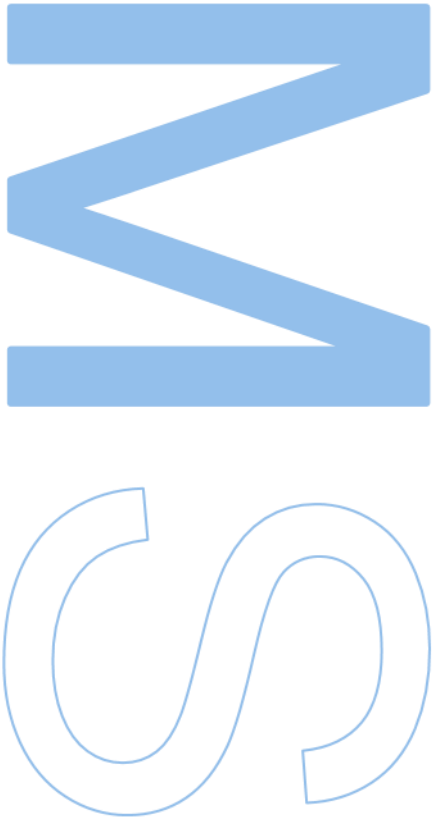
Locality-sensitive ensembles for recommender systems



Maria Gomes Vale de Melo
Master's degree in Computer Science
Computer Science Department
2022

Orientador
João Vinagre, Professor Auxiliar Convidado, Faculdade de Ciências da Universidade do Porto


U.PORTO
FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Abstract

One of the main goals of a recommender system is to in some way guide the user to products or services that match the user's preferences. Even though data is always being collected and the information on a user keeps evolving, most systems encountered presently train their models in batch settings, not taking advantage of the live stream of data.

Since it is reasonable to group similar users, to capture local phenomena, it is also intuitive to model this subset of users independently to obtain better results for different types of users. In this thesis we propose a connection of this three concepts, an incremental clustering which will feed a incremental ensemble method whose combined results produce improved recommendations.

Different types of algorithms were taken into considerations, keeping three requirements in mind. The clustering algorithm should be incremental, fuzzy – so it can provide soft membership probability –, and we should be able to control the number of clusters so it coincided with the number of ensemble nodes. A non-negative incremental matrix factorization clustering algorithm and an fuzzy adaptation of the DBSCAN (Density-based spatial clustering of applications with noise) algorithm with control of clusters were the more suitable ones for the intended application.

There were select 4 different recommendation datasets. The clusters produced by the two different algorithms for all 4 datasets were analysed and incrementally used to populate the ensemble's nodes, the result of the recommendations produced were evaluated using a recall metric – Recall@20 – using prequential evaluation.

Both algorithms show improved results, when compared to state-of-the-art algorithms. In the case of the non-negative matrix factorization algorithm there were improvements for 3 of the 4 datasets. As for the DBSCAN based algorithm there were improvements in all 4 datasets.

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

Resumo

Com a explosão da internet e da informação nas últimas décadas, os sistemas de recomendação provaram ser uma forma eficaz e muito lucrativa de usar dados. Esta rentabilidade incentivou a progressão nesta área de estudo, com novos métodos e melhorias.

Um dos principais objetivos de um sistema de recomendação é, de alguma forma, orientar o utilizador para um produto que se assemelhe às preferências do usuário. Mesmo dado o constantes fluxo de dados recolhidos e a evolução dos dados de um usuário, a maioria dos sistemas encontrados atualmente treina seus modelos em *batch*, não aproveitando a entrada constante de dados. Além disso, foi estabelecido há muito tempo que podemos usar interações de utilizadors semelhantes para recomendar a um determinado utilizador - *collaborative filtering*.

Visto que é razoável agrupar utilizadores semelhantes, para capturar fenômenos locais, também é intuitivo modelar esse subconjunto de usuários de forma independente para obter melhores resultados para diferentes tipos de usuários. Nesta tese propomos uma conexão desses três conceitos, um agrupamento incremental que alimentará um método de ensemble incremental cujos resultados combinados produzem recomendações mais eficazes.

Diferentes tipos de algoritmos foram levados em consideração, mantendo três requisitos em mente. O algoritmo de *clustering* deve ser incremental, *fuzzy* para que possa fornecer probabilidade de associação *soft*, e devemos ser capazes de controlar o número de *clusters* para que coincida com o número de nós do *ensemble*. Um algoritmo de *clustering* incremental de factorização de matrizes não negativas e uma adaptação *fuzzy* do algoritmo DBSCAN com controle do número de *clusters* encontrados que melhor se adquearam à aplicação pretendida.

Foram selecionados 4 conjuntos de dados diferentes, adequados para recomendação. Os *clusters* produzidos pelos dois algoritmos diferentes para todos os 4 conjuntos de dados foram analisados e usados incrementalmente para popular os nós do *ensemble*, o resultado das recomendações produzidas foram avaliados usando uma métrica de *recall*, Recall@20 num método de avaliação prequencial.

Ambos os algoritmos obtiveram resultados positivos, no caso do algorithm baseado na factorização de matrizes não negativas melhorias foram registadas para 3 dos 4 data-sets. Já no caso do algoritmo baseado no algoritmo DBSCAN (Density-based spatial clustering of applications with noise) foram registadas melhorias para os 4 data-sets quando comparados aos algoritmos

state-of-the-art, sobre as circuntâncias de teste determinadas. Embora com algumas limitações, em particualar a complexidade temporal do algorithm baseado no clássico DBSCAN.

1

[1]Keywords: sistemas de recomendação, algoritmos incrementais, *clustering*, modelos *ensemble*, modelos locais

Acknowledgements

Both this dissertation and my journey so far in the higher education system have been specially marked by the people who accompanied me in these endeavors.

I would firstly like to thank the professors and colleagues who supported and inspired me through these 5 years at FCUP. A special thank to Prof. João Vinagre for guiding me in this dissertation, and for always being available.

Finally, a very special thanks to my family for their unwavering belief in me.

Contents

List of Tables

List of Figures

Acronyms

DBSCAN Density-Based Spatial Clustering of Applications with Noise

NMF Non-negative matrix factorization

SGD Stochastic Gradient Descent

SVD Singular Value Decomposition

ISGD Incremental Stochastic Gradient Descent

BISGD Bagging Incremental Stochastic Gradient Descent

UBISBGD User Based Bagging Incremental Stochastic Gradient Descent

CLUBISBGD Cluster-based User Based Bagging Incremental Stochastic Gradient Descent

Chapter 1

Introduction

Technology is quickly integrating most aspects of our lives, from the purchases we make to our social networks, with the now wide access to the internet we seem to leave behind data on most of our actions. Matters that were usually of concern to soft sciences, such as social-dynamics or finances, became computable given the large availability of data.

Recommender systems are a particular field, enabled by the growing size of available datasets, that has been gaining relevance over the past decades. With platforms as Netflix and Amazon relying heavily on these systems, using the enormous amount of data generated by their users to assess their choices and try to show them products they want to see. And all this for a very good reason, in 2016 Netflix evaluated their recommender system on $1 billion per year, this system is so valuable because it keeps customers from canceling, more so a great portion of the platform traffic is generated by these recommendations, keeping users engaged. In [5] the following statement was made: "Consumer research suggests that a typical Netflix member loses interest after perhaps 60 to 90 seconds of choosing, having reviewed 10 to 20 titles (perhaps 3 in detail) on one or two screens," "The user either finds something of interest or the risk of the user abandoning our service increases substantially".

Given the value of these systems, this area has been the subject of attention of many studies, and as a result there are constant improvements.

1.1 Motivation

Although this is a broadly studied area there are problems yet to solve.

We intuitively know that these gigantic data set are constantly changing either by adding or deleting data, and constantly growing. Every time a user sees on their screen a product or clicks on it, new data is generated. It would be expected that these interactions would affect the recommender system real time, but most recommendation systems are still executed in batch, modeling the whole data set in scheduled periods instead of updating an already

calculated model. There are two flagrant problems with this approach: (1) the user does not get a customized experience of their actions in real time, and (2) there are computational resources being recurrently consumed for the same computation on a slightly altered data set.

In this dissertation we will be focusing on incremental ensemble models. Ensembles have been shown to improve various machine learning tasks [7] with significant increase in accuracy. Taking into consideration the promise of ensemble models and the necessity of adapting recommender systems to the stream environment of today, incremental ensemble models seem to be a key factor in the production of efficient incremental recommender models. There has been still little exploration in this field, thus its investigation holds much promise.

When it comes to recommender systems collaborative filtering is one of the main approaches used when designing them, in these systems, items are recommended to users based on the content of the items and target user's ratings, item based collaborative filtering. On the other hand it can makes use of the information on different similar users to recommend to the target user, this is known as user based collaborative filtering.

In either of these approaches of collaborative filtering, either items or users that are similar would naturally aggregate according to their characteristics, local phenomena has different patterns of behavior must be model together by the system. Even thought it has already been shown that local models help improve recommender systems [11][1], ensemble techniques are not designed to be locality-sensitive.

The combination of these techniques, collaborative filtering, ensemble learning, the exploitation of local models using clustering, in an incremental application is a natural next step in the field.

1.2 Goals of Research

In this dissertation it is intended to develop an algorithm that combines an previously developed online ensemble recommender system and a novel incremental clustering technique to exploit local phenomena.

By doing so, our goals for this paper are the following:

1. Review the state-of-the-art on incremental clustering, ensemble recommender systems and local models for recommendation
2. Study and develop a fully incremental ensemble method, able to learn and combine local models
3. Carry out empirical studies on real-world datasets, and compare our method with alternatives

To reach this, we settled the following sub-problems:

1. Adapt or develop new clustering algorithms that meet the following requirements:
   (a) The clustering algorithm must be *incremental*
   (b) The clustering algorithm must be *fuzzy*, in order to use the degree of belonging to a certain cluster to infer the distribution of each pair user-item across the nodes of the ensemble
   (c) The clustering algorithm must allow the *control of the number of clusters*, for the same reason it mus be fuzzy, to allow the inference of the distribution of the data points across the ensemble
2. Develop an online ensemble algorithm able to capture and adapt to local phenomena, from an existing online ensemble algorithm and the developed incremental clustering algorithm.
3. Assess the resulting algorithm on real-world data.

1.3 Contributions

This thesis contributions are:

1. A novel online ensemble learning technique for recommendation
2. An extension of streamRec[1], an existing python library for online recommendation

1.4 Thesis Outline

This dissertation is organized in 7 chapters, as follows: in Chapter 2 we explore core concepts to the development of the proposed contributions. We delve into different branch's of Machine Learning, with a particular focus in clustering methods, looking into different approaches to clustering . Besides clustering Ensemble methods are also explored in this chapter, exploring the concepts of Bagging, Boosting and Stacking. The last topic explored in this chapter is Recommender Systems, taking particular attention to the use of Matrix Factorization in these methods.

In following the chapter, Chapter 3, we delve into the current literature on the diverse matters we connect in these dissertation. Looking into different approaches to incremental clustering, incremental ensemble recommender systems and locality sensible applications of those same recommender systems. The chapter is closed of with an assessment of challenges expected to face in the development of the new contributions, given the state of the art.

Chapter 4 details the functioning of the proposed algorithms, starting of by the selection of the clustering algorithms to be used as a base for the development. The adaptation of

[1] https://github.com/joaoms/streamRec

the two selected algorithms is then described, and their integration in a user based bagging recommendation algorithm detailed.

To assess the quality of the proposed algorithms, in Chapter 5 the select datasets for evaluation are described, and the methodology in each the algorithms are evaluated is presented. The results of the subjection of the algorithms to said evaluation are presented, in the order and manner mandated by the presented methodology.

The presented results in Chapter 5 are analysed and discussed, in order to culminate in the final Chapter 6 in determination of final assessments in on the work developed and delineation of pertinent questions to be explored in the future, either to fix some limitations of the presented algorithms or to improve their performance.

Chapter 2

Background

In this chapter we will briefly explore some key subjects that represent base knowledge for the understanding of the dissertation.

These subject all fall under the umbrella of Machine Learning, we will be focusing mainly on three Machine Learning problems. We will start by introducing the subject of Machine Learning in itself, followed by the first concrete Machine Learning problem, Clustering which is a central point of this dissertation's work. We will then move on to Ensemble Methods and finally Recommender Systems, all essential to comprehend the work presented. These three fields of Machine Learning are used are the building blocks of this dissertation.

2.1 Machine Learning

Machine Learning is an application of Artificial Intelligence , that focus on giving systems the power to learn autonomously without being explicitly programmed by providing it data to learn from. These algorithms analyse the input data with the goal of identifying patterns in data and, based on the data presented, make better predictions in the future.

Machine learning can be categorized in four different ways:

1. **Supervised Machine Learning**

   The system is give both the input and the objective feature, which we can see as the desired output. Recommender systems modeling is an example of supervised machine learning, where the objective feature is the user's choice.

2. **Unsupervised Machine Learning**

   In this approach no objective feature is given, the goal is for the algorithm to find underlying patterns, structures or relations in the data. An example of the use of this approach is locality modeling, Clustering, where it is intended for the algorithm to find aggregations of similar data points.

3. **Semi-Supervised Machine Learning**

   This Machine Learning technique falls somewhere in between supervised and unsupervised learning, the provided data contains a mixture of both entries containing an objective feature and not.

4. **Reinforcement Learning** This Learning method that interacts with its environment, performing actions and discovering errors or rewards. Trial and error search and delayed reward are the main characteristics of reinforcement learning.

2.1.1 Clustering

Many applications need grouping data elements. For a variety of data mining applications, dividing a large number of data points into a smaller number of groups aids in summarizing and comprehending the data. The answer to this necessity is Clustering.

We can classify Cluster algorithms as , Hard and Fuzzy Clustering. In the former, a data point either belongs to a certain cluster or not, hence hard clustering. In the later, a data point has a degree of belonging to all the clusters, having a greater probability of belonging to ones than to others.

2.1.1.1 Representative-Based Algorithms

These are the simplest of all clustering algorithms, they rely on an intuitive notion of distance to cluster data points. In this context distance most times signifies the degree of similarity between two data points.

Clusters are created in on shot and no hierarchical relationships are established. This is accomplished with use if a set of partitioning representatives, these may be either created as a function of the data points in the cluster or selected from the original data. These methods produce high quality clusters, which is equivalent to discovery high quality representatives.

After the representatives are set, a distance function can be used to assign the data points to the cluster of the closest representative.

The norm is that the number of clusters,k, is set by the user. Given an data set D containing n data points denoted X1...X2, the goal of the clustering algorithm is to determine the k representatives Y1...YN that minimize the sum of the distances between the data points and their representatives:

$O = \sum_{i=1}^{n}[min_j Dist(\bar{X}_i, \bar{Y}_j)]$

The most well known Representative-Based Algorithms is the K-Means algorithms. In the k-means algorithm, the sum of the squares of the Euclidean distances of data points to their closest representatives is used to quantify the objective function of the clustering.

2.1.1.2 Hierarchical Clustering Algorithms

These algorithms usually use distances to cluster the data, but is not an obligation, many hierarchical algorithms use other clustering methods as a subroutine for constructing the hierarchy. The reason why an hierarchy is useful is that the different levels of clustering granularity provide different application-specific insights, providing a taxonomy of clusters, which can be browsed for semantic insights.

Hierarchical Clustering Algorithms can be put in two different classes, Bottom-up and Top-Down.

Bottom-up algorithms, also known as agglomerative, successively agglomerates individual data points into higher-level clusters. The objective function used to decide the merging of clusters may vary between methods.

Top-Down algorithms, also known as divisive, successively partition the data points into a tree-like structure. This approach provides flexibility when it comes to choosing the trade-off between the balance in the tree structure and the balance in the number of data points in each node.

2.1.1.3 Probabilistic Model-Based Algorithms

These algorithms are soft algorithms, here each data point has an assignment probability, different from zero, to many (usually all) clusters. Contrary to the prior presented algorithms, which are hard clustering algorithms, these are said fuzzy. A soft solution to a clustering problem may be converted to a hard solution by assigning a data point to a cluster with respect to which it has the largest assignment probability.

Non-negative Matrix Factorization NMF (nonnegative matrix factorization) is a dimensionality reduction technique designed specifically for clustering. To put it another way, it embeds the data in a latent space that makes grouping easier. This method works well with non-negative and sparse data matrices. In text applications, for example, the $n * d$ document-term matrix always has non-negative elements. Furthermore, this matrix is sparse since most word frequencies are 0. A relevant difference between NFM and other dimensionality reduction methods is it does not guarantee or need to contain orthonormal vectors. A very well know example of the use of this method is for document clustering [22].

Let D be a document-term matrix, when n is the number of documents defined on a lexicon of size d. The NMF algorithm reduces the data dimensionalitty to a k-dimensional basis system, where each basis vector is topics. These vectors are vectors of non-negatively weighted words defining that topic, every document has a non-negative coordinate referent to each basis vector. Given this we can define the belonging to a cluster by finding the biggest value within the coordinates of the document across any if the k vectors.

We can, using this method, classify each document by topic, not that these does not dictate the belonging of a document do a topic, but gives a value that translate the proximity of the document to all topics, given this we can consider this a soft clustering method.

2.1.1.4 Density-Based Algorithms

The form of the underlying clusters is already specified implicitly by the underlying distance function or probability distribution, which is one of the key difficulties with distance-based and probabilistic approaches. The basic idea behind these methods is to find fine-grained dense areas in the data first. These serve as the "building blocks" for assembling the clusters in any shape. These can alternatively be thought of as pseudo-data points that must be meticulously re-clustered into groups of any form. As a result, the majority of density-based algorithms may be classified as two-level hierarchical algorithms. Because there are fewer building blocks in the second phase compared to the quantity of data points in the first phase, more comprehensive analysis may be used to organize them into complicated forms.

DBSCAN In the DBSCAN algorithm, density-based spatial clustering of applications with noise, density characteristics of data points are used to merge them into clusters. The individual data points in dense regions are used as building blocks after classifying them on the basis of their density. The density of a data point is defined by the number of points that lie within a radius Eps of that point (including the point itself). The densities of these spherical regions are used to classify the data points into core, border, or noise points. Where a core point is so that contains within its radius a minimum number of points, MinPoints, set by the user. A border point is such that is not a core point but its reachable by one, its contained in the radius of a core point. A noise point, or outlier, does not have in its radius the minimum amount of points and it not reachable by a core point.

The DBSCAN clustering method goes as follows once the core, boundary, and noise points have been established. The first step is to create a connection graph based on the data. Each node corresponds to a core point, and an edge is placed between them. If and only if they are within Eps of each other, they constitute a pair of core points.

This graph's connecting components are all identifiable. These are the clusters that have been built around the main points. After that, the border points are given to the cluster with the highest level of connectedness. The resultant clusters are referred to as outliers, whereas noise points are referred to as outliers.

2.1.2 Ensemble Methods

Ensemble methods are statistical and computational learning procedures, similar to human learning behavior of seeking several opinions before making any decision. The principal of these

methods is the use of the combination of different "opinions" in order to make a better decision.

The ensembles are sets of learning systems, each ensemble computes its own decision, the ensembles then combine the their insights into an overall decision. Empirical studies showed that in classification and regression problems, ensembles improve on single learning machines [21].

Ensemble methods can be decomposed in two main classes[8]:

1. **Parallel**

   In this class, base learners are generated at the same time in a parallel manner. Being them different models, or simply modeling different fractions of data at each learning instance.

2. **Sequential**

   These methods make use of different learners to learn sequentially, with the first learners fitting simpler models to the data, those models are assessed and their errors collected. The next model goal is to solve the previous models errors.

With these two classed there are three types of methods that been focus of special attention. and outputs a weighted average of the predictions.

2.1.2.1 Bagging

Bagging is a parallel ensemble method used for improving unstable estimation or classification schemes. It consists in fitting multiple decision trees in different samples of the same data set and averaging for prediction.

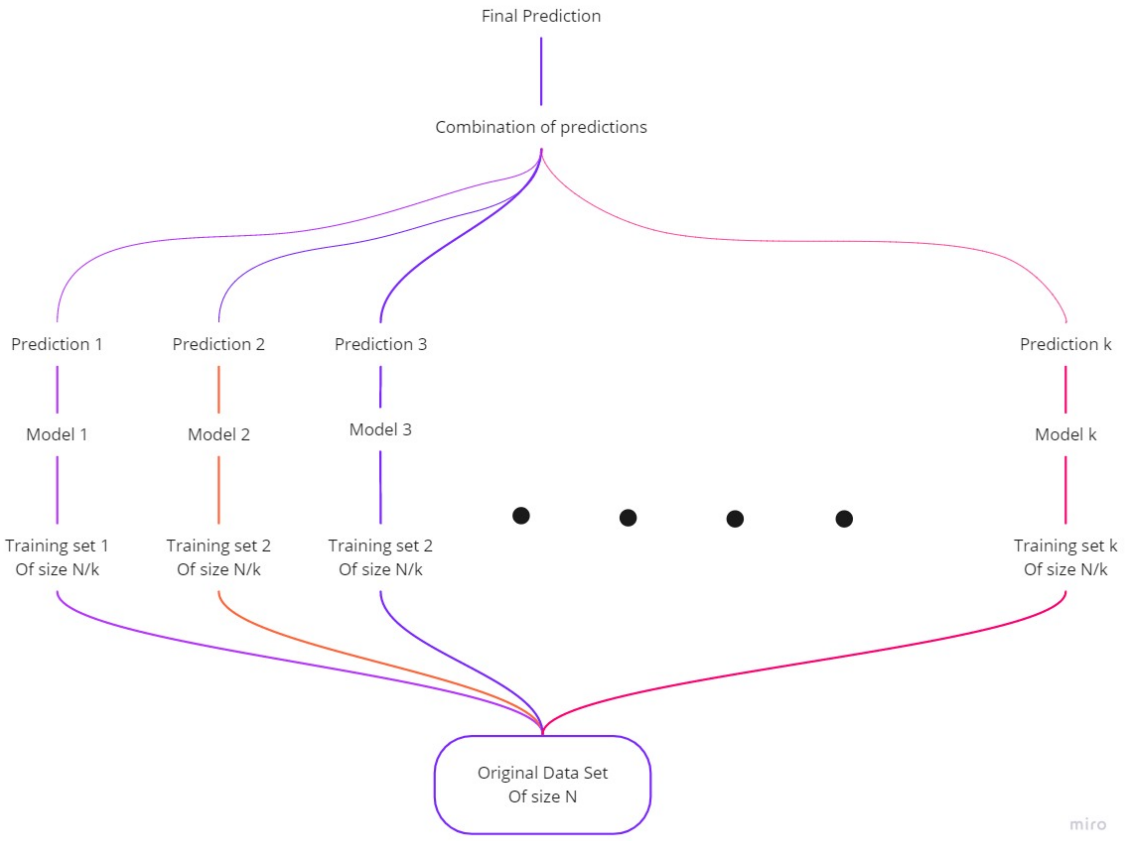
This is accomplished by random sampling with replacement from the data set, the replacement implies that some data points may appear multiple times in each training data set, and others not appear at all. Even so, each data point has the same probability of being included in the training data set. This approach decreases the variance and adjusts the prediction.

The different select training data sets are used to train multiple models, the result is an ensemble of different models with different predictions. This different predictions are used to calculate and over all result, being it more robust then a singular model. In the case of a linear regression problem the average of all predictions can be used as final prediction, in classification problems the majority vote is taken as the answer.

A popular application of this method is Decision trees which tend to have high variance.

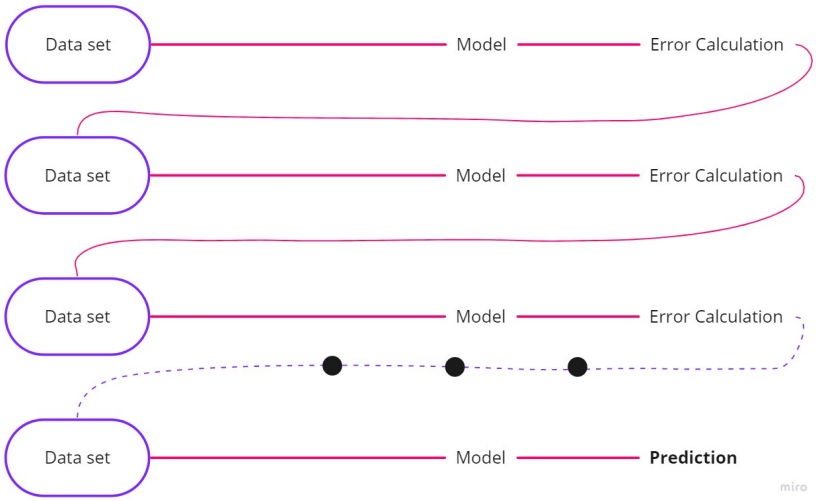
2.1.2.2 Boosting

Boosting methods, which are sequential ensemble methods, add sequentially ensemble members that correct the predictions made by previous models and outputs a weighted average of the predictions.



This model in general decreases the bias error and builds strong predictive models. Boosting uses multiple learners. The data samples are weighted ,so some of them may take part in the new sets more often.

In each iteration, data points that are hill predicted are identified and their weights are increased so that the next learner will focus on imitating that previous error [2].



2.1.2.3 Stacking

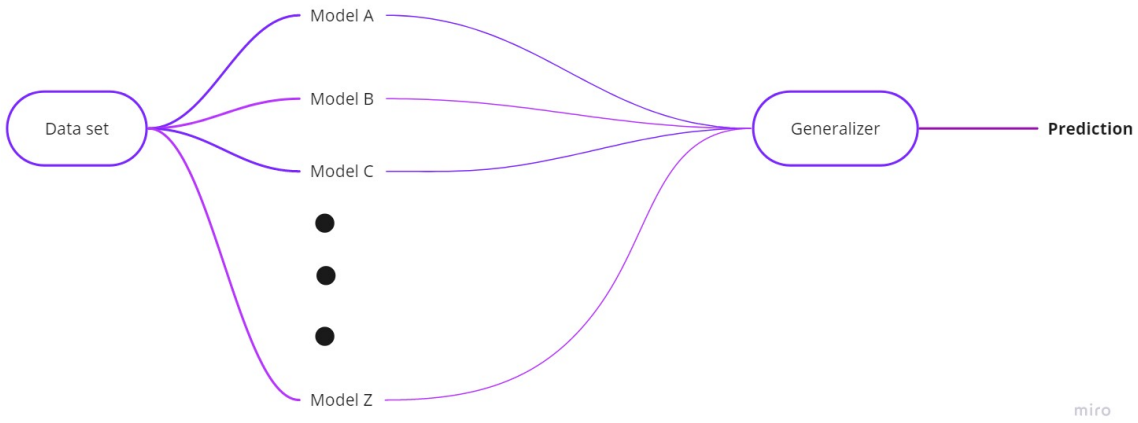
Stacking involves fitting many different models types on the same data and using an extra model to learn how to combine the result of said models. This method falls into the category of parallel ensemble.

Contrary to bagging, in stacking, the models are typically different and fit on the same data set and not sampling training data sets. Unlike boosting, in stacking, a single model is used to learn how to best combine the predictions from the contributing models, instead of a sequence of models that correct the predictions of prior models.

Architecturally speaking a staking model involves at least two base models, and a singular meta-model that combines the predictions of the base models. **Base-Models**: Fit the training data and whose predictions are compiled. **Meta-Model**: Model that learns how to best combine the predictions of the base models.

The meta-model is trained using out-of-sample data and predictions from base models. That is, non-training data is supplied to the base models, predictions are formed, and these predictions, along with the predicted outputs, form the input and output pairs of the training dataset used to fit the meta-model.

In the case of regression, the outputs from the base models used as input to the meta-model may be real values, probability values, probability like values, or class labels; in the case of classification, the outputs from the base models used as input to the meta-model may be probability values, probability like values, or class labels.



2.2 Recommender Systems

Recommender Systems are an important machine learning class, whose focus is to generate meaningful recommendations to a collection of users of items they are interested in. The use of recommendations systems are wide spread and very common, from online shopping to music services or even online dating services.

We ca define the recommendation problem as follows:

1. Set of users U
2. Set of items S to be recommended to the users
3. Each user in U is defined with a user profile that includes user characteristics, including its preferences
4. The task is to learn the utility function that measures the usefulness of item s to user u and predict the utility value of each item to each user

Recommendation task can be classified as Item Prediction, where the objective is to predict a ranked list of items that the user will be interested in. Or a an Rating Prediction, here we aim to predict the rating that a user will give to an item not seen before.

This problem can be approached in a number of ways:

1. **Content-based filtering**

   The user will be recommended items similar to the ones he previously had an positive interaction with.

2. **Collaborative filtering**

The user will be recommended items that people with similar past preferences positively interacted with previously.

3. **Hybrid approaches**

   Its a mixture of both content-based and collaborative filtering techniques, taking advantage of the characteristics of both.

Recommender Systems help user navigate the overwhelming offer of items. Typically, recommendations are made to fit the needs of a broad group of users or a cluster of users who get a variety of options. According to the user's choices, personalized recommendation systems attempt to predict the most suited items or services.

As previously presented in recommender systems there are two classes of entities, users and items. Users have preferences for certain items, and such preferences must be inferred from the data. The data is represented as a utility matrix, with a value for each user-item pair representing what is known about the user's degree of preference for that item.

Values are drawn from an ordered collection, such as numbers 1–5 denoting the number of stars a user assigned to an object. We assume the matrix is sparse, which means that the majority of the elements are "unknown." We have no specific information on the user's preference for the item if the rating is unknown.

This matrix can be decomposed in two matrices, U and V. Where U represents users and V items, both covering a common latent feature space. This feature space is predefined, and it is by rule of thumb smaller that the original utility matrix rank.

2.2.1 Matrix Factorization

Matrix decomposition is a method to breaking down a matrix into its constituent elements so that more complex matrix operations may be calculated more easily. Matrix decomposition methods, also known as matrix factorization methods, are essential to compute even the simplest tasks like solving systems of linear equations, computing the inverse, and computing the determinant of a matrix.

Along side the previously referred application, NMF for clustering, Matrix factorization besides making algebra computation extremely efficient has become popular in recommendation systems methods.

There are a plethora of different Matrix factorization methods, here we will present two of the most commonly used in recommendation problems:

Unrestricted Matrix Factorization: Is a classic matrix factorization, given the data matrix X:

$X \approx UV^T$

Here U and V may have mixed signs, so can the input data. Given that can define:

$$SVD : X_{\pm} \approx U_{\pm} V_{\pm}^{T}$$

NMF: Given a nonnegative input date, U and V must be negative.

$$NMF : X_{+} \approx U_{+} V_{+}$$

Semi-NMF: Given an mixed sign input, we restrict only V to be nonnegative.

$$Semi - NMF : X_{\pm} \approx U_{\pm} V_{+}$$

In particular when it comes to applications in recommender systems is the method of unrestricted matrix factorization, inspired by SVD and sometimes refereed in the literature as SVD, a popular choice. In this application matrix factorization characterizes both items and users by vectors of factors inferred from positive interactions between users and items, be R the matrix containing these vectors.

The singular vector decomposition algorithm decomposes the R matrix into two latent factor matrices U^{p*z} and V^{q*z} where p is the number of users, q of items, and z the common latent features. We want the product of the latent matrices to be as close as feasible to R:

$$R_{ui} \approx \tilde{R}_{ui} = U_u V_i^T \text{ , being u a given user and i a given item}$$

Since we are working with positive-only feedback, we assume $R_{ui} = 1$ for each positive interaction, this is each user-item pair (u, i) occurring in the data set, and $R_{ui} = 0$ otherwise. This decomposition is achieved by minimizing the squared error between 1 and $\tilde{R}_{ui}$ for all data points in the data.

Chapter 3

State-of-the-art

In this chapter we will analyse literature related to these two subjects, Incremental Clustering and Incremental Recommendations Systems, which are cornerstones subjects for the work developed in this dissertation.

3.1 Incremental Clustering

By definition, an incremental process, in data context, is so that each data entry is computed upon when it enters the system, making it so that the computation is executed on a single pass over the data.

3.1.1 Incremental DBSCAN

Traditional Clustering algorithms are design for batch execution, but many adaptations have been studied in order to fit the on stream demand. One of those variations is the Incremental DBSCAN [23].

This particular approach proved to outperform the classic algorithm in most cases. The intuition of the incremental implementation is to calculate an affected area around the added point only once and classify it according to the outcome of this calculation. In the traditional DBSCAN algorithm it is defined by the user the radius of the neighborhood and the minimum number of points of the neighborhood [2, section 2.1.1.4]. In the incremental DBSCAN algorithm we use that radius to calculate the affected area.

At the entry of a new data point, P, in the data set the neighborhood of P, if the neighborhood size is equal or greater than the set minimum, then we know it is a core point. The same analysis is operated on the points in the neighborhood, those with big enough neighborhoods are too core points, these are the seed points for new clusters.

If no seed points are found P is an outlier, if that is not the case we look in all the already

existing cluster checking for matches on our seed points.

Suppose no matches are found, these means our P cluster is isolated, a new cluster is created and added to our record of clusters. On the other hand if only one match with a cluster is found P and our seeds are added to that existing cluster creating a new bigger one, eliminating the old one.

The last hypotheses is that if more than one match is found, in these case the P cluster servers has a bridge between two or ore clusters. Creating a cluster that harbors the match clusters and the P cluster.

This very elegant adaptation of the classical DBSCAN solves one of our requirements for the clustering algorithm, who's goal is to integrate a incremental recommendation systems. But it does lack on the number of clusters control and on the fuzzy approach.

3.1.2 Incremental Matrix Factorization

In 2014 Vinagre, J., A.M. Jorge, and J. Gama, presented an incremental adaptation to the matrix factorization [17]. This particular paper proposed an incremental recommendation algorithm based on matrix factorization, although the incremental matrix factorization did not serve the purpose of clustering the method translates.

The research proposed methods to calculate incrementally using just the currently available data at each step, based on a basic Stochastic Gradient Descent (SGD). Inspired on the Latent Semantic Indexing [4], which uses Singular Value Decomposition(SVD) of large document matrix to index large collections of text documents. This principal is used for the Collaborative Filtering problem in [17], using an user-item matrix to find a latent feature space. Given a user-item matrix R with positive only data (records only the existence of an positive interaction between user and item or not), the algorithm decomposes it into two factor matrices U and V, as in SVD, where U spans the user space and V the item space given the prediction, $\hat{R}$, is:

$$\hat{R}_{ui} = U_u \cdot V_i$$

The model is trained by minimizing the L_2-regularized squared error for known values in R and the predictions in $\hat{R}$

$$\min_{U.,V.} \sum_{(u,i)\in D} (R_{ui} - U_u \dot{V}_i^T)^2 + \lambda(||U_u||^2 + ||V_i||^2)$$

In the proposed algorithm, Incremental Stochastic Gradient Descent(ISGD), at the entry of a new data point $< u, i >$ there are made adjustments to the factor matrices U and V in a singular step, given the positive only nature of the R matrix the error is calculated as follows:

$$err_{ui} = 1 - \hat{R}_{ui}$$

Algorithm 1 ISGD

1: $Data : D = < u, i >, a\ finite\ set\ or\ a\ data\ stream$
2: $input : feat, \lambda, \eta, iter$
3: $output : U, V$
4: **for** $< u, i > \in D$ **do**
5: **if** $u \notin Rows(U)$ **then**
6: $U_u \leftarrow Vector(size : feat)$
7: $U_u \sim N(0, 0.1)$
8: **end if**
9: **if** $i \notin Rows(V)$ **then**
10: $V_i^T \leftarrow Vector(size : feat)$
11: $V_i^T \sim N(0, 0.1)$
12: **end if**
13: **for** iter **do**
14: $err_{ui} \leftarrow 1 - U_u \dot{V}_i^T$
15: $U_u \leftarrow U_u + \eta(err_{ui} V_i - \lambda U_u)$
16: $V_i \leftarrow V_i + \eta(err_{ui} U_u - \lambda V_i)$
17: **end for**
18: **end for**

The algorithm showed in tests improvement of performance in most cases when compared to state of the art algorithms.

Given this effective incremental matrix factorization algorithm an adaptation could be made for clustering, the feat factors naturally acts as a sort of fuzzy clustering, and the number of clusters could be control exactly by controlling the number of features. The main problem with this adaptation is the non guaranty of normalization of the matrices, and the user/item vectors. The the values given in a item or user vector by the SGD must translate the same meaning across all vectors.

3.2 Incremental Ensembles for Recommendation

In the aforementioned paper [17], incremental matrix factorization is used to produce an incremental recommendation system. An iteration on this algorithm was presented in 2017 [19], here it was purposed and algorithm based in the ISGD algorithm that took advantage of ensemble methods in an incremental manner.

Prior to this proposal, stream-based ensemble models have been extensively studied. The Random Forests algorithm (Breiman, 2001) , which combines several randomized decision trees and aggregates their predictions by averaging, is well known and it has been successful in data stream mining [16]. The study on this area although extensive was focused only on classification and regression problems and not recommendation systems.

AdaBoost[15] is a well-known Ensemble method. It trains Ensemble members in rounds, and at each round increasing the importance of training examples that were misclassified in the previous round. The final Ensemble is then aggregated using weights $\alpha_0 ... \alpha_N$ calculated during the training. Building on the principal that weights can be transferred between Convolutional Neural Networks, and between subsets of a same network in order to speed up in the initial training phase, Incremental AdaBoost was constructed. Following this intuition: since each subsequent round of AdaBoost increases the importance given to the errors made in the previous round, the network at a certain round t can be "recycled" at round $t+1$ to learn a new re-sampled training set. [10]

The work done by Vinagre et al. [19] on the other hand studied the use of these incremental ensemble methods in benefice of recommendation systems. In this proposed method the ensemble technique used is Bagging [2, section 2.1.2.1]. The problem rises when we consider the distribution of each example across each node of the ensemble, which is classically calculated as a derivation of the size of the data set,N. Given that in a streaming scenario N is not know, this is overcome by the assumption The problem is that this would still require knowing N beforehand. However, if we assume that $N \to \infty$, then the distribution tends to a Poisson(1) distribution.

While in classic Bagging the data is blindly distributed across the bootstrap nodes, making it so that each individual user is different in each of the nodes, in the proposed method all $< u, i >$ pairs of the same user are used the same number of times for training at each node. In this way it is secured the completeness of all user profiles in each of the bootstrap nodes.

This algorithm has shown great promise, and when compared to state of the art algorithms shows an improvement in predictions. But it lacks the locality sensibility aim on this dissertation.

3.3 Locality-sensitive Incremental Recommendation Systems

In batch Recommendation systems it is well known that a pre-clustering of the data points brings advantages, augmenting the quality of the predictions[13].

But when it comes to online recommender systems the research is not as extensive, none the less it was already proven the advantages of pre-clustering in this scenario too. In 2021 in the dissertation by Lúcia Moreira [9], it was shown that the execution of a pre-clustering step on the data to then be processed by a incremental recommendation algorithm, namely the User-based Bagged ISGD, improved the accuracy of predictions.

In the aforementioned dissertation different clustering algorithms were tested, all showing the increase in the predictions quality. But none of them adequate to integrate the incremental recommendation algorithm. Since they must face the same limitations addressed in the present thesis, the clustering algorithm must be fuzzy, incremental and have the ability to set the number of cluster before hand.

It must be fuzzy do that a given user is not only profiled in one of the nodes of the ensemble,

it must be incremental so that the over all algorithm is incremental and we must set the number of clusters before hand to correctly distribute the users across the nodes. In this scenario the set number of cluster substitutes the Poisson(1) distribution mentioned in 3.2.

3.4 Advances beyond the state of the art

The state of the art will be in this thesis the foundation for new developments. Considering the work developed by Lúcia Moreira in [9], in this dissertation we will further explore the concept of a locality sensible recommender system. By adapting the User-based Bagged ISGD algorithm to incorporate an incremental clustering algorithm as a definer of the user distribution in the bagging nodes. This adaptation requires and selection of the highest scoring $\frac{1}{3}$ clusters for each user to define to each nodes of the bagging the user should be insert to.

One of the proposed algorithms uses the mention incremental matrix factorization 3.1.2, to produce an implicit clustering through the matrix factors. To do this the factor values for each user must be normalized in order to ensure that the values between users for the same factor can be compared, but also ensuring the preservation of the order between the factor values in the user's vectors.

The second proposed algorithm is based in the incremental DBSCAN algorithm 3.1.1, this algorithm presents more challenges in its adaptation. Being that it does not allow the control of the number of clusters, nor is it fuzzy. The distance metric in the algorithm must also be adapted to fit the data, since we aim to measure the similarity between users and not their euclidean distance.

Chapter 4

Proposed algorithms

Through this chapter we will see in detail the different steps and fronts of implementation of this new recommender system approach. This chapter is divided in 4 sections. The first one addresses the exploration of existing clustering algorithms, specifically directed at an assessment of their fitness to the set criterion and/or to be modified to fit those same criterion. In the second section we will describe the adaptation to the existing NMF algorithm, and on the third the adaptation of the DBSCAN algorithm. In the final section we will walk through the adaptation of the algorithms in 4.2 and 4.3 to the aforementioned BISGD algorithm.

4.1 Selecting Clustering Algorithms

Revising our Clustering algorithm requirements, the algorithm must be incremental, fuzzy and allow control over the number of cluster. It must be incremental so the result of its integration in the incremental bagging recommendation algorithm is too. More so it must be fuzzy so the corresponding belonging values may serve as the heuristic for the distribution of the user item pairs along the bagging nodes. Lastly it must allow the control of the number of cluster for the same purpose of defining the distribution of the pairs across the nodes.

Representative-based 2.1.1.1 clustering algorithms like K-Means, allow the control of the number of clusters. Traditionally these algorithms are nor incremental, nor fuzzy, nonetheless adaptations of algorithm have been studied. The incremental adaptations as the one studied in [12], although functional they are very sensitive to errors in estimating distortion. The cousin of K-Means, C-Means is also a representative-based algorithm, contrary to K-Means C-Means is fuzzy, but still not incremental.

Hierarchical Clustering Algorithms traditionally do not fit the requirements, still incremental versions of hierarchical algorithms have been studied and been successful. We can see in [14] adaptations of the Cobweb and Classit incremental algorithms for document clustering. Other incremental hierarchical algorithms have been widely used for document clustering and text summarizing [20]. On the other hand the core intuition of hierarchical algorithm is not compatible

with fuzziness.

Probabilistic Model-Based algorithms are by definition fuzzy. Within this class of clustering algorithms we can find Non-negative Matrix Factorization(NMF),the number of features of the matrix factorization can be used as an implicit number of clusters control. An incremental version of this algorithm has been developed in [18]. Meeting in this way two of the three requirements.

Density-Based Clustering algorithms do not traditionally fill any of our three requirements. One of the most well known Density-Based Clustering algorithms is DBSCAN, although it does also not fulfill any of the requirements, it has a very successful incremental adaptation. The IncrementalDBSCAN algorithm has results comparable the traditional DBSCAN and proves to have competitive results when compared to other state of the art incremental algorithms. This particular algorithm due to its partitioned nature shows promise to be adapt in terms of number of cluster control and fuzziness.

4.2 Incremental Normalized Non Negative Matrix Factorization

The incremental matrix factorization described in 3.1.2, is used with the purpose of recommendation but it can be adapted for clustering purposes. As is the algorithm randomly initializes the with the normal distribution each user vector of the matrix, normal distribution does not give us in this case any particular advantage when it comes to the outcome of each of the models. By using clustering to define the distribution of points along the nodes of the bagging we can guarantee that for each node there exists a criterion of belonging, and therefor for those points that particular model depending on the grade of belonging gives a more adjusted result.

Here we propose the use of incremental matrix factorization as an implicit clustering, for this purpose each factor of the factorization represents a cluster to which it's relative value to other factor defines the grade of belonging of said user to the cluster. The Incremental Stochastic Gradient Descent does not then guarantee any value relation between the final values of a given user's vector and another. In this way the magnitude of the value of a the user vector for a particular user cannot be compared to the value of the same feature for another user. Since the optimization of the values of the features is incrementally performed one user at a time, not taking into consideration future of previous users.

If user a has value 0.7 for the feature y in it's vector, and user b has a value of 0.3 for the same feature, this does not necessarily mean that the degree of belonging of a to the cluster y is greater then that of b.

To overcome this limitation we normalize the vectors, so that the scalar product of all vectors corresponds to one, therefore giving comparative meaning to the values of different user vectors. To guarantee the correct normalization of the values we must first restrain the vectors from having negative values. To do so we use the sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}}$$

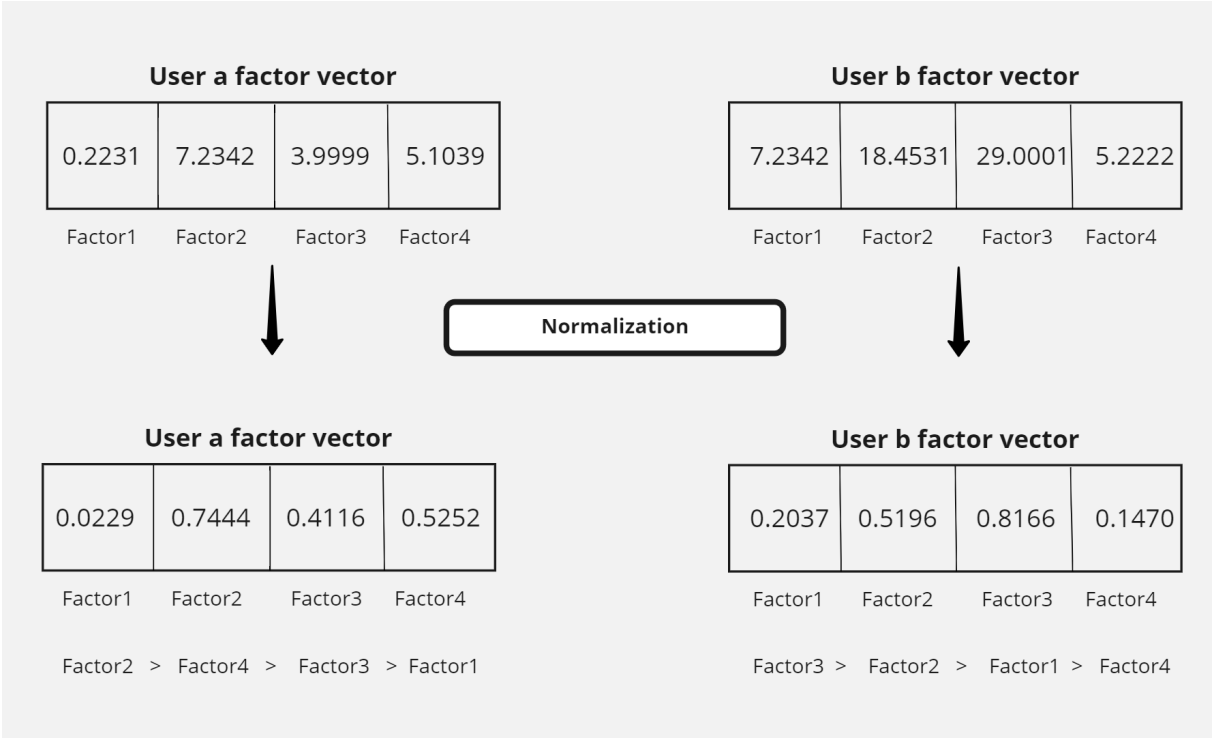


Figure 4.1: User vector normalization

As it's shown in the figure 4.1, although for user a the Factor2 has the same value of the Factor1 for user b (7.2342) this value is not uniformly representative of the grade of belonging to the respective factor clusters.

This function guarantees that each value of the vector will range between 0 and 1.

The normalization can be translated by the following formula:

Given any vector $V = (x, y, z)$ its norm is $|V| = \sqrt{(x^2 + y^2 + z^2)}$,

To normalize the vector we then divide it's values by the norm:

$$V/|V| = (x/|V|, y/|V|, z/|V|)$$

After these adaptations the Incremental Non Negative Matrix Factorization Clustering algorithm is:

Algorithm 2 INMFC

$Data : D = < u, i >,\ a\ finite\ set\ or\ a\ data\ stream$
$input : feat, \lambda, \eta, iter$
$output : U, V$
for $< u, i > \in D$ **do**
  if $u \notin Rows(U)$ **then**
    $U_u \leftarrow Vector(size : feat)$
    $U_u \sim N(0, 0.1)$
  end if
  if $i \notin Rows(V)$ **then**
    $V_i^T \leftarrow Vector(size : feat)$
    $V_i^T \sim N(0, 0.1)$
  end if
  for iter **do**
    $err_{ui} \leftarrow 1 - U_u \dot{V}_i^T$
    $U_u \leftarrow U_u + \eta(err_{ui} V_i - \lambda U_u)$
    $V_i \leftarrow V_i + \eta(err_{ui} U_u - \lambda V_i)$
  end for
  $U_u \leftarrow sigmoid(U_u)$
  $V_i \leftarrow sigmoid(V_i)$
  $U_u \leftarrow normalize(U_u)$
  $V_i \leftarrow normalize(V_i)$
end for

These changes create an uniformity of meaning of the feature values across different user vectors.

4.3 Incremental k-Controlled Fuzzy DBSCAN

As previously mentioned the classic DBSCAN clustering algorithm 2.1.1.4 lacks many of the requirements defined for the desired clustering method, it is not incremental, it does not allow the control of the number of clusters and it is not fuzzy. The algorithm discussed in 3.1.1 presents a solution for the incremental requirement, and there have been developed fuzzy variations of the classic algorithm such as in [6]. The fuzziness can easily be achieved by using the distance metric as a soft classification of the belong to said cluster, using in particular the distance of each point to the clusters core.

Still these solutions lack the control of the number of clusters, this can be achieved by setting the number of desired clusters k. The first k points will be set as cores of k clusters, has more points are added to the set it's neighborhood is evaluated. If a point p's neighborhood size is big enough, has at least many points as the defined parameter *MinPts* which represents the minimum number of points for a cluster, the point in said neighborhood that minimizes the

distance to other points is considered a potential core.

Elements of the core-hood of potential cores with a core-hood of size at least *MinPts* are added to the *Update Seed list(UpSeed)*. *UpSeed* is iterated and it's checked if the element belongs to an already defined cluster, said cluster is registered in the *Found Clusters List(foundClusters)*. In the instance of no clusters being found, the current clusters are evaluated to check if they have the minimum required size, this test fails in the first iterations given to the initialization of the clusters with a single point. In these cases said cluster is dissolved and replace by a p and it's neighborhood.

If all clusters have the minimum size the smallest cluster is found, if the smallest cluster's neighborhood is smaller then the potential size of the p cluster it's closest cluster is found. Otherwise, if the size of the found cluster is greater then the potential size o *p's* cluster then p is considered an outlier. In the instance of the distance between the smallest and it's closest being smaller then *eps*(the maximum distance between points for them to belong to the same neighborhood) these two are merged and p is expanded into a cluster. Otherwise the smallest cluster is dissolved and p is expanded into a cluster.

If a single cluster is found in *foundClusters* p is added to said cluster. In the case of existing multiple found clusters points of the potential p cluster and p are added to the largest of said found clusters.

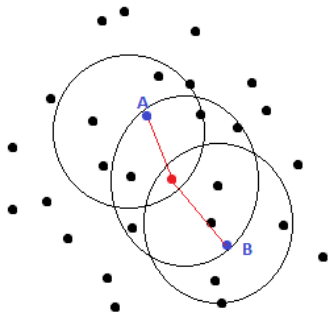


Figure 4.2: Density reach-ability

In figure 4.2 we can see the reach-ability principle used in this algorithm. Point **A** and **B**, although they are further apart from each other than *eps*, are still reachable trough their neighborhood.

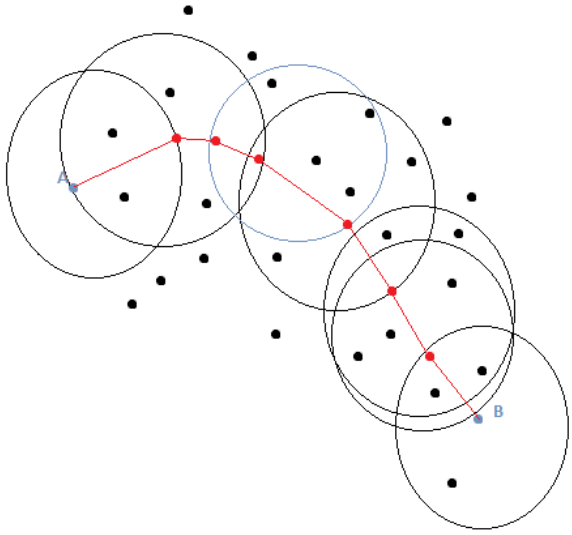


Figure 4.3: Density connect-ability

On the other hand, in figure 4.3 we can see that even if two points neighborhood do not overlap they can still be connected.

In this particular algorithm the distance measured between points, in order to calculate neighborhoods, is derived from the users factor vectors. Using the incremental matrix factorization refereed in 3.1.2, and using the vector normalization from 4.2 we have vector that represent the characteristics of each user. And by calculating the norm of their difference we can evaluate their similarity.

Given two user vectors $V = (x_v, y_v, z_v)$ and $W = (x_w, y_w, z_w)$, their norm is respectively:

$$|V| = \sqrt{(x_v^2 + y_v^2 + z_v^2)} \text{ and } |W| = \sqrt{(x_w^2 + y_w^2 + z_w^2)}$$

Their normalized vectors are:

$$V/|V| = (x_v/|V|, y_v/|V|, z_v/|V|) \text{ and } W/|W| = (x_w/|W|, y_w/|W|, z_w/|W|)$$

And the distance between them can be represented by:

$$dist = \sqrt{((x_v - x_w)^2 + (y_v - y_w)^2 + (z_v - z_w)^2)}$$

Initially, to calculate the distance between two users it was considered the cosine similarity. But due to the sparsity of data, this measure proved to be inefficient returning a similarity of 0 between most users.

Algorithm 3 K-IncDBSCAN

```
Data : StreamD = < u, i >_1, < u, i >_1, ...
input : eps, MinPts, num_clusters, cl_num_iterations, cl_learn_rate, cl_regularization
output : userclusterdictionaryDic
for < u, i >∈ D do
    if seen_points <= num_cluster then
        Add < u, i > to Clusters and Add < u, i > to Clusters_Cores
    else
        neigh = neighborhood(< u, i >)
        if size of neigh >= MinPts then
            pc = point of neigh that minimizes distance
            Add pc to Potencial_Cores
        end if
        for p in Potencial_Cores do
            pneigh = neighborhood(p)
            if size of pneigh >= MinPts then
                Add points of pneigh to UpSeed
            end if
        end for
        if size of UpSeed < 1 then
            Add< u, i > to Outliers
        else
            for seed in UpSeed do
                if seed is in Clusters and cluster not in foundClusters then
                    countFoundClusters = countFoundClusters + 1
                    Add cluster to foundClusters
                end if
            end for
            if size of foundClusters == 0 then
                find smallest cluster : smallestCluster
                if size of smallestCluster < potencialSize then
                    remove smallestCluster from Clusters
                    expand < u, i > and add to Clusters
                else
                    Add< u, i > to Outliers
                end if
            end if
            if size of foundClusters == 1 then
                Add < u, i > and seeds in UpSeed to foundCluster
            end if
            if size of foundClusters > 1 then
                Find in foundClusters the closest cluster to < u, i >
                Add < u, i > and seeds in UpSeed to closest
            end if
        end if
    end if
end for
```

4.4 Cluster-based Bagging Algorithm

In the dissertation produced by Lúcia Moreira, [9], it was proposed the improving of an incremental recommender system that makes use of an ensemble method. The strategy proposed made use of the results of a recommendation system to determine the distribution of users through the ensemble nodes. In this work although a batch clustering was used,it proved the concept that clustering was able to improve the results when compared to the state-of-the art algorithm which used normal distribution to populate the ensemble nodes.

The proposed algorithm in said dissertation was the Cluster-based Bagging Algorithm.

Algorithm 4 Cluster-based Bagging Algorithm

1: $Data : Stream\ D = < u, i >_1, < u, i >_1, ...$
2: $input : latent\ features\ z,\ regulization\ \lambda,\ learning\ rate\ \eta, bootstrap\ nodes\ M\ iter\ user\ fuzzy\ cluster$
3: $probabilities\ C\ with\ dimensions\ (number\ of\ users,\ number\ of\ nodes),\ threshold\ t$
4: $output : factormatrices U^m V^m$
5: **for** $< u, i > \in D$ **do**
6: **for** $m \leftarrow 1 to M$ **do**
7: **if** $C_u^m \in Top\ t\ x\ M\ (or\ t\ x\ 1/M)\ highest\ probabilities\ for\ u\ to\ belong\ to\ any\ node\ then$ **then**
8: **if** $u \notin Rows(U)$ **then**
9: $U_u \leftarrow Vector(size : feat)$
10: $U_u \sim N(0, 0.1)$
11: **end if**
12: **if** $i \notin Rows(V)$ **then**
13: $V_i^T \leftarrow Vector(size : feat)$
14: $V_i^T \sim N(0, 0.1)$
15: **end if**
16: **for** iter **do**
17: $err_{ui} \leftarrow 1 - U_u \dot{V}_i^T$
18: $U_u \leftarrow U_u + \eta(err_{ui} V_i - \lambda U_u)$
19: $V_i \leftarrow V_i + \eta(err_{ui} U_u - \lambda V_i)$
20: **end for**
21: $U_u \leftarrow sigmoid(U_u)$
22: $V_i \leftarrow sigmoid(V_i)$
23: $U_u \leftarrow normalize(U_u)$
24: $V_i \leftarrow normalize(V_i)$
25:

This algorithm is heavily based on the UBISBGD algorithm, using it as a base to which the cluster distribution is there to feed. This algorithm with be abbreviated in the remainder of this dissertation as CLUBISBGD. The following algorithms take this application as an inspiration to implement an incremental version of this one.

4.4.1 INMF-CLUBISBGD

The INMF-CLUBIDBGD uses the incremental clustering produced by the Incremental Normalized Non Negative Factorization 4.2 to serve feed at each data entrance as the cluster which defines the distribution of users in the ensembles nodes just as seen in CLUBISBGD 4.4.

Taking into consideration the algorithm 4.2, at each entry $< u, i >$ the algorithm is called, the resulting factor vector of that entry is used to defined the distribution on that entry in the ensemble nodes. The $\frac{1}{3}$ highest scoring factors are selected and implicitly mapped to the ensemble nodes, thus defining the distribution of the entry, adding it to the corresponding bagging nodes.

4.4.2 DBSCAN-CLUBISBGD

Similarly to INMF-CLUBISBGD (Section 4.4.1), the DBSCAN-CLUBISBGD adapts the Incremental k-Controlled Fuzzy DBSCAN (Section 4.3) to the CLUBISBGD algorithm, feeding incrementally the bagging nodes using the cluster distribution.

Considering the algorithm **DBSCAN-CLUBISBGD**, at each entry $< u, i >$ the algorithm is called, the resulting clustering of that entry is used to calculate the distance of $< u, i >$ to all clusters cores, the closest $\frac{1}{3}$ clusters are selected, and $< u, i >$ is added to the corresponding bagging nodes.

Chapter 5

Evaluation

Given the algorithms presented in sections 4.4.1 and 4.4.2, we will in this one describe the tests endured by them. Firstly we will present the data sets used in the tests, we will then define the tests that were executed. The results of said tests will be presented.

5.1 Datasets

In order to test the presented algorithms it was used real world data. These datasets should comply to some requirements, given that the algorithms at cause are user based recommender algorithms the data should be appropriate for the tests. More so the data should also be positive only given the positive only nature of the algorithms. For that we select 4 datasets which contain positive only user item interactions.

All of the 4 datasets are composed of two columns, first column containing the user's identifiers and the second containing the item's identifiers. Each entry represents an user-item positive interaction. The first dataset *ppf2* containing 111941 entries, the second *ml1m* containing 226310 entries, the third *palco2010* containing 588851 entries and the last on *ymusic* 476886 entries.

The *ppf2*[1] dataset is collected from a music social network, which contains music playlist activity, the dataset *palco2010* was collected from the same network consisting of a music streaming log. The *ml1m* data set is a binary version of the Movielens-1M movie ratings[2]. Lastly the *ymusic* dataset is a subset of the Yahoo! Music dataset3[3], which originally contains ratings given by users to music tracks in a 0 to 100 scale. This data contains only the ratings lower then 90 which was then randomly sampled for 2000 users.

After preliminary evaluation, the **DBSCAN-CLUBISBGD** was found to be too complex, and therefore to time consuming to endure the same testing as the **NMF-CLUBISBGD**

[1] https://rdm.inesctec.pt/dataset/cs-2017-003

[2] https://grouplens.org/datasets/movielens/

[3] https://consent.yahoo.com/v2/collectConsent?sessionId=3_cc-session_48521a78-3428-47fd-b376-6bd4084334bd

algorithm. Although this was a set back in this algorithms evaluation the results shown were promising, so a different evaluation approach was taken.

To evaluate the algorithm **DBSCAN-CLUBISBGD** each of the original data sets were sampled, sample data sets were produced by randomly sample 5% of the users and filter the data preserving order as to only include entries of said users. Each data set endures this process 5 times, resulting in the following data sets: *ppf2_sample5_1*, *ppf2_sample5_2*, *ppf2_sample5_3*, *ppf2_sample5_4*, *ppf2_sample5_5*, *ml1m_sample5_1*, *ml1m_sample5_2*, *ml1m_sample5_3*, *ml1m_sample5_4*, *ml1m_sample5_5*, *palco_sample5_1*, *palco_sample5_2*, *palco_sample5_3*, *palco_sample5_4*, *palco_sample5_5*, *ymusic_sample5_1*, *ymusic_sample5_5*, *ymusic_sample5_3*, *ymusic_sample5_4*, *ymusic_sample5_5*. The data set *lastfm* was not sampled for this purposed given that it only contains 50 distinct users.

5.2 Methodology

Given the datasets described in section 5.1, the algorithms were subjected to analysis and evaluation. Initially the clustering produced by the algorithms presented in sections 4.4.1 is analysed in order to retrieve information on their constitution. Then it was conducted a parametrization study for the , in order to choose the optimal parameterization for each of the algorithms. Finally each of the of the recommendation algorithms in sections 4.4.1 and 4.4.2 going to be tested against the state of the art algorithms UBISGD and BISGD (section 3.2).

5.2.1 Clustering Analysis

To analyse the clusters firstly it was collected and plotted information the distribution of user through the clusters, by looking into number of clusters users belong too. To do so given that these are soft clusters we defined, by empirical testing, a score threshold to define the belonging. We then analyse the distribution of scores in each clusters.

To better understand the evolution of the clusters during the execution of the algorithms their position in the ordered bagging distribution was recorded after 10%, 30%, 50%, 70% and 90% of the data was consumed. Besides that for the same intervals of data consumption the each user's cluster distribution was recorded and difference of said distribution between each of the intervals is recorded to be analysed.

5.2.2 Parameterization

In order to compare the presented algorithms to the state of the art ones, we used their optimal bagging and modeling parameters for each of the tested datasets. These parameters are *num_factors*(which define the number of factors in the recommendation factorization), *num_iterations*(the number of iterations executed in the linear regression), *learn_rate*(the

learning rate used in the linear regression) and *regularization*(the regularization used in the linear regression).

For the *ppf2* dataset these parameters are:

- *num_factors*: 90;
- *num_iterations*: 8;
- *learn_rate*: 0.3;
- *regularization*: 0.4.

For the *ml1m* dataset these parameters are:

- *num_factors*: 160;
- *num_iterations*: 8;
- *learn_rate*: 0.1;
- *regularization*: 0.4.

For the *palco2010* dataset these parameters are:

- *num_factors*: 160;
- *num_iterations*: 4;
- *learn_rate*: 0.5;
- *regularization*: 0.4.

For the *ymusic* dataset these parameters are:

- *num_factors*: 200;
- *num_iterations*: 9;
- *learn_rate*: 0.25;
- *regularization*: 0.45.

Besides these set parameter the number of clusters for all algorithms and datasets is also set to 8.

For the **NMF-CLUBISBGD** algorithm the studied parameters are *cl_num_iterations*(which defines the number of iterations executed in the clustering linear regression), *cl_learn_rate*(the learning rate used in the clustering linear regression) and *cl_regularization*(the regularization used in the clustering linear regression). The algorithm will be executed for different values and order of magnitude for each of these parameters and incrementally selected the optimal value for each of the parameters in the order they were here presented.

The same process was performed for the **DBSCAN-CLUBISBGD** algorithm, but taking in to consideration the parameters *eps*(which defines the maximum distance between points to be considered to be from the same cluster), *MinPts*(the minimum number of points for a set of points to be considered a cluster), *cl_num_iterations*(which defines the number of iterations executed in the clustering linear regression), *cl_learn_rate*(the learning rate used in the clustering linear regression) and *cl_regularization*(the regularization used in the clustering linear regression).

Although the parameterization is performed for both algorithms, we should consider given the **DBSCAN-CLUBISBGD** algorithm's limitations that the data sets used are the derived sample data sets mentioned in 5.1, one for each of the original data sets.

As for the **NMF-CLUBISBGD** algorithm parameterization, it was perform for each of the original datasets using the first 10% of the data entries.

5.2.3 Algorithms performance test

Using the optimal parameters discovered for each dataset we evaluate the whole datasets using the evaluation metric recall. More specifically it was used Recall@20 on a prequential evaluation method. The proportion of all pertinent elements in a system that are located in a list that system has retrieved is known as recall. Using a cut-off of 20, this is truncated to the first 20 elements. Recall yields 1 if the 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. The prequential process considers that for each incoming user-item pair a prediction with the current model is performed and the score of that prediction is performed by matching it to the actual observation. The model then is updated using the results of the prior iteration to the next user-item pair. For all the tests presented in this dissertation the this evaluation is performed in intervals of 100 entries.

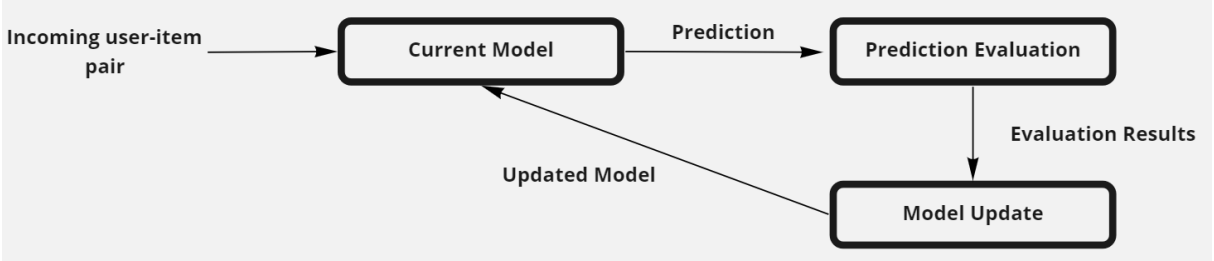


Figure 5.1: Prequential Evaluation

In the particular case of the **DBSCAN-CLUBISBGD** algorithm, it is tested with all 20 sample datasets, grouping by the original data set, the average recall of each 5 sample datasets is calculated and compared among algorithms.

The algorithm implementation as the test execution was performed using python 3.7.

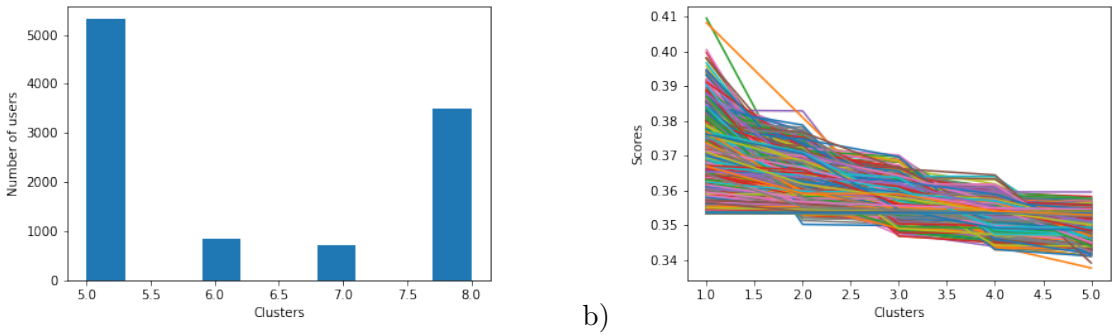
5.3 Results

In this section the methodology presented in 5.2 is applied to each of the algorithms, the results are presented in an adequate manner for interpretation, and initial assessments are taken to lay the grounds for the result discussion, where we will further explore the results.

5.3.1 Clustering analysis

Considering the methodology above described 5.2.1, we will here present and analyse the clusters for the **NMF-CLUBISBGD** algorithm and subsequently for each dataset.

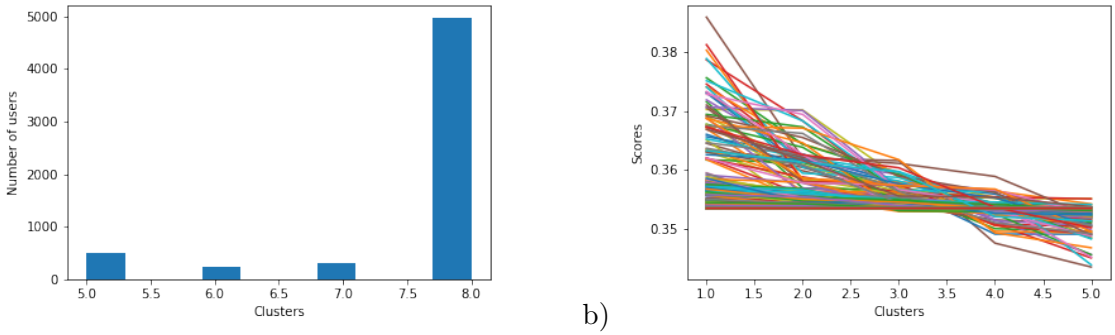
We will first analyse for each of the dataset the distribution of users along the clusters. In order to do so we set the belonging threshold at 0.1.



a) b)

Figure 5.2: User and score distribution along clusters, dataset ppf2.

In the figure 5.2 we can fin two sub-figures, to the left in sub-figure a) we can see that the majority of users belong to 5 clusters, followed by a substantial amount that belongs to 8. In sub-figure b) we can see that as a whole the cluster the scores fluctuate between 0.34 and 0.41, given that our threshold was 0.1 we can assume that this sample contains all the data executed.

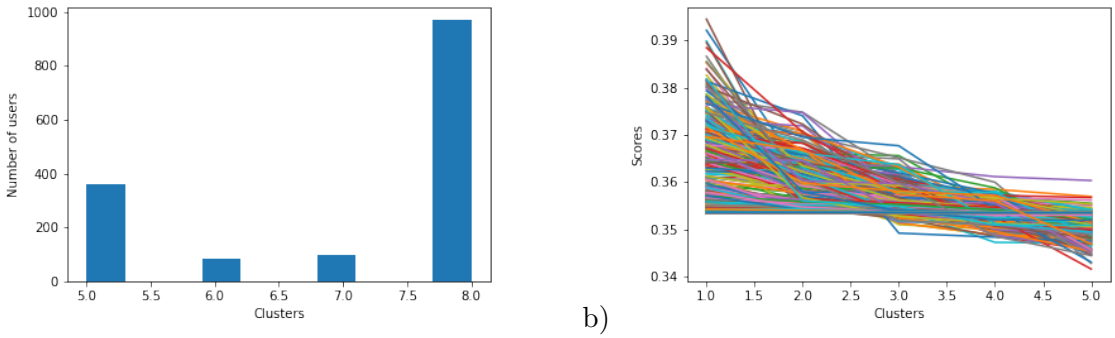


a) b)

Figure 5.3: User and score distribution along clusters, dataset ml1m.

By analysing figure 5.3 we can fin two sub-figures, to the left in sub-figure a) we can see that the substantially number of users belong to 8 clusters, with a residual number of users belonging

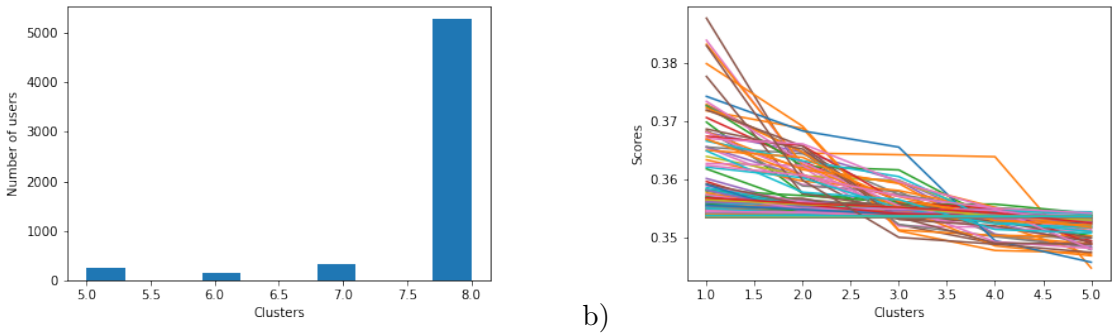
to either 5, 6 and 7 clusters in that order. In sub-figure b) we can see that as a whole the cluster the scores fluctuate between 0.39 and 0.3, given that our threshold was 0.1 we can again infer that this sample contains all the data executed.



a) b)

Figure 5.4: User and score distribution along clusters, dataset palco_2010.

In figure 5.4 we can see in sub-figure a) that the a little more then half of the of users belong to 8 clusters, followed by a substantial amount that belongs to 5, and residual number of users belong to 6 and 7 clusters, in that order. In sub-figure b) we can see that as a whole the cluster the scores fluctuate between 0.4 and 0.3, and again given that our threshold was 0.1 we can assume that this sample contains all the data executed.



a) b)

Figure 5.5: User and score distribution along clusters, dataset ymusic.

In sub figure a, of 5.5, we can see that the vast majority of users belongs to 8 clusters, with residual amounts belonging to 7, 5 and 6 in this order. In sub figure b we can see that as a whole the cluster the scores fluctuate between 0.39 and 0.3, and again given that our threshold was 0.1 we can assume that this sample contains all the data executed.

For the last 3 datasets a the majority of users belonged to 8 clusters, which would represent a very poor clustering. To avoid this the threshold would be set to a higher value, but given that in our algorithms the clusters selected for the bagging are the $\frac{1}{3}$ highest scoring clusters for that user, and we can see that scores are variable within the intervals recorded this clustering is by preliminary analysis viable.

In the following figures it's illustrated the positional changes of the clusters, for all four datasets, along the iterations.

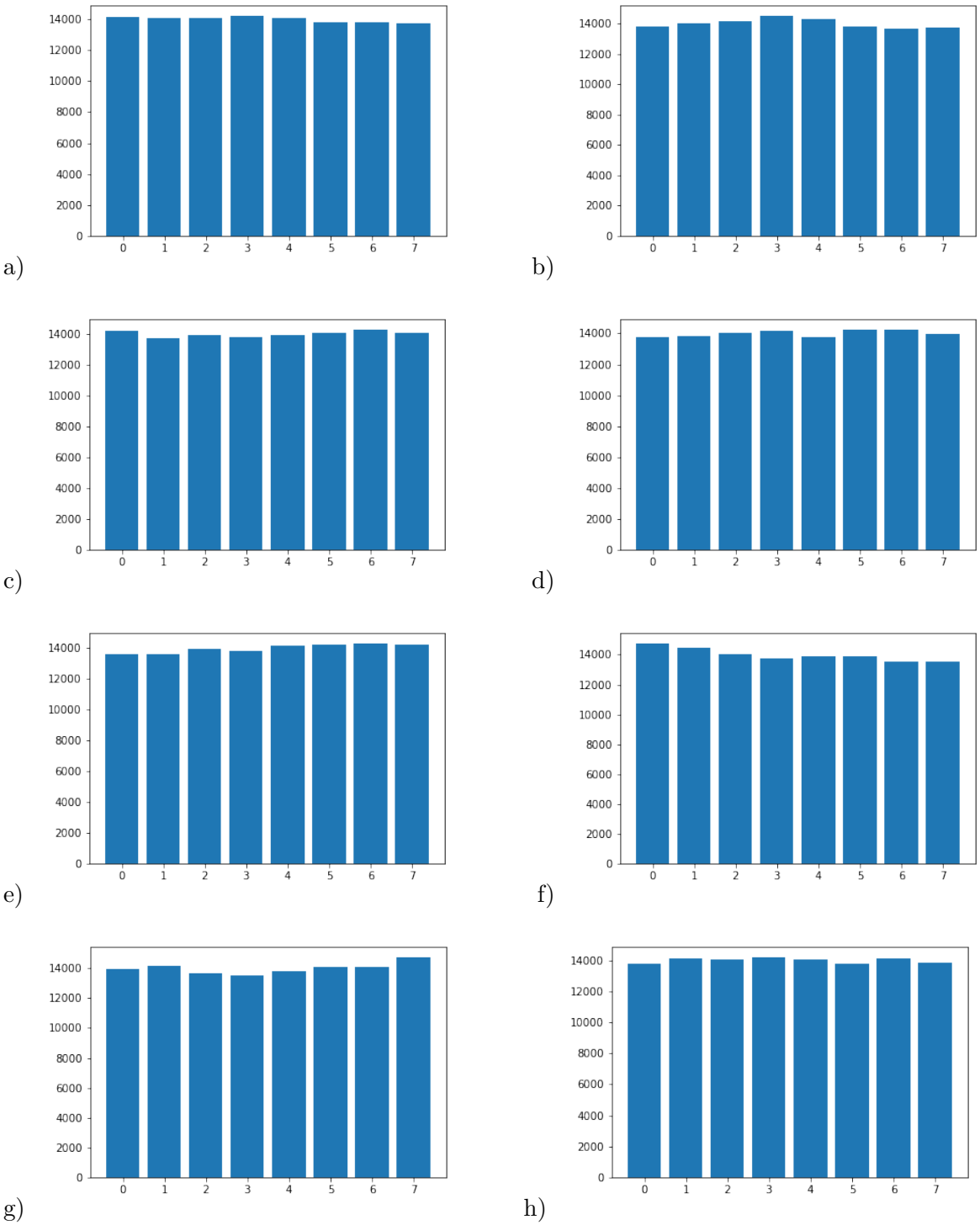


Figure 5.6: Positional changes of cluster, ppf2 dataset.

In figure 5.6 we can encounter 8 sub figures, each corresponding to one of the 8 clusters, in those figures it's represented the number of times the cluster was found in each position along the clustering execution. It is infer-able by the results that all clusters varied consistently in

position, not showing a bias towards neither of the positions. Cluster one's analysis can be found in sub figure a), cluster two's in sub figure b) and so on.

This figure 5.6 is illustrative of the results observed in all 4 datasets, all showing a lack of bias in the positioning of clusters along iterations. The results of this analysis in the remaning datasets can be found in Annex A

For this algorithms last clustering analysis, it's presented a study of the variation of user's clusters after different percentages of the data are given to the algorithm.

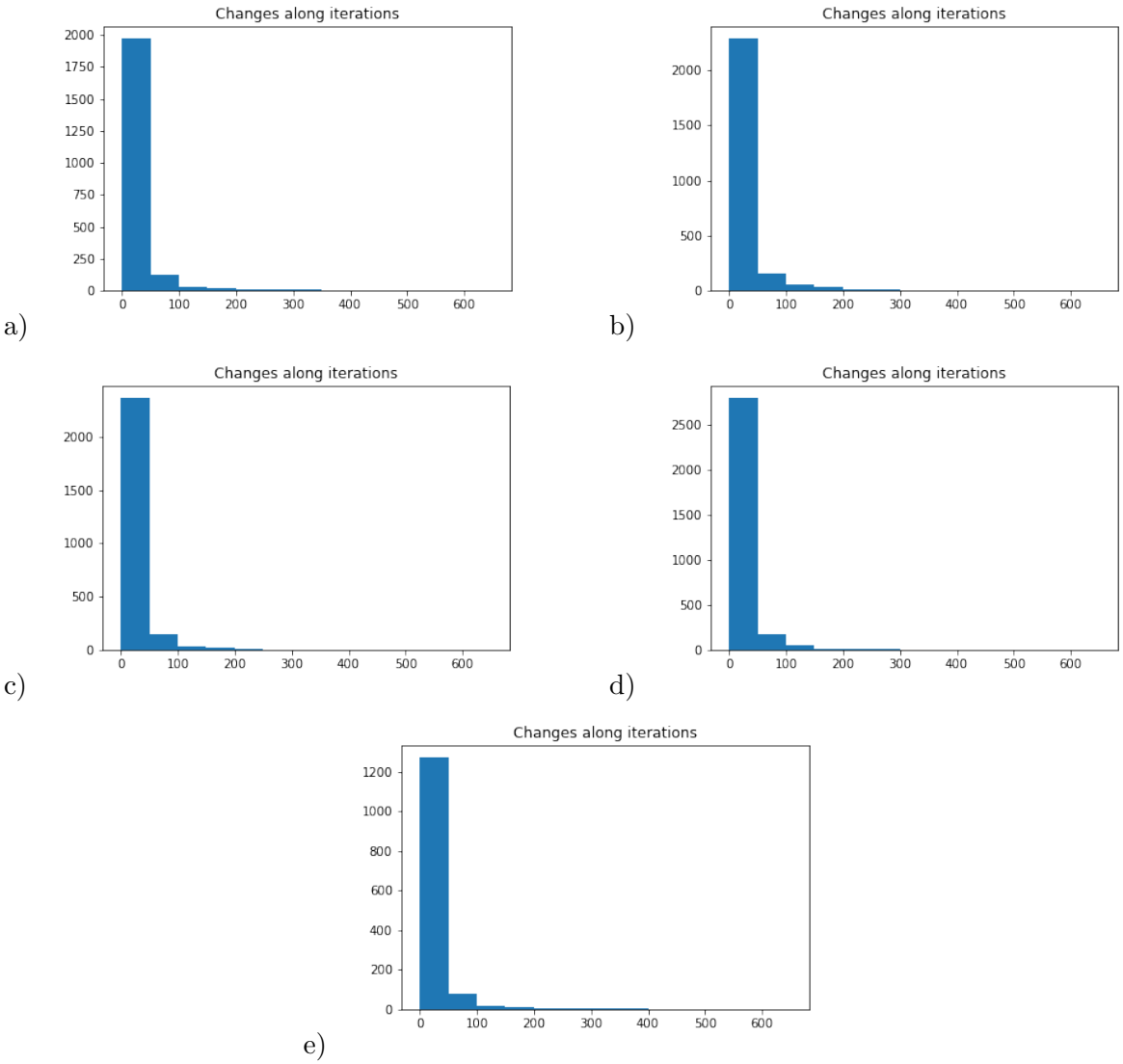
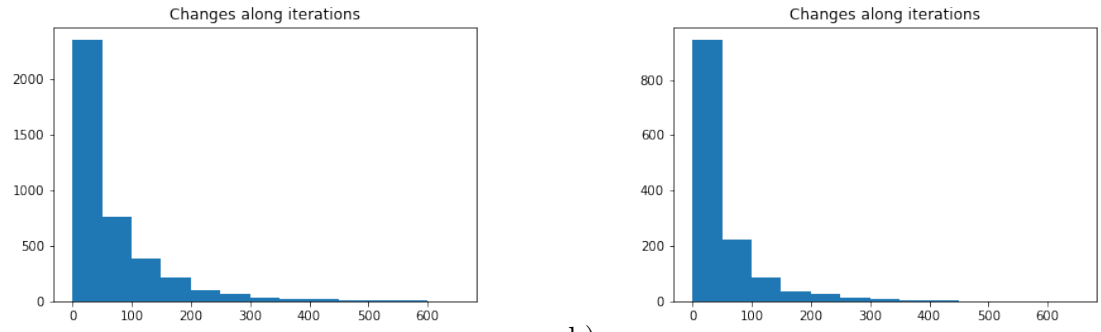


Figure 5.7: Number of changes in user's clusters, as data entry evolves, ppf2 dataset.

In figure 5.7 there are 5 sub figure, sub figure a) corresponds to the analysis after 10% of the data was consumed, b) after 30% of the data, c) after 50% of the data, d) after 70% of the data and e) after 90% of the data was consumed.

For the *ppf2* data set, we can see that most users in the first 10% of entries suffer between one and 50 changes to the clusters it belongs too, with residual amounts suffering up to 350

changes in this interval. For the other percentages of data entry the results are very similar. This shows that along the execution the clusters a user belongs too keep consistently changing showing that the clustering is responsive and does not stagnate after the initial entries.



a) b)

Figure 5.8: Number of changes in user's clusters, as data entry evolves, ml1m dataset.

In figure 5.8 we can find two sub figure a) referent to the changes recorded after 10% of the data was executed, and b) corresponding to the same analysis for 30%. When compared to the previous dataset in the first 10% and 30% the were recorded significantly more changes. But contrary to the *ppf2* dataset, no more changes are recorded after 30% of the data is entered. This could be due to the lack of entry of new users after the first 10% of data is executed.

For the *palco_2010* no changes recorded in the user's clusters after 10% of the data is entered. To better understand this behaviour a future direction could go through insert synthetic data that would force cluster changes to evaluate the behaviour of the clustering algorithm.

The *ymusic* dataset shows better results then the previous two, keeping consistent changes through the whole dataset entry. Even showing more instances of user's clusters changes over 100, and up to 600.

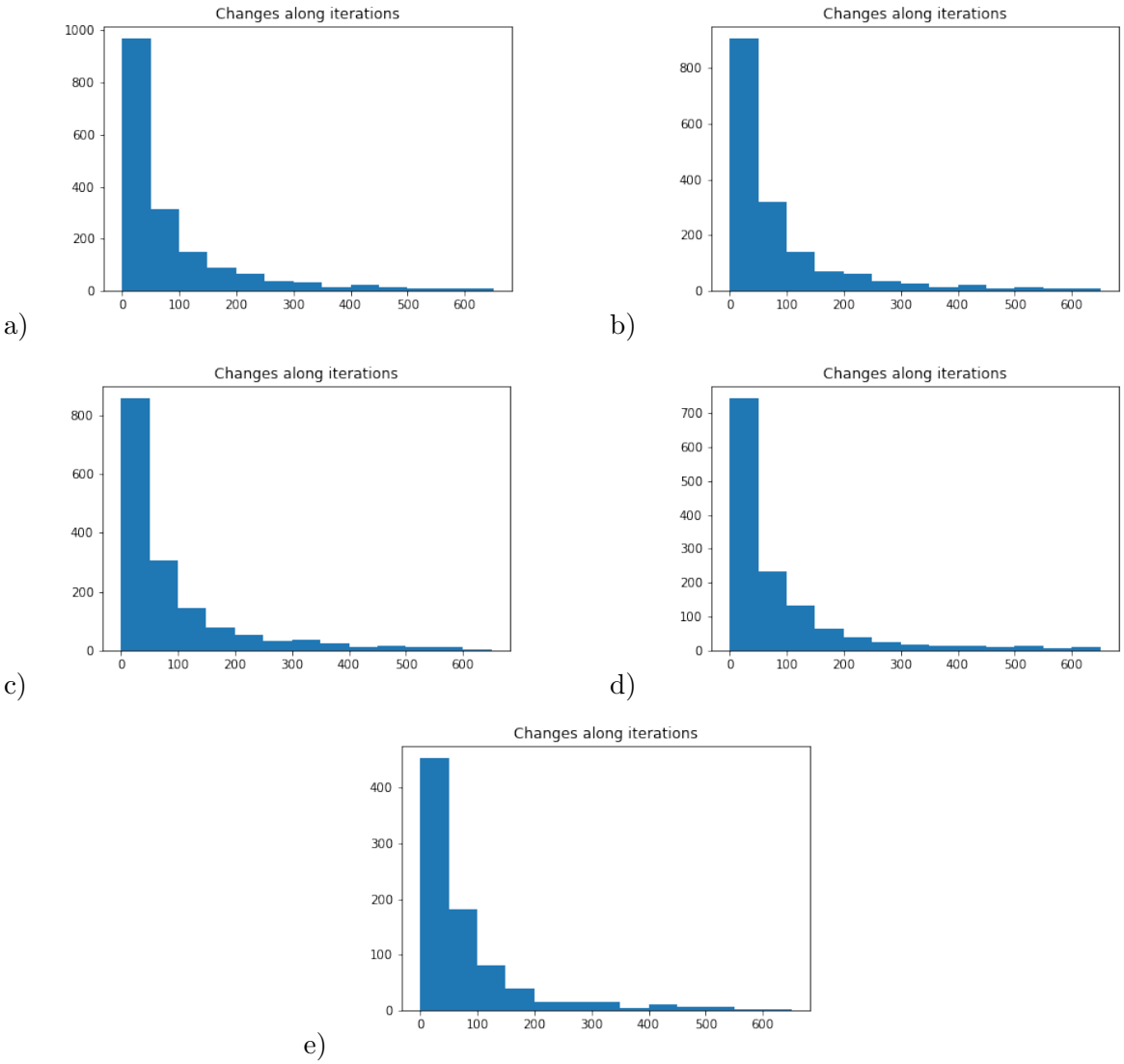


Figure 5.9: Number of changes in user's clusters, as data entry evolves, ymusic dataset.

5.3.2 Hyperparameters

Here we will present an analysis of the algorithms results with parameter variation, this analysis subsequently lead to the choice of optimal parameter used in the final performance evaluation.

5.3.2.1 NMF-CLUBISBGD

The metrics collected for this analysis were both recall and the time of execution. We should recall that for this parameterization it was used 10% of the data for all datasets.

Table 5.3.2.1 is illustrative of the parameterization executed on all data sets for the **NMF-CLUBISBGD** algorithm, in Annex B the parameterization of the remaining datasets can be found.

cl_num_iterations	cl_learn_rate	cl_regularization	recall	run_time
5	0.1	0.1	0.255	0:00:35
10	0.1	0.1	0.248	0:01:00
15	0.1	0.1	0.374	0:01:15
20	0.1	0.1	0.312	0:01:34
25	0.1	0.1	0.311	0:01:55
30	0.1	0.1	0.305	0:02:16
15	0.0001	0.1	0.261	0:01:19
15	0.001	0.1	0.289	0:01:26
15	0.01	0.1	0.369	0:01:24
15	0.1	0.1	0.268	0:01:20
15	0.01	0	0.234	0:01:16
15	0.01	0.0001	0.281	0:01:13
15	0.01	0.001	0.372	0:01:16
15	0.01	0.01	0.333	0:01:16
15	0.01	0.1	0.260	0:01:15

Table 5.1: Parameterization ppf2 dataset, NMF-CLUSISBGD

Analysing table 5.3.2.1, in blue is presented the *cl_num_iterations* parameter testing, and highlighted the highest recall value. Using the result of the first parameterization, the parameter *cl_learn_rate* is tested, here represented in the color red. Setting *cl_learn_rate* to its optimal value, it's finally evaluated *cl_regularization*. The set optimal values are: *cl_num_iterations* = 15, *cl_learn_rate* = 0.01 and *cl_regularization* = 0.001.

For the *ml1m* dataset the optimal parameters, as shown in annex B, are: *cl_num_iterations* = 10, *cl_learn_rate* = 0.001 and *cl_regularization* = 0.001.

For the *palco_2010* dataset the optimal parameters, as shown in annex B, are: *cl_num_iterations* = 25, *cl_learn_rate* = 0.001 and *cl_regularization* = 0.01.

Finally, for the *ymusic* dataset the optimal parameters, as shown in annex B, are: *cl_num_iterations* = 25, *cl_learn_rate* = 0.1 and *cl_regularization* = 0.1.

5.3.2.2 DBSCAN-CLUBISBGD

As mentioned in 5.1 the datasets used for this algorithm evaluation are samples of the original datasets presented. For the parameterization only one of each original datasets is used: *ppf2_sample5_1*, *ml1m_sample5_1*, *palco_sample5_1* and *ymusic_sample5_1*.

eps	MinPts	cl_num_iterations	cl_learn_rate	cl_regularizatio	recall	run_time
0.1	5	5	0.1	0.1	0.167	0:07:00
0.25	5	5	0.1	0.1	0.167	0:07:17
0.5	5	5	0.1	0.1	0.125	0:07:34
0.75	5	5	0.1	0.1	0.000	0:07:48
0.1	5	5	0.1	0.1	0.333	0:12:35
0.1	10	5	0.1	0.1	0.286	0:08:14
0.1	15	5	0.1	0.1	0.167	0:08:02
0.1	20	5	0.1	0.1	0.000	0:07:50
0.1	25	5	0.1	0.1	0.000	0:07:56
0.1	30	5	0.1	0.1	0.000	0:07:50
0.1	5	5	0.1	0.1	0.200	0:07:03
0.1	5	10	0.1	0.1	0.273	0:06:55
0.1	5	15	0.1	0.1	0.250	0:06:51
0.1	5	20	0.1	0.1	0.091	0:06:52
0.1	5	25	0.1	0.1	0.200	0:06:48
0.1	5	30	0.1	0.1	0.111	0:06:54
0.1	5	10	0.0001	0.1	0.000	0:06:52
0.1	5	10	0.001	0.1	0.000	0:06:48
0.1	5	10	0.01	0.1	0.000	0:06:47
0.1	5	10	0.1	0.1	0.500	0:06:51
0.1	5	10	0.1	0	0.000	0:07:13
0.1	5	10	0.1	0.0001	0.000	0:07:31
0.1	5	10	0.1	0.001	0.250	0:07:15
0.1	5	10	0.1	0.01	0.100	0:07:09
0.1	5	10	0.1	0.1	0.429	0:53:00

Table 5.2: Parameterization ppf2_sample5_1 dataset, DBSCAN-CLUBISBGD

The above table, 5.3.2.2, illustrates the parameterization process performed in this algorithm, in Annex B the parameterization of the remaining datasets can be found. Show that for dataset *ppf2_sample5_1* the oprimal parameters are as follows: $eps = 0.1$, $MinPts = 5$. $cl\_num\_iterations = 10$, $cl\_learn\_rate = 0.1$, $cl\_regularization = 0.1$.

For the *ml1m_smaple5_1* the optimal parameters, as shown in annex B, are: $eps = 0.75$, $MinPts = 10$, $cl\_num\_iterations = 20$, $cl\_learn\_rate = 0.001$, $cl\_regularization = 0.1$.

For the *palco_smaple5_1* thr optimal parameters, as shown in annex B, are: *eps* = 0.1, *MinPts* = 30, *cl_num_iterations* = 5, *cl_learn_rate* = 0.001, *cl_regularization* = 0.001.

For the *ymusic_smaple5_1* thr optimal parameters, as shown in annex B, are: *eps* = 0.1, *MinPts* = 20, *cl_num_iterations* = 10, *cl_learn_rate* = 0.01, *cl_regularization* = 0.01.

5.3.3 Performance

In this subsection we will present the final results of all algorithms, using the found optimal parameters for all datasets and executing the totality of the data of each.

Using the parameterization described in 5.2.2, both **UBISGD** and **BISGD** were tested using the whole 4 datasets.

dataset	Recall	run_time
ppf2	0.262	0:06:57
ml1m	0.060	0:10:44
palco_2010	0.169	1:48:42
ymusic	0.109	3:18:11

Table 5.3: Final results, BISGD algorithm

dataset	Recall	run_time
ppf2	0.272	0:07:13
ml1m	0.060	0:10:10
palco_2010	0.211	0:42:40
ymusic	0.107	0:33:36

Table 5.4: Final results, UBISGD algorithm

The same data is present in contrast, for the NMF-CLUBISBGD algorithm presented in this thesis.

dataset	Recall	run_time
ppf2	0.291	0:07:04
ml1m	0.065	0:15:25
palco_2010	0.197	0:03:30
ymusic	0.123	0:59:14

Table 5.5: Final results, NMF-CLUBISBGD algorithm

For the **DBSCAN-CLUBISBGD** algorithm, given the in initial found limitations of this algorithm in terms of time complexity being that it prevented the algorithm from tested in the same setting used for the **NMF-CLUBISBGD** algorithm in a reasonable time frame. The performance evaluation was made by evaluating 5 different samples of the originals datasets and

the average of the results is presented. In the following graphs the behaviour of the different algorithms is illustrated across the 5 samples for all the original datasets.

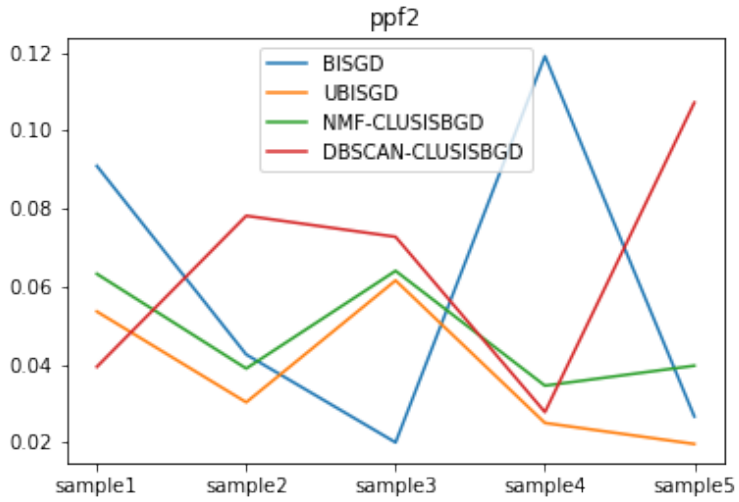


Figure 5.10: Results of samples of dataset *ppf2*

In the graph 5.10, it is difficult to infer the overall performance of the algorithms when compared with each other, given the irregularity of the results across the samples. Further in this section we will analyse the average of the results of each algorithm, providing a more clear understanding of the results.

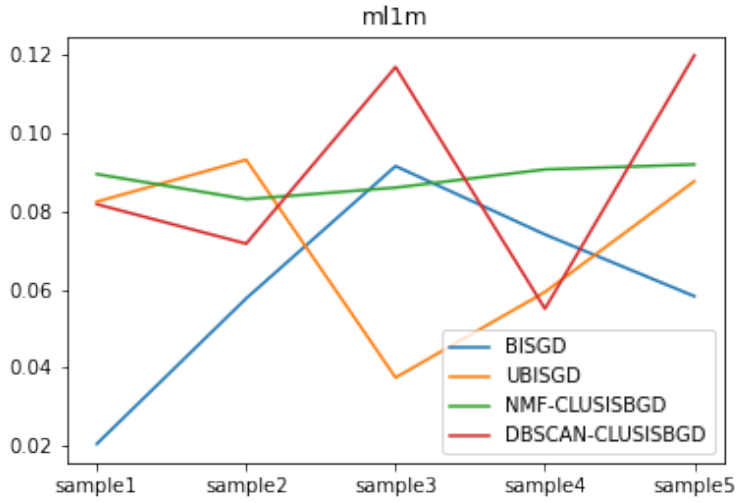


Figure 5.11: Results of samples of dataset *ml1m*

The graph 5.11, given the different spikes in the results for all the algorithms, except the **NMF-CLUBISBGD**, does not illustrate in an obvious way the overall results of the algorithms, this will be better demonstrated by the average of this results, presented further ahead in this section.

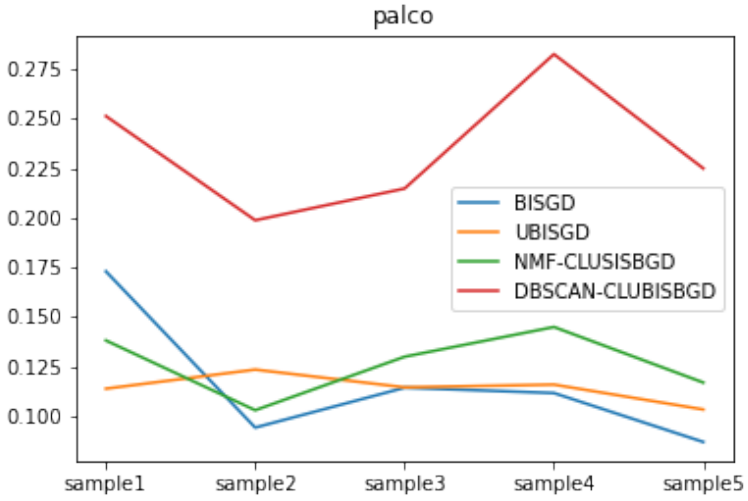


Figure 5.12: Results of samples of dataset *palco2010*

In figure 5.12 it is noticeable that the results delivered by the **DBSCAN-CLUBISBGD** algorithm are overall better then those of the other algorithms. Followed by the **NMF-CLUBISBGD** algorithm, the **UBISGD** algorithm and the **BISGD** algorithm in this order.

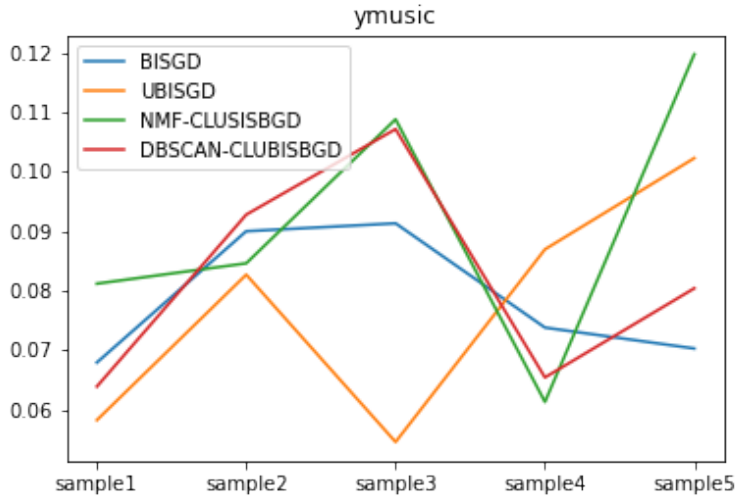


Figure 5.13: Results of samples of dataset *ymusic*

For the *ymusic* dataset, we can see in 5.13 that overall the **DBSCAN-CLUBISBGD** and **NMF-CLUBISBGD** algorithms show better results than the state-of-the-art algorithms, having with the **NMF-CLUBISBGD** algorithm better results for both the first and fifth sample.

Algorithm	Average Recall	Average run_time
BISGD	0.059 ± 0.038	0:00:9
UBISGD	0.038 ± 0.016	0:00:8
NMF-CLUBISBGD	0.048 ± 0.012	0:00:25
DBSCAN-CLUBISBGD	0.065 ± 0.028	2:44:40

Table 5.6: Average results, *ppf2* dataset

In the table 5.3.3, we can observe the average result of the different algorithms across the 5 samples of the *ml1m* dataset. The DBSCAN-CLUBISBGD algorithm shows results, recall wise, superior to all state of the art algorithms, and even better results than the NMF-CLUBISBGD algorithm. We can also observe that the time performance of the DBSCAN-CLUBISBGD algorithm is by great margin worse then the other algorithms.

Algorithm	Average Recall	Average run_time
BISGD	0.060 ± 0.023	0:00:18
UBISGD	0.072 ± 0.020	0:00:19
NMF-CLUBISBGD	0.088 ± 0.003	0:01:01
DBSCAN-CLUBISBGD	0.089 ± 0.025	0:11:37

Table 5.7: Average results, *ml1m* dataset

Considering the table 5.3.3, we can see that for the *ml1m* dataset samples, the DBSCAN-CLUBISBGD algorithm surpaces the results of the remaining algorithms, although by a small margin. Contrary to other datasets the time performance of the DBSCAN-CLUBISBGD algorithm in this instance, although still worse then the performance of other algorithms, does not worsen by as big as a margin as in other datasets.

Algorithm	Average Recall	Average run_time
BISGD	0.116 ± 0.030	0:00:17
UBISGD	0.114 ± 0.006	0:01:23
NMF-CLUBISBGD	0.126 ± 0.015	0:01:53
DBSCAN-CLUBISBGD	0.234 ± 0.029	2:08:56

Table 5.8: Average results, *palco* dataset

For the *palco* dataset in the table 5.3.3 we can see that although as stated above the **DBSCAN-CLUBISBGD** algorithm shows promising results recall wise, time wise it is by great margin the worst performing of all the algorithms.

Finally for the *ymusic* dataset, although the **DBSCAN-CLUBISBGD** algorithm shows better results then the state-of-the-art algorithms recall wise, it fall short when compared to the **NMF-CLUBISBGD** algorithm. Once again we should note that time wise the **DBSCAN-CLUBISBGD** algorithm performs poorly.

Algorithm	Average Recall	Average run_time
BISGD	0.079 ± 0.009	0:01:29
UBISGD	0.077 ± 0.018	0:01:28
NMF-CLUBISBGD	0.091 ± 0.020	0:02:00
DBSCAN-CLUBISBGD	0.082 ± 0.016	2:31:24

Table 5.9: Average results, *ymusic* dataset

5.4 Result Discussion

Taking as foundation the results presented in 5.3.3, we can make some inferences. The NMF-CLUBISBGD algorithm shows improved results, when compared to the state-of-the-art algorithms, for the datasets *ppf2*, *ml1m* and *ymusic*, showing for the remaining dataset results closed, but inferior, to the state-of-the-art algorithms.

On the other and the DBSCAN-CLUBISBGD algorithm shows superior results, recall wise, for all the datasets when compared to the state-of-the-art algorithms. But it showed in general a decrement of the time performance when compared to all other algorithms.

Although in some instances an improvement in performance, for the present algorithms, is not observed, we can still observe that clustering is an advantage in recommendation systems, and it shows that it is possible to improve results using clustering in an incremental environment.

Chapter 6

Conclusion

In this dissertation we presented two different algorithms which embedded incremental cluster into the state-of-the-art user-based bagging recommended system. The first one, **NMF-CLUBISBGD**, based on non-negative matrix factorization used the implicit clustering generated by the factor values. The second, **DBSCAN-CLUBISBGD**, based on the classic DBSCAN algorithm, required further adaptation, but keeps the density-based approach of the classic algorithm.

The results presented in 5 show that, as demonstrated in previous work [9], the clustering of users by similar item interactions improves the recall of the resulting recommendation systems. Furthermore, the results support the feasibility of clustering in an incremental environment.

For both presented algorithms the results were positive, showing improvement on the state-of-the art, although not free of limitations that invite future work.

Future directions

Considering the **NMF-CLUBISBGD** algorithm, we should note that although it presented improved results when compared to the state-of-the-art algorithms those improvements are not radical, but we should also take into consideration that in the testing environment the base line was always the optimal parameterization of the state-of-the-art algorithms. Therefor further testing of the algorithm with a more extensive parameter optimization would be a natural next step. Further more, it was shown in the clustering analysis that the clustering would stagnate after a small percentage of the entries were processed, for some datasets, to investigate this phenomena an artificial introduction of user item interactions, where we would introduced the existing user interacting with an item it had not in the past, could help assess the response of the algorithm to new information.

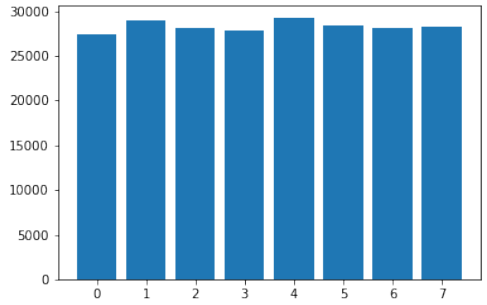
Furthermore, given the lack of positive results a particular data-set, *palco_2010*, for this first algorithm, a next step would be to further analyse the data-sets, to infer a relation between the

improvements, or not, of the results and the nature of the data-sets.

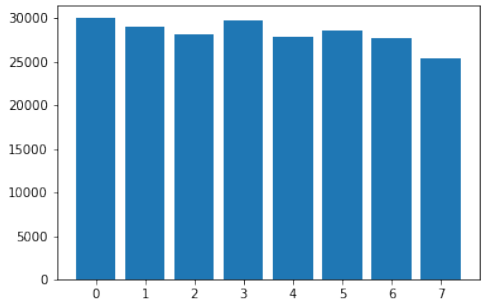
As for the **DBSCAN-CLUBISBGD** algorithm, its most apparent limitation is its time complexity, which surpasses the time marked by other algorithms by large margin. This is due to the nature of the algorithm's implementation, given that to assess the placement of a point it needs to calculate the points distance to all other points. To tackle this problem a possible approach is the use of Locality sensitive hashing [3], which would help define regions of possible clustering, therefore providing an extension of neighborhood which would serve as a smaller search space.

Appendix A

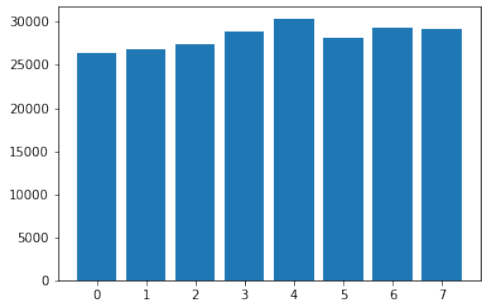
Cluster Analysis



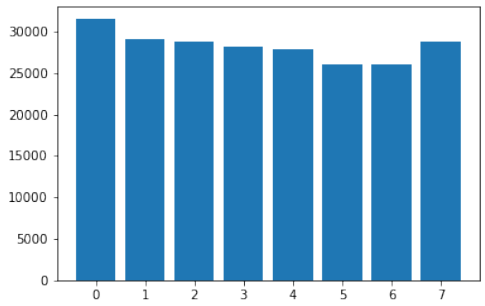
[1]



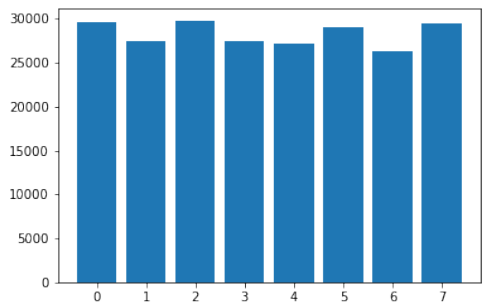
[2]



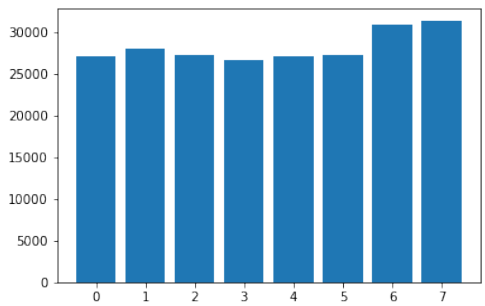
[3]



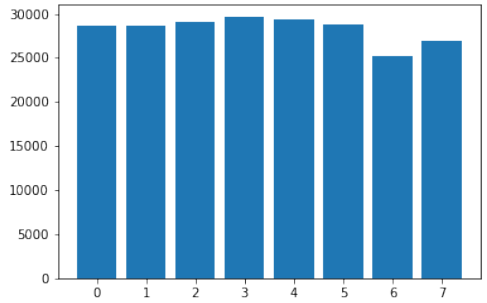
[4]



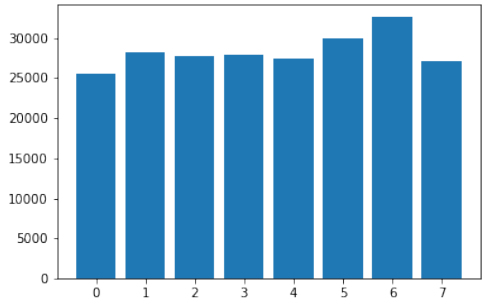
[5]



[6]



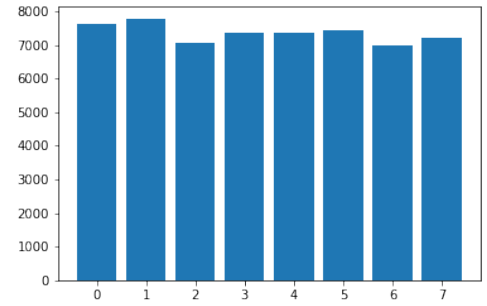
[7]



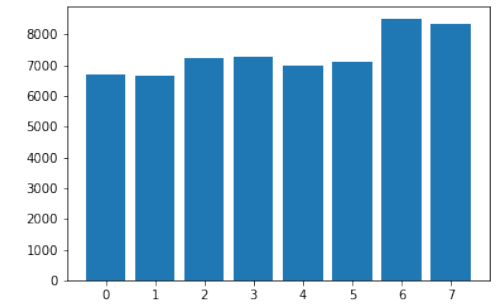
[8]

Figure A.1: Positional changes of cluster, ml1m data-set.

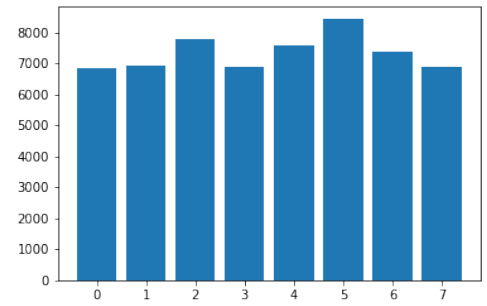
Figure A.1, presents the same analysis as the previous figure but for the data-set *ml1m*. We can draw the same conclusions for this data-set, as clusters have a balance distribution along all positions.



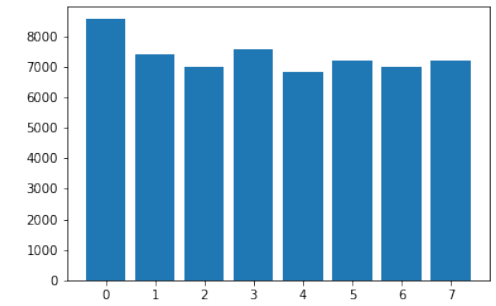
[1]



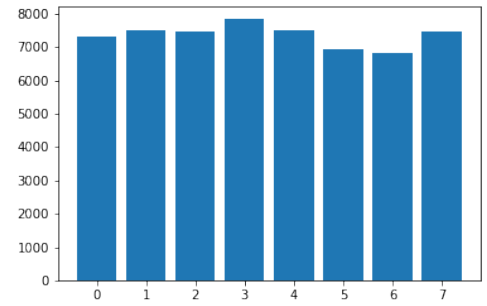
[2]



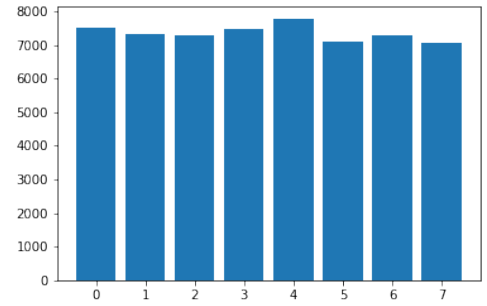
[3]



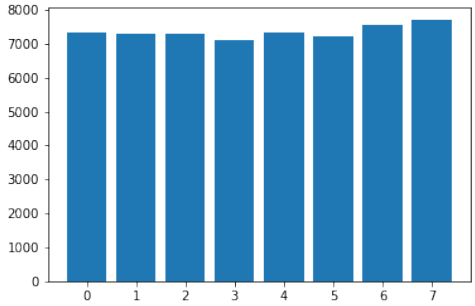
[4]



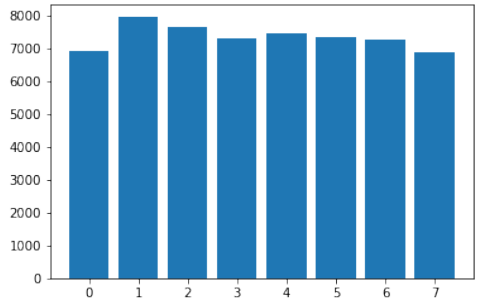
[5]



[6]



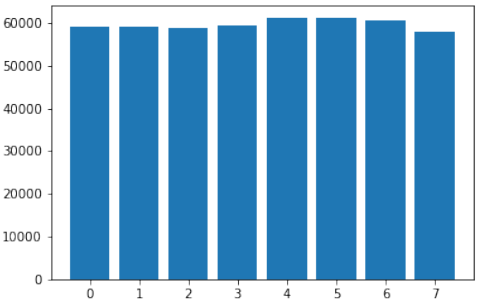
[7]



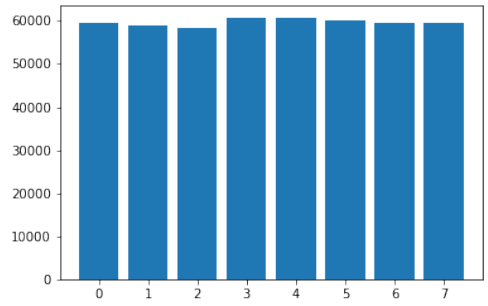
[8]

Figure A.2: Positional changes of cluster, palco_2010 data-set.

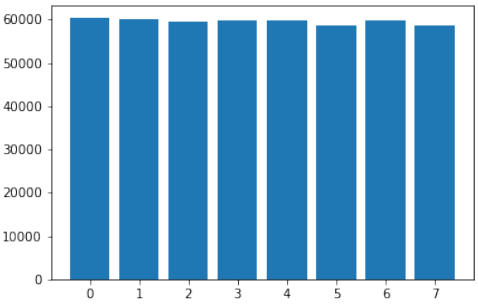
Similarly to the other data-sets in the *palco_2010* analysis in figure A.2, we can see that the clusters have a balance distribution along all positions.



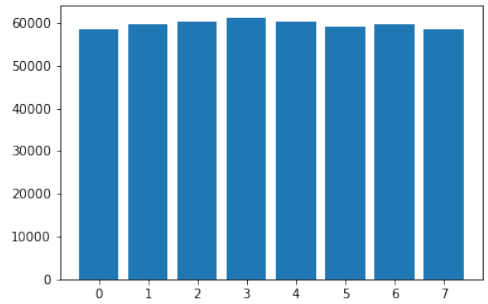
[1]



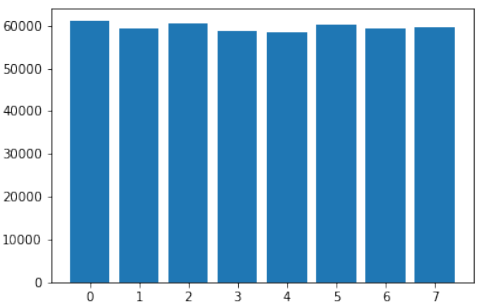
[2]



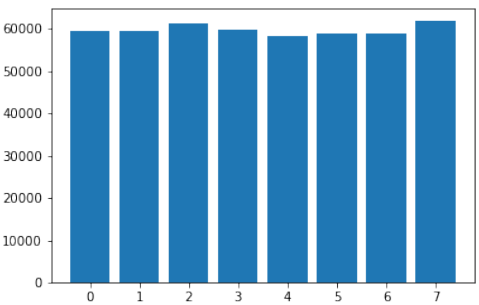
[3]



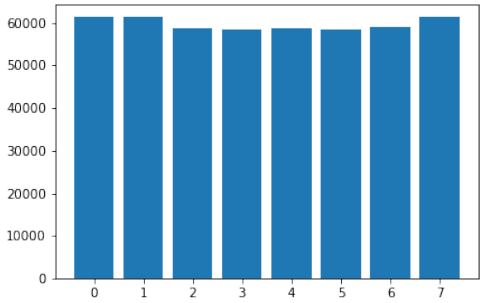
[4]



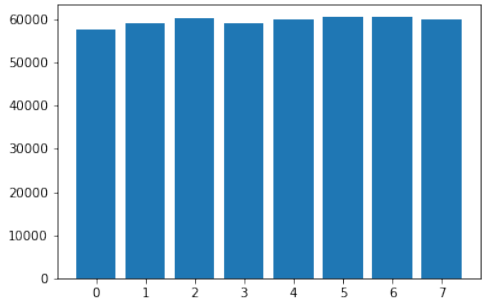
[5]



[6]



[7]



[8]

Figure A.3: Positional changes of cluster, ymusic data-set.

This last data-set *ymusic* has similar results to the previous ones, A.3 it's noticeable that the clusters have a balance distribution along all positions.

Appendix B

Parameterization

cl_num_iterations	cl_learn_rate	cl_regularization	recall	run_time
5	0.1	0.1	0.057	0:07:59
10	0.1	0.1	0.086	0:08:26
15	0.1	0.1	0.080	0:09:07
20	0.1	0.1	0.064	0:09:53
25	0.1	0.1	0.073	0:10:27
30	0.1	0.1	0.075	0:11:05
10	0.0001	0.1	0.047	0:08:31
10	0.001	0.1	0.091	0:08:28
10	0.01	0.1	0.085	0:08:20
10	0.1	0.1	0.072	0:08:23
10	0.001	0	0.061	0:08:40
10	0.001	0.0001	0.076	0:08:44
10	0.001	0.001	0.058	0:08:39
10	0.001	0.01	0.088	0:08:27
10	0.001	0.1	0.047	0:08:40

Table B.1: Parameterization ml1m data-set, NMF-CLUBISBGD

cl_num_iterations	cl_learn_rate	cl_regularization	recall	run_time
5	0.1	0.1	0.169	0:01:42
10	0.1	0.1	0.115	0:01:48
15	0.1	0.1	0.235	0:01:57
20	0.1	0.1	0.255	0:02:09
25	0.1	0.1	0.283	0:02:26
30	0.1	0.1	0.225	0:02:31
25	0.0001	0.1	0.278	0:02:24
25	0.001	0.1	0.283	0:02:29
25	0.01	0.1	0.268	0:02:28
25	0.1	0.1	0.206	0:02:29
25	0.001	0	0.171	0:02:34
25	0.001	0.0001	0.157	0:02:23
25	0.001	0.001	0.217	0:02:25
25	0.001	0.01	0.257	0:02:43
25	0.001	0.1	0.243	0:02:31

Table B.2: Parameterization palco_2010 data-set, NMF-CLUBISBGD

cl_num_iterations	cl_learn_rate	cl_regularization	recall	run_time
5	0.1	0.1	0.186	0:01:43
10	0.1	0.1	0.085	0:01:57
15	0.1	0.1	0.162	0:02:07
20	0.1	0.1	0.082	0:02:18
25	0.1	0.1	0.225	0:02:24
30	0.1	0.1	0.170	0:02:27
25	0.0001	0.1	0.182	0:02:18
25	0.001	0.1	0.18	0:02:14
25	0.01	0.1	0.132	0:02:09
25	0.1	0.1	0.260	0:02:11
25	0.1	0	0.146	0:02:21
25	0.1	0.0001	0.179	0:02:13
25	0.1	0.001	0.163	0:02:14
25	0.1	0.01	0.222	0:02:15
25	0.1	0.1	0.280	0:02:11

Table B.3: Parameterization ymusic data-set, NMF-CLUBISBGD

eps	MinPts	cl_num	cl_learn_rate	cl_regularization	recall	run_time
0.1	5	5	0.1	0.1	0.000	0:21:18
0.25	5	5	0.1	0.1	0.143	0:21:52
0.5	5	5	0.1	0.1	0.000	0:23:18
0.75	5	5	0.1	0.1	0.286	0:23:40
0.75	5	5	0.1	0.1	0.000	0:23:51
0.75	10	5	0.1	0.1	0.250	0:23:47
0.75	15	5	0.1	0.1	0.000	0:23:24
0.75	20	5	0.1	0.1	0.000	0:23:02
0.75	25	5	0.1	0.1	0.200	02:54:03
0.75	30	5	0.1	0.1	0.000	02:54:03
0.75	10	5	0.1	0.1	0.200	0:22:14
0.75	10	10	0.1	0.1	0.182	0:22:41
0.75	10	15	0.1	0.1	0.000	0:27:07
0.75	10	20	0.1	0.1	0.333	4:36:54
0.75	10	25	0.1	0.1	0.111	6:34:52
0.75	10	30	0.1	0.1	0.200	8:21:15
0.75	10	20	0.0001	0.1	0.000	5:36:22
0.75	10	20	0.001	0.1	0.250	8:22:06
0.75	10	20	0.01	0.1	0.000	1:20:25
0.75	10	20	0.1	0.1	0.000	4:39:32
0.75	10	20	0.001	0	0.143	0:22:15
0.75	10	20	0.001	0.0001	0.000	0:22:48
0.75	10	20	0.001	0.001	0.231	0:22:37
0.75	10	20	0.001	0.01	0.091	0:23:04
0.75	10	20	0.001	0.1	0.273	8:30:39

Table B.4: Parameterization ml1m_sample5_1 data-set, DBSCAN-CLUBISBGD

eps	MinPts	cl_num_$_{i}$terations	cl_learn_rate	cl_regularization	recall	run$_{t}$ime
0.1	5	5	0.1	0.1	0.167	0:09:56
0.25	5	5	0.1	0.1	0.167	0:10:37
0.5	5	5	0.1	0.1	0.000	1:00:44
0.75	5	5	0.1	0.1	0.143	0:52:45
0.1	5	5	0.1	0.1	0.000	0:10:44
0.1	10	5	0.1	0.1	0.000	0:10:14
0.1	15	5	0.1	0.1	0.000	0:10:22
0.1	20	5	0.1	0.1	0.000	0:10:43
0.1	25	5	0.1	0.1	0.000	0:10:35
0.1	30	5	0.1	0.1	0.200	0:10:11
0.1	30	5	0.1	0.1	0.143	0:10:46
0.1	30	10	0.1	0.1	0.091	0:11:02
0.1	30	15	0.1	0.1	0.500	0:11:15
0.1	30	20	0.1	0.1	0.000	0:11:08
0.1	30	25	0.1	0.1	0.125	0:11:27
0.1	30	30	0.1	0.1	0.111	0:11:20
0.1	30	5	0.0001	0.1	0.167	0:12:16
0.1	30	5	0.001	0.1	0.250	0:10:53
0.1	30	5	0.01	0.1	0.200	0:10:32
0.1	30	5	0.1	0.1	0.000	0:11:10
0.1	30	5	0.001	0	0.000	0:10:38
0.1	30	5	0.001	0.0001	0.250	0:10:11
0.1	30	5	0.001	0.001	0.000	0:09:49
0.1	30	5	0.001	0.01	0.167	0:09:57
0.1	30	5	0.001	0.1	0.182	0:10:03

Table B.5: Parameterization palco_sample5_1 data-set, DBSCAN-CLUBISBGD

eps	MinPts	cl_num_iterations	cl_learn_rate	cl_regularization	recall	run$_t ime$
0.1	5	5	0.1	0.1	0.250	0:20:34
0.25	5	5	0.1	0.1	0.000	0:22:08
0.5	5	5	0.1	0.1	0.083	0:25:20
0.75	5	5	0.1	0.1	0.125	0:23:11
0.1	5	5	0.1	0.1	0.071	0:21:27
0.1	10	5	0.1	0.1	0.143	0:21:05
0.1	15	5	0.1	0.1	0.111	0:20:29
0.1	20	5	0.1	0.1	0.273	0:20:11
0.1	25	5	0.1	0.1	0.000	0:20:04
0.1	30	5	0.1	0.1	0.167	0:19:32
0.1	20	5	0.1	0.1	0.143	0:20:36
0.1	20	10	0.1	0.1	0.167	0:20:53
0.1	20	15	0.1	0.1	0.154	0:20:49
0.1	20	20	0.1	0.1	0.105	0:20:49
0.1	20	25	0.1	0.1	0.000	0:20:29
0.1	20	30	0.1	0.1	0.111	0:20:53
0.1	20	10	0.0001	0.1	0.000	0:20:35
0.1	20	10	0.001	0.1	0.000	0:19:48
0.1	20	10	0.01	0.1	0.222	0:20:04
0.1	20	10	0.1	0.1	0.067	0:20:16
0.1	20	10	0.01	0	0.143	0:21:18
0.1	20	10	0.01	0.0001	0.250	0:21:12
0.1	20	10	0.01	0.001	0.100	0:21:09
0.1	20	10	0.01	0.01	0.167	0:20:34
0.1	20	10	0.01	0.1	0.154	0:20:23

Table B.6: Parameterization ymusic_sample5_1 data-set, DBSCAN-CLUBISBGD

Bibliography

[1] Marie Al-Ghossein, Talel Abdessalem, and Anthony Barré. Dynamic local models for online recommendation. In *Companion Proceedings of the The Web Conference 2018*, pages 1419–1423, 2018.

[2] Peter Bühlmann. Bagging, boosting and ensemble methods. In *Handbook of computational statistics*, pages 985–1022. Springer, 2012.

[3] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262, 2004.

[4] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[5] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

[6] Dino Ienco and Gloria Bordogna. Fuzzy extensions of the dbscan clustering algorithm. *Soft Computing*, 22(5):1719–1730, 2018.

[7] Shuaib Khan and VB Kirubanand. Comparing machine learning and ensemble learning in the field of football. *International Journal of Electrical and Computer Engineering*, 9(5): 4321, 2019.

[8] Gautam Kunapuli. *Ensemble Methods for Machine Learning*. Manning, 2020. ISBN: 978-3-319-14141-1.

[9] Lúcia Raquel Brandão Moreira. Online ensembles of local recommendation models. 2021.

[10] Alan Mosca and George D Magoulas. Deep incremental boosting. *arXiv preprint arXiv:1708.03704*, 2017.

[11] Rong Pan, Guandong Xu, Bin Fu, Peter Dolog, Zhihai Wang, and Martin Leginus. Improving recommendations by the clustering of tag neighbours. *J. Converg*, 3(1), 2012.

[12] Duc Truong Pham, Stefan Simeonov Dimov, and CD Nguyen. An incremental k-means algorithm. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 218(7):783–795, 2004.

[13] Ishikawa Fuyuki Quan, Truong Khanh and Honiden Shinichi. Improving accuracy of recommender system by clustering items based on stability of user similarity. 2006.

[14] Nachiketa Sahoo, Jamie Callan, Ramayya Krishnan, George Duncan, and Rema Padman. Incremental hierarchical clustering of text documents. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 357–366, 2006.

[15] Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

[16] Erwan Scornet, Gérard Biau, and Jean-Philippe Vert. Consistency of random forests. *The Annals of Statistics*, 43(4):1716–1741, 2015.

[17] A.M. Jorge Vinagre, J. and J. Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. 2014.

[18] João Vinagre, Alípio Mário Jorge, and João Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In *International conference on user modeling, adaptation, and personalization*, pages 459–470. Springer, 2014.

[19] João Vinagre, Alípio Mário Jorge, and Joao Gama. Online bagging for recommender systems. *Expert Systems*, 35(4):e12303, 2018.

[20] Dingding Wang and Tao Li. Document update summarization using incremental hierarchical clustering. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 279–288, 2010.

[21] Michael J Way, Jeffrey D Scargle, Kamal M Ali, and Ashok N Srivastava. *Advances in machine learning and data mining for astronomy*. CRC Press Boca Raton, 2012.

[22] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273, 2003.

[23] X Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the 24rd international conference on very large data, bases. Google Scholar Google Scholar Digital Library Digital Library*, 1998.