POLITECNICO DI MILANO
School of Industrial and Information Engineering
Master of Science in Computer Science and Engineering

Department of Electronics, Information and Bioengineering

Rating aware feature selection in content-based recommender systems

Supervisor: **Paolo Cremonesi**
Co-supervisor: **Maurizio Ferrari Dacrema**

Master Thesis of:
Adriana Cano,
10449566

Academic Year 2017-2018

To my brilliant friend and sister, for always being my rock and inspiration.
To mom and dad, for giving me everything.
To Ana, for always showering me with love and support.
To uncle Giovanni for all the advice and philosophy of life discussions.
To Era, for enduring the long phone calls and for the constant cheering.
To all the friends made throughout these years, for making Milan feel like home way from home.

Abstract

For many e-commerce services the amount of items being offered to the users is currently overwhelming and ever increasing. Such phenomenon, present in many different domains, makes it necessary to suggest items to users based on personal preferences for a better user experience. A system that can make such suggestions is called a Recommender System.

In essence, a Recommender System exploits the available information, being that user or item information, to make predictions about the user's ratings towards items. Content-based approaches are one of the main recommendation approaches that need item information as well as past user interactions in order to make recommendations to the user. Specifically, using past user preferences, items which are similar in content to those already liked by the user, are recommended to the latter. However, such approaches are known to have a worse performance than other approaches such as collaborative filtering or hybrid ones; one of the main reasons behind such behaviour is that item features are prone to noise and redundancy.

This thesis introduces a novel feature selection technique aiming to overcome the prominent issues of content-based approaches. Such technique utilizes past user ratings to find a feature importance through the use of a machine learning algorithm. A classification problem foundation is set up where the samples used to learn from are a feature representation of rating-aware information. The aim of such classification model is that of obtaining a feature importance which is then used for feature selection. The reduced feature set leads to dimensionality reduction, interpretability of the model being used, explainability of the results, as well as an increase in performance.

The intuitive hypothesis were backup by experiments conducted over two datasets: Movielens 10 Million and The movies dataset. The obtained results proved that our feature selection technique copes well with different types of features, almost always outperforming the state of the art baseline for a content-based approach which uses all the available features.

Keywords: feature selection, recommender system, content-based

Sommario

Per molti servizi di e-commerce la quantità di articoli offerti agli utenti è attualmente travolgente e in costante aumento. Tale fenomeno, presente in molti domini diversi, rende necessario suggerire gli articoli agli utenti in base alle preferenze personali per una migliore esperienza dell'utente. Un sistema che può dare questi suggerimenti è chiamato un sistema di raccomandazione.

In sostanza, un sistema di raccomandazione utilizza le informazioni disponibili, ovvero le informazioni sull'utente o sull'articolo, per fare previsioni sulle valutazioni dell'utente rispetto agli articoli. Gli approcci basati sul contenuto sono uno dei principali approcci alla raccomandazione che richiedono informazioni sugli articoli e le interazioni passate dell'utente al fine di fornire raccomandazioni all'utente. In particolare, utilizzando le preferenze precedenti dell'utente, elementi che sono simili nel contenuto a quelli già da lui apprezzati, le vengono poi consigliati. Tuttavia, tali approcci sono noti per avere prestazioni peggiori di altri approcci come il filtro collaborativo o approcci ibridi; una delle ragioni principali alla base di questo comportamento è che gli attributi degli articoli sono soggetti a rumore e ridondanza.

Questa tesi introduce una nuova tecnica di selezione degli attributi che mira a superare i problemi principali degli approcci basati sul contenuto. Tale tecnica utilizza passate valutazioni dell'utente per trovare un'importanza di funzionalità attraverso l'uso di un algoritmo di apprendimento automatico. Viene creata una base di un problema di classificazione in cui gli esempi utilizzati per apprendere sono una rappresentazione di caratteristiche di informazioni sensibili alla classificazione. Lo scopo di tale modello di classificazione è quello di ottenere un'importanza della funzione che viene quindi utilizzata per la selezione degli attributi. Il set ridotto degli attributi porta ad una riduzione della dimensionalità, interpretabilità del modello utilizzato, spiegabilità dei risultati, nonché un aumento delle prestazioni.

L'ipotesi intuitiva 'e stata sostenuta da esperimenti condotti su due dataset: Movielens 10 Million e The movies dataset. I risultati ottenuti hanno dimostrato chela nostra tecnica di selezione degli attributi si adatta bene a diversi tipi di attributi, superando quasi sempre le performance della baseline dello stato dell'arte per gli approchi basati su contenti che utilizzano tutti gli attributi.

Parole chiavi: selezione degli attributi, sistemi di raccomandazione, basato su contenuti

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

A recommender system is a system (platform or engine) that focuses on information filtering with the aim of suggesting to users items that they will find interesting or useful. As the world wide web continues to grow, especially in e-commerce and social media, the need for directions on what to watch, read, buy, listen to next continues to grow too hence the use of recommender systems is becoming more and more essential as time goes. The Netflix challenge proposed in 2008 is considered as the start of a new area for recommender systems that is still growing and will keep doing so considering the necessary of it.

A well-known example of a recommender system is that of Netflix, where the users watch movies and the system, based on their preferences, recommends to them other movies that they are likely to like. In order to make recommendations the system needs information about the users as well as the items that are being recommended.

There exist three principal approaches to build a Recommender System: content-based, collaborative filtering and hybrid.

1.1.1 Content-based approach

Content-based approaches are those approaches that use item information (item attributes also called features) to find similarity between items and then recommend to users items similar to those that the user had already liked. How this works in practice is that book *100 years of solitude* of Gabriel Marquez is similar to Ismail Kadare's *Broken April* being both bestsellers of the magical realism writing style and if the systems knows that a user likes *100 years of solitude* it recommends to that user the book *Broken April*. However, in literature such approaches are known to perform quite poorly with respect to the other ones and most often this is due to the fact that features are prone to noise and redudency.

1.1.2 Collaborative filtering approach

Collaborative filtering approaches instead are based on similarity of user preferences, that is to users with similar taste recommend items that one likes and the other hasn't interacted with yet. Take for example user Veronica which has liked a set

of movies *The Great Beauty*, *A star is born*, *Manchester by the sea*, *Pride and Prejudice*, *Call me by your name* while user Marina has liked *Call me by your name*, *Pride and Prejudice*, *Silver Lining Playbook*, *Juste la fin du monde*; we recommend *Silver Lining Playbook*, *Juste la fin du monde* to Veronica and *The Great Beauty*, *A star is born*, *Manchester by the sea* to Marina given that they have shown similar taste in movies. These approaches are divided mainly in two categories: item-based, where similarities between items based on the ratings given to them are found, and user-based where similarities between the users are calculated based on the ratings given to items.

1.1.3 Hybrid approach

The hybrid approach is one that combines the two previous methods, trying to surpass in the drawbacks of one method with the other: a well-known problem is that of cold-start items, which are items that are new and with no user interactions and hence unable to be used in the collaborative filtering approach. The content-based approach instead can solve this issue. Therefore using both of the approaches leads to better recommendations.

1.2 Problem definition

Optimizing the performance of content-based approaches has been the focus of many works during the recent years. One of the techniques used to do so is feature selection: while the intuitive reasoning is that more data means more information and a better performance of the algorithms in practice that is not always true. Information can be noisy, mislead the learning algorithm and therefore lead to erroneous predictions thus a lower performance. This has led to discover approaches to apply feature selection, categorized in three types: Filter, Wrapper and Embedded methods. The general strategy for all approaches is that of analyzing the impact of the features on the final prediction being done and through feature selection techniques to have a ranking of these features with respect to their impact on the performance. Once we have an importance ranking of the features we can select the best subset to keep reducing data dimensionality, often improving the performance of the model and increasing explainability of what is being done.

1.2.1 Proposed work

In this thesis we propose a new feature selection technique for content-based recommender systems. Through the use of a classification machine learning algorithm we extract the importance per feature where the samples of the model are generated in a feature representation of past user ratings. Starting with the extraction of user feedback towards the items we then proceed to generate positive and negative samples that are representation of the features in common between the items with a positive feedback and those with a negative feedback. The reoccurring features among the samples created are very likely to be relevant ones that we can keep and discard the rest without losing information nor quality of the recommender. It is the task of the classification model to calculate the importance of each feature. Once

we have such importance the feature selection is done and a reduced feature set is obtained.

The confrontation of the content-based recommendations made using only the reduced feature set with the baseline of using all available features is done. Our approach makes it possible to accompany the application of a content-based approach with an explanation and interpretation of the approach. It also shows that such technique copes well with different types of features, namely metadata and social tags. Furthermore, we show that a reduced feature set can often yield better quality in recommendations with respect to using all the features present in a dataset.

1.3 Structure

The work is structured as follows: Chapter 2 introduces the state of the art, a summary of recommender systems, feature selection in machine learning and feature selection in recommender systems as well as the mathematical tools needed, Chapter 3 proceeds with an introduction of our work, the model explained thoroughly in all its steps and Chapter 4 gives a detailed description of the datasets as well as the dataset split done to conduct the experiments. Chapter 5 presents the results obtained by applying our technique to two datasets as well as the confrontation with the baseline. In Chapter 6 conclusions are extracted and possible future works are discussed.

Chapter 2

State of the art

Recommender Systems are a sub-field of information filtering systems that have the intent of predicting a user's preferences towards an item. The purpose of a Recommender system varies from application to application, e.g. Amazon recommends products, Facebook recommends friends, pages or groups, Netflix recommends movies, Spotify recommends songs and playlists and so on. As e-commerce activities increase, so does the need to help the users towards a better experience by guiding them through recommendations of items that they are most likely to buy/watch/listen to/read instead of being lost in the huge number of items being offered. An item in the world of RS can be just about anything e.g., book, song, playlist, product, movie and application. In order to do so, information about past behaviour of users as well as information about users and items is needed. With the use of all the available information various methods have been created over the years to make recommendations possible like Collaborative Filtering, Content-Based Filtering as well as Hybrid Filtering.

This chapter is organized as follows. In Section 2.1 we introduce the basic notions of a recommender system, in Section 2.2 we introduce the different models used to make recommendations followed by the different similarities needed for many of such models in 2.3. This is followed by the introduction of the most used evaluation metrics in Section 2.4. In Section 2.5 we introduce the notion of feature selection, in machine learning as well as in recommender systems. In Section 2.6 the xGBoost algorithm is introduced followed by the introduction of the Bayesian optimization technique for hyper-parameter tuning in Section 2.7.

2.1 Fundamentals of Recommender Systems

The use of a recommender system can be very broad, vary from application to application but the goal remains always the same: suggest items to users that are of their interest. In order to make recommendations information is needed, information about the users, interaction of the users with the items and information about the items.

In literature, almost all approaches are based on the user past behaviour towards the items of the system and item content or user content, usually called item profile and user profile.

2.1.1 Recommender system taxonomy

A recommender system can be a complex concept to explain, especially considering the spectrum of human beings' preferences that it tries to include when making the recommendations. However, there is a general structure that is common for any application.

As it is described in [4], a recommender system is based on a combination of elements:

- Type of data
- The filtering algorithm: how the information is filtered to make recommendations. Can be categorized in four types:
  - **Collaborative Filtering**: the most extensively used approach which has as its fundamental core the comparison of user preferences towards items and then use the similarity between users to makes suggestions.
  - **Content Based Filtering**: an approach that uses information about the items as well as past behaviour of users. Knowing past preferences of the user through his behaviour and similarity between the items based on content of the item we can recommend items similar to those the user liked in the past.
  - **Demographic Filtering**: uses demographic information about the users such as nationality, age, gender etc. Similar to the collaborative filtering but with the difference that similarity between users is calculated using demographic information.
  - **Hybrid Filtering**: a combination of than one filtering approach with the prospect of increasing the model performance. State of the art shows that an ensemble of models has most certainly proved to be the most effective recommendation method. It is also an approach that overcomes some of the common problems of the RS world, like cold-start problem, overspecialization problem and sparsity problem.
- The model chosen: *Memory-based* or *Model-based*
- The employed techniques
- Sparsity level of the database and desired scalability
- Performance evaluation
- Desired quality of the results

2.1.2 Ratings

A rating is an interaction of a user with an item and it is the basis of any algorithm. The type of ratings depends by the system at hand, the possible data collected from the user interactions given how the system in question is built. The ratings are an indication of likes or dislikes of the users and to make it as explicit as possible ratings are usually a scale that indicates the level of like. Ratings can therefore be

continuous values, values from an interval like { -5, -3, 0, 3, 5} where the extremes -5 and 5 mean extreme dislike and like respectively, the most common case values from a set {1,2,3,4,5} or in other cases a simple option of like/dislike.

Explicit ratings

Explicit ratings are those ratings where the user gives an indicative value to the item, the most common ones being 5-point (e.g. rating product in Amazon) and in more rare cases 7-ratings or 10-ratings. This type of ratings is very informative as it give a semantic interpretation of the user's interest towards the items in the database and a clearer profile of the user is created. However, the issue with numerical rating is that of unbalance with respect to the semantic interpretation of user preferences, e.g Amazon ratings go from 1 to 5 where 5 means "love it", 4 "really like it" and 3 "like it" and the remaining two are for negative feedback. In this case we have an unbalanced setting. There exist also binary and unary ratings where binary ratings are those where the user has the option expressing his/her like or dislike and unary ratings are those that the user has only the option of expressing his like.

Implicit ratings

Implicit ratings instead are those ratings where we count on the interactions of the users with the items where by interaction is meant the simple act of a user viewing or buying an item. It is a kind of unary rating where the customer preferences are extracted from the user behaviour rather than their explicit expression of interest.

2.1.3 Data structures

The information to be used by the recommender system is quite vast and the methods to elaborate it are many but how such information is structured is the same for all scenarios. The user-item interaction information as well as item and user profiles are defined in form of three matrices, respectively User Rating Matrix, Item Content Matrix and User Content Matrix. Being U the set of all users in system, I the set of all items and F the set of all features we have the following data structures:

- **User Rating Matrix** (URM), a matrix of ‖U‖x‖I‖ size where each cell is a rating given by the user u to an item i, that is we have the user preferences stored in each row of the matrix. This matrix can be very sparse when ratings are lacking. It is very common that in a system users does not spend their time in rating the items which leads to few data and a sparse URM. From hereon, we will use the notation r_u^i to identify the interaction between user u and item i, that is the value of the URM at row u and column i.

- **Item Content Matrix** (ICM), a matrix of ‖I‖x‖F‖ size where each cell tells us the relation of the item i with feature f, that is in each row we have the content of each item through the features. The values that each cell can take vary from integer to continues and Boolean. From hereon, the notation c_i^f will be used to identify the value of the ICM at row i and column f, which is if item i has feature f.

- ✍ **User Content Matrix** (UCM) is a matrix of‖U‖x‖F‖ size where each cell tells us the relation of a user u with a feature f, that s in each row we have the user's profile. This third type of matrix is rarer and generated for the hybrid approaches.

2.2 Recommender Systems models

The basic approaches of Recommender Systems are categorized with respect to the kind of data they use to make the recommendations: the user-item interactions, so the URM, or the attribute information, so the ICM. The approaches that use the URM are called Collaborative filtering approaches while those that use the ICM are called Content-based filtering approaches. For the second type however, the recommendation system is not solely based on the attribute information, it also needs the user-item interaction to make personalized ratings to a user based on past preferences whereas in the case of Collaborative information about the whole community of users is collected and used. Both of these approaches have their pros and cons and through studies have proven that combining them together can lead to better performance of the recommender system. Such system that combine both Collaborative filtering and Content-based filtering models are called Hybrid Systems.

For these three main approaches different models are defined and the research of new possible approaches is ongoing however in the following sections the most common models in literature are summarized.

2.2.1 Content Based filtering

Content based approaches are those approaches that are based on the attribute information, more precisely the features of the items where a feature can be anything from movie genre, song artist or book publisher, so metadata or social tags given by users to the items usually through a description or a review. These attributes are a description of the item and its content hence the name of the model. These types of models combine information about the interests of the user towards the items and then explore to find other items similar to those that the user already liked based on the content of the items. For example, we have that Mandy has rated highly the book A Walk to Remember and that book has no other ratings. In this case a Collaborative Filtering approach would yield to nothing given that we don't have enough information to apply it. This is where a Content-Based approach comes into play: looking at the attribute information, we can find other books similar to *A walk to remember* like *The Notebook* of the same author and books genre or *The fault in our stars*, different author but the same genre and topic.

ItemCBFKNN

Suppose that the model wants to predict the rating $\hat{r}_u^i$ that u gives to item i. To make such prediction we need to define a similarity measure between items, sim(i,j) and make use of it. The predictions is then defined as:

$$\hat{r}_u^i = \frac{\sum_{j \in I} r_v^j sim(i,j)}{\sum_{j \in I} |sim(i,j)|}$$

There main advantages of this approach are [16]:

- **User independence**: a content-based approach is based on the sole ratings of the active user unlike a collaborative filtering approach where ratings from other active users are needed as well to generate the neighbourhood in order to make the recommendations.

- **Transparency**: as the basis of the prediction is item content it can easily be used to explain the recommendations made to a user. The features of an item are quite self-explanatory and a good indicator whether or not to trust the system at use. E.g. on Netflix after watching *Mamma Mia!* you find a section called "Because you watched *Mamma Mia!*" and the list of the recommended movies is very similar to *Mamma Mia!* because of its content, being that the genre, the same director or actors in common. The reason behind such recommendation is clear to every user of the system. In collaborative filtering instead, transparency is not possible; the only motivation behind the recommendations is that a group of unknown users with similar likes to the active user liked the item.

- **New item**: content-based approaches do not require for an item to be rated to recommend it, therefore they overcome the item cold-start problem that collaborative filtering instead faces. In the case of collaborative filtering, an item is recommended only when it has sufficient ratings by users.

However, this approach also has its shortcomings:

- **Limited Content Analysis**: content-based techniques are quite depended from the number and quality of information about the item so that the model can distinguish items to the user's taste. Domain knowledge if most often needed to better understand and evaluate the goodness of the information to be used. The insertion of features, being that manual or automatic could not be enough to distinguish aspects of items so that the recommendations made are in line with the user's preferences.

- **Over-specialization**: content-based approaches are limited with respect to unexpected suggestions that is these kinds of approaches are not able to recommend something new and different from what the user has shown interest for. This problem is called serendipity problem which refers to the lack of novelty in the recommendations made by the model.

- **New user**: while the content-based approaches solve the item cold-start problem they do not solve the user cold-start problem that is they cannot make recommendations to new users that have no rating. A fundamental aspect of the approach is that it makes use of the user's past preferences to recommend the new items and it needs sufficient information about the user preferences to work well.

2.2.2 Collaborative filtering

Collaborative filtering methods are methods that base their rating prediction in the user-item interactions as a collaborative group. The most critical challenge that this approach faces is the sparsity of the URM which is due to the fact that users are not very keen on rating everything that they watch but also due to the fact that the amount of items in a system is quite large and it's not very likely that a user has interacted with all or a major part of the item offered by the system.

These kinds of approaches are built upon the philosophy that human beings with similar taste can be a good source for recommendations among themselves. In literature Collaborative filtering approaches are the ones that yield better results with respect to Content-based filtering approaches. As defined by [1] there are two kind of collaborative filtering approaches: *memory-based* and *model-based*.

Memory based approaches

Memory based approaches are neighborhood approaches (k-NN) where the rating of a user-item is based on the user-item rating of the nearest neighbors of the user. With nearest neighbors of a user u is understood the users that are most similar to user u based on their preferences towards the items. These kinds of approaches are the earliest methods introduced to make recommendations. In a Collaborative Filtering approach the neighborhoods can be obtained in two ways:

- **User-based Collaborative filtering** are those approaches that find a similarity between users based on the ratings given to items, that is, users that have rated items similarly tend to have similar taste and therefore be considered similar and neighbors. For example, we have that Andy and Lucy rated tv-series similarly in the past and Andy has rated The office which Lucy hasn't; we can use such rating for the prediction of the interaction Lucy-The office. The notion can be expanded to an entire set of neighbors, where the prediction for Lucy - The Office can be computed as the average of Neighborhood - The office ratings in the dataset. In a more practical sense, the rows of the URM are used, finding the similarity between them through a similarity measure and from there the K most similar neighbors per user are considered.

  Consider user u, this approach predicts the rating $\hat{r}_u^i$ that user u gives to item i using the ratings given by the nearest neighbours to item i. Consider also the sim(u,v) which is the similarity between user u and user v we have:

$$\hat{r}_u^i = \frac{\sum_{v \in U} r_v^i sim(u,v)}{\sum_{v \in U} |sim(u,v)|}$$

- **Item-based Collaborative filtering** instead finds the similarity between items based on how such items are rated by users and then use this item-similarity to recommend to users items similar to past preferences of the user. For example, to predict the rating that user Jenny gives to book Love in times of cholera we need to find a set of books similar to Love in times of cholera and use the rating given by Jenny to such books for the Jenny - *Love in times of cholera rating*. In this case, columns of the URM are used to find the

similarities between items creating so the neighborhood and keeping only the top neighbors for the rating prediction.

In this case the similarity is defined between two items sim(i,j) and the prediction of the model for the rating $\hat{r}_u^i$ of item i given by user u is given as:

$$\hat{r}_u^i = \frac{\sum_{j \in I} r_u^j sim(i,j)}{\sum_{j \in I} |sim(i,j)|}$$

Model-based approaches

These approaches instead differ from the previous ones for the fact that a model is trained and stored to be used in the future to predict the ratings given by users to items. An approach much similar to machine learning and data mining models, where the model is trained on a portion of the overall data with the aim of learning general truths from it and then used for future predictions on unknown ratings that is on making new recommendations. These models can be from decision trees, rule-based models, Bayesian methods to latent factor models.

Two of the most common approaches are *Sparse LInear Model* (Slim) [18] [7] and *Matrix Factorization* [15].

- **Slim**: predicts the rating of an item i by user u by as a sparse aggregation of items already rated by user u. The model can be summarized as:

$$\tilde{A} = AW$$

where A is the binary URM and W is $n \times n$ matrix of aggregation coefficient and n is the number of items. The model than learns the W matrix using URM as the ground truth and minimizing the following regularized optimization problem:

$$\begin{aligned} minimize_w \quad & \frac{1}{2}\|A - AW\|_F^2 + \frac{\beta}{2}\|W\|_F^2 + \lambda\|W\|_1 \\ & subject\ to \quad W \geq 0 \\ & diag(W) = 0 \end{aligned} \tag{2.1}$$

- **Matrix Factorization**: is the base of the most successful latent factor models. It decomposes the URM into the product of two lower dimentionality rectangular matrices.

$$U\hat{R}M \simeq UV^T$$

In its basic for, considering L as the latent factor we represent U as $\|U\| \times \|L\|$ and V as $\|I\| \times \|L\|$ Each columns of U and V is referred to as latent vector or component while each row of U and V is referred to as latent factor. This way we can refer to U for users and V for items. When the learning is completed we can obtain the rating of a user for each item as:

$$\hat{r}_u^i = U[u] \times V[i]^T$$

The most delicate step is that of computing the mapping of each user and item to the latent factor. Various matrix factorization methods have been proposed over the years where the two most common one are Single Value decomposition and Alternating Least Square.

There are many distinctions between the memory-based approaches and the model-based one: that memory-based approaches are usually far simpler and easier to understand but have the downfall of not being able to cope with the sparsity issue usually present in the URM which in turn makes them an heuristic approach while model-based approaches are more complex and far more difficult to explain but do not face the sparsity issue nor are they a heuristic approach. However, attempts to put the two approaches together have been done and have led to promising results[14].

2.2.3 Hybrid filtering

Different approaches use different sources of information to make the recommendations: CF make use of community ratings and CBF instead make use of the item description and the user's own ratings. This makes it so that each one of them work well in different scenarios, usually all depending on the quality and quantity of the data available to learn from. As in many other fields of data science, the combinations of many different aspects of the same data can lead to a better performance with respect to using an approach on its own. An ensemble model is most often, if not always, the best approach in any machine learning or data mining application, combining different algorithms together. The same principal can be applied in recommender systems as well: hybrid approaches, that given a broad range of input put together the different types of recommender systems to perform the same task. Different models of the same type can be put together to obtain a better performance as well. The possible model combinations are endless, which [4] groups in four types: implement Collaborative filtering and Content-based filtering methods individually and aggregate their predictions, integrate Content-based filtering characteristics in a Collaborative filtering approach or vice-versa and last by constructing a model that integrates both Content-based filtering and Collaborative filtering characteristics together.

2.3 Similarities

In both Collaborative filtering and Content based approaches it is necessary to define the concept of similarity, that is to define a distance measure. To make it possible to calculate such similarity a vector or set representation of the user ratings as well as the item content is made so that mathematical approaches can be applied.

2.3.1 Cosine similarity

One of the most common similarity measures. It uses the mathematical notion of cosine of the angle between two vectors. Let A and B be two vectors of an n-dimensional space, the cosine similarity between the two is defined as:

$$cos(\mathbf{A}, \mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||}$$

where $\cdot$ is the dot product between the vectors and $|||\mathbf{A}|||$ indicates the norm of the vector A. The similarity values vary from -1 to 1 where -1 means exactly opposite, so not similar at all, 1 means overlapping, so identical and 0 means orthogonality.

2.3.2 Jaccard similarity coefficient

The Jaccard coefficient, also known as the intersection over union, is a method that measures the similarity and diversity of finite sample sets. It is obtained by dividing the size of the intersection with the size of the union of the sample set. Consider A and B two sets:

$$J(A,B) = \frac{\|A \cap B\|}{\|A \cup B\|} = \frac{\|A \cap B\|}{\|A\| + \|B\| - \|A \cap B\|\|}$$

where $\|A\|$ indicates the cardinality of set A. If both sets are empty, the Jaccard coefficient is by usually equal to 1. J(A,B) $\in$ [0,1], where with similarity equal to 1 means that the sets are completely overlapping and with similarity equal to 0 means they are completely different.

2.3.3 Tanimoto similarity

Tanimoto similarity coefficient, also known as Extended Jaccard, has the same scope of comparing sets as the Jaccard coefficient with the difference that it has been adapted for real valued vectors. Considering A and B as two vectors, the Tanimoto coefficient is defined as:

$$T(\mathbf{A},\mathbf{B}) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||^2 + ||\mathbf{B}||^2 - \mathbf{A} \cdot \mathbf{B}}$$

With A and B being vectors with binary values we have once again the Jaccard coefficient.

2.3.4 Tversky similarity

The Tversky index is an asymmetric similarity measure between two sets that takes values in the range [0,1]. It can be seen as a generalization of the Tanimoto coefficient and the Dice coefficient. Consider A and B as two sets, the Tversky coefficient can be defined as:

$$Tv(A,B) = \frac{\|A \cap B\|}{\|A \cap B\| + \alpha\|A - B\| + \beta\|B - A\|}$$

where A-B denotes the relative complement of B in A.

2.4 Evaluation

The evaluation process of any kind of model is one of the most crucial steps of the whole prediction. It can give us an understanding of the effectiveness of the recommendation system applied to the problem. There are different factors that we need to keep in consideration so quite often it is necessary to consider more than one metric at a time to evaluate the goodness of the approach used. The evaluation process is quite difficult in its nature because it can lead to the wrong conclusions, being that underestimating or overestimating the approach at question.

The evaluation of a recommender system can be done principally in two ways:

- ✍ **Online evaluation**: active users are needed that interact immediately with the recommendations made by the system. For example, for a job recommendation system like in LinkedIn we can measure the conversion rate by counting how many of the recommended job offers were opened by the user. This kind of approach is usually referred to as A/B testing. The direct impact of the approach is measured through this kind of testing. However, the need of active users to perform such evaluation is a major drawback and makes it infeasible for research purposes.

- ✍ **Offline evaluation**: are made on historical datasets. These are the most common type of methods in research as the most practical ones. It is effective because not only does it not need active users, it also gives the possibility to evaluate the proposed recommender system on different datasets and assure a greater generalization power.

For any type of evaluation there are two axes to give importance to: accuracy and ranking. While the general goal of a recommender systems is that of being accurate in its prediction, the ranking is also crucial. For example, consider user Juliette in a music streaming service that is exploring the section of recommendations of the system; we have a list of 15 songs to suggest, consider that the best five songs for Juliette are in the end of the list. For Juliette to discover these songs she would have to go through all other 10 songs first and that is not good because it is very likely that the user will get tired of listening to songs they don't like and it's also very likely that they don't make it to the end of the list of suggestions.

For the rest of the work, offline evaluation is considered therefore the metrics considered are also for this type of evaluation. The general purpose of an offline evaluation is that of filtering out the possible algorithms to apply to the problem at hand [9].

In order to choose the right metric to evaluate your model on the properties in terms of user experience that the system is aiming to satisfy are crucial and that can vary from system to system, like accuracy, scalability, robustness and so on. There are three main categories of metrics used for evaluating a recommender system: *predictive accuracy metrics*, *usage prediction metrics* and *ranking accuracy metrics*.

2.4.1 Predictive Accuracy metrics

The notion behind these types of metrics is that of measuring the predictive power of the recommender system in terms of how close the predicted rating is to the actual rating given by the user. However, being the purpose of a recommender system that of suggesting items to users which the user can decide to interact with or not the exact value predicted by the system for the user-item couple is not much of interest. It is far more interesting to see just the interaction or lack of it after the suggestion is made.

Root Mean Square Error

Root Mean Square Error (RMSE) is perhaps the most popular metric used in evaluating accuracy of predicted ratings[9]. Being r_u^i the real rating given by the user to the item, $\hat{r}_u^i$ the predicted rating and U_T as the test set of user-item pairs (u,i) for which we know the true rating r_u^i we calculate RMSE as:

$$RMSE = \sqrt{\frac{1}{\|U_T\|} \sum_{(u,i) \in U_T} (\hat{r}_u^i - r_u^i)^2}$$

2.4.2 Usage Prediction metrics

What is of particular interest in a recommender system whether the user makes something of the suggestion made to him/her or not. Consider Netflix for example, it suggests movies and you consider it to be a good recommender system if it suggests movies that you will watch, no matter what your actual rating of the movies are. The same can be said about Spotify for example, where a recommended song or playlist are listened to or not by the user is what matters. Another example can be that of LinkedIn job posting recommendation, where the act of opening a job posting is an indication that the user shows interest toward what was recommended to him. The summarization of the user reaction to recommendation can be made through the confusion matrix, that adapted to a recommender system application by [9] is as follows:

	Recommended	Not Recommended
Used	True Positive (**TP**)	False Negative (**FN**)
Not used	False Positive (**FP**)	True Negative (**TN**)

The interpretation for each outcome is:

- ✍ True-Positive: those items that were recommended and that were effectively relevant for the user
- ✍ False-Positive: those items recommended but not relevant for the user, so wrongful recommendations.
- ✍ False-Negative: those items that were not recommended but should have been
- ✍ True-Negative: items not recommended and that should not have been recommended

Precision

Precision is a metric that measures how many of the recommendations made were relevant for the users.

$$precision = \frac{TP}{TP + FP}$$

Precision values lie in the range [0,1] where with precision equal to 1 we have the ideal recommender and precision equal to 0 the opposite.

Recall

Recall is measures how many of the relevant items were recommended by the recommender system.

$$recall = \frac{TP}{TP + FN}$$

Recall values lie in the range [0,1] where with recall equal to 1 we have the ideal recommender and recall equal to 0 the opposite.

F1 measure

F1 is the harmonic average of precision and recall, takes values from 0 to 1 where 0 is the worst performance possible and 1 where precision and recall are ideal.

$$F1 = 2\frac{precision \cdot recall}{precision + recall}$$

2.4.3 Ranking metrics

For this class of metrics, we focus on the order the recommendations are presented to the user. When recommending a list of items to a user, the order in which they are presented is fundamental for the user experience: the better the quality of the ordering, the more satisfying is the navigation experience of the user is. There are two ways to measure the accuracy of the ranking: we can try to determine the correct order of a set of items for each user and measure how close a system comes to this correct order or attempt to measure the utility of the system's ranking to a user [9]. For our work, we gave importance to the user's utility and the metrics of this class are:

Mean Average Precision

Mean Average Precision (MAP) is considered the most relevant metric in the recommender systems applications. It is the mean of the Average Precision (AP) of every recommended list of length n. Average precision itself defined as:

$$AP@n = \frac{\sum_{k=1}^{n} Precision@k \cdot Rel(k)}{Number\ of\ relevant\ documents}$$

where Rel(k) is 1 if the item at rank k is relevant and 0 otherwise. The definition of MAP then becomes:

$$MAP@n = \frac{\sum_{k=1}^{n} AP@k}{n}$$

It takes values in the range [0,1] and the higher the better, where equal to 1 means ideal case of recommending only relevant items. Substantially MAP is a variant of the Precision metric that keeps track also of the ranking, so it keeps track of the ordering of the recommendation list.

Normalized Distributed Cumulative Gain

Normalized Distributed Cumulative Gain (NDCG) is a ranking metrics, with values are in the range of [0,1], that provides two special features: a weight over the ordering of the items and it allows for an evaluation of graded relevance instead of a binary one. For simplicity, we have considered a binary relevance for the experiments conducted.

NDCG is a normalized version of Distributed Cumulative Gain (DCG) and DCG itself is weighted sum of the degree of relevance of the ranked items where the weight,

or discount, is a decreasing function of the rank of the item [24]. To obtain the normalized version of DCG we divide it by the best DCG ranking result, called Ideal DCG (IDCG). The discount function though is not fixed and can be adapted to the problem in question. The most common function in literature however is the logarithmic one:

$$D(r) = \frac{1}{\log(1+r)}$$

where r denotes the item rank. DCG is then calculated as:

$$DCG = \sum_{r=1}^{n} y(r)D(r)$$

where y(r) $\in \{0, 1\}$ and y(r) = 1 means that the item at rank r is relevant and y(r) = 0 means the opposite and n is the number of recommended items. IDCG instead is defined as:

$$IDCG = \sum_{r=1}^{n} D(r)$$

The NDCG is then defined as:

$$NDCG = \frac{DCG}{IDCG}$$

NDCG values are always in the range [0,1].

Mean Reciprocal Ranking

Mean Reciprocal Ranking (MRR) is the average of the Reciprocal Ranking (RR) of all the recommendation lists across the users where RR itself measures how high the first relevant item is ranked in a recommendation list for a user. Defining $rank_u$ as the highest rank of a relevant item in the recommendation list for user u, the MRR is expressed as:

$$MRR = \frac{1}{\|U\|} \sum u \in U \frac{1}{rank_u}$$

For top-k recommendation, we define that if $rank_u$ k then $\frac{1}{rank_u} = 0$. MRR is of particular interest to evaluate the quality of a short recommendation lists like top-3 or top-5.

2.5 Feature selection

Feature selection is becoming always more essential as the amount of data increases as well as its dimensionality which also means an increase in computation time and difficulty of doing data analysis. The benefits of applying feature selection during data pre-processing are many: avoid the curse of dimensionality, reduce computation time, generalization of the dataset as well as simplification of models which leads to easier interpretation.

Feature selection is a machine learning technique that reduces the dimension of data by keeping a subset of the data variables (features) without reducing the

accuracy of the model through the identification of irrelevant information and highly correlated features among them or improving it when identifying noisy information and thus removing it. A feature is an individual measurable property of the process being observed [6]. For each item in the dataset a representation through its features is possible, providing us with information about the item and making it possible to then train a model and find the correlations between the different features as well as correlation with the output of the model that is being trained. In order to do feature selection a criterion needs to be defined for how to choose the features to keep and that can be quite challenging.

Feature selection is not to be confused with other feature engineering process: it does not introduce new features, all that it does is choose from those given as input. Feature weighting instead is another feature engineering technique where for each feature a weight is calculated and features are then multiplied with such weight before applying any model to the available data. Feature selection can be seen as special case of feature weighting where a weight equal to zero is associated to the features that are not selected.

Many approaches have been introduced during the years for the various fields that work with huge amount of data as this is a problem that has been present since day one applying Machine Learning algorithms to a problem.

2.5.1 Feature selection in Machine Learning

Applications of Machine learning algorithms can vary but an issue that remains throughout of them is that of data dimensionality. The focus of feature selection (variable elimination) is that of selecting a subset of features which best describes the data, are not redundant and are not noisy.

The many approaches that are used to do feature selection can be categorized in [6]:

- ✍ Filter methods
- ✍ Wrapper methods
- ✍ Embedded methods

Filter methods

Filter methods are the simplest kind. The core idea of these methods is that of scoring a value for each feature through a ranking criterion and then use a threshold for the selection, keeping everything above the threshold and discarding everything below it. The most important relation that we need to evaluate for a feature is that with the outcome of the model so the ranking criteria needs to take that in consideration when it is being calculated. A definition of relevance is needed where relevance can be seen as the independence of the feature from the input and the dependence with the outcome of the model. Ranking methods are a considered filtering method because they are applied before the model is trained. Two of the most common ranking methods are:

- ✍ Correlation Criteria (CR): the Pearson correlation criteria for a classification problem where x_i is a feature and Y is the target:

$$R(i) = \frac{cov(x_i, Y)}{\sqrt{var(x_i) * var(Y)}}$$

  This type of correlation can only detect linear dependencies between the feature and the target of the problem.

- ✍ Mutual information (MI): measure the dependency between two variables through the use of entropy.

  Shannon definition of entropy is:

$$H(Y) = -\sum_y p(y)log(p(y))$$

  and as the definition of entropy goes we have the uncertainty of variable Y in this case. What is needed though is the dependency between Y and the features, so the definition of conditional entropy is given as:

$$H(Y|X) = -\sum_x \sum_y p(x, y)log(p(y|x))$$

  where x is a feature. This conditional entropy gives us the entropy of Y by observing X and it tells us that by observing X the entropy of Y decreases. We obtain the decrease of uncertainty as:

$$I(Y, X) = H(Y) - H(Y|X)$$

  This gives us MI and the interpretation of the results is that if MI is equal to zero, then X and Y are independent and if greater than zero then X and Y are dependent. The calculation of MI can vary depending on the type of data but the idea that remains is that of calculating the dependency of each feature with the output and then selecting a subset of features through the use of a threshold $d < D$. This approach however does not take into consideration the inter-dependencies between the features themselves.

Filtering methods are simple, computationally cheap to apply and avoid overfitting by not depending on a learning model's accuracy for choosing the subset of features to keep but also have some drawbacks. Redundant subset can be chosen given that inter-feature dependency is not considered.

Wrapper methods

Wrapper methods instead tackle the problem in a different way. This kind of approach chooses a different subset of features at a time and relies on the performance of the learning model in selecting the best one. The model itself is used as a black box and the performance of such model is used as the objective function to optimize in testing subsets of features. To try out all the possible subsets of the feature space would lead to a NP problem and it would be inefficient considering the time it would require, especially in datasets with a large number of features. Therefore, selecting the subset of features to try out needs to be done more carefully and heuristically. There are two main approaches in doing so:

- ✍ **Sequential selection algorithms**: these types of algorithms start with an empty (full) set of features and then incrementally add (remove) features until the maximum of the objective function is obtained, so the highest accuracy of the learning model is obtained. An example of this type is Sequential Feature Selection (SFS). Sequential Backward Selection (SBS) instead applies the same approach but on reverse, starting with the entire set of features and then removing one at a time. Both these methods suffer from the lack of information about the inter-feature dependencies. There have been implemented improved version of such as Sequential Floating Forward Selection (SFFS) and Adaptive Sequential Forward Floating Selection (ASFFS).

- ✍ **Heuristic search algorithms**: these types of algorithms instead evaluate different subsets of features and then keep the one that gives the best accuracy of the learning model. The subsets to evaluate are chosen from a search space or generated through particular techniques. Genetic Algorithms (GA) are one of the most frequent methods to build the subset of features, where whether a feature is included or not is expressed in the chromosome bits representation. The global maximum of the objective function, where the objective function is the performance of the predictor, leads us to the best suboptimal subset we were looking for.

Wrapper methods are more effective than Filter methods, but they have their drawbacks. One of the main drawbacks concerns the computation time that it requires to find the best subset. For the wrapper methods for each subset a model is trained and tested to evaluate the performance of the subset. Another drawback is that of overfitting: the choice is based on the performance of the model and that can lead to overfitting which in other words means poor generalization and learning the training data too well without the ability to do well in the test set instead.

Embedded methods

Embedded methods are the third type of feature selection. This approach tires to overpass the drawbacks of the two previous methods. It's a combination of the first two: aiming to decrease the computation time of testing out feature subsets in wrapper methods, in an embedded method the feature selection process is incorporated in model training. The MI concept introduced in filter methods is very important but suffers from the lack of considerations of the feature redundancy given that it does not keep track of the correlation between the feature in the chosen subset. Embedded methods like [10] and [19] try to make up for that by aiming to maximize the correlation of the feature with the output and minimize the correlation among the selected features, thus reduce redundancy. Another interesting method in doing feature selection is that of using a classifier's feature weights to apply feature selction. [11] uses a Support Vector Machine (SVM) to do classification for a cancer classification problem. It uses the weights of the SVM for the feature as ranking criterion applying then Recursive Feature Elimination (RFE). Through RFE one or more features at a time are removed where the features to remove are those with the lowest ranking criterion. This backward feature elimination is done in an iterative fashion until the performance doesn't improve anymore.

2.5.2 Feature selection in Recommender Systems

Feature selection is of importance to the recommender systems applications as well. Each item can be described through its features usually called item profile. Such item profiles give us information about the item content and such information can be used to recommend items to users based on the content preferences of the users. However, the amount of information, that is the amount of features can be quite high due to the large number of data but also due to the diversity of such features. In the item profile features can vary from numerical attribute to labels or labels. Attributes can be qualitative (nominal, ordinal and binary) and quantitative (discrete and continues).

In addition, in the recent years social tagging is becoming more important as we go, where by social tagging is understood to be any text assigned by a user to an item, such as a tag or a brief description of the item. It is however hard and expensive to provide an exhaustive description of the items and it is even harder to provide an accurate one. The information coming from different sources and more particularly from different users can be noisy and quite complex. Thus the available features usually lack good quality which makes the information provided by them less worthy without first applying pre-processing.

In selection a subset of features we can go different ways about it, but that can be grouped in two main approaches:

- ✍ Collaborative filtering with side information
- ✍ Feature selection with collaborative filtering information

Collaborative filtering with side information

Given that feature selection is a pre-processing step usually done in machine learning and data mining it is very common to apply the same methods also to a recommender system. Such approaches can be algorithms like Genetic Algorithms(GA): [17] use a GA to apply feature selection and choose parameters of the Support Vector Machine (SVM) being used for the final prediction; they aim to optimize both at the same time and succeed in doing so by using a genetic algorithm where the fitness function is the performance of the SVM given the feature subset and the parameters of the SVM.

Another algorithm used for doing feature selection is Ant colony optimization (ACO) as used in [12], where the recommendation problem is seen as a classification task for each user and an Artificial neural network is used to make the prediction after the feature selection is done. In this work, the application of ACO is accompanied by the use of fuzzy c-means clustering: to apply ACO first they represent the feature selection problem as a graph problem and then find the weights(pheromone values) between the nodes(ants), weights that are found through the use of fuzzy c-means clustering, which finds the values of membership function of different features with respect to ratings provided by user to items. The application of ACO on the graph tells us which nodes are important and which are not thus making it possible to do feature selection.

Other standard approaches have been applied in doing feature selection, in [12] a forward greedy feature selection approach is adapted, with the addition of keeping in consideration the correlation between features themselves, something that the

greedy algorithm does not take into account. Pearson correlation is used for finding such correlation between the features and then the overall selection on the features is done by removing those irrelevant features as well as the redundant ones.

Another approach used to select the most relevant features is that of Chi-square selection, as done in [23]. In this work, an application to a specific problem of computer science publications recommendation feature selection is done using the Chi-square selection method, where features in this case are words in a document. After calculating the Chi-square value for each feature, top N features with the highest values are kept for the rest of the prediction.

[22] is yet another work concerning automated feature selection. They propose an approach were a score is calculated for each feature and then the features with the highest feature value are considered as the most relevant. The distinction between label and attributes is made: a label or a tag is an n-gram or a description associated to an item by either users, experts or automatic text-mining algorithms and an attribute is metadata features of the items like price or is-on-sale.

There is also a distinction in how the weight of the features is calculated but the same concept is applied for both type of features considering the ratio of two variable b1 and b2. The core idea is that of finding feature similarity, consider f1 and f2, by using a collaborative filtering algorithm based on users and their interaction with items containing f1 and f2 or a content based algorithm. b1 is proportional to the similarity of the feature under consideration w.r.t. relevant items according to the recommendation algorithm used. b2 is a normalizer which is proportional to the similarity of the feature under consideration w.r.t. random items. The ratio b1/b2 is considered as the normalized relevance of the feature with respect to the recommeneded items. The evaluation of the goodness of the approach is done through the use of Matrix Factorization in the case of cold-start items. In case of the cold-start items, item similarity is used using only the most relevant features, where the most relevant features are obtained from the scoring algorithm proposed by the work.

An interesting work of feature selection in a recommender system relevant for our work is that done in [3]. The similar aspects with such work are that a tree-based machine learning classification algorithm is used to distinguish what are positive and what are negative samples provided to it and make use of such algorithms to understand the importance of features in doing the classification. It is however different in the features it is analyzing (user based features like profile type or profile size and model-based features like personalized recommendations, context aware recommendations etc.), what is being classified (the samples provided to the classification algorithm) as well as the approach (Collaborative filtering) that is aimed to optimize.

The application of all these feature selection approaches however is evaluated and confronted with Collaborative filtering approaches that use side information.

Feature selection with Collaborative filtering information

In this second category of feature selection works in recommender systems we are considering approaches that use collaborative information, being that a collaborative item similarity or directly from user ratings, in order to do feature selection and then evaluate the quality of the approach in a Content-based recommender system.

In terms of the information used to obtain feature relevance and the model such feature relevance is applied to there exist two similar works: [8] and [5].

[8] introduces a wrapper approach in obtaining feature relevance through the use of collaborative information. The feature relevance is learnt in two steps, first by tuning an item-based collaborative filtering recommender and then learning the feature weight that best approximates the item-item collaborative similarity found in the previous step.

The higher performance of the collaborative filtering approaches leads to make use of such collaborative information for a better feature selection. The problem is reduced to minimization problem with objective function defined as:

$$argmin_w = \|S^{\mathbf{CF}} - S^{\mathbf{W}}\|_F^2 + \lambda|\mathbf{D}|_F^2 + \beta|\mathbf{V}|_F^2$$

where $S^{\mathbf{CF}}$ is the item-item similarity, $S^{\mathbf{W}}$ is defined as:

$$sim(i,j) = f_i^T \mathbf{W} f_j$$

$\mathbf{W}$ is a diagonal matrix that tells us how much a feature of item i interacts well with a feature of item j. It is defined as:

$$\mathbf{W} = \mathbf{D} + \mathbf{V}^T\mathbf{V} \tag{2.2}$$

where D is a diagonal matrix having as dimension the number of features, n_F , and $\mathbf{V} \in \Re^{n_F \times n_L}$. The number of latent factors n_L is treated as a parameter.

[5] proposes an approach to overcome the gap between content-based approaches and item-based collaborative filtering approaches through feature weighting. The features that it learns are such that the difference between the item similarity found by the content-based is much like the similarity found by the item-based collaborative filtering. Considering F the set of all features, the weighted similarity is defined as:

$$s_{ij}^{(w)} = \sum_{f \in F} w_f a_i f \dot{a}_j f =< w, \mathbf{a}_i \mathbf{a}_j > \tag{2.3}$$

The problem is then reduced to a minimization problem:

$$argmin_w = \|S^{\mathbf{CF}} - S^{\mathbf{W}}\|_F^2$$

where $S^{\mathbf{CF}}$ is the item-item similarity and $S^{\mathbf{W}}$ is the weights similarity, as defined in equation 2.3. It is a solution based on least squares and thus it is called Least Square Feature Weighting.

However, both of these approaches use the derived feature relevance to do feature weighting instead of feature selection.

2.6 eXtreme Gradient Boosting

xGBoost is a machine learning model used in classification, regression and ranking problems which in the course of the years have shown to be one of the most effective and fastest models to be used in various problems, seen as the to go to solution for many Kaggle competitions.

xGBoost, which stands for eXtreme Gradient Boosting, is a model built as an ensemble (Boosting) of weak learners (Decision Trees) which are built sequentially,

aiming to improve the learner using the results of the previous learner (through Gradient Descent). As any supervised learning algorithm in machine learning the steps taken to build the model are fundamentally two: define a loss function and aim to minimize it. As defined in [xGBoost: A scalable Tree Boosting System] the regularized objective function to minimize is:

$$L(\phi) = \sum_i l(\hat{y}_i, y_i) - \sum_k \Omega(f_k) \tag{2.4}$$

where $\Omega(f)$ is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2\lambda|\omega|^2} \tag{2.5}$$

where $\hat{y}_i$ is the predicted value of the model and y_i is the real value and l is the differentiable convex loss function. The term $\Omega(f)$ penalizes the complexity of the model (the regularization term added by this implementation of xGBosot). The reason for using regularization is to avoid overfitting, an issue that is present in any machine learning method. Overfitting happens when the model fits perfectly or almost perfectly the training data but fails to handle previously unseen data of the test set. Underfitting instead is the opposite, the model does not learn enough from the training data, not using the available data at their best potential and thus does not generalize well on unseen data.

The special aspect of this implementation is that the model is trained in an additive way, improving iteration after iteration.

y_t^i is the prediction of sample i at the *t-th* iteration. We use the prediction of the previous iteration for the sample and add the f_t where f_t corresponds to a regression tree function. This approach uses Regression trees instead of Decision Trees which contains a continuous score in each of the leaves and then the prediction for a sample is made by summing up the scores of the corresponding leaves when classifying the sample down to the leaves. In optimizing the second the objective function, Chen and Guestrin use a second-order approximation:

$$L_(t) \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.6}$$

where

$$g_i = \vartheta_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{2.7}$$

and

$$h_i = \vartheta^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{2.8}$$

being 2.7 the fist gradient and 2.8 the second gradient of the loss function.

The simplified version of 2.6 is:

$$\tilde{L}^{(t)} = \sum_{i=1}^{n} [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \tag{2.9}$$

Defining $I_j = \{i|q(x_i) = j\}$ as the instance set of lead j we can rewrite equation

2.9 by expanding Ω as follows:

$$\begin{aligned}\tilde{L}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T\end{aligned} \tag{2.10}$$

For a fixed structure q(x) we can compute the optimal weight w_j^* of leaf j by

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda} \tag{2.11}$$

and calculate the corresponding optimal value by

$$\tilde{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{\sum_{i\in I_j} {g_i}^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{2.12}$$

Equation 2.12 can be seen as the measure of goodness of a tree structure q, which differs from the impurity of a decision for the fact that it is derived from a broader variety of objective functions.

A particular feature of the implementation that is of particular interest to our work is how it handles sparse data, being it a *sparsity aware* algorithm. Input data for a model can be sparse due to various reasons like missing values or feature engineering process like one-hot encoding and the model takes this into consideration automatically, unlike other implementations or other approaches which either optimize only sparse data or need extra effort to handle sparse data. In case of missing values, the model does so that the instance goes in a default direction of the tree, which is learnt from the data. The main improvement is the fact that the algorithm only considers the non-missing values and for the missing values then learns the best direction in the tree. Another particular feature of the xGBoost model that is of use for our work is that it also reports the importance of the features of the dataset once the model is trained. There are five types of importance:

- ✍ Gain: a value that is the average training loss reduction gained when using the feature for splitting
- ✍ Weight: a value calculated for each feature as the number of times the feature is used to split the data across all trees
- ✍ Cover: a value for each feature calculated as the number of times the feature is used to split the data across all trees weighted by the number of data that go through these splits.
- ✍ Total gain: the total gain across all splits the feature is used in
- ✍ Total cover: the total coverage across all splits the feature is used in

2.7 Bayesian Optimization

Hyper-parameter tuning is one of the most crucial steps of any learning process. The distinction between hyper-parameter and parameter in a model is that hyper-parameters are those kinds of parameters the user has to set while model parameters are learned by the model itself through the data processing. The fact that we have to choose the values to give to the hyper-parameters raises the problem of how to choose them and how to find the best combination in case of more than one parameter to set.

The notion of hyper-parameter search is that of finding the point in the parameter space that best generalizes the dataset for the purpose it is being used by yielding the highest performance of the model on unseen data. To find the right point in such space a naïve approach would be that of trying out all point of the space but that can be very unfeasible due to the large number of parameters there are in a model. This is a very tedious task to be done manually so automatic approaches have been developed over the years. These approaches can be categorized in three main categories: Simple search approaches like *Grid search* and *Random search* and Optimization from samples, which itself is divided in Heuristic search, Model-based derivative free optimization and Reinforcement learning [21]. One approach that has attracted the most attention for its efficiency for various problems and sound theoretical justification is *Bayesian Optimization*. It's an Optimization from samples, under the Model-based derivative free optimization category. Such approach has become a standard in literature as well in various competitions, e.g as used by Creamy Fireflies in the ACM Recommender System Challenge 2018 (RecSys 2018) [2].

In the Bayesian optimization approach a *probabilistic model* is built with the scope of obtaining the performance of the parameter configuration. Such model can be for e.g. a Gaussian process, a tree-based model or deep network. An *acquisition function* is then defined based on the probabilistic model, e.g. expected improvement, upper confidence bounds, to balance exploration and exploitation during search. At each iteration a new sample is generated, optimizing the acquisition function and its evaluation is then used to update the probabilistic model.

For the conduction of the experiments in this work the skopt library was used. This implementation of the Bayesian optimization supports only Particle Swarm Optimization algorithm [13].

Problem definition

Hyper-parameter tuning is an optimization problem, where an objective function is needed and under pre-defined constraints the best solution is to be found with respect to the objective function. This can be expressed through the following formulation:

$$x^* = argmin_x f(x) \tag{2.13}$$

Some truths need to hold:

- f is a black-box function with unknown closed-form. This function needs to be defined by the user and in the case of hyper-parameter search it is the performance of the model given the set of parameters x

- ✍ f is quite expensive function to evaluate. In terms of models, the evaluation function requires the time to train the model we are using and then evaluate the performance of such model given the parameter set provided (x)
- ✍ Evaluations of $f(x)$ may be noisy

x denotes the parameters that we give to such function and that need to respect the pre-defined constraints. The general idea of this approach is that of starting out with some random points in the parameter space and from there move in informed fashion in the search of the best point in the parameter space. With informed fashion it's intended the fact that it uses information from the evaluation of the previous step in order choose the next parameters to test out.

Search process

For $t = 1 : T$ we have:

- ✍ Given observations $(x_i, y_i = f(x_i))$ and $i = 1 : t$, build a probabilistic model for the objective f. Integrate out all possible true functions, using Gaussian process regression.
- ✍ Optimize a cheap acquisition/utility function u based on the posterior distribution for sampling the next point

  $$x_{t+1} = argmin_x u(x)$$

  exploit uncertainty to balance exploration against exploitation.
- ✍ Sample the next observation y_{t+1} at x_{t+1}

Acquisition function

Acquisition function $u(x)$ specify which sample x should be tried out:

- ✍ Expected improvement : $-EI(x) = -E[f(x) - f(x_t^+)]$
- ✍ Lower confidence bound: $LCB(x) = \mu_{GP}(x) + \kappa\sigma_{GP}(x)$
- ✍ Probability of improvement: $-PI(x) = -P(f(x) \geq f(x_t^+) + \kappa)$

After setting up the environment when the Bayesian optimization has finished executing we have the best hyper-parameters ready to be used on the test set to get the final evaluation of the model being used.

Chapter 3

Model Formalization

Feature selection is a pre-processing step that has been used for long now in the field of information filtering. The available features can be of different types, from item attributes like the author of a book, publishing year or book price to social tags added by the users to even brief descriptions. However, the information provided by the features can be redundant and prone to noise, especially in the case of user provided social tags.

It is however not that common to use machine learning approaches to select the features due to computation difficulties, time and space wise. Machine learning algorithms are known to be complicated in terms of hyper-parameter tuning considering the high number of hyper-parameters present in the model. Such hyper-parameter tuning process is crucial for the learning process given that the wrong set of hyper-parameters can lead to poor performance of the model. Furthermore, the existing algorithms to choose from are many which means that considerations and experiments need to be conducted in order to choose one. This can be costly considering the time it requires to train such models. On the other hand, it is a common knowledge that such algorithms yield very good results and thus there is an ongoing research in using them for such application.

The aim of our work is that of applying a machine learning algorithm for the feature selection pre-processing step to then use the reduced feature set to make content-based recommendations. For doing feature selection collaborative information is used, extracting feature related feedback from each active user. The chapter is structured as follows. In Section 3.1 the process pipeline is introduced followed then by the description of each pipeline step in Data Pre-processing in 3.2, Classification model training in 3.3, Feature selection in 3.4 and the Recommender system in 3.5. The choices made and the motivation for such choices are described in full detail.

3.1 Pipeline

The objective of our work is that of making recommendations through a content-based approach using only the most relevant item features from all the available features present in the data. As in any other field the process of feature selection requires the ordering of features with respect to some criteria. To obtain such ordering, in our work we use URM provided information to obtain a feature importance value to then select from so our approach uses collaborative filtering information for

the feature selection process.

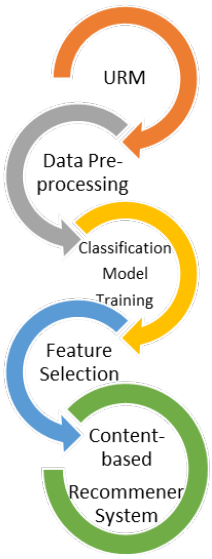

Figure 3.1: The process pipeline

As shown in Figure 3.1, the whole process has a start with the URM from which user feedback is extracted.

A feedback is a user's reaction towards an item and given that such reaction can be positive or negative, a feedback can be of two types: negative and positive.

Utilizing user's profile we can obtain the set of positive feedback and that negative feedback from which in turn we can analyze to understand what the items with positive feedback have in common and what items with negative feedback have in common. This leads us to the generation of what will become the samples for the classification model: from the positive feedback we extract positive samples and the negative feedback we extract negative samples. Once we have the samples we can then apply a classification algorithm with the aim of understanding from these samples which are the key features that help distinguish the two types from one another. For our purpose we need a classification algorithm able to give us an importance for each feature of the input variables and thus making it possible to order them. Once the features are ordered we can decide what to keep and what to discard. After the selection is done, we can use the subset of the selected features to make content-based recommendations.

The entire process of our feature selection can be summed up in four phases:

- ✍ **Data pre-processing**
- ✍ **Classification model training**
- ✍ **Feature selection**
- ✍ **Recommender system**

In a true pipeline fashion the output of one step is used as the input of the next step and it is all concluded with the recommendations being made to the user.

3.2 Data pre-processing

The first step of the pipeline is that of data pre-processing where both URM and ICM are used to elicit the information needed to do feature selection. It step can be summed up in two sub-steps: Feature extraction and sample generation. These two stages of data elaboration set the foundation for what is going to be our feature selection approach.

Figure 3.2: Data pre-processing steps

As we can see in Figure 3.2 the two steps are very specific in their source of information: for feedback extraction we make use only of the URM, so collaborative information, while for the sample generation we use the feedback extracted in the previous step as well as the ICM to produce what will then be the input data for the following step of the pipeline, that is the training of the classification model.

3.2.1 Feedback extraction

For this first step of data pre-processing we make use only of the URM given that we need to extract the negative and positive feedback per user. By extracting such feedback we are extracting information about what the user has liked or shown interest on and what the user dislikes. The definition of what is positive and negative for a user depends on the type of data available in the dataset.

Explicit dataset

For an explicit dataset, where a user gives numerical values to each item, e.g. from the range of [1,5] there are different ways to choose what is a positive feedback and what is a negative feedback:

- use user's average rating as a threshold: all those with a rating above a threshold are considered as positive
- considering positive all the rated items and as negative those not rated
- using a threshold, for example the rating 3 in a system where ratings go from 1 to 5

Trying out the three different approaches we choose the first one due to the better performance over the other two. This is a user-based definition of what is liked and not liked and makes for personalized feedback for each user.

Consider a dataset of user ratings of films and we have the ratings of user Michael.

	Roma	Emma	Estiu 1993	Youth	Drive	A star is born
Michael	5	0	4	1	1	3

Table 3.1: URM user example

The average rating given by such user is 2.8 therefore the positive feedback for Michael are *Roma*, *Estiu 1993* and *A Star is Born* while *Youth* and *Drive* are considered as negative feedback.

In the cases of very active users, so users that have given many ratings, only a portion of the overall ratings are used as feedback. In choosing which items to keep in this case the items rated the highest are added to the positive feedback set while the movies rated the lowest are added to the negative feedback set. Consider the example of user Michael in table 3.1, if were were to keep only 1 positive item and only 1 negative item in this case *Roma* would be considered as positive feedback and one movie chosen at random between *Youth* and *Drive* would be the negative feedback. With the aim of having a balanced feedback set we keep the same number of positive feedback as the number of negative feedback.

To make the distinction of positive and negative as robust as possible an Ł normalization of the URM column wise, so per item, was done. Consider vector $\mathbf{x}$ of the form:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

We can define l2 norm of the vector as:

$$|x| = \sqrt{\sum_{k=1}^{n} |x_k| l^2}$$

3.2.2 Sample generation

In a true pipeline fashion the results of the previous step, so the positive and negative feedback, are used as the input of this next step: *the sample generation*. As described in Section 3.1, the goal of our work is that of doing feature selection obtaining a feature importance from the use of a classification model. To use a two-class classification model first we need the samples to train and test such model on, so a dataset with positive and negative samples. This dataset is generated by our initial recommender dataset, so URM and ICM.

To generate the needed samples we use the feedback extracted from each user. For each user we have a set of positive items, so the positive feedback, and a set of

negative items, the negative feedback. This however, can be put down in terms of item characteristics (features) if we add information from the ICM as well. Knowing that user Michael liked the movie *Roma* means that we know that Jennis likes *dramatic*, *artistic*, *black and white* and *oscar contendors*, which are the features of the movie *Roma* present in the ICM.

In aiming to have a more general understanding of the user preferences towards the item features we considering what each couple of items with positive (negative) feedback have in common.

	Artistic	Dramatic	Oscars	B&W	Musical	Biography
Roma	1	1	1	1	0	0

Table 3.2: Item 1 feature vector space representation

	Artistic	Dramatic	Oscars	B&W	Musical	Biography
A star is born	1	1	1	0	1	1

Table 3.3: Item 2 feature vector space representation

Following the example in 3.1 we have the *Roma* and *A star is born* both have a positive feedback. It is interesting to know what these items have in common for the user to have liked them. The approach proposed by us is that of doing an element-wise multiplication of the vector space representation of such items to obtain 1 in all the features in common and 0 otherwise. In the case of the example we would have:

	Artistic	Dramatic	Oscars	B&W	Musical	Biography
Positive sample	1	1	1	0	0	0

Table 3.4: Example of a sample

The newly created sample can be seen as an incorporation of information from two different sources, a stronger definition on what is considered positive for a user and what is considered negative. For each user, we can now generate all the possible combinations between the positive feedback as well as all the possible combinations between the negative feedback producing so positive and negative samples respectively.

This kind of sample generation leads to the generation of millions of samples for the whole community of the users and thus setting the foundation for a sound classification problem.

There are however some issues that depend on the available dataset and not on the approach being used:

- ICM sparsity: It is very common that the ICM matrix is quite sparse with items that have no features at all or items with very few features.

- ✍ Noise and redundancy: features assigned to items, especially in the case of social tagging are prone to noise and redundancy and using such features as they are can lead to a lower score of the recommendations.

3.3 Classification model training

The core of a two-class classification problem is that of learning from a set of training data to distinguish between a positive and negative sample.

The possible choices of classification algorithms that come from machine learning and data mining techniques are quite vast, Decision Trees, Random Forest, Support Vector Machines, Boosted Trees just to name a few. However, not all of these algorithms are useful for our goal. The purpose of the classification model for our work is that of providing an ordering of item features by learning to classify the positive and negative samples we present to it. Understanding the effect of the features over the classification of a positive or negative sample gives us the importance of such feature when making the choice. This narrows down the type of algorithm to use for the classification step of our problem: we need a model that other than doing the classification it returns also the feature importance which is based on the internal calculations done to do classification.

Random Forest versus xGBoost

Among all the possible algorithms the tree based ones are the most suited for our work and this narrows down our choices to two algorithms used the most in literature: *Random Forest* and *xGboost*. Both of these approaches are based on decision trees and both of them provide feature importance for the features of the input. In choosing between the two experiments were carried out before picking xGBoost algorithm for our work.

Strengths

- ✍ **Quality**: from the results obtained from using an xGBoost model instead of a Random Forest one for the classification step of our work it resulted that the feature importance provided by xGBoost lead to a higher score of our final recommender system

- ✍ **Computation time and resources**: one remarkable distinction notices between the two models was in terms of the training time of the model as well as the resources it required. In the case of Random Forest, given the high dimensionality of the input samples (in the case of The movies dataset we have about 50 000 features) the computational power needed to do it all at once exceeded the available 12 GB of RAM available on the machine where the experiments were run. This lead to batch learning where the training data was divided into batches and the Random forest was incrementally trained one batch at at a time. However, such learning process increases the computational time, in the case of The Movies dataset up to hours. Such problem was not met in the case of the XGBoost model. The training time of the model does not exceed one hour even for the biggest dataset, the case of The Movies dataset with both types of features.

✍ **Model implementation**: the implementation of xGBoost offers some very useful features that can be used when using such model. One of them is the Early Stopping technique [20] from which we can benefit when doing hyper-parameter tuning, as it is explained in Section **??**. Another useful feature for our work in the model implementation concerns the handling of sparse input. xGBoost is optimized in terms of sparse inputs, as it is in our case, and handles only the non zero cases present.

Weaknesses

✍ **Model sensitivity**: one drawback of the xGBoost models is that are prone to overfitting if the hyper-parameters provided to it are not chosen carefully. Thus it requires a wary tuning process.

In a pre-processing step like feature selection the trade-off between the benefit of such step and the required time to it is always evaluated. xGBoost proved to be the better choice and it was chosen as the algorithm to use for the feature importance calculation.

With xGBoost, once the model is trained, it is relatively straightforward to retrieve the feature importance. With feature importance in the case of such algorithms is meant the impact of a feature on the generation of the boosted trees. The more a feature of the input samples is used in decision making when creating the trees, the more important it is. The value is calculated for each feature which makes it possible for a ranking of the feature or even further a selection.

3.3.1 Hyper-parameter tuning

The xGBoost algorithm is notorious for the large number of parameters that need close care so to avoid *overfitting* or *underfitting*.

To achieve the highest score as well as reliable results the available data is divided in three subsets: Train set, Validation set and Test set. The division of the dataset in such subsets is described in Section 4.2. The parameters to tune in an xGBoost model:

✍ **Tree depth** $\in$ [3,50]. The basis of xGBoost is that of learning from weak learners, so very simple trees, with depths starting from 3 and on average go up to 10. There is not however an upper bound on the depth that can be provided. Upon the consideration that the samples provided to the xGboost for our problem are of very high dimensionality, in the case of a dataset used for the experiments there are about fifty thousand features, different values up to depth 50 were tried out.

✍ **Number of trees** $\in$ [100,3000]. The lower bound by the implementation is 100 and once again a higher bound is not defined by the implementation itself. The xGBoost however has implemented the *Early Stopping* technique. Such technique does so that while the model is learning it tests out the quality of the model in the validation set and it continues to learn as long as improvements are seen in the quality of the model. If after a number of iterations, so after a number of trained trees are trained, the quality does not improve anymore then it stops generating new trees. This is done as to avoid overfitting of

the training set which would lead to poor generalization abilities. Therefore we set a very high number of trees, about 3000 trees for all the experiments conducted, and the algorithms continues to generate trees for as long as the quality on the validation set improves. If the quality does not improve in 15 iterations then the algorithm stops and it tells us that it is done training. To be able to use such model feature we need to provide to the model the Train and the Validation set. The train set is used to learn while the validation set is used to evaluate the goodness of the model while it learns and tune the hyper-parameters.

- **Learning rate** $\in$ [0.01,1]. In the search of the optimum of the model performance steps are taken and we can dictate how big of a step the model takes in each iteration. A low learning rate lowers the risk of stopping at a local optima but it can take much longer to learn while a higher learning rate can miss the optimum solution by jumping too much. The bounds of such hyper-parameter are predefined and choosing the best one is a task of the Bayesian optimization approach.

- **Regularization term alpha** $\in$ [0,1]. L1 regularization term on weight that helps reducing the execution time of the algorithm.

- **Regularization term lambda** $\in$ [0,1]. L2 regularization term on weight, used to avoid overfitting.

- **Column subsample tree** $\in$ [0.01,1]. Denotes the fraction of columns to be randomly sampled for each tree.

Evaluation procedure

The aim of our work is that of making recommendations through a classical content-based approach while using only the most relevant subset of item features. The contribution of the xGBoost model is that of providing the feature importance which is then used to do feature selection.

Therefore the better the performance of the xGBoost model is the better is the calculation of the feature importance and the better is also the performance of the recommender. The classical hyper-parameter tuning scheme is that of evaluating the quality of the model in a validation set for different hyper-parameters and then keeping the hyper-parameters that yield the highest value of the chosen metric you are aiming to optimize. For our work however, we go about this in a different way: instead of optimizing the performance of the xGBoost model itself on some metric, given the hyper-parameters of the xGBoost model, we optimize the performance of the ItemCBFKNN using only the selected features whose selection is based on the xGBoost model trained on the provided hyper-parameters.

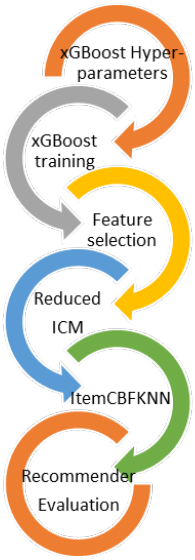

Figure 3.3: xGBoost hyper-parameter tuning

The hyper-parameter tuning of the xGBoost model can then be reduced to the steps seen in the Figure 3.3. For each set of hyper-parameter values we go to optimize the performance of the ItemCBFKNN with the reduced ICM containing only the features selected.

The metric we use to optimize for the recommender system is *MAP*, described in 2.4.3. The reason we choose to optimize such metric is that other than being considered the most important metric in recommender system literature it captures the importance of ranking as well as precision of the recommendations being made.

For the hyper-parameter tuning process we have used the Bayesian Optimization approach, as described in Section 2.7. We provide the search space for each hyper-parameter and function that evaluates the points chosen by the Bayesian optimization at each step. When the Bayesian Optimization is done we have the best hyper-parameters for our xGBoost model. During this iterative process we have to consider that the ItemCBFKNN Recommender itself also has hyper-parameters that need tuning. For such hyper-parameters we decided to use the best hyper-parameters found when optimizing the ItemCBFKNN recommender with the original ICM, so the ICM containing all features. How such hyper-parameters are obtained is explained in Section 3.5.1.

3.4 Feature selection

Once the training and optimization of the xGBoost model is done we have an importance for each feature available in the dataset. What we do now is a simple distinction between what features to keep and what to discard. The features' importance provided by the xGBoost showed that a great deal of the features have no impact in the decision making process of tree generation and thus an importance equal to 0 with only a small portion of the features with importance different from zero. The different approaches in choosing the features to keep were tried out:

- ✍ **All**: choose all features with an importance different from zero

- ✍ **Percentage**: choose only a percentage from the features with non zero importance, becoming thus the percentage a parameter to tune

- ✍ **Threshold**: choose the features with an importance over a threshold. The threshold becomes thus a parameter to tune

3.5 Recommender system

The recommender chosen to make the recommendations is ItemCBFKNN, summarized in Section 2.2.1. For the rest of the work baseline denotes the ItemCBFKNN recommender when the original ICM with all the available features is used.

3.5.1 Hyper-parameter tuning of ItemCBFKNN

The recommender itself has hyper-parameters that need to be tuned to achieve the best performance of the model with the data at hand. For both cases, the baseline and the reduced ICM with the selected features(FSItemCBFKNN), the same hyper-parameter procedure is applied. Once again we use the Bayesian Optimization approach for tuning the hyper-parameters which requires the definition of the search space for the parameters and also the function to be optimized.

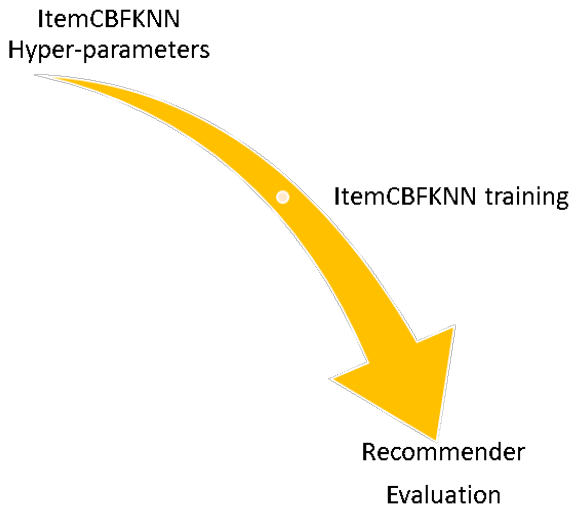

Figure 3.4: ItemCBFKNN hyper-parameter tuning

As shown in Figure 3.4 the tuning process consists of three different steps where the difference between the baseline and the ItemCBFKNN recommender we do is the ICM being used. What we aim to optimize in both cases is the MAP of the recommendations being made, which for the tuning process is the performance on the validation set.

The parameters to tune in the case of a ItemCBFKNN algorithms are:

- ✍ **Number of Neighbours** $\in$ [5,800]. The ItemCBFKNN model has as the main characteristic the number of neighbours we consider for each item.

- ✍ **Shrinkage factor** ∈ [0,1000]. The shrink factor is applied to the similarity aiming to avoid overfitting.
- ✍ **Type of similarity** ∈ Cosine, Pearson, Tanimoto, Tversky, Jaccard, Dice. To find the nearest neighbours we define a similarity between the items. Such similarity can be calculated in different ways therefore we ought to choose between the different types.
- ✍ **Similarity coefficients** alpha and beta ∈ [0,2]. In the case of some similarities these factors are part of the similarity formula used.

Chapter 4

Datasets

To evaluate the effectiveness of the feature selection technique proposed by us we used two different datasets: MovieLens 10 million and The movies dataset. The two essential sources of data needed to develop our work are the URM and the ICM, which both of these datasets provide. Furthermore, both datasets have a rich content of features which makes for a good setting for the evaluation of our work in doing feature selection.

In this chapter we do a thorough dataset analysis, section 4.1 followed by the dataset split, section 4.2, which is yet another crucial aspect in setting up a recommender system.

4.1 Dataset analysis

In this section the analysis of the URM as well as the ICM is done, for both datasets used in conducting the experiments.

4.1.1 MovieLens 10M

MovieLens 10M is a movie database released by the GroupLens mainly used for research purpose. It is a collection of user ratings for movies and information about the movies in the form of tags. As the name suggests the dataset contains about 10 million ratings, given by 72 000 users.

URM

The URM is a matrix with explicit ratings.

$$r_u^i \in \{0.5, 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5.\}$$

Each user can express his dislike, $r_u^i = 0.5$ or love $r_u^i = 5.0$ towards the movies.

MovieLens 10M URM	
Users	71586
Items	65134
Interaction	10 M
Sparsity	0.9978

Table 4.1: Movielens 10M URM statistics

For this dataset we have over 70 thousands users that interact with over 65 thousand movies. The number of interactions is over 10 million however the sparsity of the matrix is very high.

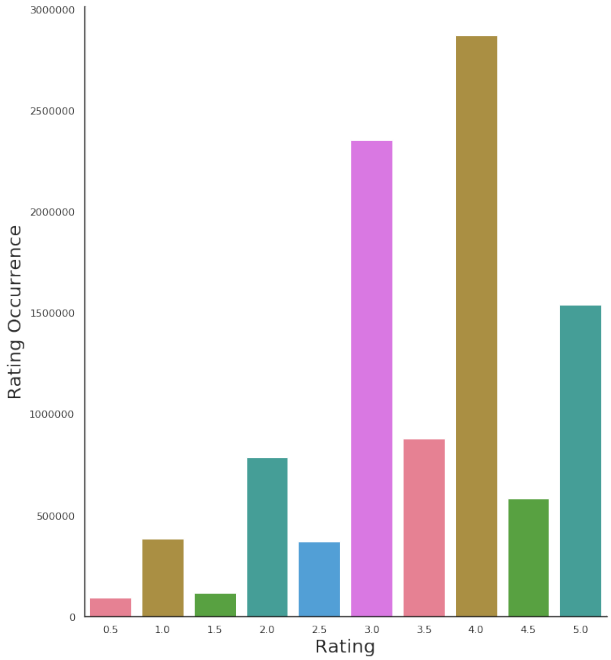

Figure 4.1: MovieLens 10M rating distribution

In Figure 4.1 we see the number of times each rating value was given by the users. The leading one is rating = 4 followed by rating = 3 with very few low ratings given to the users. We have an average rating of 3.5 which means that the users were rather positive towards the movies present in the database but not to overjoyed about them neither.

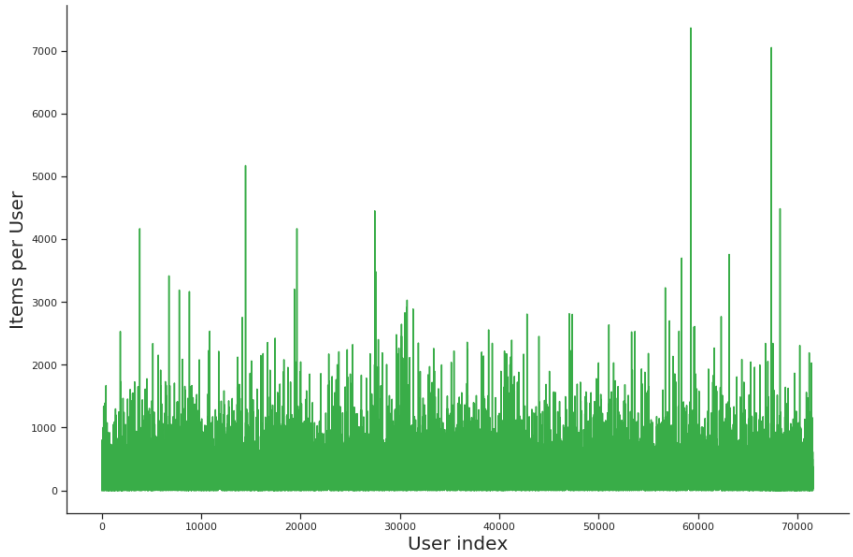

Figure 4.2: MovieLens 10M items rated by users

In Figure 4.2 we see the number of items rated by each user where few outliers are present, so very active users that have rated over 4000 movies. On average, a user has given about 140 ratings, which is a relatively high number of ratings to give. This however, is a good thing for a recommender system since it has a lot of information about the user preferences.

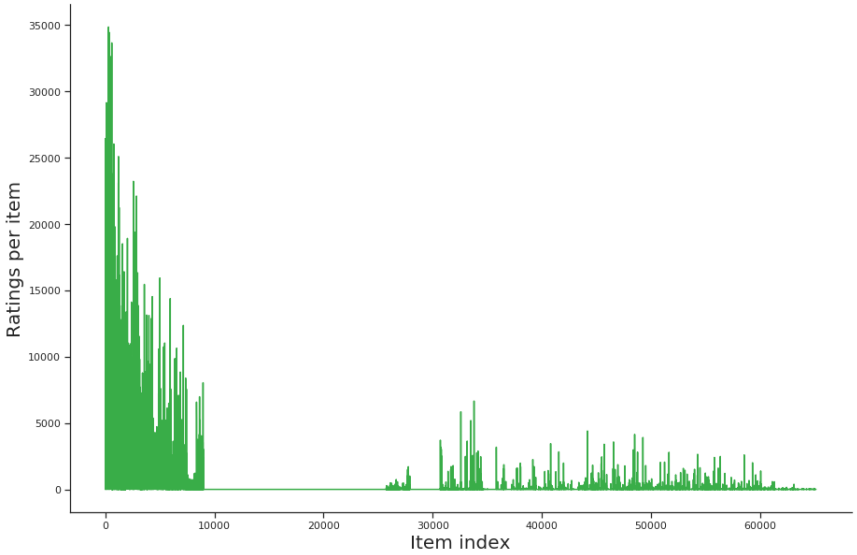

Figure 4.3: MovieLens 10M ratings per item

In Figure 4.3 instead we see the number of ratings given to an item. In this case we see that there are a large quantity of items that have little to no rating at all. However, on average an item has 153 ratings given to it. In this case the average

ICM

The information about the items instead is given in the form of social tags given by the users to the movies.

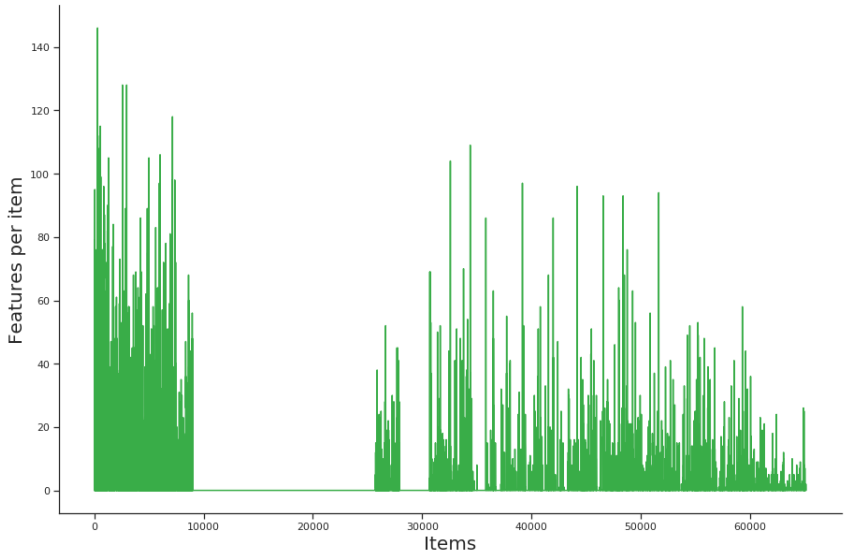

Figure 4.4: MovieLens 10M features per item

MovieLens 10M	ICM
Items	65134
Features	16529
Interaction	71155
Sparsity	0.9999

Table 4.2: Movielens 10M ICM statistics

In Figure 4.4 we see the distribution of features per items, where highest number of features an item has is 146. There are 57533 items with no features at all and 7601 items with features. On average we have that to each item it is associated a feature.

To get a clearer picture of the distribution of features per item, we see the extremes: the items with the highest number of features and the itemes with the lowest number of features.

F	Items with F features
146	1
128	2
122	1
118	1
115	1

Table 4.3: MovieLens 10M items most tagged items

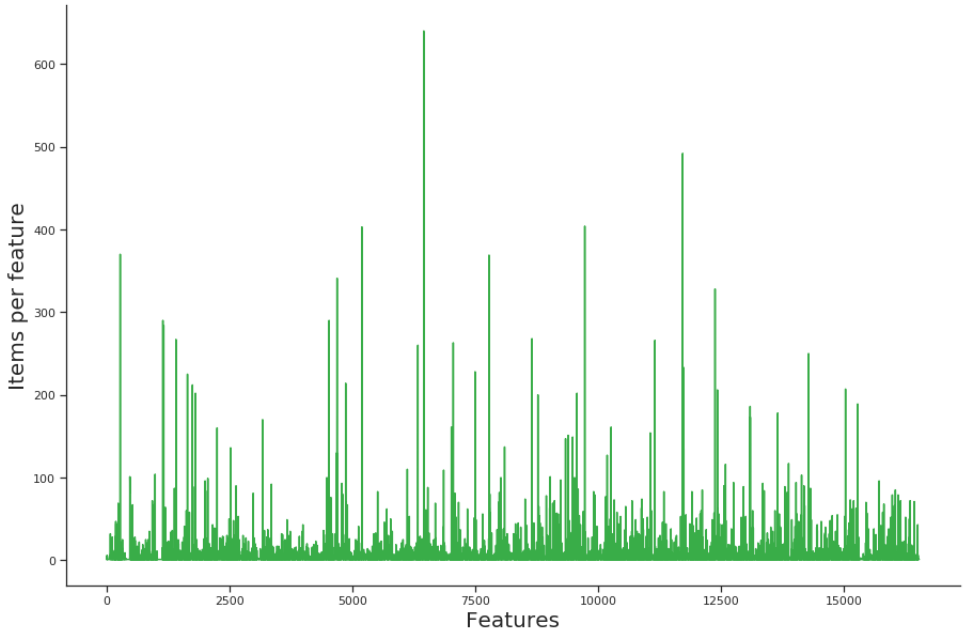

Figure 4.5: MovieLens 10M items per feature

In Figure 4.5 instead we see the distribution of the item per feature where the feature that describes most items goes up to over 600 items. On average we have that each feature is associated to 4 items.

F	Items with F features
0	57533
1	1519
2	1045
3	703
4	543

Table 4.4: MovieLens 10M least tagged items

Features examples
George Clooney
A nice romantic comedy
AFI #12
90s
3.5
9-2-2007
06 Oscar Nominated Best Movie
David O. Russell
2 endings
<3
==========

Table 4.5: MovieLens feature examples

In Table 4.5 we can see examples of the tags assigned by users to the movies. Given that social tags are assigned by users they vary a lot and can be from actor, directors to short reviews, ratings, or metadata about the movies as well as insignificant tags like the last example. It it fairly obvious that the quality of the features is low, very noise and most often redundant.

4.1.2 The Movies Dataset

The Movies dataset (TMD) is a public dataset published on the Kaggle website. It contains data collected from TMDB and GroupLens. It includes ratings, so URM, and various item information, so ICM. This second dataset is quite more complete with respect to MovieLens 20 Million, adding far more information about the items, thus making it a good example for feature selection scope of our work.

URM

In The Movies Dataset the ratings given by the users to the items are float numbers.

$$r_u^i \in \{0.5, 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5.\}$$

This dataset has a large number of users and item and even though it has over 25 million interactions it is very sparse, as it shows in Table 4.6.

URM	
Users	270882
Items	44711
Interaction	25M
Sparsity	0.9978
Average rating	3.5

Table 4.6: TMD URM statistics

The average rating given by the users is 3.5 where the maximum possible rating of 5.0 and the minimum 0.5 are both given. This tells us that users have shown a positive interest towards the movies present in the catalogue.

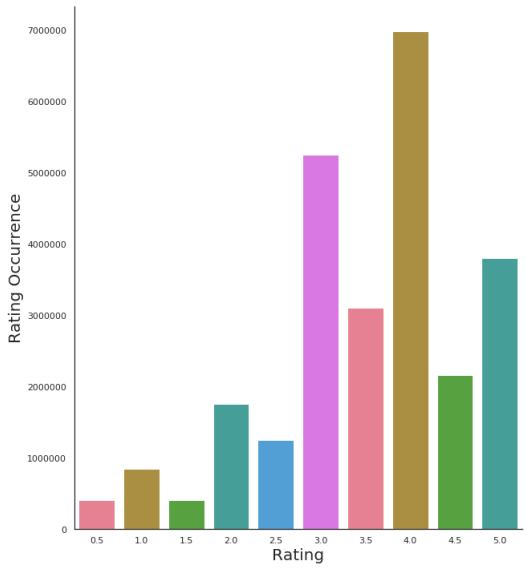

Figure 4.6: Rating distribution in TMD

In Figure 4.6 we see the distribution of the ratings where the most used rating is 4.0 followed by 3.0 and 5.0. We have 9910671 ratings below average, which is about 38% of the ratings overall while the rest of the ratings are above average. Therefore we have an imbalance of the ratings available in the dataset.

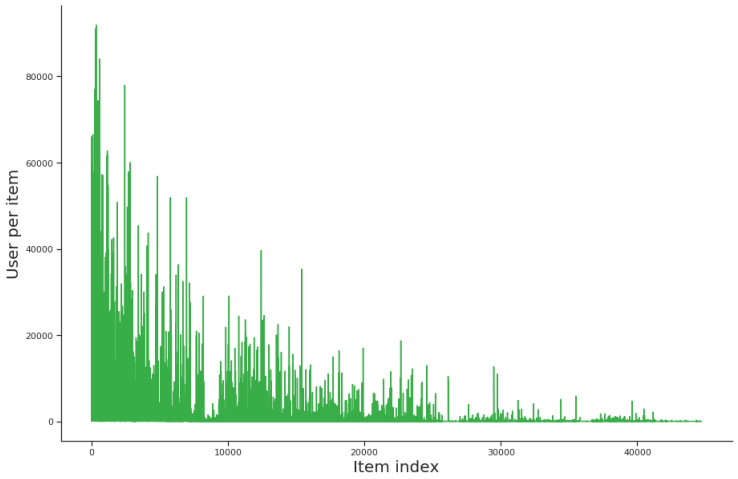

Figure 4.7: User interactions per item in TMD

In Figure 4.7 we the number of users that interact with each item where the item with highest number of interactions has 91921 ratings and the item with the lowest interactions has only 1 rating.

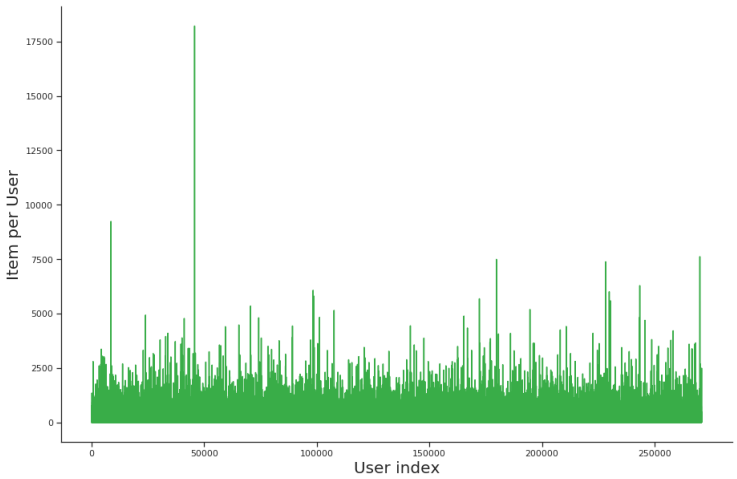

Figure 4.8: Items rated per user in TMD

In Figure 4.8 instead we see the distribution of items rated per user. In this case we have just a few outliers, so just a few users with unusual behaviour in the sense that give too many ratings with respect to the average ratings given per user, which is about 95 ratings per user.

ICM

For this dataset the features describing the items are of two types: metadata and credits. The metadata ICM provides information about a movie's metadata like genre, production company, release year, production country spoken language and so on. The credits ICM instead provides information concerning the cast and crew of the movies. We also have put both of them together to create a unique ICM. The experiments were conducted on the three different ICM.

Metadata

The features available for this type of ICM are about movie metadata.

Metadata features
Genre
Release year
Production company
Production country
Original language
Spoken language
Collection

Table 4.7: The metadata feature categories

In Table 4.7 we can see the seven different types of features in the metadata category. All these features are essential aspects of a movie that can tell us quite a lot for the user's preferences whether he/she is a fan of action movies or adventure,

period dramas or movies from the 70s, Walt Disney productions or Paramount pictures, movies in Italian or Dutch to different collection of movies.

Metadata	ICM
Items	44711
Features	3076
Interaction	221383
Sparsity	0.9984

Table 4.8: Metadata ICM statistics

In Table 4.8 instead we have some general information about the ICM regarding the metadata. Overall we have 3076 metadata features that describe 44711 items. The low density of the matrix tells us that the items are not fully described by the features and that per each item not all, if just a few features are associated to it.

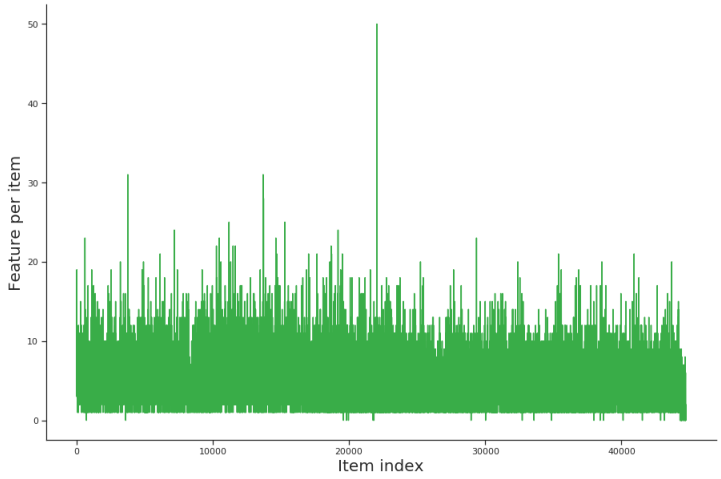

Figure 4.9: Features per item in metadata

In Figure 4.9 we see this exact relation between items and the features that describe it. We notice some outliers, so items with too many features example one with 50 features that describe it or just a few cases with over 30 features.

Items with F features	F
46	0
1907	1
5190	2
6803	3
7518	4
7149	5

Table 4.9: Items with specific number of features

However, the average number of features per user is 5 and in Table 4.9 we can see that there are quite a lot of items with very few features.

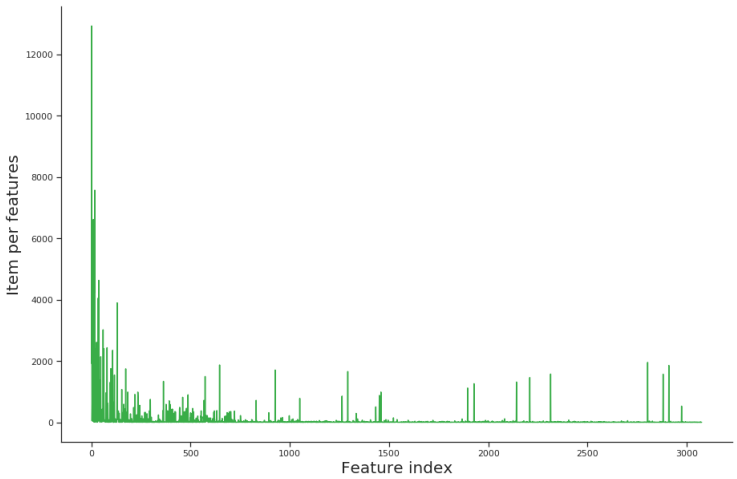

Figure 4.10: Items per feature in metadata

In Figure 4.10 instead we see the number of items a feature is associated to, so the feature occurrence. It is pretty obvious that we have a lot of rare features and very few popular ones.

Features with occurrence O	**O**
2	3
23	4
633	5
1	6613
1	7567
1	12917

Table 4.10: Top and bottom feature occurrences in

On average we have about 71 occurrences per feature and the reason behind such high number is due to the features that have a very high number of occurrences, as seen in Table 4.10. We have 2 features that appear only 3 times, 23 features that appear 4 times and so on. The feature with the highest frequency is only one and it appears 12917 times.

Credits

In this second ICM information about the people involved in the movie is provided, so the cast and crew of movies, names of actors and names of members of the crew.

Credits features
Cast
Crew

Table 4.11: The credits feature categories

It is very common for users to be driven by the cast of a movie given that the actors are a fundamental aspect of a film.

Credits	**ICM**
Items	44711
Features	46026
Interaction	525020
Sparsity	0.9997

Table 4.12: Credits statistics

Overall we have 46026 features of this type and as it can be seen in Table 4.12 once again we are faced with the sparsity issue. This however is fairly considerable since we have a high number of features and also the fact that the features are actors and crew and for each movie the number is not that high. The number of cast and crew depends a lot from the type of movie, where some require more crew to film the scenes while others require very few.

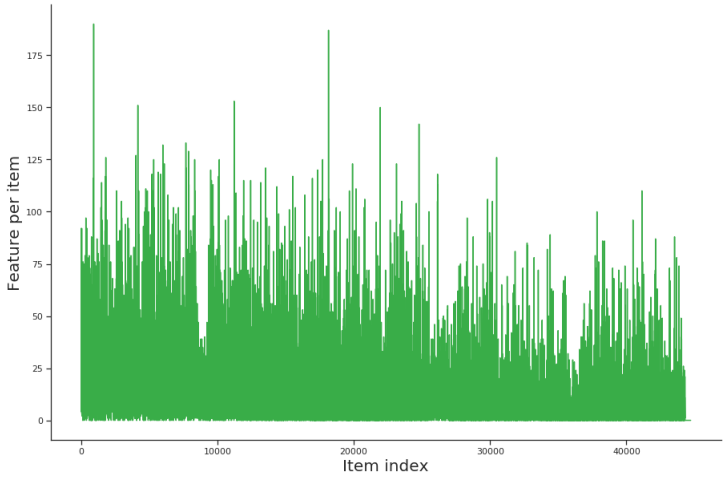

Figure 4.11: Features per item in credits ICM

In Figure 4.11 we can see this behaviour for all items. On average we have 11 features per item, so 11 cast and crew members.

Items with F features	**F**
3685	0
3164	1
2589	2
2387	3
2401	4
2392	5

Table 4.13: Items with lowest number of features in credits ICM

Items with F features	F
1	190
1	187
1	186
1	153
1	151

Table 4.14: Items with highest number of features in credits ICM

The low average of features per item can be explained by looking at the items with the highest number of features, Table 4.14 and at the items with the lowest number of features, Table 4.13. There are a few items with a very high number of features and a lot of items with very few features.

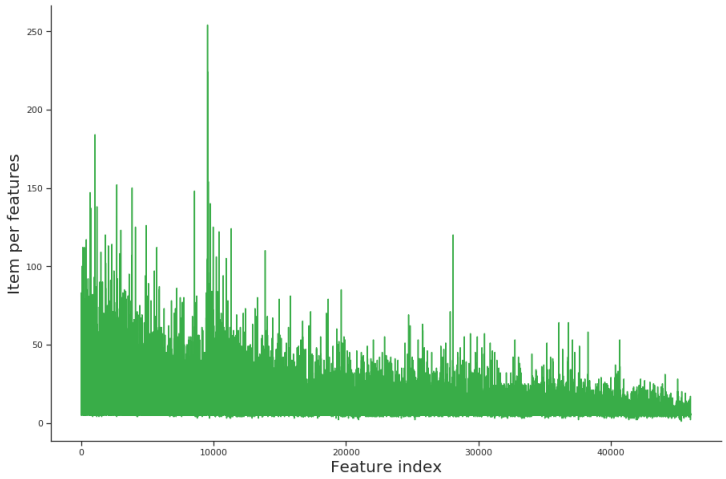

Figure 4.12: Items per feature in credits ICM

In Figure 4.12 instead we see the occurrences of each feature in the dataset, so for each feature we see how many times it is used to describe an item.

Features with O occurrence	O
1	1
11	2
49	3
1	184
1	224
1	254

Table 4.15: Highest and lowest feature occurrences in credits ICM

We see in Table 4.15 that the features with very high occurrence are a few while the features with few occurrences are more common.

Both

In this last ICM we have put together the metadata with the credits to obtain a complete description of the films.

Both	**ICM**
Items	44711
Features	49102
Interaction	746403
Sparsity	0.9997

Table 4.16: Both ICM statistics

In this third case we have put together metadata and credits to create a huge ICM with 49102 features to describe the items. This way the information for each item is more complete and when applying feature selection we can see which of the two types of features are more informative and indicative for the performance of the recommender.

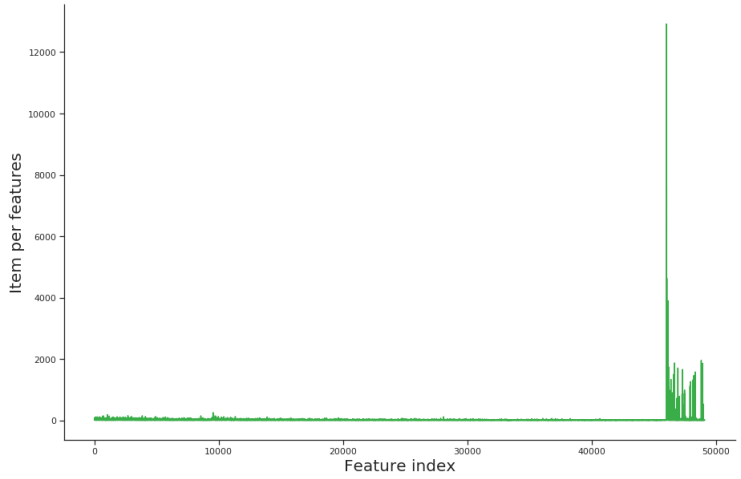

Figure 4.13: Items per feature in Both ICM

For this ICM the number of items per feature is the same as putting together side by side the items per feature of the previous two ICM-s.

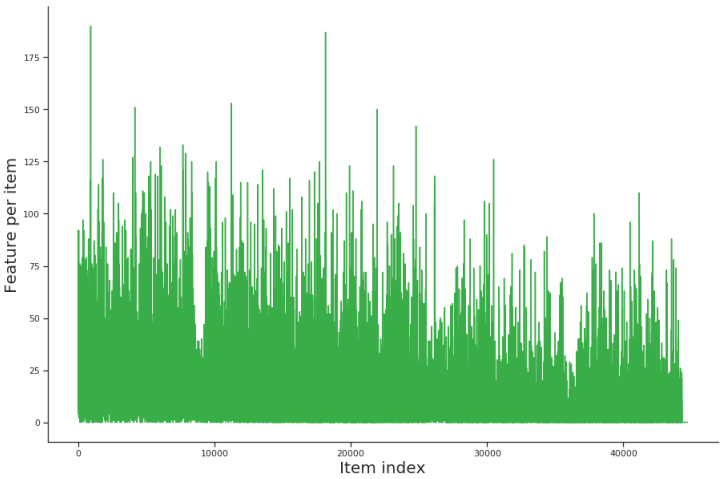

Figure 4.14: Features per item in both ICM

In Figure 4.14 instead we see the representation of each item through the features.

F	Items with F features
0	37
1	460
2	1203
3	1511
4	1671
5	1871

Table 4.17: Items with lowest number of features in both ICM

We see that there are only 37 items with no features at all then the rest described by few features

F	Items with F features
198	1
195	1
189	1
159	1
158	1

Table 4.18: Items with highest number of features

In Table 4.18 instead we see the opposite side where the items described high number of features are few.

4.2 Dataset split

As in any machine learning and recommender system to achieve generalization over a dataset and to have reliable results the division of datasets in three different subsets

is crucial. One of the most common division approaches is that of splitting the dataset in train, validation and test set. These subsets are then used respectively for training the algorithms, tuning the parameters and then testing the goodness of it on unseen data.

In a recommender system the division of the dataset is usually referred to the division of the URM, more precisely the interactions of the users with the items given that this is what we are aiming to learn.

The goal of every recommender system is that of predicting ratings of users for items they have not rated yet. To be able to test out how good our model is in doing such prediction, the test set is created and then the predictions are confronted with the actual ratings of the test set. The generation of the test set is done by diving the overall interactions in an 80-20% ratio where 80% of the data is held for the train set used to learn from and 20% used to test out the goodness of the model. The division of the interactions is done per user, in the sense that for each active user we apply the 80-20% ratio so as in the extreme cases of users with 1 or 2 ratings we have that such ratings are in the train set instead of the test set.

However, we need to keep in consideration the hyper-parameter search of the model, that is we need to have another portion of the available data to use when doing the hyper-parameter tuning as to find the best configuration. To generate such set we have applied the same approach of Train-Test split to the Train set to obtain the Train-Validation split. It was crucial to keep separated the test set from all the learning process throughout all the work, being that of the xGBoost as well as the recommender itself, to be sure that the results obtained are precise.

4.2.1 MovieLens 10 Million

Applying the described split procedure we obtain the three needed sets.

Train set	
Users	71568
Items	65134
Interactions	8000559
Sparsity	0.9982

Table 4.19: MovieLens 10M train set

The complete train consists of about 8 million interactions, which is 80% of the initial interactions present in the dataset. This makes it possible to learn about the users and then evaluate the goodness of the model

Test set	
Users	71568
Items	65134
Interactions	1999495
Sparsity	0.9995

Table 4.20: Movielens 10M test set

The remaining interactions go in the test set, so about 2 millions of them. Even in this case the matrix sparsity is quite high.

Validation set	
Users	71568
Items	65134
Interactions	1600135
Sparsity	0.9996

Table 4.21: Movielens 10M validation set

The subset of interactions needed for the model validation, in our case validation of xGBoost as well as the recommender system consists of about 1.6 million interactions.

New train set	
Users	71568
Items	65134
Interactions	6400424
Sparsity	0.9986

Table 4.22: MovieLens 10M new train set

After splitting the train set to obtain the validation set what remains are about 6.4 million of interactions, which will be used for the training of the models to then be used for the parameter tuning.

4.2.2 The Movies Dataset

Train set	
Users	270882
Items	44711
Interactions	20779247
Sparsity	0.9983

Table 4.23: TMD train set

In Table 4.6 we can see the part of the URM that becomes that basis for the training of our models which contains about 80% of the interactions of the original URM.

Test set	
Users	270882
Items	44711
Interactions	5193834
Sparsity	0.9996

Table 4.24: TMD test set

The test set instead for this dataset is described in Table 4.24 where the remaining of the interactions is left to evaluate the goodness of the approach.

Validation set	
Users	270882
Items	44711
Interactions	4154153
Sparsity	0.9986

Table 4.25: TMD validation set

The validation set for The movies dataset which is used to tune the various hyper-parameters present in the model, as described in 4.25, is made of about 4 million interactions which are chosen from the Train set presented in Table 4.23. The train set then becomes:

New train set	
Users	270882
Items	44711
Interactions	16625094
Sparsity	0.9996

Table 4.26: TMD new train set

We have thus defined all the necessary components to proceed with the results as well as the conclusions of our work.

Chapter 5

Experimental results

5.1 Implementation details

All experiments were conduced using Python version 3.6. Various libraries were used: Numpy, Scipy, pandas, xgboost, sklearn. The functions and classes provided by such libraries are well-known to be efficient when working with sparse matrices and mathematical operations. The experiments were run of a server with 12GB of RAM and 8 cores, making it possible to run the experiments in parallel as well.

5.2 Experiments

The following experiments were done on both datasets and the results obtained were in the same line for all the cases. To avoid repetition the results are reported for one dataset per experiment.

5.2.1 xGBoost impact

xGBoost is the machine learning algorithm that we opted to use for the our feature selection technique and it plays a crucial role for the process. The importance of the features is what we use to decide on which features to keep and which ones to discard. The various experiments showed that even though we introduced samples with a high number of features to the model, up to 50 000 features for one dataset, it is able to distinguish just a small portion of them as relevant.

The two most important hyper-parameters of an xGBoost model are the number of trees and the depth of such trees. To tune the two hyper-parameters Bayesian optimization was used, as explained in Section 3.3.1, therefore the following plots are produced from the different runs of the Bayesian optimization. The entire set of hyper-parameters present in the xGBoost are not optimal, just the points of the hyper-parameters space that the search algorithm looks at when optimizing the objective function provided to it.

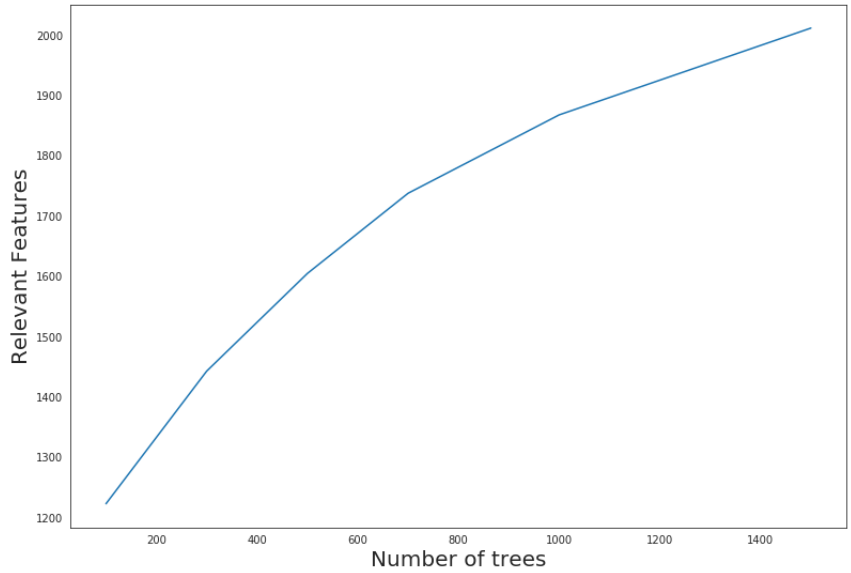

Figure 5.1: Number of tree variation in MovieLens 10M

In Figure 5.1 we see how the variation of the number of trees the model learns leads to a variation of the number of relevant features found by it. That is due to the fact that the more trees there are the more features are explored and thus with the increase of the trees increases also the number of relevant features where relevant ones are those with an importance different from zero. However, the number of features considered as relevant after the learning is done does not give us a clear picture of how these subset of features affects our recommender.

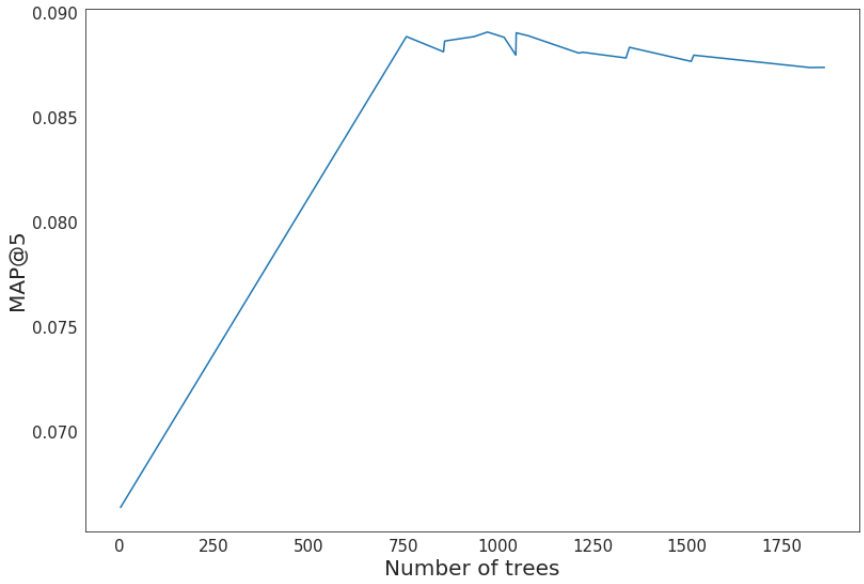

Figure 5.2: xGBoost number of trees impact on MovieLens 10M

In Figure 5.2 instead we see how the increase of the number of trees affects the overall performance of the recommendations obtained with the reduced set of features. The core idea of the xGBoost model is that a new tree aims to improve the performance of the previous tree learnt. Adding trees over a limit does not improve the performance of the model, it just goes to overfit the training data. However,

the implementation of the model has incorporated Early Stopping strategy where even though we set up a very high number of trees, even up to 10000, when the performance of the model does not improve on the validation set the model stops learning and the stopping point is the number of trees sufficient to learn from the available training data.

We see that the impact on the recommender system (validation set)of the number of tree variation and such variation has a great impact on the quality of the predictions made. We see also that after a certain number of trees the model does not improve anymore for the current dataset. This however is also due to all the other hyper-parameter variation.

The other key hyper-parameter for the model is the depth of the trees.

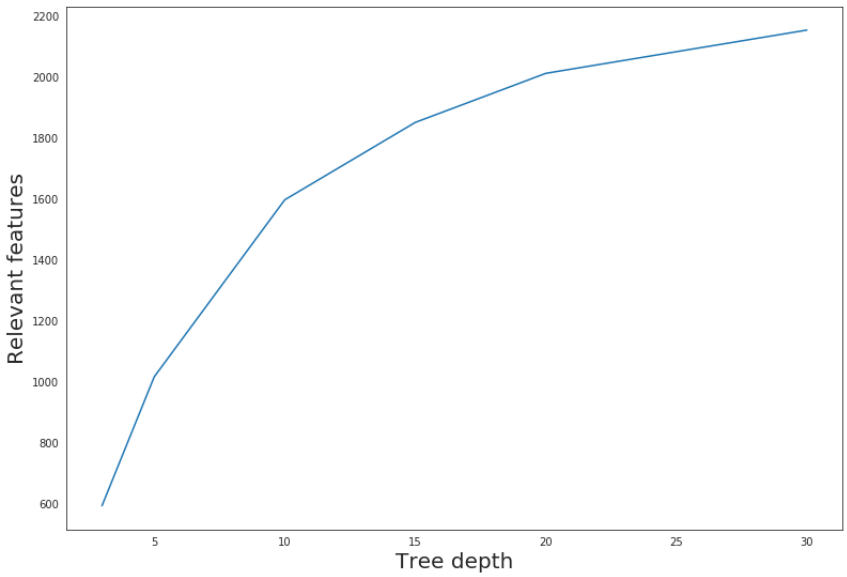

Figure 5.3: xGBoost depth impact on MovieLens 10M relevant features

In Figure 5.3 we see how the change of tree depth affects the number of relevant features. The deeper the tree, the more splits it has therefore the more features are used to build the tree. This leads to the increase of the relevant features, so the features that contribute to the decision making increase. However, deeper trees can easily lead to overfitting the training data and not necessarily mean that the performance of the model overall improves.

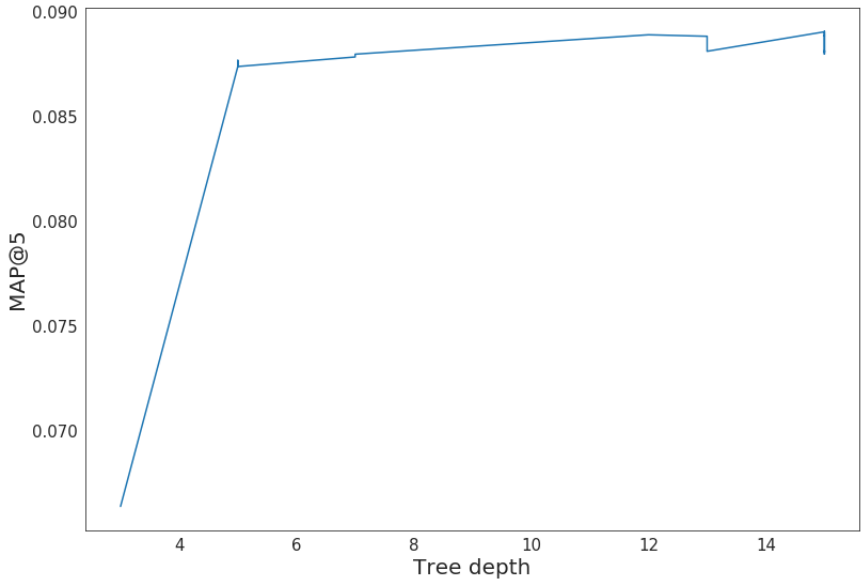

Figure 5.4: xGBoost tree depth impact on MovieLens 10M

In Figure 5.4 instead we see the impact that it has on the final recommendation performance and in the case of MovieLens we see that with a depth equal to 15 we obtain the highest performance.

5.2.2 Feature selection

In Section 2.5 we talk about the different ways to select the features once we have the importance provided by the xGBoost algorithm.

	MAP@5	Precision@5	Recall@5	NDCG@5
Threshold: average importance	0.1606	0.2057	0472	0.0962
Threshold: above zero	**0.1637**	**0.2086**	**0.0480**	**0.0980**
Percentage: 30 %	0.1582	0.2045	0.0471	0.0949
Percentage: 50%	0.1595	0.2050	0.0473	0.0960
Percentage: 70%	0.1606	0.2058	0.0472	0.0962

Table 5.1: Different feature selection techniques in MovieLens 10M

In Table 5.1 see the results of applying the different techniques, evaluated on the test set. For all experiments the importance of the features calculated by the xGBoost model is used.

The first case is one that using as a threshold the average importance calculated for all features and then keeping only the features with an importance equal or higher than that.

The case of a threshold ≥ 0 yields the best results and it was thus used for the rest of our approach. Given that the resulting importance of the features provided by xGBoost has a large quantity of features with importance equal to zero keeping all the features with an importance difference from zero is also the logical approach.

The last three approaches instead are referred to cases where choosing a percentage of the features with an importance different from zero. This case also shows us that with only 30% of the relevant features we can obtain a considerably high quality in prediction. It is however best to keep all of them.

5.2.3 Technique stability

In Section 3.2.1 we describe how the feedback extraction process it's done. For each user we keep the feedback, positive and negative, and such number can be changed.

Kept feedback	**MAP@5**	**PRECISION@5**	**RECALL@5**	**NDCG@5**
3	0.1585	0.2042	0.0471	0.0953
5	0.1624	0.2068	0.0476	0.0976
7	0.1625	0.2078	0.0477	0.0970
10	0.1621	0.2075	0.0476	0.0969

Table 5.2: Varying number of feedback kept in MovieLens 10M

The results in Table 5.2 we see the impact of keeping a different number of feedback per user, where for each user we keep the same number of positive feedback as negative ones (e.g. first case we keep 3 positive feedback and 3 negative feedback).

This variation of the features kept leads also to an increase of the number of samples.

Kept feedback	**Positive samples**	**Negative samples**
3	209124	209124
5	690741	690725
7	1414012	1413880
10	2814344	2817069

Table 5.3: Varying number of feedback impact on number of samples

As we see in 5.3 with the increase of the feedback kept we have an increase in the number of samples generated as well. However, such increase in number of samples does not have an impact on the quality of the recommendations done after a certain threshold. Our technique shows to be stable even with the increase of the number of samples provided to the xGBoost.

5.2.4 Not selected features

In all four cases where our feature selection technique was applied a low number of features was selected with respect to all the available ones. The remaining features were deemed as not relevant. However, it is interesting to see how the quality of

the recommendation changes when adding even the ones not deemed important up until all features are used.

The movies dataset: Both

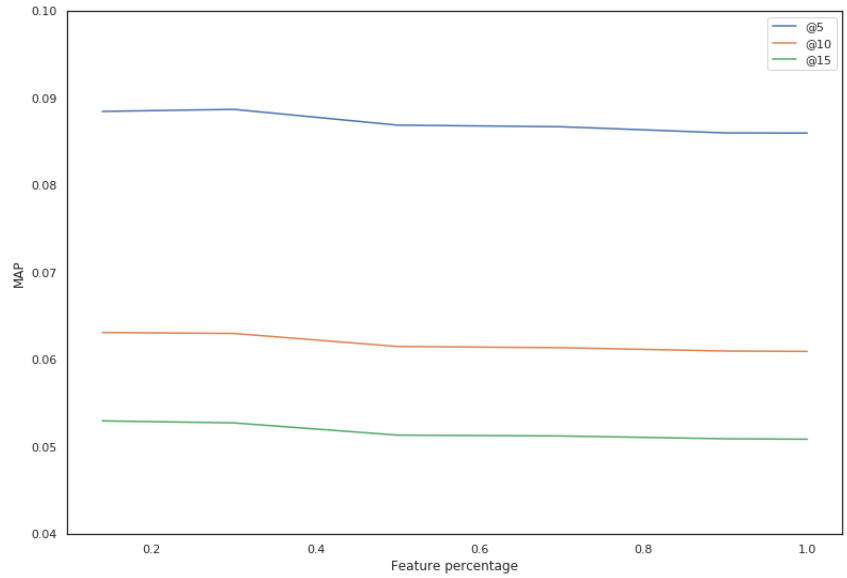

Figure 5.5: TMD Both MAP performance with feature percentage variation

For The movies dataset, ICM containing both metadata and credits, our feature selection technique is able to distinguish the 14% from the overall features as the most relevant ones while the remaining 86% are discarded. Starting by using only the selected features and then adding the discarded features a group at a time we see in Figure 5.5 that the performance, in this case the MAP metric, does not improve, if anything it gets slightly worse.

This proves that the selected features are indeed the most informative ones.

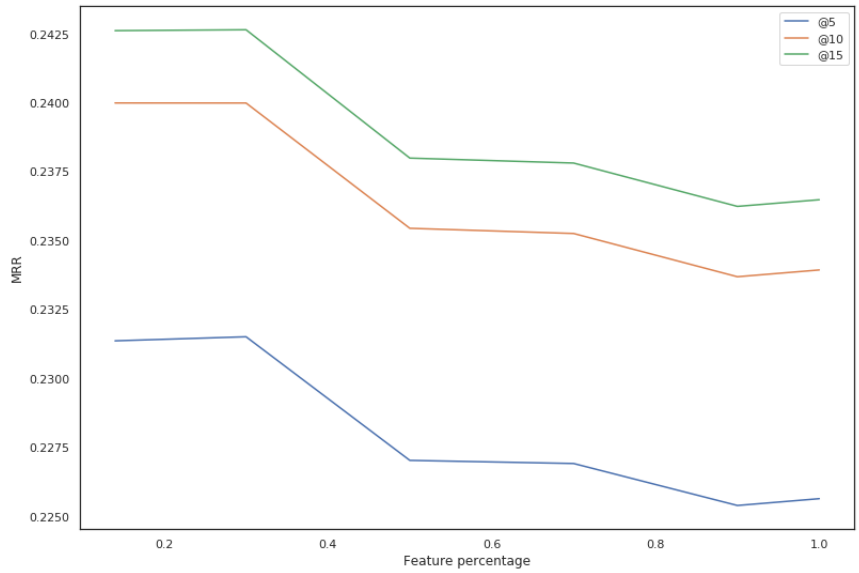

Figure 5.6: TMD Both MRR performance with feature percentage variation

In Figure 5.6 instead we see the effect of adding the discarded features from our approach on the MRR metric. There is a significant worsening of the performance with the more discarded features we add which means that the relevant items are ranked higher when the irrelevant features are added. This means that the discarded features add noise to our prediction and it is better to not use them when recommending.

The two graphs show to us that while for accuracy metrics the feature selection approach improves by little for the ranking metrics like MRR it has a great impact. Adding the features that are deemed as irrelevant by our technique leads only to a poorer performance of the recommender.

5.3 Results

In this section the experiments conducted in the two datasets are analyzed, on Movielens 10 million ICM as well as the three available ICM matrices, Metadata, Credits and both put together. Every experiment is done the same in all 4 cases so that the conclusions extracted can be confronted and generalized.

The following results are showed for two cases:

- **ItemCBFKNN**: the baseline, where all the features available are used
- **FSItemCBFKNN**: the Feature selection ItemCBFKNN, where the recommendations are made using only the selected features by our technique

5.3.1 MovieLens 10M

As described in Section 4.1.1 MovieLens 10M is a dataset that consists of user provided tags for movies. We have 16529 feature overall to use for the recommendations.

Feature selection	
All	16529
Selected	1853
Percentage	11%

Table 5.4: MovieLens 10M feature selection

The application of our technique leads to the selection of only 1853 features, which is just 11% of the initial features in the dataset.

Explainability

Top 10 selected features
Tumey's DVDs
Imdb top 250
Seen more than once
DVD
Erlend's DVDs
Classic
Owned
Seen at the cinema
Action
Eric's Dvds

Table 5.5: MovieLens 10M top 10 selected features

From the top 10 features selected by our technique we see different kind of information. The most important feature, "Tumney's DVDs", is a tag that upon thorough search on the internet results to be the tag associated to a user who has rated a large amount of movies and thus has become a reliable source of information. The same thing can be said about "Erlend's DVDs" and "Eric's DVDs". The "Imdb top 250" tells us that the movies tagged with this tag are popular and well-liked ones. Others like "DVD" and "Owned" instead tell us these movies are good enough to end up buying them. "Seen more than once" tells us that it's that good of a movie to be watched more than once. Tag "action" describes the genre of the movie as does "classic". We see that the movies deemed relevant by our approach are tags that provide different type of information but nonetheless are able to give a good description of a movie.

Least important features
awsom
Hong Kong
strong
story of one family
it thought it was funny but it wasn't
monkey
space program
Vince Vaughn
Cate Blanchett

Table 5.6: MovieLens 10M 10 least important features

In Table 5.6 instead we see the least important features from those selected by our technique. The tags vary from actor's names to short reviews as well as information about the movies like "space program".

Non selected features
ha ha ha
hack the system
grocery store
looooooooooooong
fxg

Table 5.7: MovieLens 10M non selected features examples

In Table 5.7 instead we see examples of non selected features. It is rather quick to understand why such tags were discarded and it is also an indicator that our technique is able to distinguish what is a relevant tag and what is not.

IDF confrontation

Given that features are reoccurring, especially in the case of tags where a tag can be associated to many items in describing them a confrontation with the frequency of a tag's appearance is interesting and informative. To make the confrontation more robust the Inverse Document confrontation (IDF) of each term is calculated. IDF is the numerical statistic that is used to reflect on how important a word is to a document or a corpus. In our case this can interpreted as the importance of a features over all the items we are tagging. It is often used as a weighting factor in information retrieval. Consider D a set of documents with cardinality N, so $N = |D|$ and t a term. IDF can be defined as:

$$IDF(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{5.1}$$

For our application, where a feature is considered as a term and the document set if the item set, the definition of IDF becomes:

$$IDF(f, I) = \log \frac{|I|}{|\{i \in I : f \in i\}|} \tag{5.2}$$

The higher the value of the IDF is, the more frequent is the term we are considering while the lower it is the rarer the term is present in the set of documents in consideration.

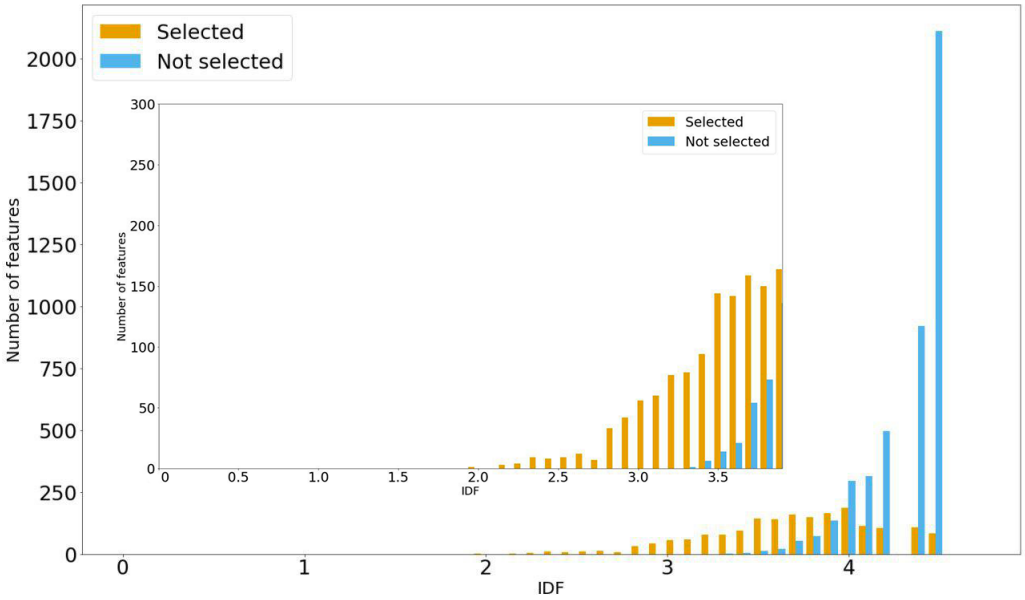

Figure 5.7: Movielens 10M selected features confrontation with IDF

Diving the features in two categories as those selected by our approach and those non selected, in Figure 5.7 we see the histogram of such two classes with respect to the IDF values of the features. In the y-axis instead we see the number of features for each category in the specific IDF value. Going from left to right on the the x-axis we have an increase of the IDF value, which means a decrease in frequency of the terms. The smaller graphic instead is an enlargement of the graph where the x-axis ranges from 0 to 3.5 value of IDF and the number of features is up to 300. This enlargement is done to make clear as well as emphasize the behaviour of the two categories of the features.

The more we go on the right the rarer the features are and from this graph we can see that a high number of rare features are not considered as relevant by our technique. However, we also see that rare features are not completely discarded, on the contrary, a considerable amount of the features selected are chosen from these less occurring features. We notice also in the enlarged graph that there are features with a lower IDF value, so more frequent features, that are discarded by our technique.

Such visualization of the two categories of the features gives a clear idea that our technique is able to find relevant features when they are frequent as well as when they are not. While it is affected by the feature frequency we can see that is not completely dependent from it.

Numerical results

	MAP	PRECISION	RECALL	NDCG	MRR
ItemCBFKNN	0.1629	0.2084	0.0472	0.0949	0.3870
FSItemCBFKNN	**0.1637**	**0.2086**	**0.0480**	**0.0980**	**0.3958**

Table 5.8: MovieLens 10M results at cuoff 5

	MAP	PRECISION	RECALL	NDCG	MRR
ItemCBFKNN	**0.1161**	**0.1656**	**0.0724**	0.1178	0.4002
FSItemCBKNN	0.1150	0.1630	**0.0724**	**0.1203**	**0.4082**

Table 5.9: MovieLens 10M results at cuoff 10

In Table 5.8 and 5.9 we see results of two cases. For the Movielens 10M our feature selection technique is able to outperform the baseline for all metrics in the case of cutoff = 5 while with cuteoff = 10 MAP and Precision are slightly lower than the baseline. The two metrics are strongly correlated to one another thus the lower performance on both of them is reasonable. However, NDCG and MRR instead show an improvement, meaning that the ranking of the recommendation is done better with less features.

This leads to the conclusion that the initial feature set is made out of noisy and redundant features that do not contribute to the recommendations made, if not making it worse and that the ranking quality of the recommendations improves with the case of reduced feature set.

5.3.2 The Movies Dataset: Metadata

Explainability

The metadata ICM matrix of The movies dataset is described in detail in Section 4.1.2. We have 3076 features overall to describe the items of the dataset.

Feature selection	
All	3076
Selected	488
Percentage	16%

Table 5.10: Metadata feature selection

After applying our feature selection technique on the metadata ICM the algorithm is able to distinguish from all the features only 488 which have an impact on the performance of the recommender which is about 16% of the all available features.

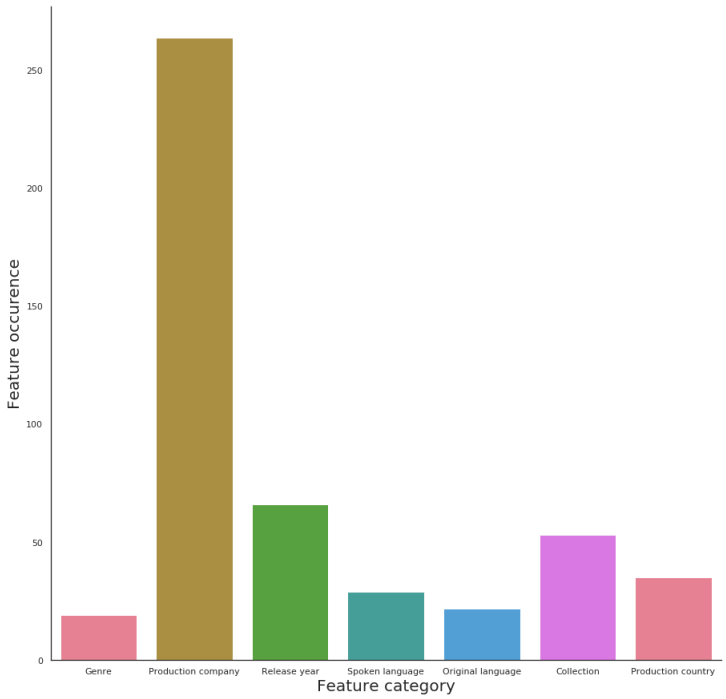

Figure 5.8: Metadata feature category occurrence on selected features

In Section 4.1.2 where the metadata ICM is described we saw that there are 7 different types of metadata categories. In Figure 5.9 we see how each of these categories are distributed in the selected features. We see that the most recurrent feature category is that of production company followed by release year and collection. The rest are less recurrent among the selected features.

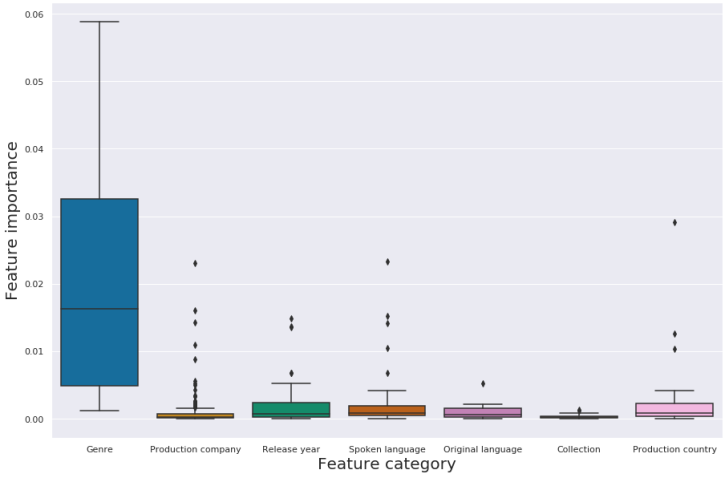

Figure 5.9: Metadata feature category distribution on selected features

However, the occurrences of the categories do not give us a good understanding of which of the categories has the most impact on the performance of the model. In Figure 5.9 we get a clear picture of the category importance distribution. We see that genre is the category with the highest values of importance even though the occurrences of such category were low with respect to the other categories. This means that a movie's genre is a driving feature that tells a lot about a user's

preferences thus makes the recommendation of movies with the same genre is a good way to go about it.

IDF confrontation

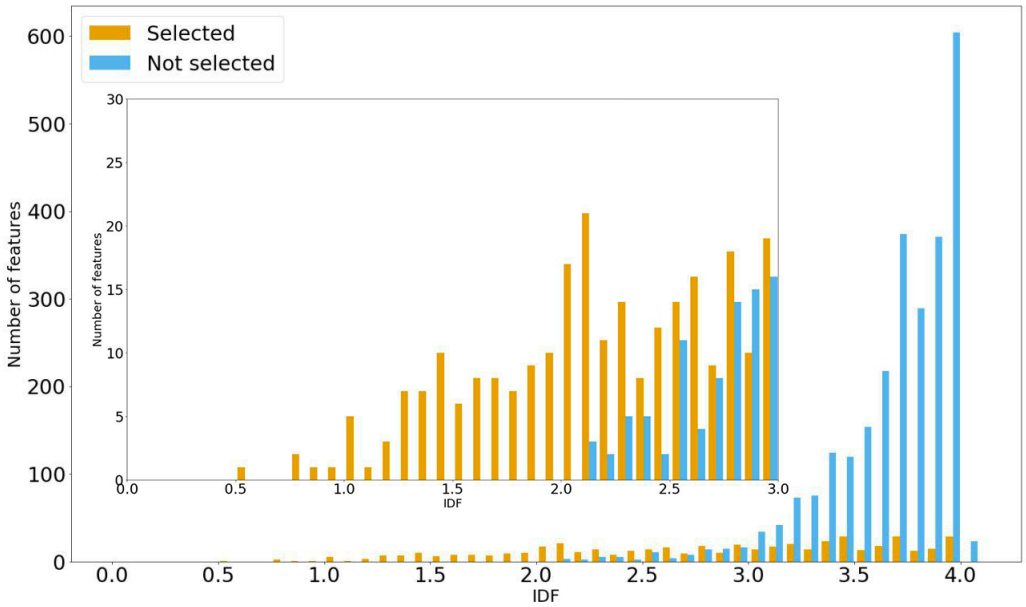

Figure 5.10: Metadata feature frequency in confrontation with IDF

The definition of IDF, as given in Equation 5.2, tells us that the rarer the features is the higher the value of the IDF is. While the selected features are very few, 488, we see that in terms of frequency there is no trend whatsoever, that is, the selected features are distributed in different IDF values. We see that a lot of rare features are discarded as irrelevant by our technique. However, it is interesting to notice that there are rather frequent features that are discarded, as seen in the enlargement.

Numerical results

	MAP	Precision	Recall	NDCG	MRR
ItemCBFKNN	**0.0533**	**0.0709**	**0.0215**	**0.0370**	**0.1708**
FSItemCBFKNN	0.0518	0.0683	0.0212	0.0363	0.1667

Table 5.11: Metadata results at cutoff 5

	MAP	Precision	Recall	NDCG	MRR
ItemCBFKNN	**0.0353**	**0.0507**	**0.0290**	**0.0426**	**0.1771**
FSItemCBFKNN	0.0342	0.0499	0.0289	0.0417	0.1730

Table 5.12: Metadata results at cutoff 10

In Table 5.11 and 5.12 we see the numerical confrontation of our approach with the baseline in the case of metadata ICM. For this ICM our approach performs slightly worse than using all features. From this we can conclude that the metadata ICM is rather noisy given that only with 16% of the features we get a comparable performance with the baseline, nonetheless our technique was not able to find a feature subset able to improve the performance of the ItemCBFKNN approach.

5.3.3 The Movies Dataset: Credits

Explainability

The detailed description of the credits ICM is given in Section 4.1.2. We have an overall high number of features, 46026, available in the dataset.

Feature selection	
All	46026
Selected	10295
Percentage	22%

Table 5.13: Credits feature selection percentage

The resulting features selected by our approach are about 22% of all the cast and crew of the initial dataset. Our technique manages to capture a subet of the actors and crew of the movies that best describe the movies of the dataset without loss in recommendation quality.

Top 10	**selected features**
Steven Spielberg	Tom Hanks
Kathleen Kennedy	Samuel L. Jackson
John Williams	Dale E. Grahn
Hans Zimmer	Debra Zane
Alan Silvestri	John Lasseter
Stan Lee	Frank Welker
Michael Kahn	Scott Rudin
Dan O'Connell	Frank Marshall
James Horner	Danny Elfman
John Ratzenberger	Janet Hirshenson

Table 5.14: Top 20 selected credits features

In Table 5.14 we see the 20 most important features selected by our approach, with an importance decreasing from top to bottom. Some of the names are quite familiar, like actors Stan Lee, Tom Hanks or Samuel L.Jackson or director Steven Spielberg. We have well-known music composers like John Williams and Hans Zimmer.

The rest of the names are quite unfamiliar, even for a cinephile like me however a look up of the names shows that indeed they are important: Kathleen Kennedy,

which is the second most important feature is a movie producers and current president of the LucasFilm, which is the production company of the famous *Star Wars saga*. Alan Silvestri, which is the fifth most imporant feature, is a music composer of movies like *Forrest Grump*, *Back to the future triology*, *Cast Away* and many others. Michael Kahn instead is a film editor who is known to be Steven Spielberg's right hand(over 30 collaborations), hence it has worked on many blockbusters. James Horner is another music composer that has composed the soundtrack of *Titanic*, *Avatar*, *Brave heart* and many other.

Looking up the rest of the names we see that all of them are part of the crew, like editors, music composers, producers and so, working with well known directors and very famous films.

5 least important features
Jose Pablo Cantillo
Gregory Jbara
Joel Hynek
Jemaine Clement
Sam Levene

Table 5.15: Credits least 5 important features

In Table 5.15 instead we see the least 5 important features that are however selected by our feature selection process. Looking them up we can see that they are all actors with at least a couple of movies in their career.

Non important features
Anne Vernon
Marc Michel
Jean Champion
Jacques Demy
Jean Rabier

Table 5.16: Credits not selected features

From the remaining features, those non selected, there are many actors and cast members that have appeared once or very few times in a movie or crew members that have worked in non important films thus deemed not important. In Table 5.16 we can see some examples of these discarded features.

IDF confrontation

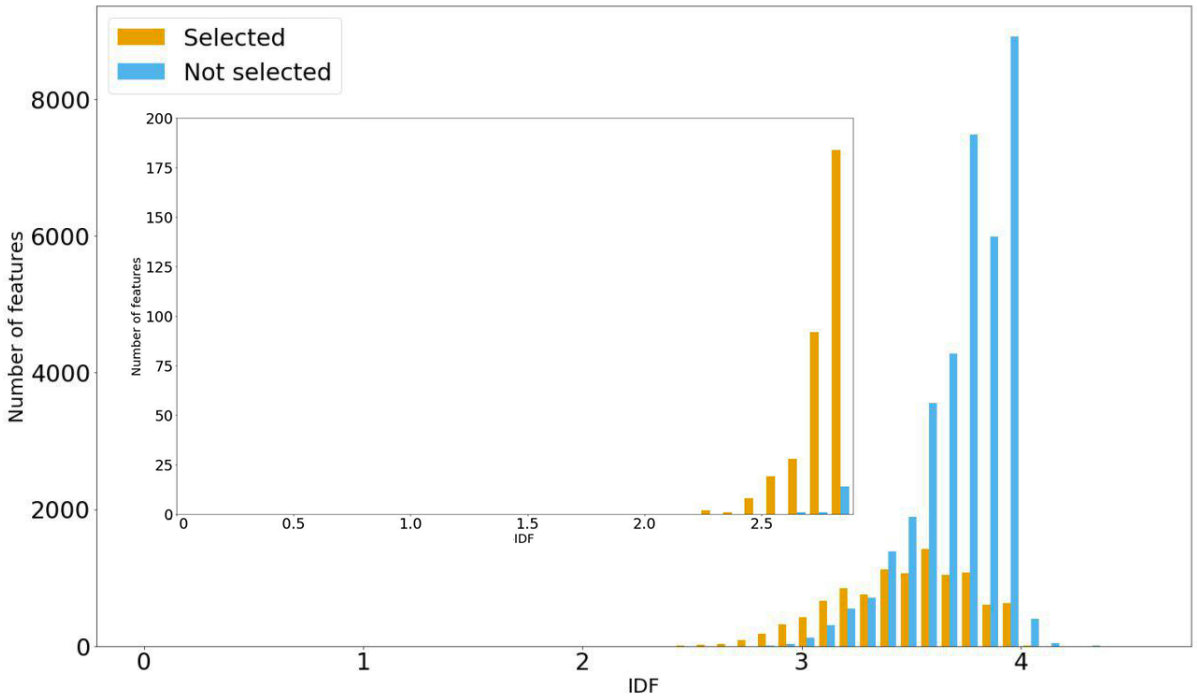

Figure 5.11: Confrontation of credits features importance with feature frequency

As per the IDF definition, as provided in Equation 5.2, the more on the right we go the rarer the features are. We see that the features selection occurs on the far right. A great deal of rare features are discarded however the selected features are also mainly concentrated on rare features. This means that however correlated our technique may be with the feature frequency it does not completely depend on that.

Numerical results

	MAP	Precision	Recall	NDCG	MRR
ItemCBFKNN	0.0781	0.1037	**0.0323**	0.0512	0.2158
FSItemCBFKNN	**0.0815**	**0.1065**	0.0320	**0.0523**	**0.2194**

Table 5.17: Credits results at cutoff 5

	MAP	Precision	Recall	NDCG	MRR
ItemCBFKNN	0.0559	0.0805	0.0457	0.0616	0.2245
FSItemCBFKNN	**0.0580**	**0.0829**	**0.0492**	**0.0623**	**0.2282**

Table 5.18: Credits results at cutoff 10

In Table 5.17 and 5.18 we can see the numerical results. We see a significant improvement in all the metrics presented, being that at a cutoff of 5 as well as a cutoff

of 10. This means that our technique is able to capture the movie content information with just 22% of the initial features, which is a significant reduction of the feature space.

5.3.4 The Movies Dataset: Both

Given that our third type of ICM for this dataset is the collection of the previews two datasets put together the rest of the analysis is done confronting the effect each of these two subset has on the overall performance.

Explainability

Putting together the metadata with the credits we obtain 49102 features which is exactly the sum of metadata with credits features.

Feature selection	
All	49102
Selected	6835
Percentage	14%

Table 5.19: TMD selected features percentage

Our technique is able to select only 14% of the initial feature set, which is still a combination of the two different sources put together.

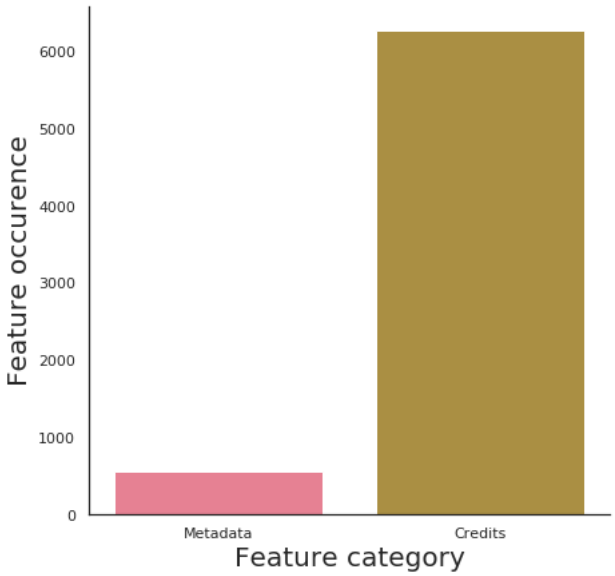

Figure 5.12: Metadata and credits occurrence in selected features

From the 6835 selected features in Figure 5.12 we see that just a few are metadata features and the rest all comes from the credits category. These are sound proportions, given that we have 3076 metadata features overall and 46026 credit features.

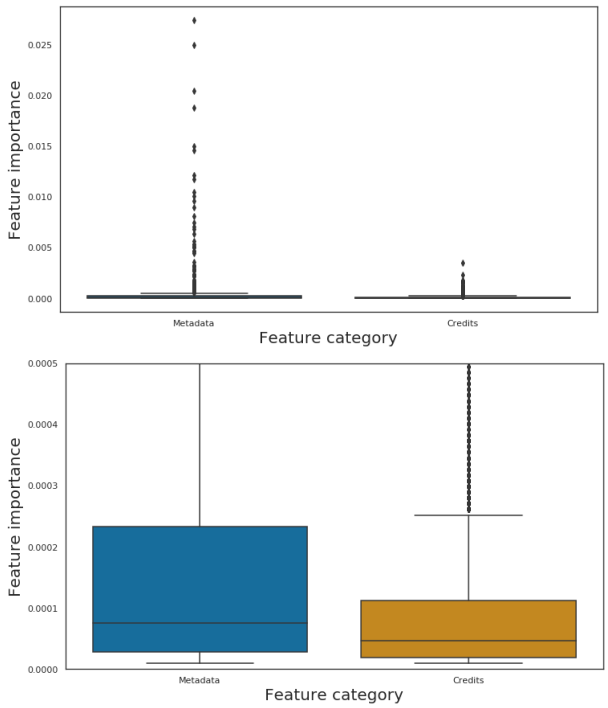

Figure 5.13: Metadata and Credits features in Both ICM

However, this is not a good indicator of what is more important between the two categories. In Figure 5.13 we see the distribution of each category importance, that is just the selected features. From the first image we understand that the average importance values are quite low for both categories and that in the case of metadata features there some outliers which have a high importance. In the second image instead, which is an enlargement of the first, we see that metadata features actually have a higher importance in confront to the credit ones, despite the low number of metadata features selected.

IDF confrontation

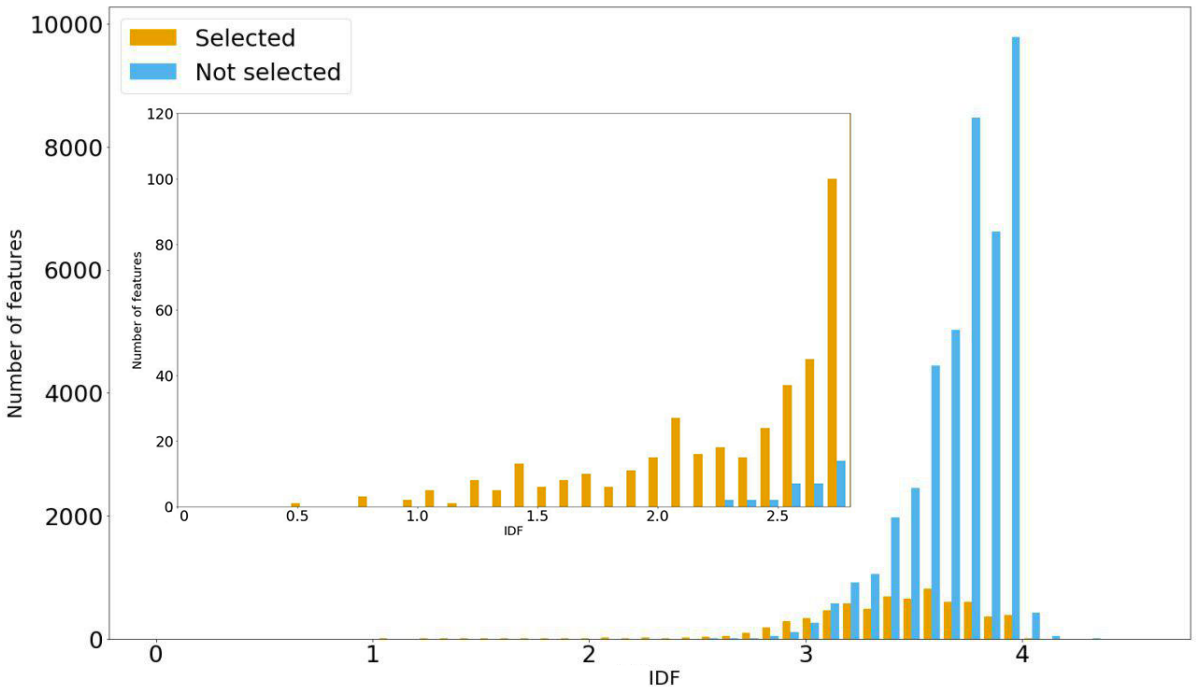

Figure 5.14: TMD Both features importance confrontation with IDF

In Figure 5.14 we see once again the confrontation with the IDF and analyse the correlation of our technique with feature frequency. In this case the similarities with the previous cases of credits and metadata ICM confrontation with IDF are evident. It is in line with the results obtained before where the selection is concentrated on rare features, adding in this case also some more frequent features like in metadata.

Numerical results

	MAP	Precision	Recall	NDCG	MRR
ItemCBFKNN	0.0860	0.1109	0.0328	0.0534	0.2256
FSItemCBFKNN	**0.0885**	**0.1140**	**0.0342**	**0.0557**	**0.2314**

Table 5.20: Results on both with cutoff 5

	MAP	Precision	Recall	NDCG	MRR
ItemCBFKNN	0.0609	0.0847	0.0455	0.0636	0.2252
FSItemCBFKNN	**0.0631**	**0.0878**	**0.0479**	**0.0665**	**0.2400**

Table 5.21: Results on TMD both feature types with cutoff 10

Numerical results show us that our approach manages to outperform the baseline using only 14% of the initial features. We can conclude that our feature selection technique is able to select the features that are informative enough to describe the content of the movies in the dataset.

Chapter 6

Conclusions

In this thesis we propose a new feature selection technique for content-based approaches in Recommender Systems. Our technique captures the feature importance through the use of a machine learning algorithm which learns from past user ratings. We assessed the performance of the content-based approach when using the features selected by our technique and compared it to the baseline which uses all the available features.

We applied our technique on two different movie datasets, which both have explicit ratings. The available features of these datasets were distinct: one dataset has social tags where as the other one has movie metadata as well as cast and crew describing the movies. It is worth pointing out that our technique is indifferent to the context of feature space that is being used.

Feature selection techniques lead to dimensionality reduction; in fact, our technique uses a fraction of the initial available features ranging from 10% to 22%. Nevertheless, in almost all the datasets, this did not lead to loss of recommendation quality with respect to the baseline. The different metrics that we used to evaluate the quality of our technique showed that in almost all cases it outperforms the baseline and, in particular, the ranking metrics showed a significant improvement.

Content-based approaches have the advantage of being easily explainable to the users, however, the wide range of features available makes it difficult. Applying our technique such explainability is rendered easier and clearer given that not only is the feature space far smaller, but we also have an order of importance for the features being used.

A comparison between the selected features and the not selected ones with their respective IDF showed that while our selection technique is correlated to the feature frequency, the two concepts do not overlap. In all four ICM-s our technique is able to discard rare features just like it is able to keep rare features due to their relevance for the recommendations. The same can be concluded about the frequent features.

Moreover, our technique requires a low computational time in each step (data pre-processing, classification model training and feature selection) and also a reasonable computational power to yield the results.

To conclude, our proposed technique proved to be a robust feature selection technique able to single out just a small amount of features that best describe the items of the dataset and thus making it possible to apply a more efficient content-based recommendation approach.

Bibliography

[1] C. C. Aggarwal et al. *Recommender systems*. Springer, 2016.

[2] S. Antenucci, S. Boglio, E. Chioso, E. Dervishaj, S. Kang, T. Scarlatti, and M. F. Dacrema. Artist-driven layering and user's behaviour impact on recommendations in a playlist continuation scenario. In *Proceedings of the ACM Recommender Systems Challenge 2018*, page 4. ACM, 2018.

[3] A. Bellogín, I. Cantador, P. Castells, and Á. Ortigosa. Discerning relevant model features in a content-based collaborative recommender system. In *Preference learning*, pages 429–455. Springer, 2010.

[4] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-based systems*, 46:109–132, 2013.

[5] L. Cella, S. Cereda, M. Quadrana, and P. Cremonesi. Deriving item features relevance from past user interactions. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pages 275–279. ACM, 2017.

[6] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[7] E. Christakopoulou and G. Karypis. Hoslim: Higher-order sparse linear method for top-n recommender systems. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 38–49. Springer, 2014.

[8] Y. Deldjoo, M. F. Dacrema, M. G. Constantin, H. Eghbal-Zadeh, S. Cereda, M. Schedl, B. Ionescu, and P. Cremonesi. Movie genome: alleviating new item cold start in movie recommendation. *User Modeling and User-Adapted Interaction*, pages 1–53, 2019.

[9] A. Gunawardana and G. Shani. Evaluating recommender systems. In *Recommender Systems Handbook*, pages 265–308. Springer, 2015.

[10] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[11] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[12] H. Hasija. An effective approach of feature selection for recommender systems using fuzzy c means clustering along with ant colony optimization and neural networks. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2017.

[13] J. Kennedy. Particle swarm optimization. *Encyclopedia of machine learning*, pages 760–766, 2010.

[14] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

[15] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[16] P. Lops, M. De Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In *Recommender systems handbook*, pages 73–105. Springer, 2011.

[17] S.-H. Min and I. Han. Recommender systems using support vector machines. In *International Conference on Web Engineering*, pages 387–393. Springer, 2005.

[18] X. Ning and G. Karypis. Slim: Sparse linear methods for top-n recommender systems. In *2011 IEEE 11th International Conference on Data Mining*, pages 497–506. IEEE, 2011.

[19] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (8):1226–1238, 2005.

[20] L. Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[21] Y. Quanming, W. Mengshuo, J. E. Hugo, G. Isabelle, H. Yi-Qi, L. Yu-Feng, T. Wei-Wei, Y. Qiang, and Y. Yang. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

[22] R. Ronen, N. Koenigstein, E. Ziklik, and N. Nice. Selecting content-based features for collaborative filtering recommenders. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 407–410. ACM, 2013.

[23] D. Wang, Y. Liang, D. Xu, X. Feng, and R. Guan. A content-based recommender system for computer science publications. *Knowledge-Based Systems*, 157:1–9, 2018.

[24] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on Learning Theory*, pages 25–54, 2013.