C., et al., *On the Equivalence of Nonnegative Matrix Factorization and K-means - Spectral Clustering.* Lawrence Berkeley National Laboratory, 2005.

33. Connor, M. and J. Herlocker, *Clustering items for collaborative filtering*, in *ACM SIGIR Workshop on Recommender Systems*. 1999, https://www.csee.umbc.edu/~ian/sigir99-rec/: University of California, Berkeley, USA.

34. Koren, Y., *Factorization meets the neighborhood: a multifaceted collaborative filtering model*, in *KDD '08: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, ACM Digital Library. p. 426–434.

35. Xu, B., et al., *An exploration of improving collaborative recommender systems via user-item subgroups*, in *21st International conference on World Wide Web*, ACM, Editor. 2012. p. 21-30.

36. Weston, J., R.J. Weiss, and H. Yee, *Nonlinear latent factorization by embedding multiple user interests*, in *7th ACM conference on Recommender systems*, ACM, Editor. 2013: Hong Kong, China. p. 65-68.

37. Lee, J., et al., *Local low-rank matrix approximation*, in *30th International Conference on Machine Learning*, ACM, Editor. 2013: Atlanta, USA. p. 82-90.

38. Lee, J., et al., *Local Collaborative Ranking.* Www'14: Proceedings of the 23rd International Conference on World Wide Web, 2014: p. 85-95.

39. Siddiqui, Z.F., et al., *xStreams: Recommending Items to Users with Time-evolving Preferences.* 4th International Conference on Web Intelligence, Mining and Semantics, 2014.

40. Khawar, F. and N.L. Zhang, *Modeling Multidimensional User Preferences for Collaborative Filtering*, in *2019 Ieee 35th International Conference on Data Engineering*. 2019. p. 1618-1621.