Bocconi

Bachelor of Science in Economics, Management and Computer Science

Enhancing Collaborative Filtering with Mixture Models: Theory and Applications

Advisor:
Prof. ANTONIO LIJOI

Bachelor of Science thesis by:
TANCREDI DORSI
student ID no 3161375

Academic Year 2023-2024

Bocconi
Università Commerciale
Luigi Bocconi

Contents

1 Introduction

Recommender systems are everywhere in the digital world, influencing the choices we make on different platforms such as e-commerce websites and streaming services. The primary objective of these systems is to suggest items that a user is likely to appreciate. Recommender systems can be categorized in different types, among which one of the most common is collaborative filtering. Collaborative filtering operates under the assumption that users with similar preferences will continue to behave similarly in the future. This method can be further categorized into memory-based and model-based approaches. Memory-based methods leverage direct similarities between users or items to make recommendations, while model-based methods use statistical or machine learning models to understand the underlying structure of data correctly and then compute predictions. However, traditional collaborative filtering methods often face challenges such as data sparsity, and difficulties in capturing the complex nature of user preferences. This thesis investigates the possibility of improving collaborative filtering through the application of mixture models, which offer a promising solution to these limitations. Mixture models are probabilistic models that describe the distribution of data as a combination of multiple distributions. In the thesis we focus first on the theoretical foundations of mixture models and then on their application in the domain of collaborative filtering, both with a memory-based and a model-based approach.

The structure of the thesis is the following: Chapter 2 introduces mixture models, discussing their basic definitions, parameter estimation methods and how to represent them graphically [13][15][14]; Chapter 3 provides a brief overview of recommender systems [1][16]; Chapter 4 focuses on collaborative filtering, detailing both memory-based and model-based approaches [1][4][5]; Chapter 5 transitions from theory to practice, presenting the implementation and evaluation of 4 mixture models, roughly following the order of the models presented in [11], using the MovieLens 100k dataset [8].

2 Mixture models

2.1 Basic definition

As the name suggests, mixture models are used to describe the distribution of a random variable as a combination, or mixture, of multiple distributions. To mathematically represent this concept, we resort to the use of latent variables: in our case the latent variable, denoted as z, indicates the distribution, the mixture component, to which a specific data point belongs. Hence, the usual mathematical notation to write the distribution of a random variable X, which we assume follows a mixture of g distributions, is as follows:

$$p(\mathbf{x}) = \sum_{k=1}^{g} p(z = k|\boldsymbol{\theta})p(\mathbf{x}|z, \boldsymbol{\theta}) \tag{2.1}$$

As we can see, the distribution is made up of a sum of other distributions where

- $p(z = k|\boldsymbol{\theta})$ represents the probability of belonging to the specific distribution k, often referred to as *mixing weights.* We can define $\boldsymbol{\pi}$ the vector of the prior probabilities, where π_k denotes the probability that a data point belongs to component k, given the parameters of the component distributions $\boldsymbol{\theta}$
- $p(\mathbf{x}|z, \boldsymbol{\theta})$ represents the probability distribution of the variable X given the belonging to the specific class z, also often referred to as *component distribution* of the mixture. While it is not always the case, we assume that the component distributions are all parametric and define as $\boldsymbol{\theta}$ the vector containing all the parameters of the component distributions

2.2 Parameter estimation via maximum likelihood

Assuming we want to model a set of data $\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_n$ with a mixture of g distributions, we need to estimate two main sets of parameters:

- the vector of mixing weights $\boldsymbol{\pi}$, i.e. the prior probabilities of belonging to a cluster

- the vector of parameters of each of the component distributions $\boldsymbol{\theta}$. For clarity, from now on we will call $\boldsymbol{\theta}_k$ the vector of parameters of distribution k and $\boldsymbol{\theta}$ the vector comprising all the vectors $\boldsymbol{\theta}_k$.

Some comments can also be made regarding finding the optimal number of mixture components, which we assume here. In general, it is advisable to start with a small number of clusters (which are also faster to compute) and verify every time whether some peculiarities of our data are not correctly represented [6]. There are several measures that can be used to evaluate the goodness of a fit of a mixture model. Some examples are information criterias, such as the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC), as well as approximate likelihood ratio tests, like the bootstrap likelihood ratio test [7].

The most direct approach to estimate the unknown parameters is maximum likelihood estimation in the canonical way:

$$\frac{\partial L(\boldsymbol{\Psi})}{\partial \boldsymbol{\Psi}} = \mathbf{0} \tag{2.2}$$

where

$$\boldsymbol{\Psi} = (\boldsymbol{\pi}^T, \boldsymbol{\theta}^T)^T \tag{2.3}$$

and

$$L(\boldsymbol{\Psi}) = \prod_{i=1}^{n} p(\mathbf{x}_i|\boldsymbol{\theta}) \tag{2.4}$$

By taking the logarithm to facilitate the optimization of the function we get

$$\log L(\boldsymbol{\Psi}) = \sum_{i=1}^{n} \log p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{g} p(z = k|\boldsymbol{\theta}) p(\mathbf{x}_i|z, \boldsymbol{\theta}_k) \right) \tag{2.5}$$

From the maximization of the log-likelihood function we can derive, through some manipulations, the following conditions that the MLE $\hat{\boldsymbol{\Psi}}$ must satisfy [12]:

$$\hat{\pi_k} = \sum_{i=1}^{n} \frac{\tau_k(\mathbf{x}_i|\hat{\boldsymbol{\theta}})}{n} \quad (k = 1, \ldots, g) \tag{2.6}$$

and

$$\sum_{k=1}^{g} \sum_{i=1}^{n} \tau_k(\mathbf{x}_i|\hat{\boldsymbol{\theta}}) \partial \log p(\mathbf{x}_i|\hat{\boldsymbol{\theta}}_{\boldsymbol{k}}) / \partial \boldsymbol{\theta} = \mathbf{0} \tag{2.7}$$

$\tau_k(\mathbf{x}_i)$ indicates the probability that a data point $\mathbf{x}_i$ belongs to the k^{th} component distribution and we can easily compute it through Bayes' theorem:

$$\tau_k(\mathbf{x}_i|\hat{\boldsymbol{\theta}}) = p(z = k|\mathbf{x}_i, \hat{\boldsymbol{\theta}}) = \frac{p(z = k|\hat{\boldsymbol{\theta}}) p(\mathbf{x}_i|z, \hat{\boldsymbol{\theta}})}{\sum_{k'=1}^{K} p(z = k'|\hat{\boldsymbol{\theta}}) p(\mathbf{x}_i|z, \hat{\boldsymbol{\theta}})} \tag{2.8}$$

While the direct optimization of the observed data log-likelihood proves to be challenging for several reasons, the results in (2.6) and (2.7) indicate an iterative approach to the maximization problem [13]. This solution is known as the Expectation-Maximization (EM) algorithm.

2.3 The EM algorithm

The EM algorithm circumvents the problem of maximizing the log-likelihood of the observed data by first defining the complete data log-likelihood, assuming we have observed z_{ik}, i.e. the latent variable taking value 1 if point i belongs to the component k and 0 otherwise. Hence, we consider

$$\log L_c(\boldsymbol{\Psi}) = \sum_{i=1}^{n} \log p(\mathbf{x}_i, \mathbf{z}_i|\boldsymbol{\theta}) = \sum_{k=1}^{g} \sum_{i=1}^{n} z_{ik} (\log p(z = k|\boldsymbol{\theta}) + \log p(\mathbf{x}_i|z, \boldsymbol{\theta}_k)) \tag{2.9}$$

We now initialize in some way all the parameters of the vector $\boldsymbol{\Psi}$, which we define as $\boldsymbol{\Psi}^{(0)}$. We then define the auxiliary function Q, which is the expected value of the complete data log-likelihood, given the data we observed and employing the parameters $\boldsymbol{\Psi}^{(0)}$:

$$Q(\boldsymbol{\Psi}, \boldsymbol{\Psi}^{(0)}) = \mathbb{E}_{\boldsymbol{\Psi}^{(0)}}[\log L_c(\boldsymbol{\Psi})|\mathbf{x}] \tag{2.10}$$

Due to the linearity of the log-likelihood in z we get

$$Q(\mathbf{\Psi}, \mathbf{\Psi}^{(0)}) = \mathbb{E}_{\mathbf{\Psi}^{(0)}} \left[\sum_{k=1}^{g} \sum_{i=1}^{n} z_{ik} (\log p(z = k|\boldsymbol{\theta}^{(0)}) + \log p(\mathbf{x}_i|z, \boldsymbol{\theta}_k^{(0)})) \right] \tag{2.11}$$

$$= \sum_{k=1}^{g} \sum_{i=1}^{n} \mathbb{E}_{\mathbf{\Psi}^{(0)}}[z_{ik}] (\log p(z = k|\boldsymbol{\theta}^{(0)}) + \log p(\mathbf{x}_i|z, \boldsymbol{\theta}_k^{(0)})) \tag{2.12}$$

Hence, the computation of the expected value of the whole log-likelihood boils down to the computation of $\mathbb{E}_{\mathbf{\Psi}^{(0)}}[z_{ik}]$ for every point i and component k, given the observed data and using the current parameters $\mathbf{\Psi}^{(0)}$. Being z_{ik} a binary variable taking either value 0 or 1, its expected value is the probability that point i belongs to component k, which we have already defined in (2.8) as $\tau_k(\mathbf{x}_i)$. Therefore, the function Q looks like this:

$$Q(\mathbf{\Psi}, \mathbf{\Psi}^{(0)}) = \sum_{k=1}^{g} \sum_{i=1}^{n} \tau_k(\mathbf{x}_i|\boldsymbol{\theta}^{(0)}) (\log p(z = k|\boldsymbol{\theta}^{(0)}) + \log p(\mathbf{x}_i|z = k, \boldsymbol{\theta}_k^{(0)})) \tag{2.13}$$

Once we have computed the Q function, we proceed to the second step of the algorithm, i.e. the maximization step. In this step we simply update the current parameters $\mathbf{\Psi}^{(0)}$ with the parameters maximizing the Q function

$$\mathbf{\Psi}^{(1)} = \arg\max_{\mathbf{\Psi}} Q(\mathbf{\Psi}, \mathbf{\Psi}^{(0)}) \tag{2.14}$$

We then perform the expectation step and the maximization step iteratively, updating everytime the parameters. We keep performing this operation until the algorithm converges, i.e. the log-likelihood improves by less than a predefined threshold.

Once we get the estimates of the unknown parameters of the model, we can exploit them for one of the most common uses of mixture models: clustering [15]. The idea behind this application of mixture models is rather simple. We use the probability of belonging to a certain cluster k for each point i, namely $\tau_k(\mathbf{x}_i|\boldsymbol{\theta})$, to perform a soft assignment of each point to a cluster. This approach is useful because it accommodates the inherent uncertainty and variability in real-world data. Differently from traditional clustering methods such as k-means, which allocate each data point exclusively to one cluster, mixture models provide a probabilistic assignment, giving the likelihood that a data point belongs to each cluster. Furthermore, mixture models are flexible and can handle clusters of different shapes and sizes, making them suitable for a wide range of applications. In this thesis we focus on one of them: collaborative filtering, a common technique used in recommender systems. We exploit mixture models in two ways: firstly, we perform a probabilistic clustering which helps us in computing the final target prediction; secondly, we use the results of the model to perform a hard clustering, thus assigning just one cluster to each data point, to try another prediction approach.

2.4 Probabilistic graphical models

When dealing with complex joint distributions, as it is often the case in mixture models, a key tool that we can use to represent them are probabilistic graphical models. There exist two types of probabilistic graphical models: directed and undirected ones; we focus only on the former ones, also known as Bayesian networks. These models exploit graphs to easily explain the relationship between variables in a distribution. More specifically, they manage to encode the conditional independence structure of a joint distribution. We say two random variables X and Y are conditionally independent given a third random variable Z when

$$p(x, y|z) = p(x|z)p(y|z) \tag{2.15}$$

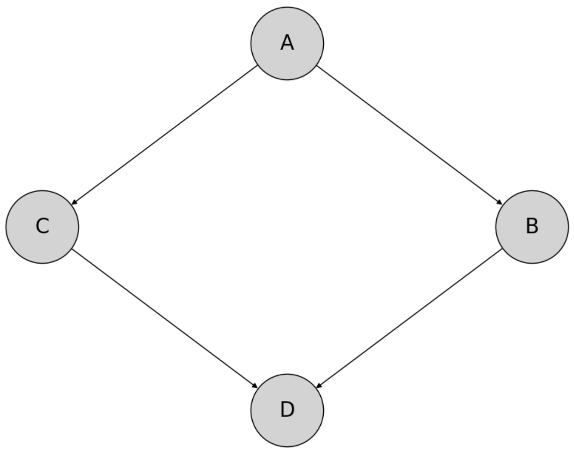
or, equivalently,

$$p(x|y, z) = p(x|z) \tag{2.16}$$

What these formulas mean is that, once we know Z, knowing Y does not provide any additional information regarding X, and viceversa.

In a graphical model, each node of the graph corresponds to a random variable. The presence and absence of edges between the nodes represent the relationships of conditional independence. More specifically, each variable in a Bayesian network is independent of its non-descendants given its parent nodes [14]. Let's use an example to explain the concept better.

Let's consider the random variables A, B, C and D, whose joint probability can be represented in the following way:



The chain rule of probability affirms that we can write the joint distribution as

$$p(a,b,c,d) = p(a)p(b|a)p(c|a,b)p(d|a,b,c) \tag{2.17}$$

By looking at its graphical representation we can simplify the above formulation. In fact, C is conditionally independent of B given its parent A, hence we can reduce it to $p(c|a)$. Furthermore, D is conditionally independent of A given its parents B and C, hence it can be reduced to $p(d|b,c)$. We can therefore rewrite the joint distribution as

$$p(a,b,c,d) = p(a)p(b|a)p(c|a)p(d|b,c) \tag{2.18}$$

In Chapter 5 we describe every model also with its graphical representation, which makes the understanding easier and more straight-forward.

3 Recommender systems

3.1 Introduction

In our daily life we often encounter objects that heavily rely on the use of recommendation algorithms. It is enough to think that every time we buy an item on Amazon, we listen to a song on Spotify or we watch a movie on Netflix we passively observe the results of these systems. Recommender systems have in fact a vast range of applications, which goes from e-commerce websites to streaming services, and from social networks to news websites. The main goal of these algorithms is to recommend to a user an item he will like, such as a product, a song or a movie. From a commercial standpoint, recommender systems play a fundamental role for several reasons [16]:

- they increase sales and revenues, by suggesting new items that the user would not have otherwise purchased
- they enhance the overall user experience, as they allow for a more personalized and tailored experience, thus also increasing user satisfaction and user fidelity
- they enable the sale of diverse items by introducing users to products and genres they may not have explored independently

3.2 Goals of recommender systems

In the first place, it is important to define properly the goal of a recommender system. The problem itself can actually be stated in two different ways [5][1]:

- A prediction problem, where the goal of the system is to predict the rating for each (user, item) pair. Therefore, we train the model on the available values, and then try to infer a prediction for the missing ones.
- A ranking problem, where instead of trying to predict all of the possible user-item combinations, we simply try to find for each user the top-k items, or viceversa.

In practice, these definitions often represent two phases of the same problem: first, predicting all possible missing entries, and subsequently ranking the best possible options based on the predictions.

3.3 Types of recommender systems

Recommender systems are usually grouped into three main categories: content-based filtering, collaborative filtering and hybrid approach.

The key idea behind content-based filtering is that the system uses the item's features to find similar items to recommend to a user. Hence, these systems analyze a user's past behavior and, based on the characteristics of the items the user interacted with and rated highly, suggest similar items. So, for example, if I watched and really liked *James Bond: Skyfall*, a content-based recommender system will suggest me some movies with analogous characteristics, such as *Mission - Impossible: Rogue Nation* or *Jason Bourne: The Bourne Identity.*

Collaborative filtering (CF) recommender systems, differently from content-based filtering systems, rely on the ratings of users to make predictions. The main idea behind these systems is that if a user has similar tastes to other users, he is likely to appreciate items that the users similar to him have appreciated. A deeper look into collaborative filtering will be given in the next section.

Recommender systems based on the hybrid approach combine elements from collaborative filtering and elements from content-based filtering to find items to suggest. In this way, they manage to leverage the advantages and mitigate the disadvantages of each of the two types. This comes at the cost of a higher complexity in their development and implementation, a much more important computational cost and some difficulties in finding the right equilibrium between the two approaches.

4 Collaborative filtering

4.1 Introduction

As previously introduced, a collaborative filtering recommender system uses the ratings provided by users to identify similar users or similar items and recommend items accordingly. The key element in collaborative filtering is represented by a tuple, representing the user-item interaction, composed of (user, item, rating). While there are different types of ratings, this work focuses only on a 5-star rating scale. The standard way to represent data in collaborative filtering is the user-item matrix: on the rows of this matrix we find the users, on the columns we find the items, while the entries of the matrix represent the rating given by a user to an item. An example of user-item matrix with 4 items and 5 users looks something like this:

	Item1	Item2	Item3	Item4
User1	1		2	3
User2	4	5	1	3
User3		5		5
User4	3		3	4
User5	5	4		5

In collaborative filtering we have a distinction between two main methods:

- Memory-based methods
- Model-based methods

4.2 Memory-based methods

Memory-based collaborative filtering, also known as neighborhood-based collaborative filtering, relies on the concept of similarity among users or items to compute the final prediction. In fact, once chosen a suitable similarity measure, similarity scores are computed. These scores are later used as input in a prediction function. The similarity scores can either be predicted between the users or between the items. The two main approaches of memory-based CF derive from this distinction:

- User-based collaborative filtering: the algorithm identifies similar users, based on their rating patterns. Hence, it computes a similarity measure between the rows of the user-item matrix. Once found the most similar users for each user, the system suggests to the user items that similar users have liked.
- Item-based collaborative filtering: the algorithm determines similar items, based on how they are rated by users. Hence, it computes a similarity measure between the columns of the user-item matrix. Once found similar items, the system suggests items similar to what the user has liked in the past. This approach resembles in some way the content-based approach, but similarity between items is only established from the user preference patterns and not from any other external feature.

4.2.1 Similarity measures

As we have seen, similarity is a key concept in collaborative filtering because these systems rely heavily on the resemblance between users or items to make recommendations. A natural question that therefore arises is regarding which similarity measures are the best to use. During the years several measures have been proposed, each one with its advantages and disadvantages. We list below 3 of the most popular ones for user-based and item-based collaborative filtering [1].
One of the most common measures is the renowned Pearson's correlation coefficient. This famous metric is often used to measure the strength of a linear relationship between two variables and is widely used in statistics. Its values range from -1, representing a strong negative correlation, to 1, representing a strong positive correlation. In the context of user-based CF its formula is

$$PCC(u,v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \overline{r}_u)(r_{vi} - \overline{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \overline{r}_v)^2}} \tag{4.1}$$

where r_{ui} and r_{vi} are the ratings of users u and v to item i, I_{uv} represents the set of items rated by both users u and v, and $\overline{r}_u$ and $\overline{r}_v$ denote the means of the ratings of users u and v respectively for items in I_{uv}.

The equivalent formula for item-based CF is

$$PCC(i,j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \overline{r}_i)(r_{uj} - \overline{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \overline{r}_i)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \overline{r}_j)^2}} \tag{4.2}$$

where r_{ui} and r_{uj} are the ratings of user u to items i and j, U_{ij} denotes the set of users who have rated both items i and j, and $\overline{r}_i$ and $\overline{r}_j$ denote the average rating of items i and j respectively for users in U_{ij}.

Another frequent similarity measure is the cosine similarity. This measure exploits the cosine of the angle between two vectors to determine their similarity. Similarly to Pearson's correlation coefficient, values for cosine similarity range from -1, where we have two opposite vectors, to 1, where we have two proportional vectors. In the context of user-based CF its formula is

$$Cosine(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{u \in I_u} r_{ui}^2} \sqrt{\sum_{v \in I_v} r_{vi}^2}} \tag{4.3}$$

where r_{ui} and r_{vi} are the ratings of item i given by user u and v respectively, I_u and I_v denote the set of items rated by users u and v respectively and I_{uv} the set of items rated by both users u and v. The equivalent formula for item-based CF is

$$Cosine(i,j) = \frac{\sum_{u \in U_{ij}} r_{ui} r_{uj}}{\sqrt{\sum_{u \in U_i} r_{ui}^2} \sqrt{\sum_{u \in U_j} r_{uj}^2}} \tag{4.4}$$

where r_{ui} and r_{uj} are the ratings of items i and j given by user u, U_i and U_j denote the set of users who rated items i and j respectively and U_{ij} the set of users who rated both items i and j.

Many times instead of the simple cosine similarity it has been used the adjusted cosine similarity where ratings are mean-centralized by subtracting the average rating of the item in user-based methods and the average rating of the user in item-based methods. Hence their formulas are

$$AdjCosine(u, v) = \frac{\sum_{i \in I_{uv}} (r_{ui} - \overline{r}_i)(r_{vi} - \overline{r}_i)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \overline{r}_i)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \overline{r}_i)^2}} \tag{4.5}$$

and

$$AdjCosine(i, j) = \frac{\sum_{u \in U_{ij}} (r_{ui} - \overline{r}_u)(r_{uj} - \overline{r}_u)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \overline{r}_u)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \overline{r}_u)^2}} \tag{4.6}$$

4.2.2 The prediction function

Once a similarity measure is established, the next step is to apply a prediction function. In user-based CF the most common formula used to generate prediction takes a weighted average of the neighboring users' ratings, where the similarity score is used as weight:

$$\hat{r}_{ui} = \overline{r}_u + \frac{\sum_{v \in N} sim(u, v)(r_{vi} - \overline{r}_v)}{\sum_{v \in N} |sim(u, v)|} \tag{4.7}$$

where N denotes the set of neighboring users and *sim* the similarity measure being used. The mean-centering process proves to be necessary in order to avoid that different rating scales among users influence incorrectly the final prediction [1]. In fact, some users may rate on average negatively, while some other users may rate on average with really high scores. With mean-centering we ensure that these individual rating scales are accounted for, resulting in more accurate and unbiased predictions.

In item-based CF, we use the same approach of computing a weighted average of the neighboring items using similarity scores as weights, without any mean-centering:

$$\hat{r}_{ui} = \frac{\sum_{j \in S} sim(i, j) r_{uj}}{\sum_{j \in S} |sim(i, j)|} \tag{4.8}$$

where S denotes the set of neighbouring items and *sim* the similarity measure being used. How many neighbours should we pick in this computation? In some systems all the possible users or items are considered neighbours. In some other systems the number of neighbours is selected based on a threshold of the similarity measure, or based on a predetermined number of top-k most similar elements [1].

4.3 Model-based methods

Model-based collaborative filtering leverages statistical and machine learning models to predict user preferences. As in traditional machine learning, model-based approaches create a compact representation of the user-item interaction matrix. These models learn underlying patterns and latent features from the data, capturing the complex relationships between users and items. Thanks to this aspect, model-based methods have usually higher accuracy than memory-based methods. They are also, in general, more scalable to large datasets, because, once the model is trained, making predictions is more computationally efficient than having to compute similarity scores. On the other hand, they suffer from a much higher complexity in their implementation and a harder interpretability. From a practical point of view, the implementation of model-based methods depends on the model chosen and can widely vary due to that.

4.4 Mixture models for collaborative filtering

Mixture models applied to collaborative filtering have been extensively studied due to the several advantages they offer. Mixture models, in fact, can effectively address the issue of data sparsity, a common challenge in collaborative filtering [9] [11]. In fact, in most collaborative filtering settings users only rate a small portion of the items available (just consider the number of movies you rate on Netflix or similar platforms compared to the whole catalog); this leads to a user-item matrix which is mostly empty. The use of latent variables to capture underlying preference patterns helps in making accurate predictions even with limited data. This approach also reduces the number of parameters needed in the model and therefore minimizes overfitting. Furthermore, mixture models provide a flexible framework that can represent overlapping and complex user preferences [11].

They allow users to belong to multiple clusters, capturing the varied nature of individual preferences. This flexibility is beneficial for making personalized recommendations that reflect more precisely the interests of users. We can also exploit mixture models in memory-based collaborative filtering to perform hard clustering before computing similarity scores. Once each point is assigned to a cluster, we compute the similarity scores for an item or a user by only considering the items or users in the same cluster. This method makes the similarity computation much more efficient by reducing the number of similarities to compute for each item or user, at the cost of some accuracy in the final predictions [1].

5 From theory to practice: applying the mixture models

We now look at 4 models of increasing complexity that can be employed in the context of collaborative filtering. After reviewing their theoretical properties, we implement them from scratch using Python. We train and test the models on the MovieLens 100k Dataset [8]. This dataset was collected by the GroupLens Research Project at the University of Minnesota and comprises 100000 ratings from 943 users on 1682 movies. Each rating can take integer value from 1 to 5. The dataset also contains some basic demographic information about the users, but we ignore it. The dataset is split into a training and test set, where the test set contains exactly 10 ratings for each user (hence a total of 9430 ratings) and the training set the remaining 90570 ratings. For the evaluation of the models the most important aspect is the accuracy, which we measure through the mean absolute error (MAE):

$$MAE = \frac{\sum_{r \in Test} |\hat{r} - r|}{N_{Test}} \tag{5.1}$$

where N_{Test} is the total number of ratings in the testing set. We also set two benchmark MAEs to have a clearer understanding of the performance of the different models. The first benchmark MAE is computed by applying a memory user-based approach, where similarities are computed using cosine similarity and scores are predicted with formula (4.7); predictions are computed considering only the top 10 most similar users. We refer to this model in the result tables as "User-Based No Cluster" and its correspondent MAE is 0.770. The second benchmark MAE is computed by employing a memory item-based approach, where similarities are still computed with cosine similarity and we use formula (4.8) for predictions; when making predictions, only the top 10 most similar items are considered. We refer to this model as "Item-Based No Cluster" and its MAE is 0.780.

Some other components are also important in the evaluation of a recommender system, such as novelty (proposing items that the user has never previously seen), serendipity (proposing unexpected items) and diversity (proposing a varied list of items) [1]; nevertheless, they are often difficult to quantify and to evaluate correctly, therefore we ignore them in our analysis.

5.1 Cluster model

5.1.1 Theory

The first model we look at is the cluster model presented in the context of collaborative filtering by Breese, Heckerman and Cadie [3]. The main assumption behind the model is that users can be categorized based on some common preferences and rating patterns. Each user in the dataset belongs to a cluster and, given the cluster, the ratings are conditionally independent. We can write the probability model as follows:

$$p(z = k, r_1, \dots, r_n|\boldsymbol{\theta}) = p(z = k|\boldsymbol{\theta}) \prod_{i=1}^{n} p(r_i|z = k, \boldsymbol{\theta}) \tag{5.2}$$

where z is the unobserved class variable, representing the users' clusters, r_i is the rating value to item i and $\boldsymbol{\theta}$ is the vector of parameters of the component distribution; conditionally on $z = k$ and $\boldsymbol{\theta}$, r_i is from a multinomial distribution on the rating values (from 1 to 5) and $p(z = k|\boldsymbol{\theta})$ is a multinomial distribution on the user classes.

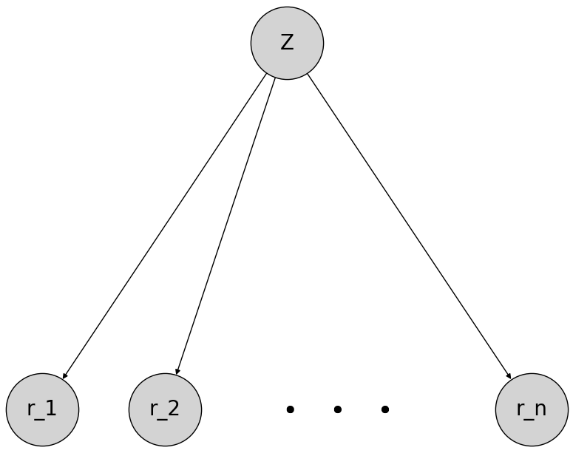


Figure 1: Graphical representation of the cluster model

From this formula we can derive the probability distribution of ratings to a single item i

$$p(r_i|\boldsymbol{\theta}) = \sum_{k=1}^{g} p(z=k|\boldsymbol{\theta})p(r_i|z=k,\boldsymbol{\theta}) \tag{5.3}$$

where g is the total number of clusters.

The parameters we need to estimate are the prior probabilities $p(z|\boldsymbol{\theta})$ and the parameters of the component distributions $\boldsymbol{\theta}$, i.e. the probability of each rating from 1 to 5, given each one of the classes. In order to estimate these parameters of the model we use the EM algorithm. As we have seen in section 2.3, the first task of the E step is to compute the probabilities of belonging to each cluster, given the observed data and the current parameters. Given that we are assuming to cluster users, we compute the probability that a user u belongs to cluster k, given its ratings, as follows:

$$\tau_k(\mathbf{r}_u|\boldsymbol{\theta}) = p(z=k|\mathbf{r}_u,\boldsymbol{\theta}) = \frac{p(z=k|\boldsymbol{\theta})\prod_{i\in U} p(r_{ui}|z=k,\boldsymbol{\theta})}{\sum_{k'=1}^{g} p(z=k'|\boldsymbol{\theta})\prod_{i\in U} p(r_{ui}|z=k',\boldsymbol{\theta})} \tag{5.4}$$

where U is the set of items rated by user u.

From a practical standpoint, the high number of probability values that have to be multiplied between them pose an important issue: underflow, which happens when a number is smaller than what the computer program is able to actually represent. Especially in the E step, the product at the numerator becomes so small that Python rounds it to 0. This is actually a common problem in naive Bayes classifiers and can be solved using the LogSumExp function. Instead of computing the numerator as it is, we compute its logarithm: this allows us to work with larger negative numbers and additions, both elements that are easier to deal with. Once we have computed the logarithm of the numerator for each one of the components, which we will call x_1, x_2, ..., x_g, we pass the vector of results in the LSE function

$$LSE(x_1, x_2, \dots, x_g) = \log\left(\sum_{i=1}^{g} e^{x_i}\right) \tag{5.5}$$

But, given that when exponentiating we could still incur in harmful underflow, we perform the following shift:

$$LSE(x_1, x_2, \ldots, x_g) = \log\left(\sum_{i=1}^{g} e^{x_i}\right) = \log\left(\sum_{i=1}^{g} e^{a} e^{x_i - a}\right) = \log\left(e^{a} \sum_{i=1}^{g} e^{x_i - a}\right) \tag{5.6}$$

where $a = \max\{x_1, x_2, \ldots, x_g\}$. Finally,

$$LSE(x_1, x_2, \ldots, x_g) = a + \log\left(\sum_{i=1}^{g} e^{x_i - a}\right) \tag{5.7}$$

This procedure is often employed in these situations because it ensures numerical stability by preventing the exponentiation of very large or very small numbers [2]. Once this computation is done, we have to perform the last step to normalize the probabilities, which consists of

$$\tau_k(\mathbf{r}_u|\boldsymbol{\theta}) = \exp\left(x_j - \left(a + \log\left(\sum_{i=1}^{g} e^{x_i - a}\right)\right)\right)$$

We now need to update the parameters in the maximization step. The updates for this step are

$$p(z = k|\boldsymbol{\theta}) = \frac{\sum_u \tau_k(\mathbf{r}_u|\boldsymbol{\theta})}{N} \tag{5.8}$$

where N is the total number of users and

$$p(r_i = r|z = k, \boldsymbol{\theta}) = \frac{\sum_u \tau_k(\mathbf{r}_u|\boldsymbol{\theta}) \mathbb{1}(r_{ui} = r)}{\sum_u \tau_k(\mathbf{r}_u|\boldsymbol{\theta})} \tag{5.9}$$

where $\mathbb{1}$ is an indicator function taking value 1 when the condition in the parenthesis is satisfied and 0 otherwise. The only thing left out to do is to predict the final ratings. We do so by multiplying the probabilities of belonging to a cluster given the observed ratings and the expected value of the rating given the cluster

$$\hat{r}_{ui} = \sum_{k=1}^{g} \tau_k(\mathbf{r}_u|\hat{\boldsymbol{\theta}}) \sum_{r=1}^{5} p(r_{ui} = r|z = k, \hat{\boldsymbol{\theta}}) \cdot r \tag{5.10}$$

Besides the prediction through formula (5.10), we also employ the probabilities $\tau_k(\mathbf{r}_u|\hat{\boldsymbol{\theta}})$

computed in the E step to implement a memory-based approach. We assign to each user the cluster to which it has the highest probability of belonging; we then compute similarities through cosine similarity, but only among users of the same cluster. We then compute the prediction by picking the top 10 most similar users of the same cluster and applying formula (4.7), as done for the benchmark user-based model. We refer to this model as "User-Based Cluster".

5.1.2 Results

We run the model 9 times, with the number of classes increasing each time from 2 up to 10. For the final evaluations we consider the model with 6 classes. Each time we initialize the probabilities $p(z = k|\boldsymbol{\theta})$ and $p(r_i = r|z = k, \boldsymbol{\theta})$ randomly from a Dirichlet distribution. We verify that the EM algorithm converges by checking the values of the log-likelihood:

$$L(\boldsymbol{\Psi}) = \sum_{(i,u)\in Train} \log\left(\sum_{k=1}^{g} p(z = k|\boldsymbol{\theta})p(r_{ui}|z = k, \boldsymbol{\theta})\right) \tag{5.11}$$

where $Train$ is the set of observations in the training set.

As we could imagine, the final value of the log-likelihood, that we get once the EM algorithm converges, increases with the number of classes. This is due to the fact that the higher the number of classes, the better the model can fit the data. Furthermore, the EM algorithm usually converged in around 20 steps.

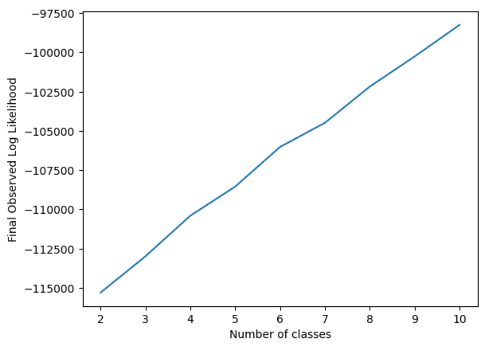


Figure 2: Final value of the log-likelihood for each model

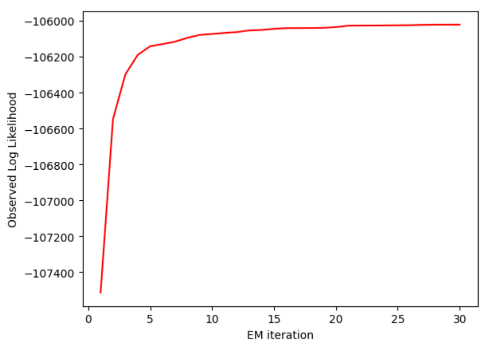


Figure 3: Log-likelihood at each iteration of the EM algorithm for the model with 6 classes

The MAE of the models are summarized in the table below:

Model	MAE
Cluster Model	1.054
User-Based Cluster	0.791
User-Based No Cluster	0.770
Item-Based No Cluster	0.780

The results show that the Cluster Model does not seem to perform too well on our data, having quite disappointing results compared to a memory-based approach. On the other hand, the results of the User-Based Cluster model are quite interesting, showing that even with a split in clusters and therefore a reduction in the training time, the prediction only suffers by a 0.02 increase in the MAE.

5.2 Aspect Model

5.2.1 Theory

The second model we look at is the aspect model by Hofmann and Puzicha [9]. This model introduces one important novelty compared to the previous one: it does not model only ratings, but also describes the probability distributions of users and items. More specifically, it assigns a latent variable z to each observation of user and item (u, i). The user variable u and the item variable i are conditionally independent given the latent class z. The joint probability can in fact be written as

$$p(i, u) = \sum_z p(z)p(i|z)p(u|z) \tag{5.12}$$

where $p(z)$ is a multinomial on the latent classes, $p(i|z)$ is a multinomial on the items, given the latent class, and $p(u|z)$ is a multinomial on the users, given the latent class. Considering the implementation on the MovieLens dataset, we can interpret $p(i|z)$ as the probability that the movie i falls under the genre z, while $p(u|z)$ as the probability that the user u prefers or is interested in the genre z. Hence, z works as a mediator linking users and items. Each latent class z represents an aspect that captures a typical pattern of user preferences and item characteristics. We can also rewrite the probability distribution in (5.12) as

$$p(i,u) = p(i)p(u|i) \tag{5.13}$$

where

$$p(i|u) = \sum_z p(z|u)p(i|z) \tag{5.14}$$

This reformulation is particularly interesting because it clearly shows that the conditional probability in (5.14) is a convex combination of typical preference patterns $p(i|z)$. Furthermore, we can also reach a dual formulation of $p(i,u)$ by inverting i and u in formula (5.13) and, consequently, defining $p(u|i)$ instead of $p(i|u)$; similarly, this new formulation shows how $p(u|i)$ is a convex combination of probabilities $p(u|z)$. The key reasoning behind this model is that users can have multiple interests, and items can appeal to users for various reasons; hence, the aspect model allows each user and each item to be associated with multiple latent classes, capturing the varied nature of preferences and item characteristics. This avoids rigid groupings of users or items into fixed clusters. This model is later extended to include the distribution of the ratings, which are of course fundamental for the final predictions. The paper offers many possible distributions to incorporate ratings; the one we use in this paper models the joint distribution in the following way:

$$p(i,u,r) = \sum_z p(z)p(u|z)p(i|z)p(r|z,i) \tag{5.15}$$

As we can see, we assume that the final rating depends directly from the item and the latent class, but it is conditionally independent of the user given the class.

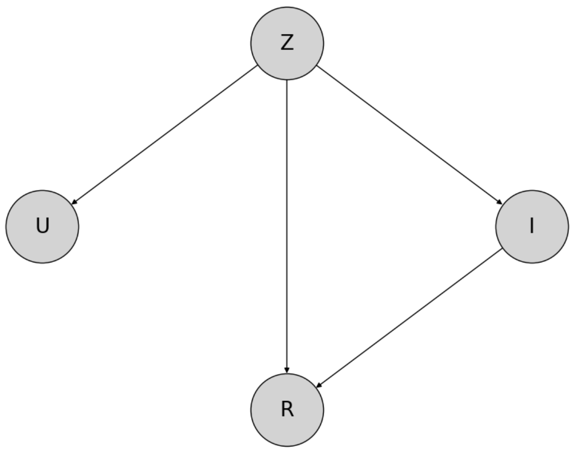


Figure 4: Graphical representation of the aspect model

We now define the steps of the EM algorithm to compute the parameters. The probabilities of belonging to each latent class for each observed item, user and rating combination are computed with the following formula:

$$p(z|i,u,r) = \frac{p(z)p(u|z)p(i|z)p(r|z,i)}{\sum_{z'} p(z')p(u|z')p(i|z')p(r|z',i)} \tag{5.16}$$

The M step updates are computed as follows:

$$p(z) = \frac{\sum_{i,u,r} p(z|i,u,r)}{N} \tag{5.17}$$

$$p(u|z) = \frac{\sum_{i,u',r} p(z|i,u',r)\mathbb{1}(u'=u)}{N \cdot p(z)} \tag{5.18}$$

$$p(i|z) = \frac{\sum_{i',u,r} p(z|i',u,r)\mathbb{1}(i'=i)}{N \cdot p(z)} \tag{5.19}$$

$$p(r|z,i) = \frac{\sum_{i',u,r'} p(z|i',u,r')\mathbb{1}(i'=i)\mathbb{1}(r'=r)}{\sum_{i',u,r'} p(z|i',u,r')\mathbb{1}(i'=i)} \tag{5.20}$$

where N is the total number of observations. Lastly, we compute the predictions for the model:

$$\hat{r}_{ui} = \sum_{r=1}^{5} r \frac{p(i,u,r)}{\sum_{r'} p(i,u,r')} \tag{5.21}$$

5.2.2 Results

As for the previous model, we run it 9 times with classes going from 2 to 10. For the final analysis we use the model with 6 classes. We initialize each time the probabilities $p(z)$, $p(u|z)$, $p(i|z)$ and $p(r|z,i)$ randomly from a Dirichlet distribution. We check the progress of the model through the log-likelihood:

$$L(\mathbf{\Psi}) = \sum_{(i,u,r)\in Train} \log\left(\sum_z p(z)p(u|z)p(i|z)p(r|z,i)\right) \tag{5.22}$$

where $Train$ is the set of all observations in the training set.

As expected, the final log-likelihood of the models increases with the number of classes. The EM algorithm takes way more iterations than the previous model before stabilizing around a value.

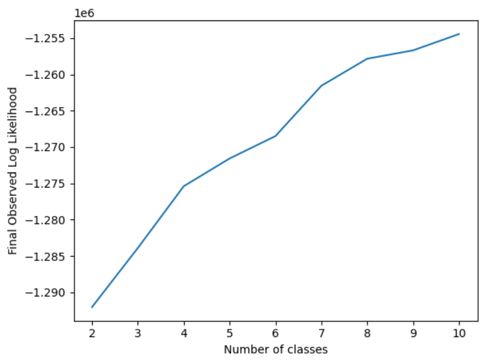


Figure 5: Final value of the log-likelihood for each model

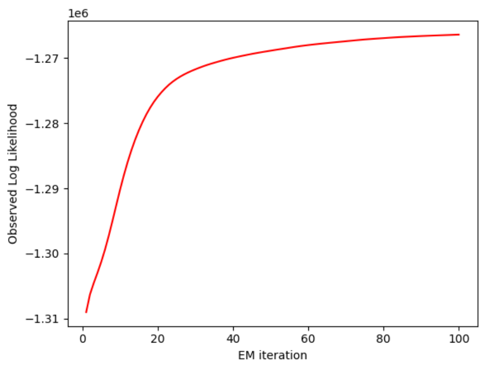


Figure 6: Log-likelihood at each iteration of the EM algorithm for the model with 6 classes

The MAE of the model and the previously defined benchmarks are summarized below:

Model	MAE
Aspect Model	1.104
User-Based No Cluster	0.770
Item-Based No Cluster	0.780

Unfortunately, the results of the aspect model are even worse than the ones of the cluster model and do not even go close to the benchmark we defined earlier. While the additional specifications of the distributions of items and users are probably beneficial, the model should be more structured to compute accurate predictions. We see a more detailed approach in the next model.

5.3 Flexible Mixture Model

5.3.1 Theory

The third model we look at is the flexible mixture model (FMM), proposed by Si and Jin [17]. This model aims at improving the aspect model by introducing one important modification: instead of using just one class variable z linking users and items together, in the FMM we treat them separately. We have therefore two different latent variables: z_u, used to cluster users, and z_i, used to cluster items.

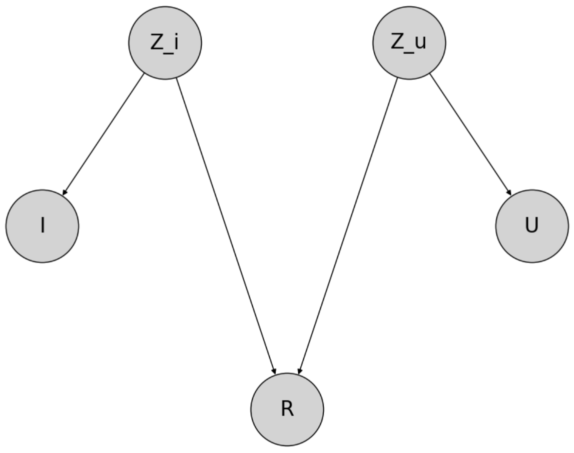


Figure 7: Graphical representation of the flexible mixture model

As we can also see from the graphical representation of the model, the distribution of the users and the items depend from their respective latent class, and, furthermore, these latent classes also influence the final rating. The final ratings are conditionally independent of the users and the items, given their respective latent classes. The joint distribution of items, users and ratings is

$$p(i, u, r) = \sum_{z_i, z_u} p(z_i)p(z_u)p(i|z_i)p(u|z_u)p(r|z_i, z_u) \tag{5.23}$$

where $p(z_i)$ and $p(z_u)$ are both multinomial distributions across respectively item and user classes; conditionally on the item class z_i, i is from a multinomial distribution on the specific items; similarly, conditionally on the user class z_u, u is from a multinomial distribution on the specific users; finally, $p(r|z_i, z_u)$ is a multinomial distribution on the rating values, given the two latent classes. As usual, we employ the EM algorithm to train the model. In the E step we compute the probabilities of belonging to the latent class given each observed item, user and rating combination but, since we have two latent variables, we compute their joint distribution

$$p(z_i, z_u|i, u, r) = \frac{p(z_i)p(z_u)p(i|z_i)p(u|z_u)p(r|z_i, z_u)}{\sum_{z_i', z_u'} p(z_i')p(z_u')p(i|z_i')p(u|z_u')p(r|z_i', z_u')} \tag{5.24}$$

Once completed the E step, we now need to maximize with respect to all of the parameters. We do so with the following formulas:

$$p(z_i) = \frac{\sum_{i,u,r} \sum_{z_u} p(z_i, z_u|i, u, r)}{N} \tag{5.25}$$

$$p(z_u) = \frac{\sum_{i,u,r} \sum_{z_i} p(z_i, z_u|i, u, r)}{N} \tag{5.26}$$

$$p(i|z_i) = \frac{\sum_{i',u,r} \sum_{z_u} p(z_i, z_u|i', u, r)\mathbb{1}(i' = i)}{N \cdot p(z_i)} \tag{5.27}$$

$$p(u|z_u) = \frac{\sum_{i,u',r} \sum_{z_i} p(z_i, z_u|i, u, r)\mathbb{1}(u' = u)}{N \cdot p(z_u)} \tag{5.28}$$

$$p(r|z_i, z_u) = \frac{\sum_{i,u,r'} p(z_i, z_u|i, u, r)\mathbb{1}(r' = r)}{\sum_{i,u,r'} p(z_i, z_u|i, u, r')} \tag{5.29}$$

where N stands for the total number of observations.

Finally, we can define the prediction formula, for a specific user u and item i, as

$$\hat{r}_{ui} = \sum_{r=1}^{5} r \frac{p(i,u,r)}{\sum_{r'} p(i,u,r')} \tag{5.30}$$

As also done in the cluster model, we exploit the results to try a memory-based approach. Through Bayes' theorem, we compute the probabilities $p(z_i|i)$ and $p(z_u|u)$

$$p(z_i|i) = \frac{p(i|z_i)p(z_i)}{\sum_{z'_i} p(i|z'_i)p(z'_i)} \tag{5.31}$$

$$p(z_u|u) = \frac{p(u|z_u)p(z_u)}{\sum_{z'_u} p(u|z'_u)p(z'_u)} \tag{5.32}$$

We then assign to each user and item the respective cluster to which it has the highest probability of belonging. We use the users' clusters to apply a user-based model as previously done, and use the items' clusters to apply an item-based model with the same characteristics of the user-based one: we compute the similarities only among items of the same cluster and compute predictions by picking the top 10 most similar items. We refer to the two models as "User-Based Flexible Cluster" and "Item-Based Flexible Cluster".

5.3.2 Results

We run the model 9 times, each time increasing both the number of user classes and items classes from 2 up to 10. For each run we initialize the probabilities $p(z_i)$, $p(z_u)$, $p(i|z_i)$, $p(u|z_u)$ and $p(r|z_i, z_u)$ randomly from a Dirichlet distribution. For the final evaluation we consider the model with 6 user classes and 6 item classes.

We control the correct functioning of the model through the log-likelihood:

$$L(\mathbf{\Psi}) = \sum_{(i,u,r)\in Train} \log \left(\sum_{z_i} \sum_{z_u} p(z_i)p(z_u)p(i|z_i)p(u|z_u)p(r|z_i, z_u) \right) \tag{5.33}$$

where $Train$ is the set of observations in the training set.

As usual, the final value of the log-likelihood for each model increases with the number of classes. This model takes roughly the same number of iterations of the aspect model to converge.

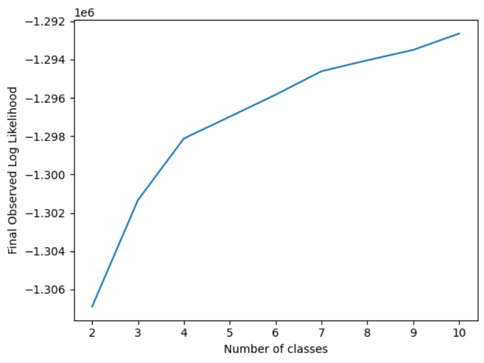


Figure 8: Final value of the log-likelihood for each model

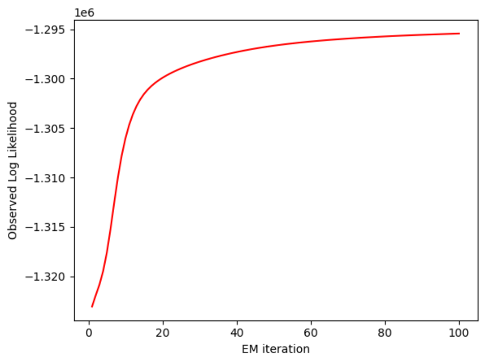


Figure 9: Log-likelihood at each iteration of the EM algorithm for the model with 6 user classes and 6 item classes

The MAE of the models are summarized in the table below:

Model	MAE
Flexible Mixture Model	0.761
User-Based Flexible Cluster	0.793
Item-Based Flexible Cluster	0.855
User-Based No Cluster	0.770
Item-Based No Cluster	0.780

It is interesting to see that the MAE of the FMM is the first one to go below that of both benchmarks. This results should be considered promising as we did not do any sort of fine-tuning for the number of the latent classes, but we just picked the mid-value between 2 and 10 for both z_u and z_i. Considering the possibility of optimizing these parameters, this model could probably bring even more interesting results. Looking at the memory-based models, one thing is particularly noteworthy: while the user-based model with clusters performed similarly to the one of the cluster model, the item-based model suffered more from the division in clusters, increasing by around 0.07 its MAE.

5.4 Decoupled Model

5.4.1 Theory

The fourth model we look at is the decoupled model proposed by Jin, Si and Zhai [10]. This model adds a layer of complexity to what we have already seen because it tries to model separately user preferences and user rating behaviours. In fact, two new latent variables regarding the users are introduced: z_R, which accounts for rating patterns, and z_P, which accounts for user preferences. This separation is done because even users who share similar preferences may have really different rating patterns, therefore the model should distinguish between users who have the same tastes, but vary in the strictness of their grades. Hence, $p(z_P|u)$ describes the probabilities over the types of preferences that user u may have, while $p(z_R|u)$ describes the probabilities over the types of rating behaviours that user u may show. As for the flexible mixture model, we have a class z_i that clusters the types of items. We also introduce another new variable, z_{pref}, which depends from z_P and z_i; this is a binary variable that indicates whether the class of users z_P likes or not the class of items z_i. The model could also be extended by introducing a more nuanced scale of preference, thus allowing z_{pref} to take more than 2 values. Finally, the ratings depend on classes z_R and z_{pref}, reflecting how the user's rating behaviour z_R and the preference condition z_{pref} interact to produce the observed rating. Each data point consisting of user, item and rating is considered an observation of item and rating conditioned on the the user, hence the probability is

$$p(i, r|u) = \sum_{z_P, z_R, z_i} p(z_P|u)p(z_R|u)p(z_i)p(i|z_i) \left(\sum_{z_{pref}=0}^{1} p(z_{pref}|z_P, z_i)p(r|z_R, z_{pref}) \right) \quad (5.34)$$

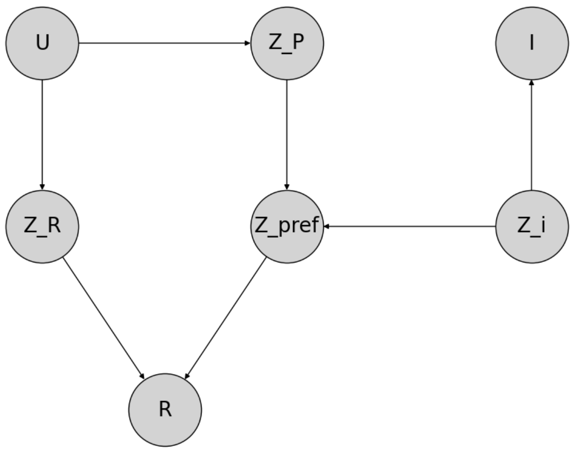


Figure 10: Graphical representation of the decoupled model

To estimate all the parameters of this distribution we first define the E step of the EM algorithm, which computes the joint probability of the 4 present latent classes, given each observed user, item and rating combination:

$$p(z_P, z_R, z_i, z_{pref}|i, u, r) = \frac{p(z_i)p(i|z_i)p(z_P|u)p(z_R|u)p(z_{pref}|z_P, z_i)p(r|z_R, z_{pref})}{\sum_{z'_p, z'_R, z'_i, z'_{pref}} p(z'_i)p(i|z'_i)p(z'_P|u)p(z'_R|u)p(z'_{pref}|z'_P, z'_i)p(r|z'_R, z'_{pref})} \tag{5.35}$$

We then update all the parameters in the M step, using the following formulas:

$$p(z_i) = \frac{\sum_{i,u,r} \sum_{z_P, z_R, z_{pref}} p(z_i, z_P, z_R, z_{pref}|i, u, r)}{N} \tag{5.36}$$

$$p(i|z_i) = \frac{\sum_{i',u,r} \sum_{z_P, z_R, z_{pref}} p(z_i, z_P, z_R, z_{pref}|i, u, r)\mathbb{1}(i' = i)}{N \cdot p(z_i)} \tag{5.37}$$

$$p(z_P|u) = \frac{\sum_{i,u',r} \sum_{z_i, z_R, z_{pref}} p(z_i, z_P, z_R, z_{pref}|i, u', r)\mathbb{1}(u' = u)}{\sum_{i,u',r} \sum_{z_i, z'_P, z_R, z_{pref}} p(z_i, z'_P, z_R, z_{pref}|i, u', r)\mathbb{1}(u' = u)} \tag{5.38}$$

$$p(z_R|u) = \frac{\sum_{i,u',r} \sum_{z_i, z_P, z_{pref}} p(z_i, z_P, z_R, z_{pref}|i, u', r)\mathbb{1}(u' = u)}{\sum_{i,u',r} \sum_{z_i, z_P, z'_R, z_{pref}} p(z_i, z_P, z'_R, z_{pref}|i, u', r)\mathbb{1}(u' = u)} \tag{5.39}$$

$$p(z_{pref}|z_P, z_i) = \frac{\sum_{i,u,r} \sum_{z_R} p(z_i, z_P, z_R, z_{pref}|i, u, r)}{\sum_{i,u,r} \sum_{z_R, z'_{pref}} p(z_i, z_P, z_R, z'_{pref}|i, u, r)} \tag{5.40}$$

$$p(r|z_R, z_{pref}) = \frac{\sum_{i,u,r'} \sum_{z_i,z_P} p(z_i, z_P, z_R, z_{pref}|i, u, r)\mathbb{1}(r' = r)}{\sum_{i,u,r'} \sum_{z_i,z_P} p(z_i, z_P, z_R, z_{pref}|i, u, r')} \tag{5.41}$$

where N is the total number of observations. Finally, we compute the final prediction:

$$\hat{r}_{iu} = \sum_{r=1}^{5} r \frac{p(i, r|u)}{\sum_{r'} p(i, r'|u)} \tag{5.42}$$

We also exploit $p(z_P|u)$ and $p(z_R|u)$ for a memory user-based approach and we use Bayes' theorem to compute $p(z_i|i)$

$$p(z_i|i) = \frac{p(i|z_i)p(z_i)}{\sum_{z_i'} p(i|z_i')p(z_i')} \tag{5.43}$$

which we use for a memory item-based approach. These three models are exactly computed like the previous memory-based ones and we refer to them as "User-Based Decoupled Cluster (z_R)", "User-Based Decoupled Cluster (z_P)" and "Item-Based Decoupled Cluster".

5.4.2 Results

Given the high training time required for this model and the high variability of the possible combination of parameters, we run the model once using the parameters employed in the original paper. This corresponds to 5 classes for the variable z_i, 10 classes for the variable z_R and 3 classes for the variable z_P. As also already mentioned above, the model could have more than two classes for the variable z_{pref}, but in this implementation we stick with two. The parameters $p(z_i)$, $p(i|z_i)$, $p(z_P|u)$, $p(z_R|u)$, $p(z_{pref}|z_P, z_i)$ and $p(r|z_R, z_{pref})$ are initialized randomly sampling from a Dirichlet distribution. We check the convergence of the EM algorithm through the log-likelihood:

$$L(\mathbf{\Psi}) = \sum_{(i,u,r)\in Train} \log \left(\sum_{z_P,z_R,z_i,z_{pref}} p(z_P|u)p(z_R|u)p(z_i)p(i|z_i)p(z_{pref}|z_P, z_i)p(r|z_R, z_{pref}) \right) \tag{5.44}$$

where $Train$ is the set of observations in the training set.

This last model is the one which took definitely more EM iterations to reach a plateau.

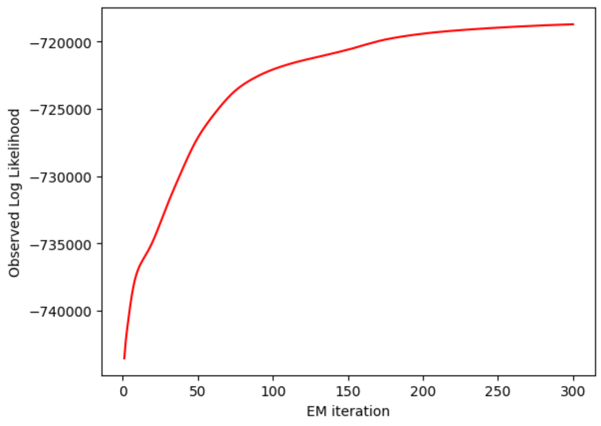


Figure 11: Log-likelihood at each iteration of the EM algorithm for the decoupled model

The results of the model are summarized below:

Model	MAE
Decoupled Model	0.892
Item-Based Decoupled Cluster	0.836
User-Based Decoupled Cluster (z_R)	0.779
User-Based Decoupled Cluster (z_P)	0.791
User-Based No Cluster	0.770
Item-Based No Cluster	0.780

Unfortunately, the decoupled model does not outperform the score of the FMM. Nevertheless, we can make a similar point to the one done for the FMM: given that we have 4 different latent variables, fine-tuning the number of classes in each variable is essential to achieve the best results. The results of the memory-based models are in line with what we have seen previously: the user-based models with clusters are not too affected by the division in clusters, with the model based on z_R slightly better than the one based on z_P probably because of the lower number of classes (3 instead of 10); the item-based model with clusters suffers a bit more from the division in clusters, worsening the MAE by around 0.05.

6 Conclusion

This thesis has explored the application of mixture models to enhance collaborative filtering in recommender systems. By addressing the limitations of traditional collaborative filtering methods, mixture models offer an interesting approach to improve recommendation accuracy.

We began by presenting the theory behind mixture models and collaborative filtering. The study then moved to practical applications, where we showed the results of 4 different mixture models applied to the MovieLens 100k dataset: the cluster model, the aspect model, the flexible mixture model, and the decoupled model. For each of these models we first implemented a classical model-based approach, with predictions based on the probability distribution assumed by the model; for the first, third and fourth model we also exploited some parameters that we had already computed to try a memory-based approach through hard clustering of users or items.

The cluster model showed some limitations in its ability to predict accurate recommendations compared to memory-based methods. However, it highlighted the potential of employing hard clustering to improve the efficiency of similarity computations, without losing too much accuracy. The aspect model introduced the idea of modeling the probability distribution of users and items too, and linking their interactions through a latent variable. Despite its innovative approach, the aspect model did not surpass the performance of simpler benchmarks, demonstrating the need for further refinement. The flexible mixture model proved to be a significant improvement, outperforming the predetermined benchmarks. In fact, by separating the clustering of users and items, it managed to model the user-item interactions better than all the other models analyzed. Lastly, the decoupled model introduced an additional characteristic by distinguishing between user preferences and rating behaviours. Although this model did not achieve the highest accuracy, it underscored the possible benefits of incorporating user behaviour patterns into our models.

Future research should focus on exploring hybrid approaches that combine mixture models with other recommendation techniques. Joining a CF system based on mixture models with a content-based system that exploits other data besides user ratings may lead to even more personalized and accurate recommendations. Additionally, a detailed study on the optimization of the number of classes in mixture models could yield interesting results.

In conclusion, this thesis has shown that mixture models are a valuable tool for collaborative filtering in recommender systems. By modeling complex distributions with latent factors, mixture models can make more precise, flexible, and customized recommendations, eventually improving user satisfaction and engagement.

References

[1] Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, New York, 2016.

[2] Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. Accurately computing the log-sum-exp and softmax functions. *IMA Journal of Numerical Analysis*, 41(4):2311–2330, 2021.

[3] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *14th Conference on Uncertainty in Artificial Intelligence*, pages 43–53, 1998.

[4] Christian Desrosiers and George Karypis. A comprehensive survey of neighborhood-based recommendation methods. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender systems handbook*, pages 107–144. Springer, New York, 2010.

[5] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.

[6] Andrew Gelman, John B. Carlin, Hal S. Stern, David B. Dunson, Aki Vehtari, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, New York, 3 edition, 2013.

[7] Kevin J. Grimm, Russell Houpt, and Danielle Rodgers. Model fit and comparison in finite mixture models: A review and a novel approach. *Frontiers in Education*, 6:613–645, 2021.

[8] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4), 2015.

[9] Thomas Hofmann and Jan Puzieha. Latent class models for collaborative filtering. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, 1:688–693, 1999.

[10] Ron Jin, Luo Si, and ChenXiang Zhai. Preference-based graphic models for collaborative filtering. *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 329–336, 2002.

[11] Rong Jin, Luo Si, and Chengxiang Zhai. A study of mixture models for collaborative filtering. *Information Retrieval Journal*, 9(3):357–382, 2006.

[12] Geoffrey McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions.* John Wiley & Sons, New York, 1997.

[13] Geoffrey McLachlan and David Peel. *Finite Mixture Models.* John Wiley & Sons, New York, 2000.

[14] Kevin P. Murphy. An introduction to graphical models. *Rap. tech*, 96:1–19, 2001.

[15] Kevin P. Murphy. *Machine learning: a probabilistic perspective.* MIT Press, Cambridge, 2012.

[16] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In Francesco Ricci, Lior Rokach, and Bracha Shapira, editors, *Recommender systems handbook*, pages 1–35. Springer, New York, 2010.

[17] Luo Si and Rong Jin. Flexible mixture model for collaborative filtering. *Machine Learning, Proceedings of the Twentieth International Conference*, pages 704–711, 2003.