Technical University of Crete

Master Thesis

A Bayesian Personalized Recommendation System

Author:
Konstantinos Babas

Supervisor:
Dr. Georgios Chalkiadakis
(Assistant Professor)

A thesis submitted in fulfilment of the requirements
for the degree of Master of Science of Electronic & Computer Engineering

in the

School of Electronic & Computer Engineering

June 24, 2014

"Follow your dreams and live the way you like. . . "

TECHNICAL UNIVERSITY OF CRETE

Abstract

Intelligent Systems Laboratory
School of Electronic & Computer Engineering

Master of Science of Electronic & Computer Engineering

A Bayesian Personalized Recommendation System

by Konstantinos Babas

In this research, we propose a novel Bayesian approach for personalized recommendations. We succeed in providing recommendations that are entirely personalized, based on a user's past item "consumptions", building a representative *user model* which reflects agent's corresponding beliefs. Having a set of items, our agent has to select the one which better matches her beliefs about a specific user, in order to recommend it and receive the corresponding reward. In our approach, we model *both* user preferences *and* items under recommendation as *multivariate Gaussian* distributions; and make use of *Normal-Inverse Wishart* priors to model the recommendation agent beliefs about user types. We interpret user ratings in an innovative way, using them to guide a Bayesian updating process that helps us both capture a user's current mood, and maintain her overall user type. We produced several variants of our approach, and applied them in the movie recommendations domain, evaluating them on data from the MovieLens dataset. We developed a generic & domain independent system, able to face the scalability challenge and able to capture user preferences (*long-term* and *short-term*). Moreover, we dealt with the *exploration* vs *exploitation* dilemma in this domain, via the application of various exploration algorithms (e.g., *VPI* exploration). Ours is a completely *personalized* approach, which exploits *Bayesian Reinforcement Learning* in order to recommend an item or a *top-N* group of items, without the need of ratings prediction. We do not employ a *Collaborative Filtering* or *Content-based* or *Preference Elicitation* technique, but we are still able to provide successful recommendations. Furthermore, we tackle the famous "cold-start" problem via the use of Bayesian and VPI explorations. Our algorithms are shown to be competitive against a state-of-the-art method, which nevertheless requires a minimum set of ratings from various users to provide recommendations—unlike our entirely personalized approach.

In Greek...

Στη συγκεκριμένη έρευνα, προτείνουμε ένα καινοτόμο *Μπαεσιανό Σύστημα Προτάσεων*. Το σύστημά μας παρέχει εντελώς εξατομικευμένες προτάσεις βασιζόμενο στις παλαιότερες "καταναλώσεις" αντικειμένων του χρήστη, δομώντας ένα αντιπροσωπευτικό *μοντέλο χρήστη* το οποίο αντικατοπτρίζει τις πεποιθήσεις του πράκτορα. Έχοντας ένα σύνολο αντικειμένων, ο πράκτοράς μας πρέπει να επιλέξει εκείνο που ταιριάζει περισσότερο στις πεποιθήσεις του σχετικά με ένα συγκεκριμένο χρήστη, προκειμένου να του το προτείνει και να λάβει την αντίστοιχη ανταμοιβή. Στην προσέγγισή μας, μοντελοποιούμε τις προτιμήσεις του χρήστη και τα αντικείμενα προς πρόταση ως *πολυδιάστατες Gaussian* κατανομές, και χρησιμοποιούμε *Normal-Inverse Wishart* priors για να μοντελοποιήσουμε τις πεποιθήσεις του πράκτορα σχετικά με τον τύπο του χρήστη. Ερμηνεύουμε τις βαθμολογίες του χρήστη με καινοτόμο τρόπο χρησιμοποιώντας τις για να καθοδηγήσουμε το Bayesian updating, που μας βοηθά να ανιχνεύσουμε την τρέχουσα διάθεση του χρήστη και να διατηρήσουμε το γενικό τύπο του. Επίσης, δημιουργήσαμε διάφορες παραλλαγές της προσέγγισής μας και τις εφαρμόσαμε στον τομέα της πρότασης ταινιών, αξιολογώντας τις σε δεδομένα που προέρχονται από το MovieLens. Με τη συγκεκριμένη εργασία, καταφέραμε να διαχειριστούμε τις προκλήσεις που προκύπτουν σε αυτόν τον τομέα. Έτσι, καταφέραμε να αναπτύξουμε ένα γενικό, ανεξαρτήτου τομέα, σύστημα, το οποίο αντιμετωπίζει εύκολα την πρόκληση της επεκτασιμότητας και είναι ικανό να αιχμαλωτίσει τις προτιμήσεις του χρήστη (*μακροπρόθεσμες* και *βραχυπρόθεσμες*). Ακόμη, διαχειριστήκαμε επιτυχώς το δίλημμα *εξερεύνηση* έναντι *εκμετάλλευσης*, εφαρμόζοντας διάφορους αλγόριθμους εξερεύνησης (πχ., εξερεύνηση VPI). Αναφερόμενοι στη συνεισφορά αυτής της έρευνας, η συγκεκριμένη είναι μια αποκλειστικά *εξατομικευμένη* προσέγγιση, η οποία αξιοποιεί Μπαεσιανή *Ενισχυτική Μάθηση* προκειμένου να προτείνει ένα αντικείμενο ή ένα *top-N* σύνολο αντικειμένων, χωρίς την ανάγκη πρόβλεψης βαθμολογιών. Δεν χρησιμοποιούμε μια *Collaborative Filtering* ή *Content-based* ή *Preference Elicitation* τεχνική, όμως είμαστε ικανοί να παρέχουμε επιτυχημένες προτάσεις. Επιπλέον, αντιμετωπίσαμε το περίφημο πρόβλημα "cold-start" μέσω της χρήσης Μπαεσιανής εξερεύνησης και VPI. Τέλος, ο αλγόριθμός μας φαίνεται πως είναι ανταγωνιστικός απέναντι σε μια νέα, εξελιγμένη μέθοδο, η οποία παρ' όλα αυτά χρειάζεται ένα ελάχιστο σύνολο βαθμολογιών από διάφορους χρήστες προκειμένου να παρέχει προτάσεις — σε αντίθεση με τη δική μας πλήρως εξατομικευμένη προσέγγιση.

Acknowledgements

First of all, I would like to thank my supervisor, Georgios Chalkiadakis, for his patience and his significant guidance all these years. My collaboration with Dr. Chalkiadakis provided with the equipment for becoming a better researcher.

I thank my colleagues, Charilaos Akasiadis, Manolis Orfanoudakis, Alexis Georgogiannis, Angelos Hliaoutakis, Antonis Igglezakis and Giannis Skoulakis for our constructive coexistence. Thanks to Evangelos Tripolitakis for providing me with processed data from the MovieLens dataset and experimental results about the *LSMF* algorithm. Special thanks to my teachers for doing their best and equipped me with essential knowledge. Thanks to all people of the Intelligent Systems Laboratory.

My gratitude to my parents for their past, present and future understanding and support!!! Finally, many thanks to anyone I might have forgotten to thank.

Contents

List of Figures

List of Tables

Abbreviations

ALS	**A**lternating **L**east **S**quares
BRX	**B**ayesian **R**andom e**X**ploration
BVPI	**B**oltzmann-**VPI**
BYLI	**B**ayes**Y**ou**L**ike**I**t
CF	**C**ollaborative **F**iltering
KL	**K**ullback-**L**eibler
LSMF	**L**arge, **S**parse **M**atrix **F**actorization
NIW	**N**ormal-**I**nverse **W**ishart
PE	**P**reference **E**licitation
VPI	**V**alue of **P**erfect **I**nformation
WWW	**W**orld **W**ide **W**eb

Symbols

c_l	center of *l-th* cluster
$gain_i^j$	gain of user type j from item i; VPI exploration
I	the recommended item selected by VPI exploration
i_i	item i; VPI exploration
j	user type; VPI exploration
M	maximum available demo/item rating
n	number of samples; Bayesian updating
p_i	*i-th* item in specific cluster
r_i	user reward from item i; VPI exploration
S	scatter matrix; a statistic element for Bayesian updating
$T = c \cdot \alpha^t$	Temperature parameter, c a constant, a <1 and t the time step; Boltzmann exploration
U_i	utility of item i; Boltzmann exploration
V_i	aggregated value of item i; VPI exploration
$\overline{x}$	sample mean; Bayesian updating
x_i	*i-th* samples; Bayesian updating
κ_n	real number; parameter of the NIW
Λ_n	precision matrix; parameter of the NIW
μ	mean; parameter of the multivariate Gaussian
μ_n	mean vector; parameter of the NIW
ν_n	degrees of freedom; parameter of the NIW
Σ	covariance matrix; parameter of the multivariate Gaussian
σ^2	variance

Dedicated to my lovely parents Nikos & Anna, and to my brother Giorgos.

Chapter 1

INTRODUCTION

Making decisions on what movie to see, what kind of music to listen to, or even what financial investment to make can be a hard problem when someone is presented with a multitude of choices. This is partly because nowadays users are confronted with huge amounts of available information, which sometimes renders the decision making a complex and tedious process, and might lead to poor decisions. Research in recommendation systems attempts to understand user needs, preferences and mood, and help her make a decision. A *Recommender System* is an information system which produces suggestions for items to users on their request. Behind the recommender, an advanced and sophisticated algorithm operates and combines data regarding items and user preferences in order to make recommendations. Additionally, an advanced recommendation system has the ability to recommend items regardless their type, i.e., the recommendation algorithm is generic and able to manage different type of items, and the system's design is flexible with the ability to be easily adapted to changes, accordingly to items type.

When a user doesn't have enough knowledge to make a decision, she has the need for recommendations. Recommendation systems have to predict what item user likes, but to do so, they should take into consideration user constraints and preferences. This can be done either explicitly or implicitly. Explicit extraction of user preferences is feasible, when the user is allowed to use quantified metrics to express her liking for items. By contrast, the system can also infer user preferences via observing and interpreting user actions. The user feedback is stored in the system for further elaboration. Exploiting information about items, user's past feedback, and system's beliefs about the user, a recommender can generate new recommendations, that are "personalized", i.e., they are expected to match user preferences.

In the last few years, the scientific community has shown strong interest in the research on recommender systems [1]:

- Many media companies and web giants, like Netflix, IMDb, Amazon and YouTube, are developing and incorporating sophisticated recommenders in their systems.
- Scientific conferences take place every year with exclusive domain of interest the recommendation systems. Also, there exist dedicated issues in academic journals related to that domain.
- Books, tutorials at scientific conferences and courses at institutions of higher education have been recently presented with main subject the recommender systems.

Typically, most recommendation methods require pre-training on data gathered from many users, who are *classified according to their inferred similarity in preferences.* A key assumption in research on recommender systems is that people are influenced during their decision making by recommendations provided by others with similar desires. Moreover, many approaches require much user involvement in a potentially cumbersome, lengthy interaction with the system. This can impose serious limitations to the usability of a recommendation system, especially when a user wishes to make a fast decision that is also dependent on her current mood. Also, a recommender system should navigate the user without making her feel restricted, but autonomous and self-determined.

Most established recommendation systems exploit user ratings over a large number of items via the use of *collaborative filtering (CF)*, *content-based* methods, or a combination thereof. *Content-based methods* usually make recommendations by analyzing the content of textual information about an item. On the other hand, *collaborative filtering* is based on the assumption that, if two users rate n items similarly, they will probably rate other items similarly as well. So, collaborative filtering techniques use ratings from a specific user on some items (e.g., movies), and combine them with ratings of other users on a set of items in order to infer the ratings of that user on unrated items. Additionally, *preference elicitation (PE)* techniques try to collect user preferences in order to construct the user's *utility function.* Most PE methods set queries to the user asking her to evaluate, order, or constrain potential system outcomes.

It is in the interests of vendors and services providers to take the use of recommender systems into account, since, by exploiting this technology, they can [1]:

- *Better understand user needs*; (this is actually the main goal of a recommender system).
- *Increase their sales*, because users receive recommendations with items that better match their preferences.

- *Sell large variety of items*, because sometimes the recommenders tend to recommend items that user likes them, but she would never select among others.
- *Increase user satisfaction*, since the user receives recommendations that she likes and the recommender improves user's interaction with the system.
- *Earn user fidelity*, because a properly functioning system treats every user as an individual, and appears to know what to offer her based on previous interactions.

In this research, we design and evaluate a *Bayesian recommendation agent*, based on a *simple*, *fast*, and *easy-to-use* elicitation and modelling process; and apply it in the movie recommendations domain. By means of this process, our agent is able to recommend a user items (e.g., movies) that best fit *both* her long-time preferences and current mood. To achieve its objective, the agent maintains *item types* (summarizing item characteristics); and *user types* (corresponding to modeled user preferences). Crucially, these are *both* represented as *multivariate Gaussian distributions* over ranges of values (ratings), describing the degree to which an item is composed of certain attributes (e.g., movie genres); and the degree to which a user likes the particular attributes, respectively. This allows for the establishment of a correspondence between user and item types: *intuitively, a user model is viewed by the agent as being an amalgamation of items this user likes*. We term this the *"you are what you consume"* idea (we discuss its origins in Chapter 2). The employment of *Normal-Inverse Wishart (NIW)* conjugate priors to model agent beliefs about types guarantees that these beliefs can be readily updated.

Now, to learn about a specific user's current mood, we first *only* ask her to rate a small number of *demonstrative items* (*"demos"* for short; e.g., movie trailers), also represented by multivariate Gaussians (just like other items), and which are selected based on the current agent beliefs about user type. Each rating is treated as a unit, that is, we do not ask the user to rate different item attributes. We then employ these ratings in an innovative way, interpreting them as an indication of the *number of samples* to take from the corresponding demo type; i.e., to indicate a demo's "weight", or, to put otherwise, the "degree of correspondence" that the specific demo has with user preferences and current mood. This is reasonable, as a user's rating of, e.g., trailers, intuitively reflects her current mood—which is, nevertheless, not independent of her long-term tastes. The agent takes this into account: each demo presented was already selected according to perceived *user type*—in this manner, long-term user preferences *are* incorporated in the choice of demos.

The samples thus collected are then utilized by a Bayesian updating process that infers a *temporary demo-based user type*, given demo ratings. Having the demo-based user type, we can then search for the best possible match (employing a *KL-divergence* metric) with

an *item* to actually recommend—and which is *not* necessarily one of the demos shown earlier. The selected item is then presented to the user, who *rates* it, leading to an update of her overall *user type*.

The overall recommendation process effectively corresponds to a *Bayesian exploration* approach in this domain—since, when selecting demos or items to present to the user, our method optimises with respect to Bayes-updated beliefs, rather than taking explicit "exploration" actions. Moreover, intuitively our method allows for a better user experience, and implicitly helps the user "discover" more about her own real preferences, as it does not rigidly disallow suggestion possibilities or prune vast parts of the search space (as methods relying on explicit user statements about their preferences or ratings probably would), but only makes suggestions given its probabilistic beliefs regarding user preferences. For interest, we devised and tested certain additional *variants* of our approach, which employ alternative (Bayesian and non-Bayesian) exploration methods when selecting a demo or item to present to the user. One of these variants most probably constitutes the first adaptation of the well-known *Value of Perfect Information* Bayesian exploration heuristic [2, 3] in the recommendations domain.

For a recommendation system, the absolute goal is its reliable operation. To confirm the proper operation, the evaluation of the recommendation technique, and thus of the developed recommender, is vital. Before commercial use, the system has to undergo an evaluation procedure. Using well known benchmark datasets, scientists can perform extended simulated evaluations in order to ensure the correctness of their algorithms. Additionally, those datasets offer a chance for comparison between various techniques. After launching, the evaluation can be performed using real users. Users interact with the recommender and their interaction produces useful conclusions about system's usability and efficiency. If such an evaluation is costly and risky, then focused experiments can be performed by requesting specific users to interact with the system following predesigned scenarios.

We evaluated our approach via simulations in the movie recommendations domain. As a note, the use of multivariate Gaussians and Normal-Inverse Wishart distributions to model types and corresponding beliefs in this particular domain is novel, as is the use of "trailer" ratings for inference purposes. Our experimental results are highly encouraging, demonstrating as they do that the agent quickly learns to recommend movies that receive high user ratings. In particular, our agent manages to exhibit performance that is comparable to that of a popular, state-of-the-art, collaborative filtering-based method for movie recommendations. Importantly, it does so without the need of looking at pre-gathered/pre-processed data involving the current user or her peers. As such, the ever-present "cold start" problem, a constant challenge to CF methods due to ratings

scarcity (see, e.g., [4] for a discussion), does not apply to our personalized method—since it does not employ any ratings of other users whatsoever, and does not aggregate past ratings to predict future ones.

We now summarize the contibutions of this work. As mentioned before, this is a completely new approach for the recommendation systems domain:

1. First of all, this method is completely *personalized*, that is, we produce recommendations based only on previous observations of a specific user. We don't use, nor do we have the need of other users recommendation data in order to make recommendations to the specific one. We simply observe the behavior of the user through her interaction with the system and recommend her items that she likes.

2. We propose a novel Bayesian *Reinforcement Learning* technique, which does not rely on any given recommendation method. That is, we do not do *Collaborative Filtering* or *Preference Elicitation* techniques, and we do not use content information about items (like *Content-aware* methods do), or social relationships between users. We perform a *model-based learning* of user preferences using Bayesian updating of the user's model.

3. We focus on user *preferences* and *mood* in combination to item features, drawing inspiration from our *"you are what you consume"* concept. We model item features as multivariate Gaussian distribution and build a *user model* through Bayesian updating using the models of the recommended items. This updated model reflects user's preferences. Additionally, by exploiting demo items in a novel way, we can also capture temporary changes of user mood, while many state-of-the art methods do not have this ability.

4. This is the first time, that a non-Preference Elicitation (PE) recommendation approach does not need to predict ratings or probability distributions over ratings. We simply receive user ratings on recommended items and incorporate them in our algorithm in order to update our *beliefs* about her, i.e. to update user model. With this model at hand, we recommend items that best match it; that is, items that better match users preferences. [1]

5. We are the first who integrate elements of Bayesian reinforcement learning theory in a recommendation system. Exploiting *VPI exploration* [2, 3], we are able to "learn" a user model more efficiently.

[1] PE techniques do not manage ratings, but answers to queries. Based on those answers, they recommend items that maximize user's utility function.

6. Our algorithm is able to capture *temporal effects* caused by the acquisition of ratings sequentially over time, since it holds a record for each user. In contrast, other methods struggle to face the problem and need to make ad-hoc decisions or use heuristics in order to take temporal effects into account, and to "fit" a huge number of data-describing parameters to the dataset.

7. We easily incorporate new data in our system and we can tackle the problem of "cold-start", which most approaches struggle to overcome.

Our approach is quite simple and fast, but still generic and effective. At this point, we should also state that the main ideas of this work, along with certain preliminary experimental results, have recently appeared in an article of ours published in *RecSys'13* [5].

The remaining chapters are organized as follows: Chapter 2 contains the necessary background knowledge related to the proposed algorithm. In Chapter 3, we provide an overview of previous work in the recommendations domain, and give a brief description of our method, presenting the challenges characterizing this domain, along with the way we face them. Next, we provide a detailed presentation of our approach in Chapter 4 describing its main aspects, the applied exploration methods, and its application to the movies domain. In Chapter 5, we present the results of the experiments we ran, and their interpretation. In the end, we write down the conclusions derived from our research, and our thoughts about future work (Chapter 6).

Chapter 2

BACKGROUND

In this chapter, we introduce our *"you are what you consume"* idea and provide the necessary background regarding the proposed recommendation approach. *Bayesian updating* is the "tool" that our agent uses in order to update her beliefs about user preferences. Exploiting the prior knowledge in the form of *conjugate prior*, and the current observations, the recommender produces a *posterior*, which represents the updated deliefs. Additionally, we give a description about the *exploration vs exploitation* problem, which has to do with the dilemma of gathering more information or making the best decision based on current information. Finally, we talk about *Bayesian* exploration and *VPI*, which are used in our recommender.

2.1 Recommender systems: Main concepts and goals

A recommender system is an intelligent information system, i.e., it possesses adaptation and learning abilities. Via learning, the recommender can become more effective and improve the user experience. By this, the user is stimulated to better understand her interests and, simultaneously, the system is enabled to collect further information and build a more accurate user model. From the user's perspective, the effectiveness of a recommender system depends on multiple factors. A system should be trusty and transparent. It has to allow users to explore new, not-tasted items, to provide them with item details and to offer them the ability to refine the set of recommendations [6].

Regarding the necessary data in order to make recommendations, some methods are based only on user ratings, while others exploit more complex data, like relation and trust between users, content information and ontological structures, which may provide more informative feeds to the recommendation process. Let us make a discrimination

between the data that a recommender has at its disposal during its life cycle, which are *observations* and information about *items* and *users*.

First of all, *items* are the objects that the system has to recommend. They are stored in the system's database using a unique id or more sophisticated models. An item can be characterized by a number of *features*. These features can be used by a recommender in order to build a model. Additionally, each item has a specific value for each user, which can be translated as the user's satisfaction after the use of the item or the cost to purchase it. The cost of an item becomes issue of high importance when the item has high economic value, e.g., real estate, shares portfolio, investments. The system should take into consideration all those facts.

Users, who interact with a recommender system, have their own characteristics and preferences, and there are significant differences between groups of them. Recommendation techniques have to model user preferences for incorporating and exploiting them. In the literature, we meet many types of user modeling. Some methods use simple vectors or lists that contain user ratings. Some others use modeled information about user likings, personal information or behaviors, while others exploit relationships and trust between users or groups of them.

A recommender system should create relations between users and items in order to be able to make recommendations. Such relations can be created based on *observations* of user-system interaction. Depending on the system's design and the recommendation technique, the user leaves her traces back explicitly or implicitly. That is, observation data can be ratings, tags, interpreted behaviors (e.g., click on, pause, leave page, etc.), queries or answers.

Herlocker et al. [7] define 11 tasks that a recommender offers or a user can succeed using a recommender:

1. Find good items: The system recommends to a user a ranked list of items that she likes them.
2. Find all good items: The recommender has the ability to recommend all the available good items.
3. Exploitation of content information: The system recommends items exploiting the available content information.
4. Recommend a sequence of related items: The system recommends related items, which supplement a sequence, for better and continuous user satisfaction.

5. Recommend a group of items: The system recommends a group of related items which can be considered as a complete one.

6. Browse items: The user has the ability to browse the available items without "consuming" anyone of them.

7. Find a reliable recommender: The recommender offers an additional functionality that allows the user to test its behavior.

8. Improve the profile: The user can provide additional information to the system about her preferences.

9. Express beliefs: The user wishes to contribute on enhancing the knowledge-base of the system providing her ratings and likings.

10. Help others: The user wishes to help other users with her opinion for items that she had already tasted.

11. Influence others: Some users just aim to influence other users' decisions.

Here, we should remark that such a system must serve equally the needs of both the system itself and the user.

2.2 The *"you are what you consume"* idea

Aristotle, the ancient Greek philosopher, quoted that *"We are what we repeatedly do. Excellence, then, is not an act, but a habit."*[1]. This quotation says that doing something again and again leads to acting perfectly out of habit. Interpreting this in terms of recommendations, we can say that observing user's sequential actions, we are able to indirectly infer an accurate model about her habits—our ordinary actions reflect our habits. Thousands of years later, Steve Jobs quoted the following: *"It's not the customers' job to know what they want."*. Intuitively, this is a key assumption for the domain of the recommendations.

Generally, the items we consume leave their marks on us; they characterize us uniquely. The products, that someone uses, can shape the person she will be [6, 8]. Thus, the higher the value of a consumed item, the higher its impact on customer's life. Additionally, tasting new items often causes changes in someone's character, making her different [6, 8]. All these facts form the intuition behind our *"you are what you consume"* idea, under which we developed the proposed approach.

[1] According to William James Durant, an American writer, historian, and philosopher.

Respectively, *consumer choice* theory is part of economics, which tries to explain the relationship between consumer choices and profit [9–11]. The key idea behind this theory is that, the consumer tries to purchase products that increase her satisfaction, in relation to the amount of money she affords to spend. Customers buy expensive items if their budget is high, and less expensive ones if their budget is low. These choices are made in an effort to maximize their perceived benefit from the transactions [9, 12]. Regarding customer preferences, they are desires of an individual for the consumption of products, which are translated into choices of items to consume. There is no decision making without preferences [13]. The main reason we are interested in preferences, is that we exploit them to model the way that people make choices.

Furthermore, customers need to achieve equilibrium between preferences, available money and item's value. The combination of these three factors differentiates the customers and determines the decision to purchase a product—still, this combination makes the decision making more complicated. Clearly, the preferences, in combination with the value of the item, play significant role in the purchase decision [13]. In our case, and interpreting the *"you are what you consume"* "literally", our idea was to sequentially build a user model, which is an amalgamation of items the user has shown to prefer to consume. This is where recommendation systems come into the picture.

2.3 Bayesian updating

A key component of our recommendation agent is performing probabilistic inference regarding the types of the system's users, by means of *Bayesian updating*. In many cases, there exists the need of estimating the parameters of an unknown probability distribution—a *model*; and Bayesian updating can be employed to this end. Key to a computationally feasible application of Bayesian updating, is the use of *Bayes rule*, in conjunction with proper *conjugate priors* describing our beliefs about the model [14].

In some detail, in the face of new data (observations) regarding the unknown model, Bayes rule takes into account a *prior* distribution (reflecting our belief on the values of the model's parameters), a *likelihood function* and a marginal probability, in order to derive a *posterior*. The likelihood function brings together the prior and the observations and follows the form of the model. The posterior represents an updated *belief* about the prior, taking into account the new data. One can then use these posterior beliefs to derive new estimates of the parameters of the unknown distribution.

Both prior and posterior distributions must be of the same family—i.e., they must have the same algebraic form. If that is the case, they are termed *conjugate distributions*.

Additionally, the family of the *likelihood function* gives rise to the choice of the prior's family. If the prior is appropriately selected to be *conjugate* for the *likelihood*, then the posterior will be of the same family as the prior [15]. *Conjugacy* offers a closed form for the posterior, allowing for the easy update of the prior—via straightforward manipulations of the prior's *hyperparameters* in the face of new evidence.

In our case, the data provided to the agent is in the form of a multivariate Gaussian, corresponding to *demo* or *item types*. We also need to maintain the *demo-based user type* or the overall *user type*, which will also be multivariate Gaussians. Since we are not aware of the underlying type parameters—determining its *mean* and *covariance matrix*—we can model the *conjugate prior* as a *Normal-Inverse Wishart* distribution *(NIW)* [16, 17]. We can then readily update the *prior* hyperparameters using samples drawn from the data[2] to get the *posterior* ones:

$$\mu_n = \frac{\kappa_0}{\kappa_0 + n}\mu_0 + \frac{n}{\kappa_0 + n}\overline{x} \tag{2.1}$$

$$\kappa_n = \kappa_0 + n \tag{2.2}$$

$$\nu_n = \nu_0 + n \tag{2.3}$$

$$\Lambda_n = \Lambda_0 + S + \frac{\kappa_0 n}{\kappa_0 + n}(\overline{x} - \mu_0)(\overline{x} - \mu_0)^T \tag{2.4}$$

$$S = \sum_{i=1}^{n}(x_i - \overline{x})(x_i - \overline{x})^T \tag{2.5}$$

where $\overline{x}$ is the sample mean, n is the number of the samples and x_i are the samples from the data. Also, $\mu_0, \kappa_0, \nu_0, \Lambda_0$ are the known hyperparameters of the prior NIW distribution. Specifically, ν_0 represents its *degrees of freedom*, while μ_0 (the *mean vector*) and Λ_0 (the *precision matrix*) are the hyperparameters that specify the multivariate Gaussian component of the prior; and S is a *scatter matrix*, a statistic characterizing the model's covariance matrix and intuitively providing a measure of the samples' dispersion. Finally, the model's parameters, *mean (μ)* and *covariance matrix (Σ)*, can be calculated given the updated beliefs, using an *Inverse Wishart* and a *Gaussian*:

$$\Sigma \sim IW(\Lambda_n, \nu_n) \tag{2.6}$$

$$\mu \mid \Sigma \sim N(\mu_n, \Sigma/\kappa_n) \tag{2.7}$$

[2]In our case, we will be utilizing user ratings to determine the number of samples to draw from different data distributions.

In our system, during the recommendation process, the very first Normal-Inverse Wishart (NIW) prior has to be an *uninformative* one [16], which means that the *hyperparameters* of the NIW take the following values: $\kappa_0 = 0,\ \nu_0 = -1,\ \mid \Lambda_0 \mid = 0$. Thus, the *updated hyperparameters* of the NIW posterior after observing n samples become: $\mu_n = \overline{x},\ \kappa_n = n,\ \nu_n = n - 1,\ \Lambda_n = S = \sum_{i=1}^{n}(x_i - \overline{x})(x_i - \overline{x})^T$. The next time the user will use the system, these posterior hyperparametes will become the hyperparameters of the *informative* NIW prior, and the new posterior will be updated through the aforementioned formulas.

2.4 Exploration vs exploitation problem

A sophisticated recommender system must have the ability to explore user preferences [1]. Because of that, the agent has to face the dilemma of recommending items that better match agent's beliefs about the user or items that may gradually improve user satisfaction and help to offer better future recommendations. This is the *exploration vs exploitation problem* of *Reinforcement Learning* [18]. In a recommendations context, without performing exploration, it is hard to improve user satisfaction and receive higher ratings, but exploration might lead to temporary poor performance regarding user ratings. In other words, exploitation helps agent to choose actions whose outcomes come with high rewards, while exploration refers to the execution of actions, which provide better understanding of the environment and can significantly improve the agent's reward. Consequently, an intelligent system has to find the optimal balance between making the best decision given current information and gathering more information.

Two commonly used types of heuristic exploration techniques are *greedy* and *random* exploration. *Random* exploration selects actions randomly based on a uniform distribution, while *greedy* exploration always selects the action with the highest expected reward. There exists a specific kind of *greedy* exploration, called ϵ-*greedy*. This exploration selects the action with the highest reward, but with a small probability ϵ, selects a random action. Furthermore, there exist exploratory techniques that decrease their exploration tendency; these are *greedy in the limit exploration (GLIE)* techniques. Example of a GLIE technique is the *Boltzmann exploration*, where an action is executed based on the Boltzmann probability. The *temperature parameter* T of that probability is proportionate to the exploration tendency, i.e., as T is decreased over time, agent becomes less exploratory.

2.5 Bayesian exploration & VPI

Many exploration techniques perform a kind of controlled exploration taking into account the value of each possible action regarding the rewards received by the outcomes. Unlike most "explicit" exploration techniques that use heuristics to guide exploration, *Bayesian exploration* exploits prior knowledge of rewards and computes the posterior distribution of them based on current observations. This posterior is used to guide the exploration. *Value of perfect information (VPI)* gathers information through exploration [2, 3, 19, 20]. It exploits the fact that the information *gain* is higher in unexplored states; so, by quantifying the value of information, this technique is able to optimally balance exploration and exploitation.

In some detail, Bayesian exploration exploits agent's beliefs about the environment where she acts. Those beliefs enclose the modeled agent's uncertainty about the state of environment. The agent performs exploration based on her prior knowledge and the current observations acting somewhat greedily and following an implicit exploration policy. Each action is selected according to current state and beliefs. In *model-based* Bayesian exploration, models can be represented by prior distributions and be easily updated in order to produce posterior distributions, which represent agent's updated beliefs. The posteriors are centered to the most possible environment states. For the updating of the model the distributions must be sampled and the samples should be incorporated in the model. Remarkably, an important advantage of the Bayesian approach is the use of distributions; with them the agent's uncertainty can be quantified and be taken into account during the action selection procedure.

On the other hand, *VPI exploration* exploits the additional knowledge about uncertain situations. Using this information, VPI can compute the *gain* from exploratory actions and balance it against the cost of suboptimal actions, based on agent's prior beliefs. Moreover, exploring action a from state s has an expected *cost*. This cost is the difference between the expected value of the current best action, and the expected value of a. Then, the *value* of exploring a at s is defined as the difference between the *gain* and the *cost*. The agent that performs VPI exploration, always selects the action that maximizes that *value*. We present a detailed description of the VPI method in the subsection 4.3.1.

Chapter 3

RELATED WORK

A multitude of methods has been used for developing systems for automated recommendations. Most established recommendation systems exploit user ratings over a large number of items via the use of *collaborative filtering (CF)* methods, *content-based* methods, or a combination thereof, in order to make recommendations, while others use techniques for learning user interests and recommend her a set of items. Additionally, some approaches exploit *preference elicitation (PE)* theory or incorporate *Bayesian networks* to recommendation systems. Here, we review these approaches and highlight the distinctive differences between these methods and our own.

3.1 Collaborative Filtering & Content-based techniques

Content-based methods [21, 22] usually make recommendations by analyzing the content information of an item. They essentially postulate that items are recommended according to the level of similarity between their description and the user's profile information. On the other hand, collaborative filtering [23, 24] is based on the assumption that, if two users rate n items similarly, they will probably rate other items similarly as well. So, collaborative filtering techniques use ratings from a specific user on some items (e.g., movies), and combine them with ratings of other users on a set of items in order to infer about the ratings of that user on unrated items.

Additionally, the CF approaches usually use matrix factorization [25, 26] to produce recommendations. They exploit the fact that there exist associations between users and items and these associations are characterized by strengths, which can be quantified. Those algorithms receive as input ⟨user, item, strength⟩ tuples and insert the data in a square user-item interaction matrix. Then, they factor that matrix in order to recommend items. The *Alternating Least Squares (ALS)* [25] is a sophisticated matrix

factorization algorithm, which creates a new factor matrix containing only 0 and 1; 1 indicates the existence of a user-item interaction — while 0 its absence. This modification makes the factorization easier. The procedure is repeated for some times, to improve results' accuracy, and the final matrix is given as input to the next phase of the recommendation algorithm. In the end, ALS recommends the items with the highest corresponding user-item values.

Examples of hybrid systems that combine CF and content-based techniques are those of [27–30]. In [27], *Content-Boosted Collaborating Filtering* uses a content-based predictor to enhance existing data about a user and after that it uses collaborating filtering to provide personalized recommendations. Debnath *et al.* [28] use a *collaborative social network graph* to assign weights on attributes of content-based recommendation depending on how important they appear to be to the users; while Barbieri *et al.* [31] intertwine Bayesian methods with CF, in order to group users into "communities" based on their rating patterns. Another hybrid approach, which combines "social" (collaborative) filtering and content-based techniques, uses ratings and additional information about each movie, in order to *classify* movies as "liked" or "disliked" [30]. In [29], the authors predict a user's choices based on the choices of others, but also take into account her answers to system questions. Their recommendation approach uses *machine learning* and *cluster analysis*. Using a support vector machine for prediction, the system clusters the movies, selects movies from the dataset, and poses questions to the users. The answers help the system refine the user-relevant movie set to recommend.

Another interesting application of CF on recommendations is introduced in [32]. Authors use two popular collaborative filtering learning algorithms, *probabilistic Principle Component Analysis* and *Non-negative Matrix Factorization*, on their interactive story generation system, in order to compose a personalized story according to user story-telling preferences. Plot points are included into the story by taking into account user ratings on story parts she has already read, and ratings from other users.

Two recent works for CF are presented in [33, 34]. In [34], the authors try to face the recommendation in situations with binary relevance data. They introduce a new Collaborative Filtering method, called as *Collaborative Less-is-More Filtering (CLiMF)*, which is adapted to cases when only binary relevance data exists, e.g., the relationship of two users in a social network. By optimizing a well-known evaluation metric, the Mean Reciprocal Rank (MRR), this method models the available data. Authors define the Reciprocal Rank (RR) for a given recommendation list of a user, by measuring how early in the list is the first relevant recommended item. The connection between MRR and RR is that the MRR is the average of RR across all recommendation lists for each user. *CLiMF* optimizes a lower bound of the smoothed RR for learning the

model parameters, which are latent factors of users and items, and are used to generate recommendations. Experimental results show that the *CLiMF* is effective and scalable, and overtakes other state-of-the-art CF methods.

Shi *et al.* [33] are confronted with the problem of top-N context-aware recommendations for implicit feedback systems, handling this as a ranking problem in CF. In the cases of top-N recommendation systems, the quality of the recommendation lists that contain items of binary relevance can be quantified using Mean Average Precision (MAP). The authors propose a recommendation model for implicit feedback domains by directly optimizing MAP. Specifically, they introduce a generic CF model that is based on a generalization of the matrix factorization technique to manage context-aware recommendations. Using tensors, they extend the matrix factorization to *tensor factorization*, which extends the dimensionality of data. Therefore, that work proposes a new context-aware recommendation method using *tensor factorization for MAP maximization (TFMAP)*, which is designed to work with implicit feedback domains and optimizes MAP for learning the model parameters. Additionally, the authors propose a fast learning algorithm for the *TFMAP* exploiting properties of the Average Precision (AP) measure. Experiments demonstrate that *TFMAP* behaves better than some state-of-the-art context-aware and context-free approaches.

In [35], authors introduce a CF recommendation framework, the *OrdRec*, which doesn't treat user feedback as a simple number, but makes the assumption of the existence of an order among the observed feedback ratings. This allows users have different internal rating scales. The framework can be incorporated to CF algorithms that manipulate numerical values, making them able to manage ordinal values. An important characteristic of this technique is that it can predict a full probability distribution of the item ratings, instead of a single rating for that item. Furthermore, authors exploit an ordinal scale in order to map user actions during their interaction with the system. The specific framework was adapted to a matrix factorization CF model called SVD++, and turns its predictions into a probability distribution over an ordered set of values. So, given the predicted rating distributions, the method tries to rank and recommend items to each user. Experiments show that this approach is accurate and efficient.

Liu and Yang in [36] propose a CF method, "EigenRank", that faces the item ranking problem by modeling user preferences derived by user ratings. They measure the similarity between users based on the correlation of their item rankings, rather item ratings, and rank items based on the preferences of similar users. This ranking-oriented approach directly addresses the item-ranking problem without the need of rating prediction phase. Also, authors introduce two algorithms for item ranking: a greedy algorithm and a novel random walk model.

Liu *et al.* with their work [37] propose the probabilistic latent preference analysis (pLPA) model for ranking predictions by modeling user preferences with respect to a set of items rather than the ratings of individual items. User preferences are inferred from user's observed ratings and items are modeled by a mixture distribution. Furthermore, the pLPA model has the ability to capture user preferences and community structures. It is based on a statistical model of the data and allows learning and inference on large-scale data. Here, authors try to model user preferences in the form of paired comparisons, which are binary responses that indicate whether a user likes an item more than another. Experiments demonstrate that the method performs better than several CF approaches.

3.2 Preference Elicitation techniques

Utility theory-inspired *preference elicitation (PE)* techniques have also been tried out in various recommendation domains (see, e.g., [38, 39]). PE tries to collect user preferences in order to construct the user's *utility function*. To do so, most Preference Elicitation techniques set queries to the user asking her to evaluate, order, or constrain potential system outcomes; while others try to translate a user's interaction with the system to preferences. After inferring user preferences, all Preference Elicitation methods make recommendations that maximize the user's utility function. Example of a recommendation system employing PE is the "Apt Decision Agent", which attempts to learn user preferences in the domain of rental real estate [39]. In that system, the user provides the features of the apartment that she prefers and the system responds with a set of apartments. Subsequently, that set is pruned based on the importance of each apartment's feature independently, since the user can state if and how much she is satisfied about each feature. Although we consider utility theory to be extremely useful, we decided against using Preference Elicitation in our work, as it would require the user to participate in a lengthy and tiresome procedure of answering questions and setting constraints.

Work in [38] is not related to recommender systems, but it is a representative example about how PE techniques make decisions and how they exploit the *VPI* theory on the decision making process. Those systems have to make decision with partial user's utility information about posible outcomes, so authors propose an approach that uses a prior probability over user's utility function, perhaps influenced by other similar users. After that, they select queries for the utility elicitation based on the VPI measure of each query, in order to maximize user's utility.

In [40], authors propose a new technique for the computation of dominated queries; queries with inferior user utility. The system is based on the assumption that the number

of possible user utility functions is finite, so the number of the query questions can be limited. This method handles the limitations, like expensive computations and long list of query suggestions, that a conversational system can face, by using a new technique for the computation of dominated queries. Authors show that the number of suggested queries is reduced and the computation of those queries is simplified. Additionally, the system performs close to optimal, even if it doesn't contemplate the true user profile. It is shown that progressively expanding the number of profiles contemplated by the system, the utility of the final recommended products can be increased.

This technique models a product using a n-dimensional Boolean vector with n attributes, since authors consider products that can be described by their features. When a Boolean vector's attribute is equal to 1, it means that the product has that feature. In this paper, only products with one discrete feature were considered. Also, queries are presented as Boolean vectors too, and each attribute is equal to 1, when the user is interested in products with the corresponding feature, and equal to 0, when she has not declare interest on it. A user utility function (user profile) is a vector of weights. The attributes model the importance that the user assigns to each product's feature. The greater the weight, the greater the importance. The recommender system suggests to the user how to revise queries in order to selects products with higher utility. Those revised queries are the *AdviceSet*. Queries with expected lower utility are not suggested and are considered "dominated".

3.3 Bayesian techniques

Most approaches, that try to incorporate Bayesian theory in their recommendation systems, are just limited to the use of *Bayesian networks* combined with the aforementioned techniques, in order to predict user preferences or ratings.

For example, Ono *et al.* [41] have proposed a user's movie preference modeling, using a *WWW questionnaire* and a *Bayesian network* to model the collected data. They use a small-scale interview to design a large-scale questionnaire in order to collect the necessary data. After that, they select effective variables to construct the Bayesian network and infer user preferences.

Rendle *et al.* [42] introduce a generic optimization criterion BPR-Opt for personalized ranking and present a generic learning algorithm for optimizing model with respect to BPR-Opt, the LearnBPR. They apply their method to two state-of-the-art recommender models: matrix factorization and adaptive kNN. In this work, the ranking has to be inferred from implicit user behavior. In this case, only positive observations are available.

Regarding BPR-Opt, it is derived through Bayesian analysis using a likelihood function and a prior probability for the model parameter. Experimental results indicate that this approach overtakes the standard techniques for matrix factorization and kNN for the task of personalized ranking.

In [43], authors introduce a probabilistic method for personalized recommendations. Their system, called Matchbox, exploits content information and CF information from previous users behavior for predicting the value of an item for a user. In this paper, users and items are represented by feature vectors. The model can learn user-item preferences following three alternatives: (a) observation of a rating of each user on specific item, (b) observation of binary user preferences and (c) observation of ordinal ratings. Additionally, authors propose a message passing technique for efficient inference, which uses a combination of Expectation Propagation and Variational Message Passing.

The work most relevant to ours is perhaps that of [44] and [45], which model users and items using a common representation. Both approaches use vectors for modeling purposes. [44] proposes a CF technique and uses a Bayesian network to support the inference procedure. They model items using *feature vectors*, and users using vectors which describe user's liking for each item feature. These vectors are incorporated in the Bayesian network in order to predict user ratings about unrated items using her previous ratings on other items. On the other hand, [45] introduces a Bayesian hierarchical model for content-based recommendations. They model each user as a *k-dimensional vector* sampled randomly from a Gaussian distribution, and items as k-dimensional *feature vectors*. Then, they incorporate data *from all users* in the hierarchical model in order *to predict a label-rating* of an item for a specific user.

3.4 Other approaches

There exists a multitude of systems which use semantics and social choice theory for making recommendations. Szomszor *et al.* [46] use movies *folksonomy* to enrich the current knowledge with descriptions of movies and interests of users. The system creates user profiles using folksonomy-generated movie *tag-clouds*, which reflect user interests. Folksonomies are taxonomy-like structures that emerge when large communities of users collectively tag resources, and tag-clouds are sets of keywords which depict user interests. The system uses user ratings on movies in addition to tag-clouds in order to recommend movies. Mukherjee *et al.* [47], on the other hand, developed a web-based movie recommendation system that uses *voting theory* for movie rankings, and *text-based learning*

based on *semantic movie features* (like their plot summaries) to provide recommendations. Moreover, users can pose queries to constrain the results by specifying values on some features.

In [48], the author presents a website which uses *trust* in *social networks* for making movie recommendations. The website generates predictive personalized ratings based on a *trust inference algorithm*, which employs ratings of other users on a movie and trust among users, in order to calculate the rating of a user for that movie. Liu *et al.* [49] have developed a movie social network that makes use of an *hierarchical framework* for personalized recommendations based on weekly rankings. This approach replaces traditional rankings with a prior knowledge-independent hierarchical recommendation scheme, and interactively produces personalized movie synopses for previewing. Also, the system recognizes and exploits associations among movies, to facilitate user navigation through them.

Elahi *et al.* [50] encounter cold-start and new user/item problem using Active learning (AL), which presents items to the user and asks her to rate them. Those items are selected in order to reveal user's interests in the best way and to improve the quality of the recommendations. Here, authors propose a novel AL rating request strategy that exploits the knowledge of the user's personality to predict the items a user will have an opinion about. Furthermore, this technique borrows from psychology a model called Five Factor Model (FFM) or Big Five dimensions of personality. Authors incorporate their approach into a context-aware recommender system that recommends places of interest to mobile users. This implementation takes the personality of a specific user as input for a matrix factorization model in order to predict what items the user should be requested to rate. So, using improved AL strategies, the method is able to provide personalized rating requests even to users with few or without ratings. Experiments show that the proposed AL method increases the recommendation accuracy and the number of the ratings acquired from the user.

3.5 Our approach

The recommendation technique we present in this work differs to all aforementioned approaches in many ways. First of all, we neither set questions to the user, nor use textual information regarding an item so as to elicit user preferences. Moreover, we do not rely on any kind of user classification or other users' inferred preferences, but attempt to fine-tune recommendations over time for each specific individual. That is, we progressively build a user type for each individual, which gradually converges to the real one. In contrast to most social networks, social choice-inspired and CF approaches,

we *do not* attempt to estimate user ratings; but just recommend the item which matches the user preferences more closely.

Though to some extent conceptually straightforward, to the best of our knowledge an approach such as ours has not been used before in the literature. It is a *simple* yet *generic* approach that combines elements from various techniques. The idea of modeling the types of *both* the user and the object under recommendation *by a probability distribution of the same form* is a novel one. Representing users and items as *complete probability distributions* over a collection of features, instead of as, e.g., vectors of point-values corresponding to specific features' weights, enables the implicit inference of *latent* features, or *hidden and otherwise unrepresentable feature mixtures, combinations, and interconnections*. Additionally, our translation of "demo" ratings into weights for guiding the sampling process used during Bayesian updating is innovative, and allows us to build a *temporary user type* that captures both current user mood and long-time preferences simultaneously. In our implementation, we recommend only one item, which better matches user preferences, in order to offer better user experience, but our algorithm can easily recommend a *top-N* group of items, if so requested.

An important contribution of our work is the recommendation with emphasis on the understanding of user preferences and the adaptation on them. We built our algorithm based on this characteristic, while most works in the literature do not. Some works do not take user preferences explicitly into acount [30, 32, 34, 35, 42], but they are limited to exploit ratings and correlations between users. On the other hand, there exist methods, which try to integrate the user preferences modeling procedure in their algorithms. Methods like [36, 37, 43–45, 50] use simplistic feature vectors and other simple techniques to model user preferences. Additionally, regarding PE approaches [38–40], they are based on user preferences in a unique way. They exploit utility functions in order to calculate users utility and infer their preferences.

Furthermore, unlike our approach, all of the aforementioned approaches need to predict ratings, either as a simple numeric value or as a probability distribution over ratings. They do perform simple or more advanced "learning", since they have to understand correlations between users and items, while some of them attempt to learn user preferences. Also, all the methods that try to model user preferences, update their models after every single user interaction.

At this point, it should be mentioned that, in the bibliography, there is no explicit, clear distinction between completely personalized methods and not. For example, CF methods like [27, 32] and methods that exploit data from previous users [37, 43], describe their recommendations as "personalized", because they use a *user model* in order to

recommend items. Instead, we produce completely personalized recommendations, since we only exploit past observations from the *current* user to recommend her an item.

Finally, our method is not susceptible to external influence by massive influx of biased users. By contrast, other methods have to introduce explicit (heuristic) terms, which they use to account for user bias in user ratings [51]. Moreover, we are able to capture *temporal effects* caused by the acquisition of ratings sequentially over time, while other methods like *BellKor* [52] need to make ad-hoc decisions or use heuristics in order to take temporal effects into account. Specifically, *BellKor* tries to predict future ratings by fitting the previously observed ratings, trying not to overfit the observed data. This technique uses tunable constants in order to control its algorithm. Also, it incorporates effects which are not caused by user-item interaction, using suitable predictors that however need to be modeled accurately; e.g., it attempts to capture temporal effects introducing matrix factorization with temporal dynamics.

3.6 Challenges

The research on recommender systems is confronted with numerous challenges. Here we try to summarize the challenges researchers in the domain face during the implementation of a recommender system. We now list several such challenges, identified in [1]; we took these into serious consideration when designing our approach:

- *Generic deployment*: A generic recommendation approach has the ability to be deployed in various recommenders recommending items from different natures. That is, the system can recommend, depending on the application domain, items, which vary from movies to share portfolios. Generality is guaranteed via the appropriate modeling of users and items. In our case, we design a generic modeling approach for users and items, which both are modeled in the same form using multivariate Gaussian distributions.

- *Scalability*: This might be the most important feature of a recommender system. The system must be able to manage large-scale data, since a database of a typical recommender contains hundreds of thousands of items for recommendation and millions of users. Experiments show that our technique can produce accurate recommendations even in very large datasets.

- *Diversity*: The user usually needs to explore her preferences and taste items that she never thought she likes. This is possible, if the recommendation approach recommends her items, which differ in some way. We succeed to recommend diverse items exploiting sophisticated exploration techniques.

- *Long-term and short-term preferences capture*: As "long-term" user preferences are defined the preferences that have been inferred during user-system interaction counting from the beginning. Instead, "short-term" preferences are those, which enclose current user likings. A recommender should model both long-term and short-term preferences in order to make valuable recommendations. In our case, we model long-term preferences by gradually building a main *user type* and short-term by a *temporary* user type. The temporary user type represents the *mood* of the user during current session. Exploiting both user types, we are able to produce recommendations based on both long-term and short-term user preferences.

- *Exploration vs Exploitation*: This is a mayor challenge for the modern recommendation approaches. This problem is related to the "mentality" of the recommender and its decisions, concerning whether to recommend items that are known to be accepted and highly rated by the user or to perform a kind of exploration recommending not-yet-tasted items in order to gather more information and offer better recommendations. In order to tackle this issue, we exploit advanced exploration algorithms (Bayesian and VPI exploration), which effectively balance the exploration and exploitation.

- *Proactive operation*: A proactive recommender is a recommender that makes recommendations, which are not directly associated with the user feedback. In this case, the system has the ability to interpret implicit user preferences through its interaction with the user. In our approach, we model users as multivariate Gaussians and update their forms exploiting user ratings. This fact allows our algorithm to capture user preferences that are related to latent item features and reinforces the proactive recommendations.

- *Transparency*: A system has to leave the user unaware about the recommendation process. The user just needs recommendations that she likes without further non-useful information. Our method performs a quite simple user-system interaction procedure, requesting user ratings and recommending single items.

- *Time factor*: An important feature of a recommender is the quick responses to user requests. Because of that, we apply a fast and lightweight recommendation process and the user receives a recommendation after some fractions of a second.

- *Privacy*: The privacy of user personal information is of great importance in the domain of information systems. A recommender system collects personal information and using this is able to build a complete and accurate user profile. This user profile must be stayed away from unwanted processing. Regarding this, we succeed to create safe user models that keep secure all user information. With

the exploitation of multivariate Gaussian distributions we can model user preferences in such a way that only our system is able to process them. That is, user information is incorporated to a Gaussian, so it is implicitly stored and it can be processed by our system only. Therefore, we secure user preferences and prevent someone from eliciting explicit conclusions about them.

Chapter 4

A BAYESIAN RECOMMENDATION PROCESS

Here, we describe our Bayesian recommendation system in detail. As mentioned, we were motivated in its design by the *"you are what you consume"* concept, which means we use the same distributions to model both users and items. Below, we define the form of the distributions we used for modeling items and user preferences. We describe the way a *temporary user type* is built based on specific *demo types*, and how that user type guides the decision making process in order to recommend the appropriate item. The user preferences are enclosed in an *overall user type*, which is the principal regulator of our system. Our agent is able to perform various additional action selection methods. These methods are a *VPI-based* and a *Boltzmann exploration* one, and are presented in detail in this chapter. Furthermore, we present the way we tackle large databases by clustering items in groups given their similarity. Finally, in this chapter we describe the application of these ideas in the movies domain.

4.1 User modeling

The choices for representing *item* and *user* types are key to the success of our recommendation system, as the agent needs to be able to infer a user type based on item types and user ratings. In this work, we present and apply a novel modeling technique using multivariate Gaussian distributions to model both items and users. This enables our agent to easily perform Bayesian exploration and update her beliefs about the user

preferenses. Because of that, she can also infer about latent [37] item features and build a more accurate user model.

4.1.1 Item/demo type

Each *item type* is modeled as a multivariate Gaussian, and each variable of the Gaussian corresponds to a specific item attribute (e.g., specific movie genre). The probabilities are distributed over ratings, which are provided in some scale of choice. In our system, we define *item types* to be k-dimensional Gaussians, corresponding to the k most important item attributes.

Now, let us describe the exact form (and attribute values) of a multivariate Gaussian representing an item. The *mean* is essentially determined by the ratings of the item, so it is an 1×k vector which contains values equal to that item's average rating—under the assumption that the item rating corresponds to every attribute available. Of course, one cannot be certain about the rating of an attribute not associated with the item (e.g., genres not associated with movies), but this is taken care of by the way we construct the covariance matrix.

Specifically, an item's k×k covariance matrix is constructed as a *diagonal covariance matrix* [53], assuming that the item attributes are independent of each other. Each element on the diagonal is associated to an attribute, and the element's value depends on whether the item is associated with that attribute or not. We assume that the uncertainty about the rating of the actual attributes of the item is small, and thus set the values of the corresponding elements to 1. In contrast, one has high uncertainty about those attributes *not* associated to the item, thus we assign a $\sigma^2 = 20$ for those diagonal elements (see Appendix A, for more details). Finally, note that a demo associated to a particular item has a *demo type* that is exactly the same with that of the corresponding item.

4.1.2 Beliefs as user type

The Gaussians for the *user* and *demo-based user* types have the exact same form as that of items. The agent, however, needs to store and update beliefs about these types. As already mentioned, these beliefs take the form of Normal-Inverse Wishart (NIW) priors, which can then be easily manipulated to infer the corresponding types as k-dimensional Gaussians, and match them to items as required by the system.

Demo-based user type is a temporary (current session life-cycle) type, and composed by the *demo types* and the corresponding ratings through Bayesian updating. This type is

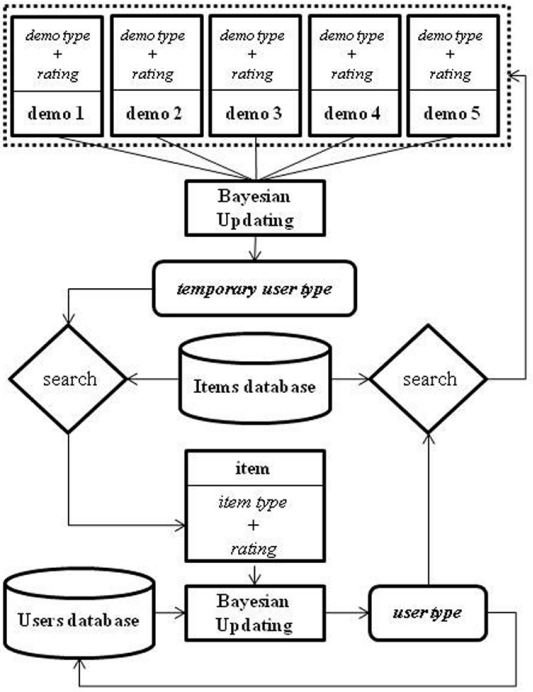


FIGURE 4.1: The overall recommendation process.

responsible for the selection of the appropriate recommended item. On the other hand, the overall *user type* is permanent and gradually built up based on the recommended item types and the user ratings on them. This user type guides the selection of the demo items during every recommendation session.

4.2 The recommendation process

We have already described the main intuitions and key ideas of our approach in the previous chapters. Here, we provide a more detailed picture of our system's architecture and overall recommendation process, also summarized in Figures 4.1 and 4.2.

When a user enters the system, she is shown and asked to rate a number of 5 demos.[1] The agent decides which demos to show based on her knowledge about the user—i.e., the stored *user type*. Thus, she presents demos that most closely match user preferences, as these have been embodied in the *user type* so far. In the case of *a new user*, the agent shows again 5 demos, but now those have few common features, since they are—out of

[1]The number of projected trailers (demos) used in our experiments was decided to be "5", as a tradeoff between receiving enough information for accurate inference of user preferences, and avoiding user distraction and frustration [54, 55]. Of course, when testing the agent with actual human subjects in real time, the number of trailers shown and their length will definitely have to be adjusted according to the users' reaction and perceived frustration levels. However, our main objective in this work is to evaluate the potential of our novel Bayesian user modeling method. Thus, while adjusting for specific real-time usage concerns in a given domain is important, current parameter choices are adequate for confirming the soundness of this approach.

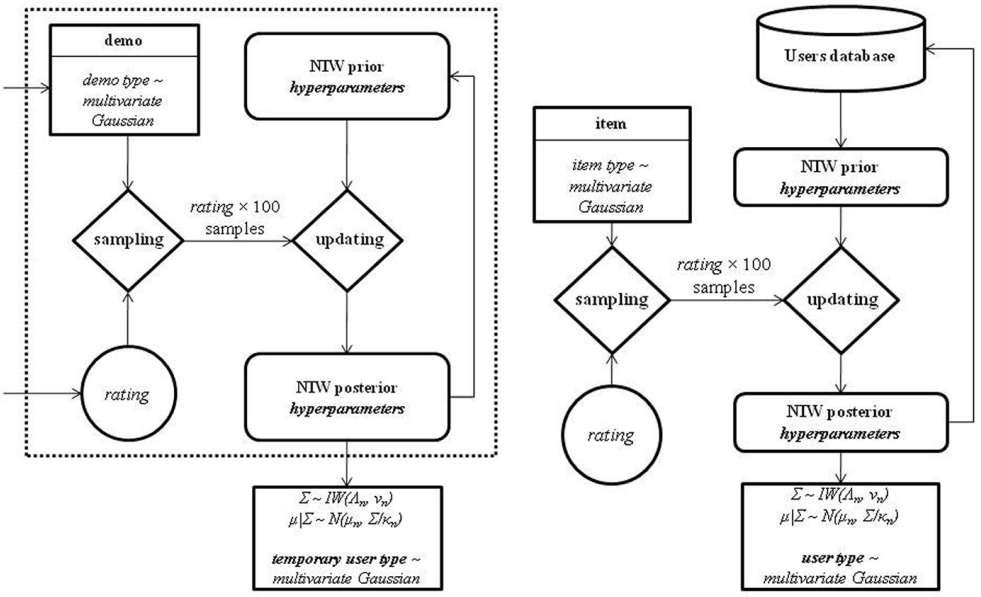


FIGURE 4.2: (a) Deriving the *demo-based user type*. (b) Deriving the *user type*.

necessity—selected randomly. Then, the user provides a rating for each demo, and a *Bayesian updating* process is used to create the *demo-based user type* from rated demo items.

Bayesian updating actually takes place two times during the recommendation process, firstly to infer a *temporary*, *demo-based user type*; and, secondly, to update the overall *user type*. The process takes into account user ratings regarding demos or items shown to the user, and, given the ratings, samples the respective *demo* or *item types* the appropriate number of times; and uses these samples to come up with an (updated) system-inferred type regarding the user. This can be done given the fact that the user type ("demo-based" or not) *is of the same form as the items' type*, and via the proper use of conjugate NIW priors. The intuition is that, in the absence of *explicit* data regarding the *user type* (which is the model whose parameters we need to infer), we utilize user ratings as an *implicit* way to indicate the extent to which a user "associates" herself with the demo or item she is shown. Therefore, we can then sample the model this item (e.g., a trailer or a movie) originates from a number of times (proportional to the degree of the user's liking of that item), and treat these samples as new evidence for the Bayesian updating process. It is important to note that, to ensure the efficiency of the update, the number of samples must be high enough. To achieve this, we multiply the *rating* with 100, thus guaranteeing that hundreds of samples are used during an update.

In more detail, the user observes and rates the first demo item (e.g., watches and rates

the first trailer). Then, the system takes *rating*×100 samples from the *demo type* distribution of that demo item, and updates the posterior's *hyperparameters* based on those *samples* and the *hyperparameters* of the *demo-based user type* prior (which has to be an *uninformative* one [16]). Subsequently, the user watches and rates the second demo and the system samples that *demo type* and updates the *hyperparameters*, in the same way as for the first one—but now the posterior of the first step has become the *informative* prior of the second step. After all these updating steps, the *hyperparameters* of the NIW distribution can be used to estimate the parameters (μ, Σ) of the multivariate Gaussian, which models the *demo-based user type*. Figure 4.2(a) summarizes this process.

Now, when the system has the final estimate of the *demo-based user type*, it has to recommend a specific item to the user. To do so, it must search the database to find the item matching that *demo-based user type* best. To this end, it uses the *Kullback-Leibler (KL) Divergence* criterion [56]. The *KL-divergence* between a Gaussian t (modeling, e.g., a demo-based user type) and a Gaussian i (corresponding, for example, to some item type), of dimension d each, is given by:

$$KL(t \parallel i) = \frac{1}{2}\log \mid \Sigma_t^{-1}\Sigma_i \mid + \frac{1}{2}tr((\Sigma_t^{-1}\Sigma_i)^{-1}) \\ -\frac{d}{2} + \frac{1}{2}(\mu_t - \mu_i)^T\Sigma_i^{-1}(\mu_t - \mu_i) \tag{4.1}$$

where Σ_t, μ_t, Σ_i and μ_i are the distributions' parameters, and $tr(\cdot)$ is the *trace* of the corresponding matrix [57]. The lower the *KL-divergence* between two Gaussian-modeled types, the greater their proximity. This is the item (e.g., the movie) recommended to the user. Notice that this item *might not* be one of the demo items shown to the user before recommendation. This effectively corresponds to a *Bayesian* rather than a heuristic *exploration* approach in this domain: the system employs its probabilistic beliefs regarding (long-term) *user type* and (short-term) *demo-based user type* in order to come up with a recommendation; and makes recommendations based on beliefs given sampled evidence, rather than, e.g., greedily matching a demo item.

Subsequently, the user puts a rating on the item (after, e.g., watching the movie). Then, the system performs another Bayesian update, resulting to a new *user type* estimate. The Bayesian updating of the *user type* is similar to the *demo-based* one; but now the *rating*×100 samples collected are drawn from the distribution corresponding to the item the user was finally recommended and rated *only* (Figure 4.2(b)). Note that *user type* and corresponding beliefs are stored for future use, unlike the *demo-based user type* which lasts only for the current session. Thus, in the case of a future system use by a known user, the hyperparameters of the posterior from which the current *user type* was inferred, will be the hyperparameters of the *(informative)* prior for deriving the next *user type*. Moreover, as stated, the updated and stored *user type* serves as a prior to guide

the system to select demos. This is also done via using the *KL-divergence* minimization criterion.

4.3 Alternative action selection methods

The "basic" Bayesian method described above behaves "greedily" wrt. beliefs when selecting a demo or item: it just picks the one with minimum KL-divergence from the Bayes-updated user type. We also devised alternative *action exploration* techniques, to assess whether these would lead to improved recommendation decisions. *Value of perfect information* method assigns a value on each available item and recommends the one that maximizes the agent's reward, while *Boltzmann* selection recommends the item with the higher probability.

4.3.1 VPI-based selection

The first of these techniques attempts to account for the expected *value of perfect information (VPI)* [2, 3, 19] characterizing the various "agent actions"—i.e., in this domain, potential recommendation choices. The rationale behind this technique is that a choice has a value not only because of its immediate benefit to the user, but also because of the information it relays with respect to user preferences. Intuitively, if a recommendation and observed user reaction lead to a reassessment of what the user really prefers, then this recommendation action carries a high information value. Thus, the *VPI exploration* technique, adapted for the recommendation domain, attempts to "simulate" various *alternative* user type reactions to future recommendations; calculates the value of information gained from these reactions when compared to the thus far modeled user type's expected behaviour; and averages out these results to come up with an *information gain estimate* that is used to "boost the desirability" of the recommendations.

In more detail, let us suppose that the reward that some user derives from a specific item is $reward = f(KL\,divergence) = M - \lfloor KL\,divergence/M \rfloor$, where M is the maximum rating the user can give to a movie (e.g., $M = 10$). We define as i_1 the item with the highest reward r_1 for the user (given the model built for her so far), and as i_2 the second best with reward r_2. Consider now an item i selected for recommendation to a user type j (possibly different to the type modeled for the user so far), and assume that this in fact represents the "actual" type for the user—therefore, presenting this user type j with an item will lead to "perfect information" regarding the value of this item for the user. (Of course, this is just an assumption the method makes, but allows it to compute a value of information estimate, via "sampling" user types and averaging out

their behaviour.) Assume that this recommendation results to a user reward of r_i for item i. We distinguish the following two cases (in all other cases, the gain due to perfect information is 0):

1. if i coincides with the item considered best for the user so far ($i = i_1$), then the *gain* from presenting the user with this item is either: *(a)* $gain_i^j = 0$, for $r_i > r_1$ (since we derived no new information from presenting the user with this item: the "perfect" information we got by fixing the user type to the assumed "actual" user type j coincides with what we had already estimated—i.e., that i_1 is the "best" item for the user); or *(b)* $gain_i^j = r_2 - r_i$, for $r_i < r_2$ (since we "learned" that the item i is actually worth less to the user than the item considered so far to be only second-best).

2. if i does not coincide with i_1, then the *gain* from presenting j with i is: *(a)* $gain_i^j = 0$, for $r_i < r_1$ (since we only observed i to be sub-optimal, as expected); or *(b)* $gain_i^j = r_i - r_1$, for $r_i > r_1$ (since we now "learned" that i was actually better than the item considered best so far).

Given this, our *VPI exploration* method works as follows: We estimate the modeled-so-far *user type*[2] by "integrating out" our NIW prior, as described in Section 2.3 (Eqs 2.6 & 2.7). Subsequently, for this integrated-out user type, we discover the i_1 item with the highest reward, and the second-best item i_2. We then sample from our NIW prior a number of s Gaussians, which represent "alternative" user types. (In our experiments, we set s=10.) After that, we calculate for each j user type (corresponding to one of the s sampled Gaussians), the rewards from presenting it with every i item, and compute the corresponding $gain_i^j$ values, as outlined above.

The next step is the calculation of the *average gain*, $\overline{gain}_i$, for presenting an item i to our user; this is computed by averaging out, over all s samples (i.e., over all user types j sampled), the $gain_i^j$ gains estimated for this item. Finally, the *VPI* method selects (and presents) the item I that maximizes the sum of the integrated-out user type's reward and the corresponding gain from selecting I:

$$I = \arg\max_i \{V_i = reward_i + \overline{gain}_i\} \tag{4.2}$$

4.3.2 Boltzmann selection

We also tried the well-known *Boltzmann exploration* [58] method to select the appropriate demo or item. At each time step t, this method assigns a selection probability to all

[2]This can be a long-term or a short-term user type, depending on whether we are attempting to select demos or actual items.

available actions:

$$Pr(i) = \frac{e^{U_i/T}}{\Sigma_{j=1}^{n} e^{U_i/T}} \tag{4.3}$$

where $T = c \cdot \alpha^t$, with c a constant and $\alpha < 1$. An action i is chosen with probability proportional to its utility U_i; and with T decreasing over time, exploration is progressively reduced.

We tried Boltzmann selection with two different utility functions in our experiments. The first of these methods, simply called *Boltzmann*, employs a U function equal to the KL-divergence between the user type and the item. The second one uses as U the metric of the VPI method described above, i.e., sets $U_i = V_i$, where V_i is the quantity in Eq. 4.2. We call this method *Boltzmann-VPI*.

4.4 Action selection methods: the details

As mentioned before, our main algorithm, *Bayesian exploration*, is by its nature greedy wrt. beliefs. That is, when the agent has to select 5 demos or an item to recommend, she always chooses those demos/items that better match the *user type* or the *demo-based user type*, respectively. Regarding VPI exploration, when it is applied, there exist significant changes at the selection phases based on the theory of the subsection 4.3.1. During the demos selection phase, the agent samples from the *user type* 10 Gaussians necessary for the VPI procedure, while when VPI is also applied for item selection, 10 additional Gaussians are sampled from the *demo-based user type* this time. We should remark that these 10 Gaussians are not directly relevant to those ones from the demos selection phase. Continuing, *Boltzmann exploration* alters the "basic" demos/items selection by assigning a probability to all available options. When the Boltzmann exploration is applied for demos selection, the 5 demos with the highest Boltzmann probabilities are selected to be shown to the user; and when it is applied both to demos and item selection, the demos are selected as before, while the recommended item is that with the highest Boltzmann probability among all available items for recommendation. Finally, the *Boltzmann-VPI exploration* method is composed from the combination of VPI and Boltzmann explorations on demos or item selection and leads to a new user utility function (where the VPI metric V_i is used as the U_i utility factor) for the Boltzmann probability:

$$Pr(i) = \frac{e^{V_i/T}}{\Sigma_{j=1}^{n} e^{V_i/T}} \tag{4.4}$$

As a remark, we also tested an additional method called *BRX (Bayesian Random Exploration)*, which didn't display essential differences from the other variants (see Appendix B, for more details on this method).

4.5 Tackling large databases

To avoid the exhaustive search of demos and items inside large databases (10's of thousands items), we apply clustering methods on the set of stored items. The clustering technique we use is the *Bregman hard k-means* clustering, adapted to the *Kullback-Leibler Divergence* [57]. This approach reduces significantly the number of comparisons between *user* and *item types*. For example, if the database contains 1000 items, which are clustered into 10 clusters of 100 items on average each one, the system has to carry out 110 comparisons: 10 to find the appropriate cluster, and 100 to find the appropriate item; instead of 1000 comparisons without clustering. Moreover, clustering is executed offline, and only occasionally: once when the database is created initially, and every time a new item is added in it. Note that after a few hundreds of recommendations leading to items being removed from their clusters, the need for reclustering arises. This is not a problem, since it can be executed off-line.

As mentioned above, the clustering was implemented using the *Kullback-Leibler hard k-means*, a variation of the *Bregman hard k-means* clustering [57]. The algorithm begins by selecting at random n items to be the clusters' centers, and then follows the principles of the standard *k-means* algorithm—but in this case the comparison between the multivariate Gaussians is performed with respect to the *Kullback-Leibler (KL) Divergence*, and the *stopping criterion* is met when the difference between the *Kullback-Leibler losses* of two succesive iterations goes below 0. The *Kullback-Leibler loss function* is given by the following formula:

$$\sum_{l=1}^{n} \sum_{p_i \in C_l} KL(c_l \parallel p_i)$$

(where c_l is a cluster center, and p_i an item in that cluster).

4.6 Application to the movies domain

We chose to apply our method to movie recommendations, an important domain that has inspired much research in recommendation systems. Here, *items* correspond to *movies*; *demos* correspond to *movie trailers*; and we use the term *trailer-based user type* to refer to a *demo-based user type*. Each *movie type* is modeled as a *multivariate Gaussian*, with each of its variables corresponding to a movie genre. The probabilities are distributed over ratings, which are provided in some scale of choice (e.g., $1-10$ or $1-5$). To create these multivariate Gaussians, we were inspired from the *MovieLens* (http://www.grouplens.org) datasets, which are actually used in our experiments below. These datasets comprise of ratings provided by thousands of users on thousands of

movies. In the *MovieLens* dataset containing 1 *million* ratings, movies are characterized by 18 specific genres. We therefore define *movie types* to be *k-dimensional Gaussians* with $k = 18$ in our experiments, involving real MovieLens ratings.

The form and attribute values of such a Gaussian representing a movie or a trailer, are described below. The *mean* depends on the overall MovieLens rating of the movie, so it is a 1×18 vector which, on each dimension, contains value equal to movie's rating. We are assured about the rating of a genre associated with the movie, but what about the rating of a not associated genre? As mentioned above, this uncertainty is modeled via the values of the covariance matrix elements.

A movie's 18×18 covariance matrix is constructed as a *diagonal covariance matrix* [53], assuming that each element on the diagonal is associated to a genre and the movie genres are independent of each other. We set $\sigma^2 = 1$ for the diagonal elements corresponding to actual movie genres. Contrariwise, we model the uncertainty about those genres *not* associated to the movie assigning a $\sigma^2 = 20$ (an empirically chosen value that is high enough so as to not "disturb" the distribution) to corresponding diagonal elements. As an example, consider the movie 'Movie' with an overall rating of 7, whose genres are *action, sci-fi, thriller*; its type is the following: the mean vector's entries all carry a value of 7, while its covariance matrix is a diagonal one, with diagonal entries corresponding to *action, sci-fi* and *thriller* variables having a value of 1, and all other (diagonal) entries being equal to 20.

In addition, the Gaussians for the overall and trailer-based user types have the same form as that of movies (i.e., 18-dimensional Gaussians). The agent needs to store and update beliefs about these user types; and as already stated, these beliefs take the form of Normal-Inverse Wishart (NIW) priors, which can be easily manipulated to infer the corresponding types as 18-dimensional Gaussians, and match them to movies as required by the system. Appendix A provides more details on the representation of the user and item models.

Chapter 5

EXPERIMENTS

We ran several sets of experiments to validate our algorithm, which we call *BayesYouLikeIt (BYLI)*, and its variants, with very encouraging results. We gathered aggregated results from various databases (simulated and real-world), and observed the behavior of specific users on some datasets. Firstly, we tested the algorithm's scalability in large simulated databases. After that, we compared *BYLI* and its variants with the *LSMF* state-of-the-art method in databases containing real-user movie preferences data, and studied the behavior of specific users performing *BYLI*. Additionally, we examined the recommendation abilities of our agent in datasets containing movies with mainly low ratings, to verify whether the few preferred movies can be accurately identified; and tested the method's adaptability on changes of user preferences. Finally, we showed that the use of demos helps the system to accurately capture user's mood. The experimental results demonstrate the efficiency and stability of our algorithm and show that we are directly comparable to state-of-the-art approaches.

5.1 Scalability and mood capture

To test the scalability of our technique, we initially ran experiments with *simulated* users on databases with *10,000*, *20,000*, and *40,000* randomly generated movies. Simulated movies representation and ratings were based on those of the popular *IMDB* website (http://www.imdb.com). Specifically, we defined *movie types* to be 16-dimensional Gaussians, corresponding to the 16 most common genres available in *IMDB*. Also, we generated 50 simulated users to interact with the system. Each simulated user is characterized by its *real user type*, a 16-dimensional Gaussian distribution with a random mean in the range of $[1 - 10]$ on each dimension, corresponding to *IMDB* ratings; and

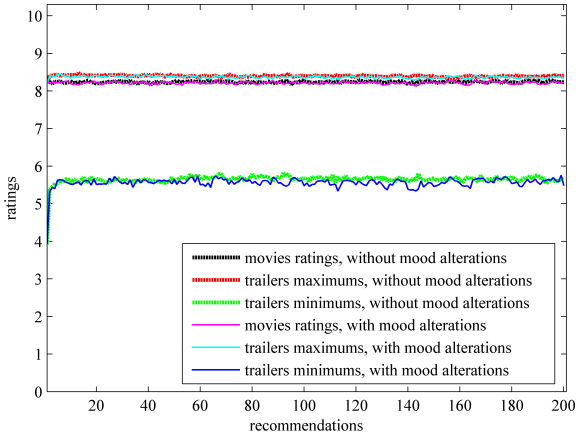


FIGURE 5.1: Comparison of *average per recommendation ratings* from 50 *simulated* users on 200 movies for 10 runs, and *average range of corresponding trailer ratings*, with and without mood alterations simulation.

a covariance matrix which is *spherical*, assuming that the *real user type* is confident regarding the degree to which user likes each genre [53].

As stated in Section 4.5, we apply clustering methods on the set of stored movies in order to avoid the exhaustive search for trailers and movies within large databases. This reduces the number of comparisons between a *user type* and stored *movie types*. In our experiments, the movies were grouped in clusters with 1,000 movies each, on average (e.g., 20,000 movies clustered into 20 clusters), based on their similarity using the *Kullback-Leibler hard k-means*, a variation of the *Bregman hard k-means* clustering [57].[1] We also had to create a *rating function* to be used by the simulated users to assign ratings. The function exploits the *KL divergence* between the *real user type* and each *trailer* or *movie type* in our system—the less the divergence, the higher the rating: $rating = f(KL\,divergence) = 10 - \lfloor KL\,divergence/10 \rfloor$ (where $0 < KL\,divergence < 100$).

In Figure 5.1, we can see that *BayesYouLikeIt* recommends movies that constantly receive high ($> 8/10$) ratings, when tested on the *20,000* simulated movies database. The experimental results for the *10,000* and *40,000* movie databases are of the same quality. We also observe that, in an average iteration, the user is shown trailers whose ratings range is about 3 degrees wide; and whose average *maximum* rating is *almost always* higher than the (average) rating received for the movie shown during that iteration. This is because Bayesian exploration: *(1)* does not *necessarily* return a movie that matches the best trailer shown in an iteration; while, at any given iteration, there are still on average many "good" movies whose trailers can be shown; and *(2)* enables the system to actually project trailers of *movies already shown* at any iteration, and thus

[1] Note that after a few hundreds of recommendations leading to movies being removed from their clusters, the need for reclustering arises. This is not a problem, as it can be executed off-line.

trailers of "preferred" movies might be shown to a user again—since these are just used to detect the *current mood* of the user. At no point is a *movie* already shown to the user actually recommended again; however, *BayesYouLikeIt* might re-use *trailers* derived from the "believed" user type to infer the *temporary* user type (though in practice this will occur only rarely in a large database).

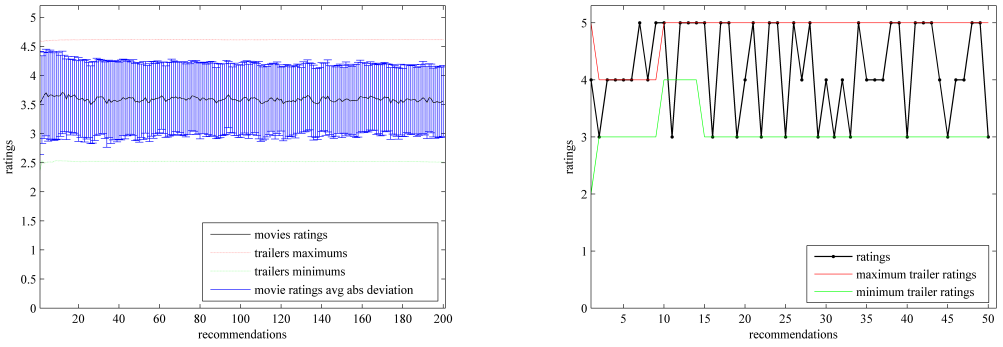
We also evaluated the ability of our agent to capture temporary changes of a user's *mood*. We simulate such mood changes by periodically changing the mean values of the *"real user type"* multivariate Gaussian distribution (which represent the preferences of the assumed "real" user). Specifically, after every 10 recommendations, we randomly change the mean of each variable of the corresponding Gaussian (via sampling a normal distribution over the range of $[1-10]$). The "mood changes" last only for 5 recommendations, and then the real user type returns to its original form. Figure 5.1 confirms that our method is able to successfully capture user's mood, since it constantly recommends movies that are subsequently rated highly. Indeed, our method's performance appears to be robust, and is not negatively affected by mood changes.

5.2 Experiments on real-world data

Furthermore, we ran experiments to test our agent on data coming from *real users*, that reflect actual human preferences and behavior. This kind of data also offer an ideal testbed for comparison with other well known techniques. We used the *MovieLens* dataset with *1 million ratings* from *6,040* users on *3,952* movies. Thus, there is no longer a need to generate simulated users and ratings: we can now have a user's rating on a movie (or trailer) by just referring to the real-world dataset.[2] Nonetheless, an existent drawback of this dataset is that the contained users are biased. Also, no clustering was used in these experiments, since the number of the available movies was small. The movies inside the system's database are modeled as in Section 4.6, with a movie's *mean* being the average rating it has in the MovieLens dataset. Ratings are integers in $[1-5]$.

We executed different sets of experiments with *200 recommendations each*, using all aforementioned *variants* for trailer/movie selection. For each such experiment, we employed *5 sets* of *100 users* each, consisting of real users with *200 or more* ratings in the MovieLens dataset. In some detail, we took the histogram of the number of ratings per user in the dataset, and sampled 100 different users for each experimental user set in accordance to that distribution. This ensures that there is no bias in the ratings used

[2]We should remark that since our methods do not predict ratings, the use of comparison metrics like *Root Mean Square Error (RMSE)* or *Mean Absolute Error (MAE)* is not possible. Instead, we compare the methods wrt. average per recommendation ratings.



(a) (b)

FIGURE 5.2: (a) *Average per recommendation ratings*, *average range* of corresponding trailer ratings and *movie ratings average mean absolute deviation* for *BYLI - Boltzmann-VPI on trailers & movies* across all users. (b) Typical run ratings of a *single*, *real* user on 50 movies, and *range* of corresponding trailer ratings.

as input in the experiments. Also, each experiment involving a specific user set was executed for *10 independent runs*. We then sequentially produced 200 recommendations to each individual user; and calculated the *average per iteration ratings* assigned by the users, across all user sets and experimental runs.

Now, as explained in Section 4, *BayesYouLikeIt* progressively builds the *user type*, and recommends the movie that best matches the *trailer-based user type* at each iteration—or the one selected according to some variant exploration criterion. Indeed, in addition to the "basic" *BayesYouLikeIt (BYLI)* algorithm, we ran experiments exploiting the VPI and Boltzmann exploration methods described in Section 4.3. We used these criteria in two different ways. First, we employed them only during the trailers selection phase of the algorithm; and second, during both the trailers and movie selection phases. Boltzmann exploration parameters were separately set to: $c = 1, \alpha = 0.5$ or $c = 1, \alpha = 0.9$ or $c = 12, \alpha = 0.9$ (a more relaxed setup); and $t \leq 3$, with $t_0 = 0$ and $t_{i+1} = t_i + 1$.

We compare our algorithm with an established recommender engine, built on the *Apache Mahout* machine learning library, adapted to yield the *Myrrix* software.[3] The algorithm used by the engine is the *large, sparse matrix factorization (LSMF)* method [25, 26], implemented using a modified version of the *Alternating Least Squares (ALS)* [25] matrix factorization algorithm. *LSMF* requires a number of ratings to be entered in the system, so that matrix factorization can be performed. Specifically, the experimental setup uses a widely used implementation of the *ALS* matrix factorization algorithm, called Myrrix, which is based on the Mahout machine learning library. Myrrix can be trained with a comma-separated file containing ⟨user id, item (movie) id, rating⟩ tuples at its initialization phase. Furthermore, it can accept ratings through the *preference* Myrrix

[3] http://mahout.apache.org; http://www.myrrix.com/design

Methods	**Average ratings**
LSMF - pretrained	3.6848
LSMF - untrained	3.6540
BayesYouLikeIt (BYLI) - Bayesian exploration	3.6112
BYLI - VPI on trailers	3.5968
BYLI - VPI on trailers & movies	3.5911
BYLI - Boltzmann on trailers, c=1, a=0.5	3.5920
BYLI - Boltzmann on trailers & movies, c=1, a=0.5	3.5592
BYLI - Boltzmann on trailers, c=1, a=0.9	3.5922
BYLI - Boltzmann on trailers & movies, c=1, a=0.9	3.5522
BYLI - Boltzmann on trailers, c=12, a=0.9	3.5919
BYLI - Boltzmann on trailers & movies, c=12, a=0.9	3.5518
BYLI - BVPI trailers, c=12, a=0.9	3.5968
BYLI - BVPI on trailers & movies, c=12, a=0.9	3.5911
Random movies recommendation	3.5736

TABLE 5.1: Comparison of average ratings of all methods (across all user sets, recommendations and experimental runs).

REST API method. Hence, depending on the state of the recommendation engine (*pretrained* with existing data or *untrained*), we initialize Myrrix with or without an initial training file. Next, for a set of iterations and a number of given users (randomly chosen given specific restrictions) we call the *recommend* REST API method of Myrrix, and check it against known ratings of each user, which constitute the test set. The actual rating of a specific user taken from the test set, is used to update the recommendation model.

In the experimental results, we include the results of a totally *random recommendation process*, just to prove that our method does not behave randomly. In this setup, there are no trailers shown to the user and a randomly selected movie is recommended to her each time.

Figure 5.2(b) helps us gain further insights in *BayesYouLikeIt* behavior when recommending movies to a single (real) user. We can see in the figure that, while learning, *BayesYouLikeIt* might occasionally recommend a movie which the user ranks *lower* than the trailers projected to her during that iteration; however, sometimes the algorithm might also recommend a movie that the user prefers to all trailers shown to her during that iteration. Over time, *BYLI* returns movies that receive consistently good ratings.

This fact is confirmed by the *average per iteration ratings* results of Figs. 5.2(a) and 5.3. Fig. 5.2(a) shows that *BayesYouLikeIt* manages to recommend movies that are on average highly rated by the users.[4] Moreover, average ratings deviation can be observed

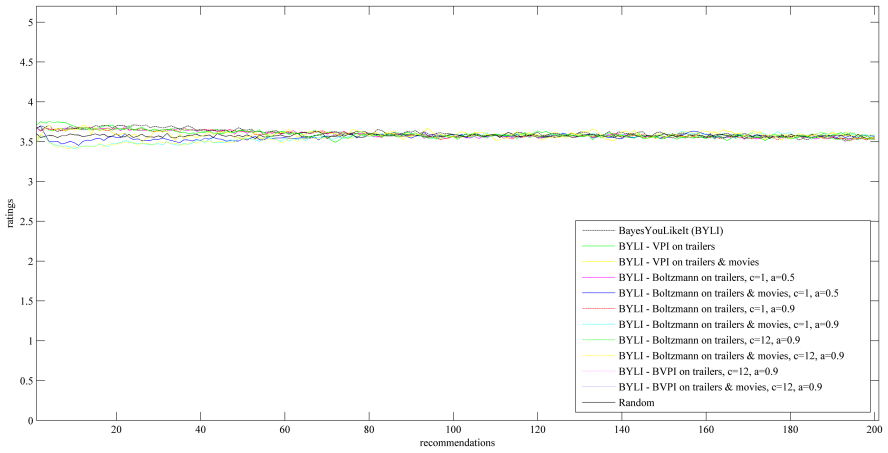
[4]Fig. 5.2(a) depicts the *BYLI-Boltzmann-VPI on trailers and movies* variant; results are similar for other *BayesYouLikeIt* variants.

to decrease over time, demonstrating an ability to "learn" and progressively become more confident on assessment of users' preferences. We note that the over time decrease in ratings average absolute deviation is a bit sharper for the variants using *VPI* (or *Boltzmann-VPI*) on both trailers and movies. This figure also shows the (average) maximum and minimum trailer ratings received by users (per iteration). Maximum trailer ratings are consistently higher than ratings received for movies recommended, for the same reasons as in the experiments with simulated users above.

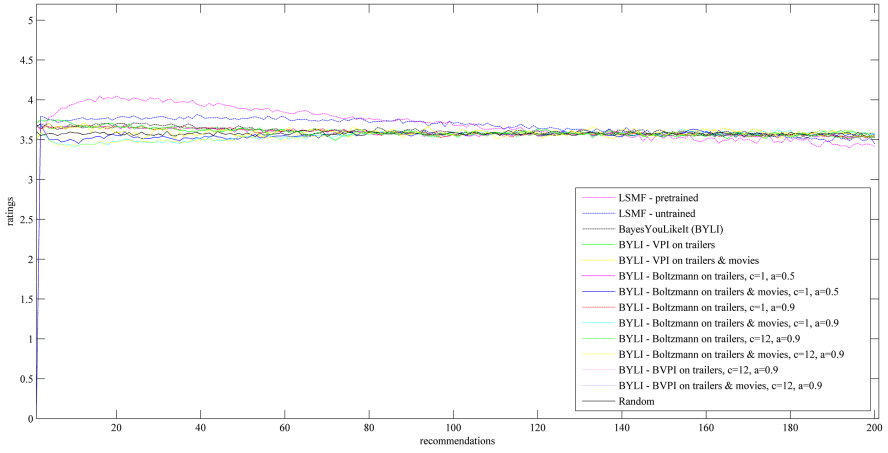
In Fig. 5.3(a), we observe that the VPI and Boltzmann exploration methods (and their combinations) are successfully intertwined with our "basic" Bayesian algorithm, since all *BYLI* variants return recommendations that receive consistently high ratings from the users. Here, we should highlight that the VPI alternatives follow a quite smooth behavior similar to this of "basic" Bayesian. Average ratings across all iterations, depicted in Table 5.1, confirm that all *BYLI* variants perform very similarly, with the "pure" *BYLI* method achieving a slightly better average score than the rest. As we mentioned before, we examined a combination of VPI and Boltzmann explorations resulting *Boltzmann-VPI (BVPI)*, which we used in our experiments. The results of this variant for all Boltzmann parameters value settings were identical with the corresponding results of the VPI variant, e.g., the *Boltzmann-VPI on trailers* with $c = 1, \alpha = 0.5$ or $c = 1, \alpha = 0.9$ or $c = 12, \alpha = 0.9$ gives results equal to those of *VPI on trailers* variant. This effect is caused by the use of the VPI method metric as the Boltzmann probability's U function. This utility function is quite strong and the VPI method completely influences the demo/item (trailer/movie) selection procedure. Because of that, we do not extensively demonstrate the results given by the Boltzmann-VPI variant.

Figure 5.3(b) then compares our method to *LSMF*. Note that *LSMF* was tested under two assumptions in these experiments. First, on the assumption that it is completely unaware of *any* user preferences at system start (this is marked as *"LSMF-untrained"* in the figures); and on the assumption that the system is *pre-trained* on ratings data from all other (i.e., the $6,040 - 100 = 5,940$ non-picked) users in the database (*"LSMF-pretrained"*). These results clearly demonstrate that *(a)* the average user ratings value of *BayesYouLikeIt* recommendations is about $3.5 - 3.7$ (out of 5); and *(b)* although our method does not depend on other users' ratings in order to recommend movies to a user, ratings received are almost indistinguishable in average quality to those received when using the *LSMF* technique.

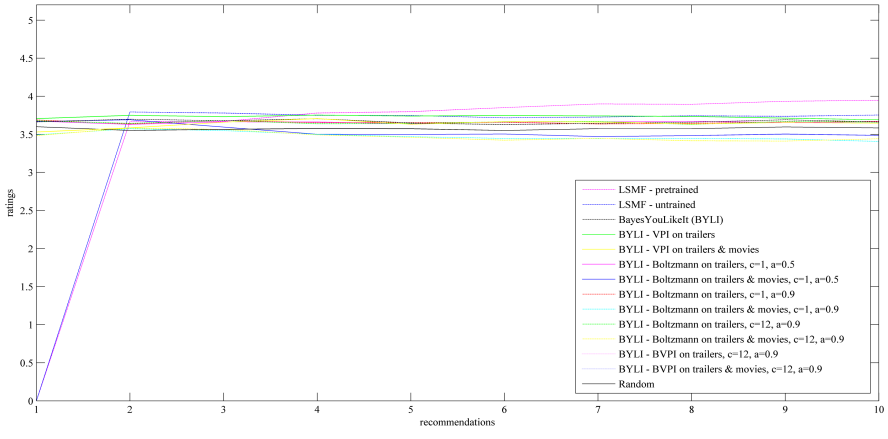
Moreover, we can observe that *LSMF*, being a CF method, has to first collect a small number of ratings from users in order to be able to return good recommendations; and thus cannot respond to a user request in a meaningful manner during the very first recommendation. This is clearly visible in Fig. 5.3(c). In contrast, our personalised



(a)



(b)



(c)

FIGURE 5.3: (a) Behavior of *BayesYouLikeIt* and its variants. (b) Comparison between *LSMF* and *BYLI*. (c) Methods behavior during the first 10 recommendations. All subfigures depict *average per recommendation ratings* across all user sets and experimental runs.

Methods	**user A**	**user B**	**user C**
BayesYouLikeIt (BYLI) - Bayesian exploration	3.5580	3.7690	3.8060
BYLI - VPI on trailers	3.5700	3.7380	3.8035
BYLI - VPI on trailers & movies	3.5810	3.7390	3.6880
BYLI - Boltzmann on trailers, c=1, a=0.5	3.5645	3.7710	3.6370
BYLI - Boltzmann on trailers & movies, c=1, a=0.5	3.5495	3.7655	3.4465
BYLI - Boltzmann on trailers, c=1, a=0.9	3.5640	3.7785	3.6515
BYLI - Boltzmann on trailers & movies, c=1, a=0.9	3.5530	3.7555	3.4465
BYLI - Boltzmann on trailers, c=12, a=0.9	3.5640	3.7720	3.6545
BYLI - Boltzmann on trailers & movies, c=12, a=0.9	3.5475	3.7600	3.4490
BYLI - BVPI on trailers, c=12, a=0.9	3.5700	3.7405	3.8105
BYLI - BVPI on trailers & movies, c=12, a=0.9	3.5830	3.7370	3.6885
Random movies recommendation	3.5535	3.7655	3.5215

TABLE 5.2: Comparison of average ratings of each user (across all recommendations and experimental runs) for each variant.

method can *immediately* suggest a good movie to the user. After the very first iterations, *LSMF* is able to exploit knowledge of "similar" users ratings, and performs strongly—but as the number of "preferred" movies drops (since our dataset contains a "closed set" of movies), its performance drops too (Fig. 5.3(b)). *BayesYouLikeIt* does not start as strong, but due to progressively converging to real user types, its performance is quite *stable* throughout all recommendations. As a final remark, running on MATLAB on a 2.20 GHz / 4 GB RAM PC, it takes a *BayesYouLikeIt* agent only about 0.3 sec on average to recommend a movie to a user.

5.3 Recommending to specific users

In order to have a better insight about the specific user's confrontation by our agent, we randomly selected 3 users from the MovieLens dataset and executed for them all the *BYLI* variants. These users belong to the 1*st* of 5 available *100-users sets*. The experimental setup was exactly the same as in the main experiments on MovieLens, i.e., 200 recommendations for 10 independent runs for each user; and the results are consisted by the calculated average per iteration ratings assigned by each user throughout the experiment. With these experiments, we can distinguish the differences between the BYLI variants executed on specific user, but also, the differences between the average ratings assigned by different users taking recommendations by specific variant. The overall average ratings of the 3 users for each variant are shown in Table 5.2.

Based on Figure 5.4, we draw the conclusion that every BYLI variant treats in different way each user, exhibiting however a "constant" behavior. In other words, although the users' average rating plots differ comparing with one another, they still follow a

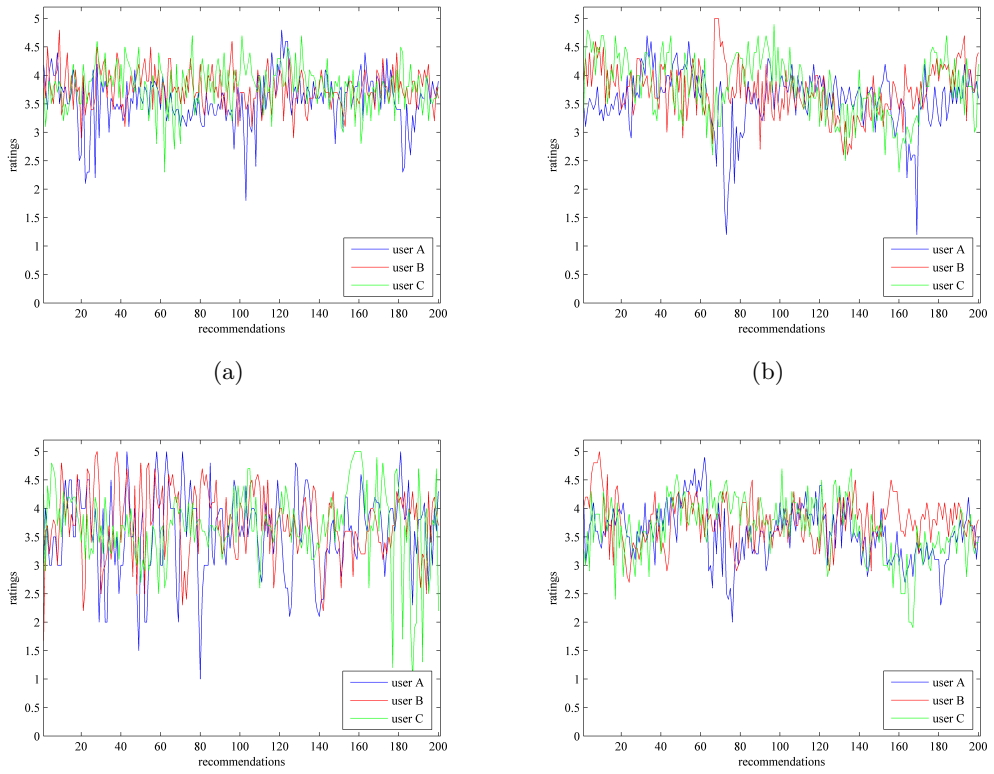
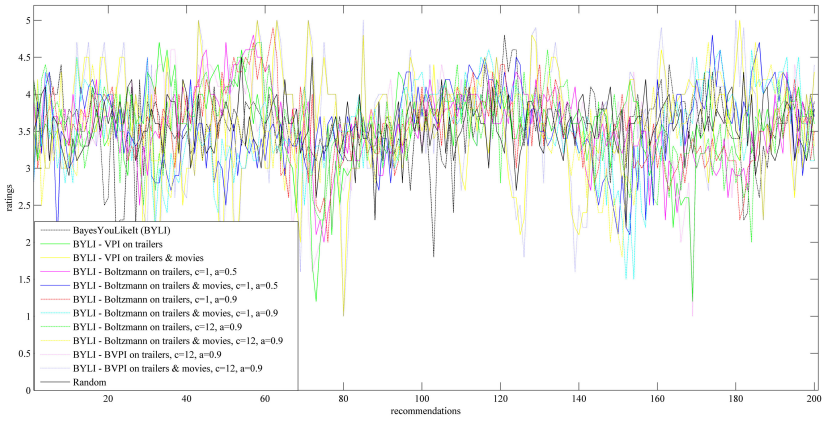


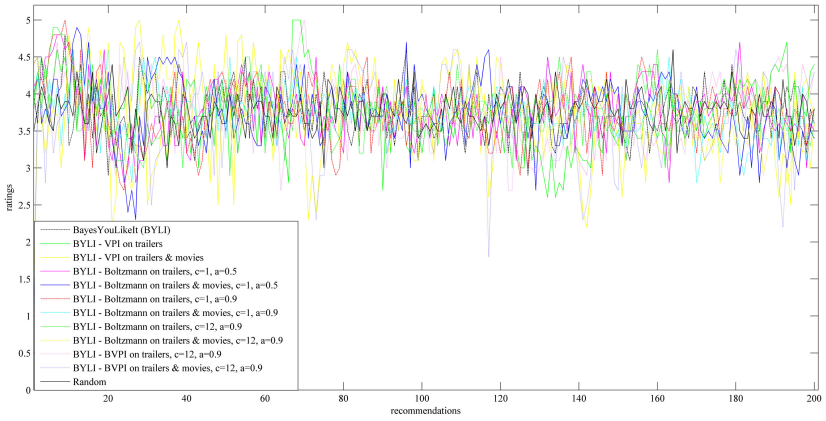
FIGURE 5.4: *Average per recommendation ratings* over all runs for 3 different users: (a) *basic* BYLI, (b) BYLI-*VPI on trailers*, (c) BYLI-*VPI on trailers & movies*, (d) BYLI-*Boltzmann on trailers*.

common "baseline" for each variant. Additionally, in those figures we see that the agent has the ability to follow the pattern of each user through different BYLI variants, i.e., the average ratings of *user B* illustrate smoother plots than the corresponding of *user A* in all variants.

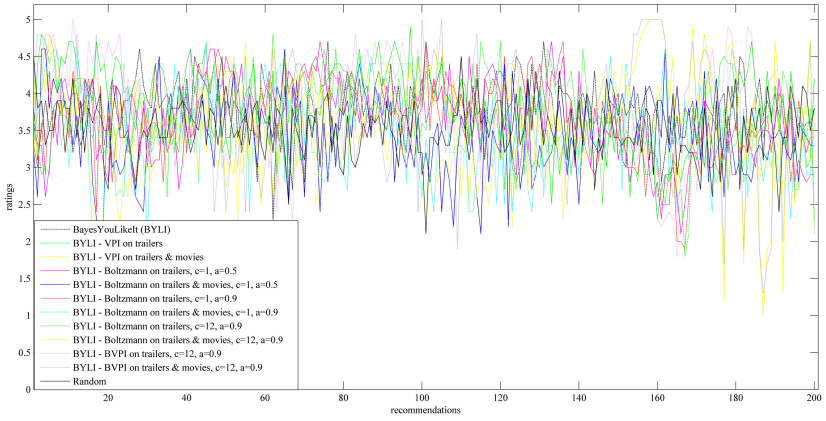
Figure 5.5 also shows that all variants apparently behave differently for every user. Furthermore, considering our agent's "average" behaviors on those 3 users, we observe that even though our agent demostrates similar behaviors around the average, there are obvious differences between them. For example, *users A & B* rated on average the recommended movies with ratings around 3.6, but for *user B* the average ratings range is narrower than the range of *user A*. These conclusions are also "summarized" in Fig. 5.6, where we can see the *overall average ratings* of the observed 3 users. In that figure, the overall average ratings of *user A* are sometimes significantly lower than the ratings of the other 2 users. Also, we can easily see that in the first 20 recommendations, *users B* & C rated highly the recommended movies with ratings above 4, which means that our agent has the ability to recommend, in the very first recommendations, movies that users like.



(a)



(b)



(c)

FIGURE 5.5: Behavior of *BayesYouLikeIt* and its variants: (a) user A, (b) user B, (c) user C. All subfigures depict *average per recommendation ratings* across all experimental runs.

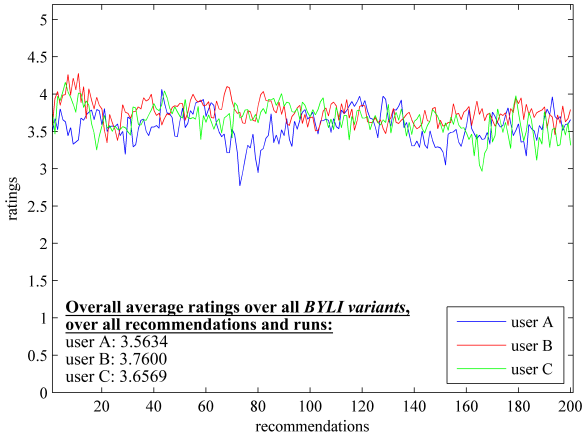


FIGURE 5.6: Comparison of the *average per recommendation ratings* of 3 users across all variants for 10 runs.

5.4 Recommending from a dataset with mainly low-rated movies (real-world ratings)

It is crucial to safely certify that our method recommends, from the early stages of the recommendation process, movies that users like them. Because of that, we designed an experiment that allows us to examine if our agent has this ability presenting suitable behavior. We used once again the MovieLens dataset. We created a set of users selecting them based on the following criteria. Each user has rated at least 100 movies and a maximum of 35 of them have received ratings 4 or 5 by that user; these movies we call *"preferred"*[5]. Additionally, the average of the *assigned-by-that-user* ratings on those movies is less than the MovieLens overall average movies rating, e.g., a *user X* has rated 115 movies, so the average of these 115 ratings is less than the MovieLens average rating. The MovieLens overall average movies rating was calculated to be 3.5816. This user set consists of 27 available users. Then, we took a subset of those users with a maximum of 20 *"preferred"* movies (8 users only) and executed the experiment again in order to test the method in an even more extreme case.

In this experiment, each user was sequentially recommended 50 movies in 10 *independent runs* using: *(a) LSMF*-pretrained, *(b)* the basic algorithm, *(c)* 2 VPI variants (*VPI on trailers & VPI on trailers & movies*), *(d)* the *random* recommendation process; we chose the basic BYLI and the 2 VPI variants, since these are the main versions of our algorithm. Then, we calculated the overall average per recommendation ratings over all experimental runs of all users. Further, the purpose of this experiment was the testing of agent's ability to recommend movies for better user experience, although

[5]Because of the constraints, the minimum possible number of *"preferred"* movies for a user was 10.

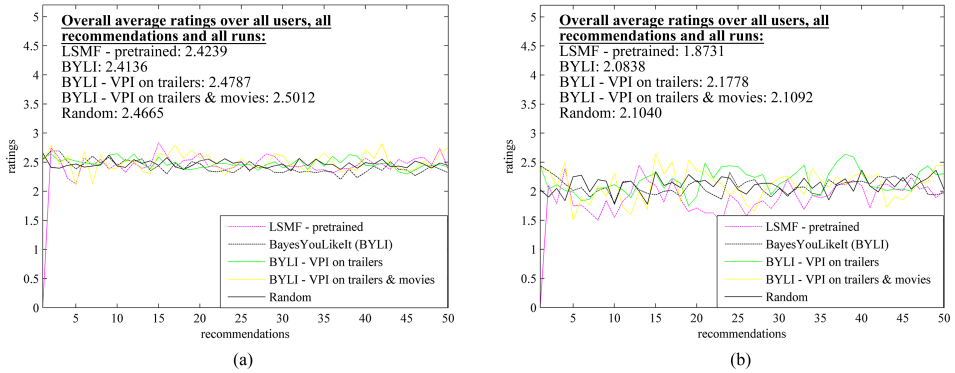


FIGURE 5.7: *Average per recommendation ratings* over all users and all runs for 50 recommendations: (a) users with less than 35 *"preferred"* movies, (b) users with less than 20 *"preferred"* movies.

the experimental dataset of each user mostly contains movies that she does not like them. Conclusions can also be drawn about the pace of agent's adaptation on user's actual preferences. The results of the experiment are presented below and show that our method can quickly understand user's liking and recommend her the desired movies.

Figures 5.7(a) and 5.7(b) show the results of the LSMF, the basic BYLI algorithm and the VPI variants. We can see that the behaviors of all variants are quite similar with the behaviors of the corresponding variants in the experiments described in Section 5.2, although the average per recommendation ratings are lower, which was expected because of the nature of the dataset. *The VPI variants behave better than both the LSMF and the basic BYLI, which means that they are able to infer faster the user preference.* It is important to comment the fact that the LSMF performs worse than all BYLI variants, especially in the case of Fig. 5.7(b). Considering that the average rating of each user over all the movies in her movies database is much lower than the MovieLens overall average movies rating (3.5816), our method succeeds to receive overall average ratings close to those averages (Tables 5.3 and 5.4). Also, note that, the behavior of the random recommmendation process is incidental, since the movies used in this experiment receive on average very low ratings.

For better insight, we illustrate results from recommendations on individuals—we selected 3 specific users from the experiment's "low rating" user set: the *best* (*user A*) and the *worst* (*user C*) available, and a *mid* (*user B*) one (Table 5.3). In Figure 5.8, we can see that there exist significant differences among the three specific users' ratings in all cases. However, we can detect similar confrontation of those users by each variant, i.e., the basic algorithm (Fig. 5.8(b)) mainly receives by users A and B ratings in the range of $[1.5 - 4]$ over 50 recommendations, while the *VPI on trailers* variant (Fig. 5.8(c)) receives ratings in the range of $[2 - 4.5]$, which means that the VPI addendum helps our method to learn user preferences sooner.Regarding the case of *user C* (the *worst*

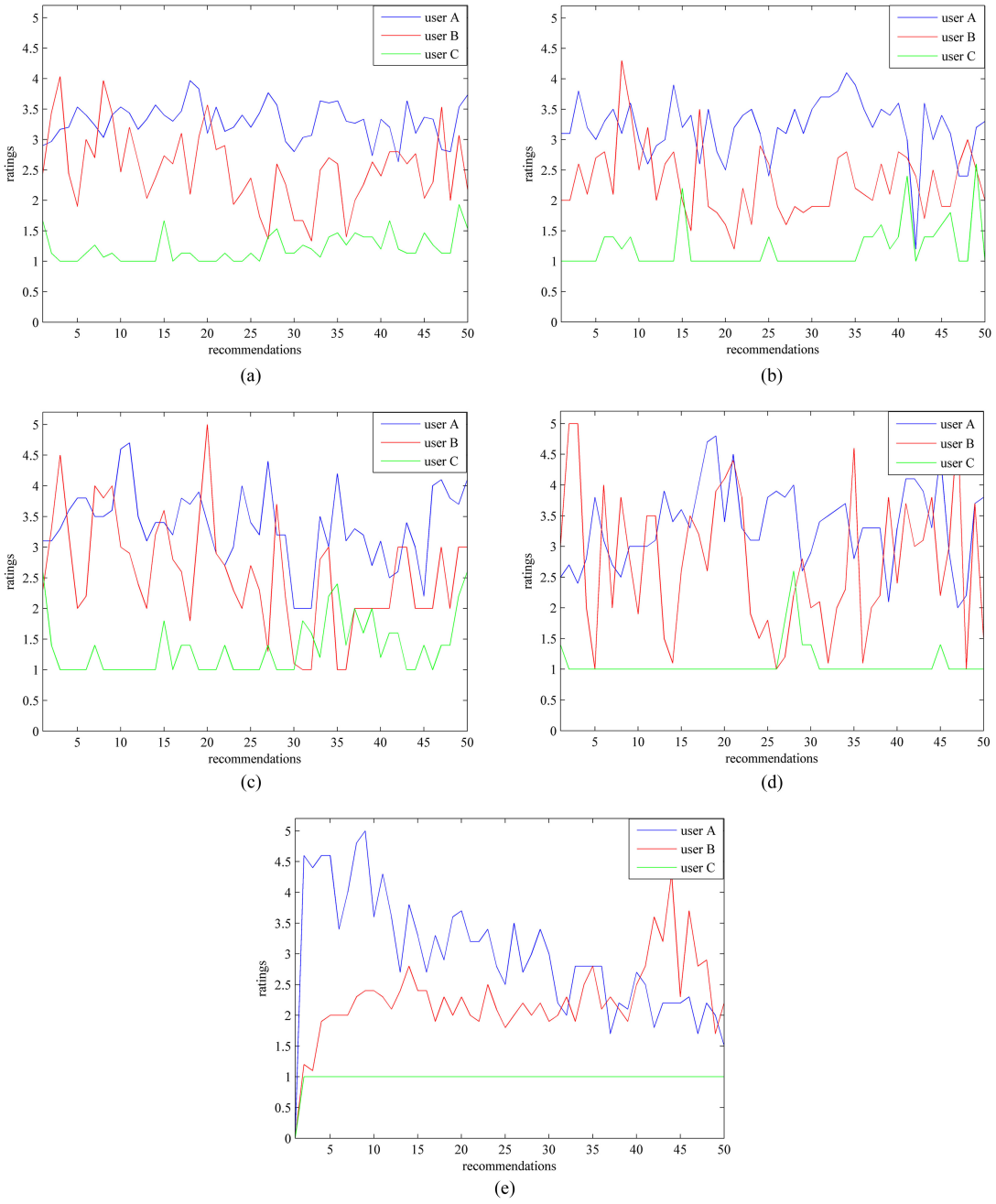


FIGURE 5.8: *Average per recommendation ratings* over all runs for 3 different users: (a) over all BYLI variants (*basic* BYLI, BYLI - *VPI on trailers*, BYLI - *VPI on trailers & movies*), (b) over *basic* BYLI, (c) over BYLI - *VPI on trailers*, (d) over BYLI - *VPI on trailers & movies*, (e) over *LSMF* - pretrained.

	user A	**user B**	**user C**
average rating	3.1000	2.5714	1.3043
total ratings	100	112	138
ratings ≥ 4	35	23	10

TABLE 5.3: The average database ratings (i.e., over all ratings provided by each of these users in the database) and the number of total ratings and ratings≥4 assigned by *users* A, B and C for the experiments of Section 5.4.

		user A	**user B**	**user C**
LSMF	*average rating*	2.9660	2.2540	0.9800
	average "preferred" per run	15.4	5.2	0
	ratio of recommended "preferred"	44%	22.6%	0%
BYLI	*average rating*	3.2020	2.3080	1.2040
	average "preferred" per run	17.5	5.4	2.3
	ratio of recommended "preferred"	50%	23.5%	23%
VPI on trailers	*average rating*	3.3500	2.5600	1.3480
	average "preferred" per run	20.1	9	3.9
	ratio of recommended "preferred"	57.4%	39.1%	39%
VPI on trailers & movies	*average rating*	3.3500	2.7240	1.0800
	average "preferred" per run	20.6	16	1
	ratio of recommended "preferred"	58.9%	69.6%	10%
Random	*average rating*	3.1000	2.6060	1.2880
	average "preferred" per run	17.4	10.7	3.4
	ratio of recommended "preferred"	49.7%	46.5%	34%

TABLE 5.4: The overall *average rating*, the average of the recommended "preferred" movies and the corresponding ratio over the available "preferred" for each user, over 10 runs and the executed variants.

one), we can see that even if our method receives higher ratings in the last stages of the experiment, it is still able to recommend the *good* movies of the dataset; in all the sub-figures of Fig. 5.8, the raise of ratings of *user* C begins mainly after the first half of the recommendation process. It is important to note that, in Figure 5.8(a), the average per recommendation ratings of *user* A are close to the overall average per recommendation ratings of the experiments on normal MovieLens datasets (Section 5.2), although her dataset contains mainly low-rated movies. Compared with the VPI variants, LSMF falls far, since it receives ratings above 3.5 only in the first 10 recommendations for the *best* user (*user* A); and learns too late the *mid* user (*user* B) — receives high ratings after the $40th$ recommendation. Considering the results of LSMF for the *worst* user (*user* C), we can state that CF methods are incapable to offer recommendations, when the user-item interaction data are few and the database contains a few good items.

Moreover, we took measurements about the number of the recommended movies that users like, i.e., the number of the movies that users rated with high ratings (4 or 5). The measurements were taken during the 10 *runs* of the 50 recommendations used in

the experiment, separately for each of the above 3 users. The results are presented in Table 5.4, and enhance our conclusion that the agent has the ability to quickly "learn" user's likings and recommend her "preferred" movies in the early stages of the recommendation process, even though the number of those movies is small. Considering that each user's dataset contains only a few good movies (35 for A, 23 for B, 10 for C), both basic BYLI and VPI variants recommend many of them until the end of the experiment's run. Interestingly, the 2 VPI variants recommend the 39%-70% of the available "preferred" movies and receive average ratings greater than the overall average ratings of the users' datasets. BYLI seems to be weaker than VPI in this case; this fact demonstrates the advantage we gain from the incorporation of the principled VPI heuristic in this setting. A remarkable conclusion derived from those results is that our approach has an advantage over CF methods, when the available data are sparse and full of low-rated items, and does not need overfitting in order to produce good recommendations. Both basic BYLI and VPI variants behave better and more stable than the LSMF, which cannot reach the overall average ratings of users' datasets, nor recommend high rate of good movies.

5.5 Recommending from a dataset with mainly low-rated movies (simulated ratings)

Following the experiment above, we designed a new one targeting the same goal, but with some variations in the experimental setup. In this case, we exploited the movies database and the users-movies correspondences provided by the MovieLens dataset, and replaced all user ratings on all movies. Now, each user's movies ratings inside the dataset follow a Beta distribution in the range of $[1-10]$ (the ratings' histogram of a typical user is shown in Figure 5.9), resulting an overall database average rating equal to 3.3775. We used 5 user sets consisting of 50 users each one; each user had rated at least 200 movies.

We simulated 200 sequential recommendations on each user for 10 *independent runs* and then, we calculated the overall average ratings as usual. For the purposes of this experiment, we tested the performance of 4 algorithms; our basic algorithm, the *VPI on trailers* and *VPI on trailers & movies* variants and a random recommendation process. Here, we try again to observe how quickly our method can learn a user model, and if can it recommend to the user the "good" movies that the database contains.

In Figure 5.10, we see the *average per recommendation ratings* over all runs over all users for the compared algorithms. The ratings are ranged around 3.5. They are low, but here we don't aim to high ratings; we aim to ratings higher in the very first recommendations than the next, which testify that our method is quickly adapted to user

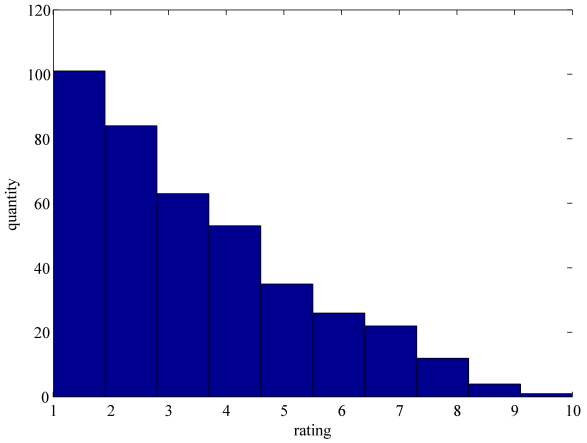


FIGURE 5.9: The histogram of the ratings for a typical user; experiments in Section 5.5.

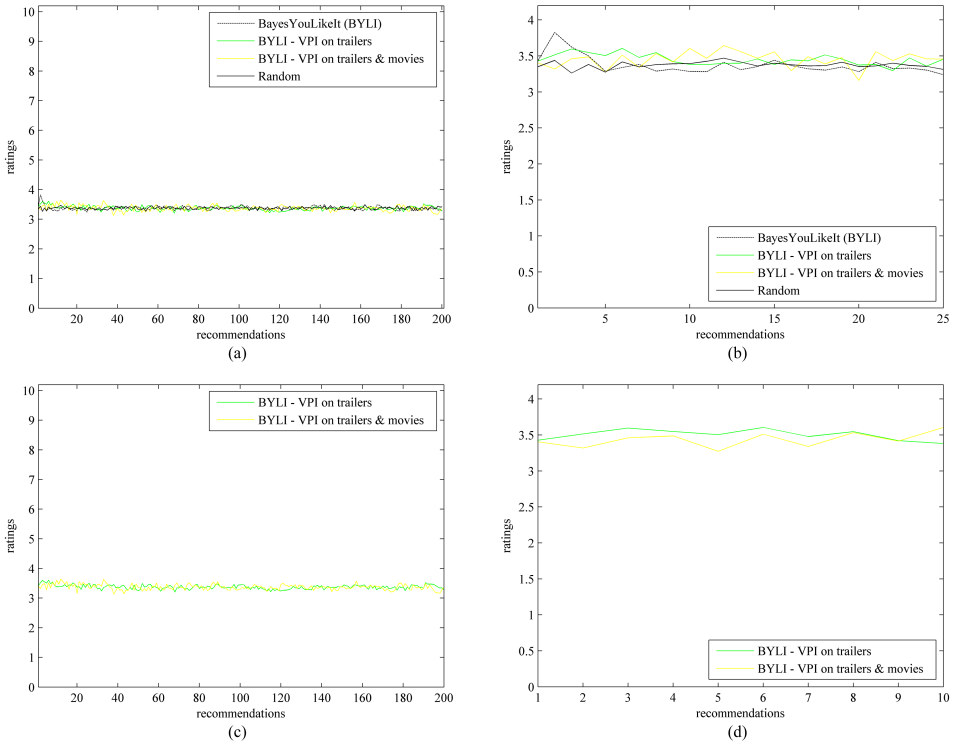


FIGURE 5.10: A. *Average per recommendation ratings* over all users and all runs: (a) 200 recommendations, (b) first 25 recommendations. B. *Average per recommendation ratings* over all users and all runs for the VPI variants (BYLI - *VPI on trailers*, BYLI - *VPI on trailers & movies*): (c) 200 recommendations, (d) first 10 recommendations.

preferences, and recommends her all the preferred movies in the early stages of the recommendation process. Figure 5.10(b) shows the overall average ratings only for the first 25 recommendations of the experiment. In this figure, we observe that the basic BYLI algorithm and the *VPI on trailers* variant overtake the 2 other methods in the first 5 recommendations. The fact that those 2 methods begin from higher rating levels and end to lower, proves that they recommend the "good" movies (movies that receive the higher possible ratings) first, and the others later.

We display the comparison of the 2 VPI variants in Figs. 5.10(c) and 5.10(d). The ratings plot of the *VPI on trailers* variant looks smoother than the corresponding plot of the *VPI on trailers & movies* variant. Furthermore, the first one recommends highly rated movies until the $10th$ recommendation, therefore it is sooner adapted to user likings, as illustrated in Figure 5.10(d).

5.6 Adapting to changing user preferences

Another major issue for a recommender system is whether it has the ability to detect preferences alternations and be adapted to them. Suppose there is a user, who used to use the system for some time and suddenly, she stopped the interaction for a long period. During use period, the agent learned the user's preferences and created her model based on the possessing beliefs. Someday, that user decides to request recommendations again by the system, but this time her preferences have radically changed and they no longer correspond to the agent's beliefs about her. The question is; can the system be adapted to this new situation and accurately adjust user's model in order to recommend her again movies that she likes?

In this experiment, we created a simulated user and a corresponding movies database. Required data about the user is her *prior user type*, which encloses agent's beliefs collected during the first period of use; and her *real user type*, which reflects "real" user's actual preferences. The movies database contains 600 movies; the 200 of them are movies that the user was recommended during an assumed initial period of use, and the *prior user type* was updated based on them. The next 200 movies were created in order to receive high ratings by the *real user type*, i.e., these movie types match perfectly the assumed "current" real user type; and the last 200 movies were randomly generated. Departing from what was assumed in all previous experiments, in this case, the *initial* prior for the user-type updating process is *informative*: that is, in this case we have an *informative prior user type*. The user was sequentially recommended all the movies of the database, i.e., 600 movies, and the experiment was executed for 10 *independent*

Database	Average rating
user A database	5.8946
user B database	5.9559
user C database	6.0022
user D database	4.7994

TABLE 5.5: The overall average ratings of the movies databases.

Methods	user A	user B	user C	user D
BayesYouLikeIt (BYLI)	5.5983	5.6133	5.3917	5.6183
BYLI - without trailers	5.5983	5.6133	5.3917	5.6183
BYLI - VPI on trailers	5.5983	5.6133	5.3917	5.6183
BYLI - VPI on trailers & movies	5.5983	5.6133	5.3917	5.6183
BYLI - VPI on movies without trailers	5.5983	5.6133	5.3917	5.6183
Random	5.5983	5.6133	5.3917	5.6183

TABLE 5.6: Average user ratings over all recommendations and all runs for each variant.

runs. User ratings were assigned on movies using a *rating function* and ranged between 1 and 10, as in the experiments of Section 5.1.

In order to test the stability of our method and the validity of the results, we simulated 4 different users and used several variants of our algorithm. Specifically, we used once again the basic BYLI algorithm, the *VPI on trailers* and *VPI on trailers & movies* variants, and a random recommendation process. Furthermore, we applied a new type of "without trailers" variants. Using these variants we aim to explore system's efficiency without the exploitation of trailers projection phase, so we modified the basic algorithm and the *VPI on trailers & movies* variant, removing that phase. Tables 5.5 and 5.6 show the average ratings of the 4 users' movies databases and the average ratings per variant for each one, respectively. In Table 5.6, we observe that the average ratings are equal in all variants for each user. This is because the user is always recommended with all the movies of the database, and rates each movie with the same rating each time.

Figure 5.11 shows the average per recommendation ratings over all runs of *user A* for all variants. It is easy to see that the VPI variants behave smoother than basic BYLI. The *BYLI without trailers* algorithm behaves smoother than the basic BYLI too, and its behaviour is directly comparable with that of the VPI variants. In Fig. 5.11(a), we observe lack of recommendation performance stability due to the presence of trailers. That is, the *trailer-based* user type is changed at a slow pace, and because of that, the agent has to be adapted to changes over and over again. Also, because our method selects trailers and hence movies based on the initial *informative prior*, the agent receives representative ratings a bit too late; and this fact causes a slow down in the learning process of the real user. This is the reason why the ratings in the first 100 recommendations are lower compared to all other variants. This phenomenon is not observable

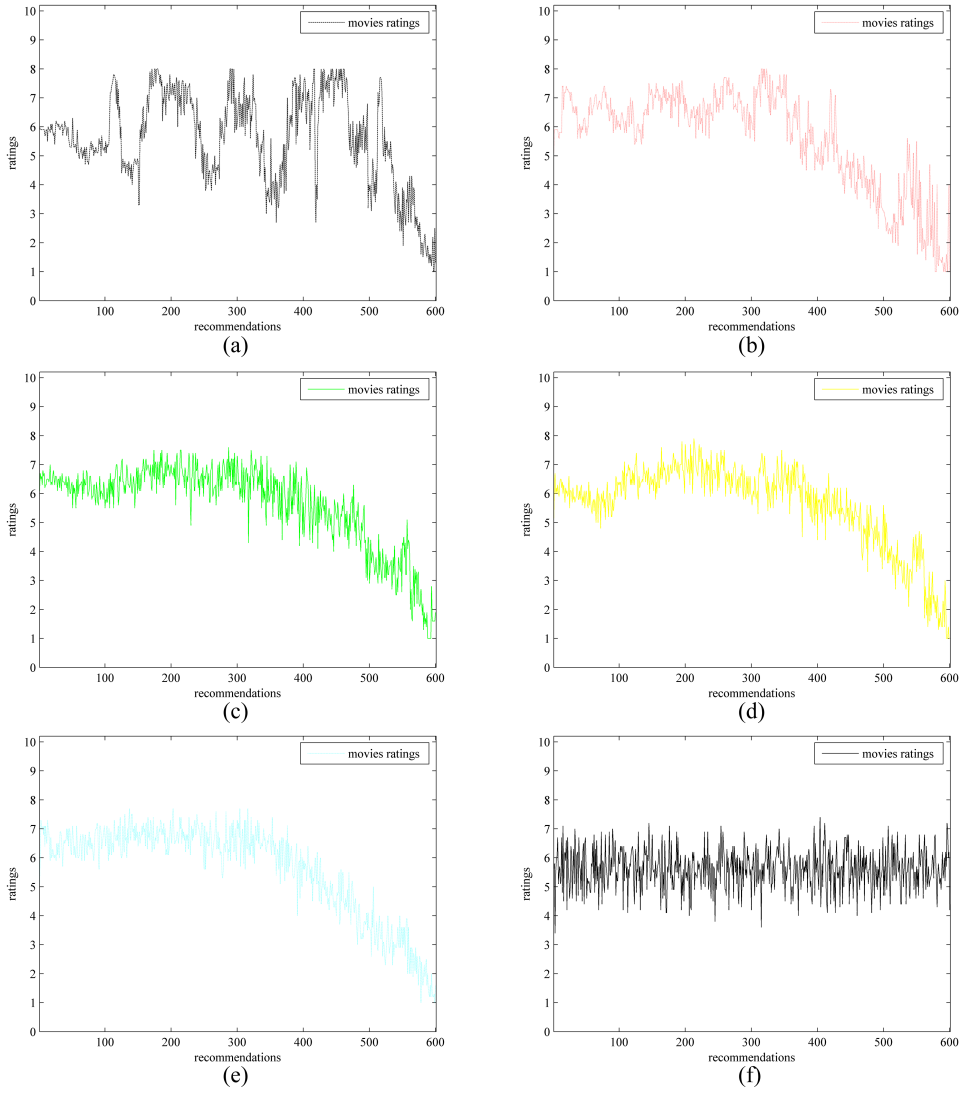


FIGURE 5.11: *Average per recommendation ratings* over all runs for *user A*: (a) BYLI, (b) BYLI *without trailers*, (c) BYLI - *VPI on trailers*, (d) BYLI - *VPI on trailers & movies*, (e) BYLI - *VPI on movies without trailers*, (f) Random.

for variants which exploit VPI, since they have the ability to learn faster, as also shown earlier.

The *VPI on movies without trailers* variant which combines VPI and absence of trailers, appears to behave better than all the others, since it receives high ratings from the beginning, and after the middle of the recommendation process the ratings are smoothly decreased (Figure 5.11(e)). Considering the nature of the movies database, this is a desirable plot, since, the database contains 200 movies that user likes, 200 randomly generated movies, and 200 movies that user does not like them (since she used to like them before her preferences' change); so in this case, our agent essentially recommends mostly preferred movies until the $300th$ recommendation and then, recommends the best available ones until the end of the experiment.

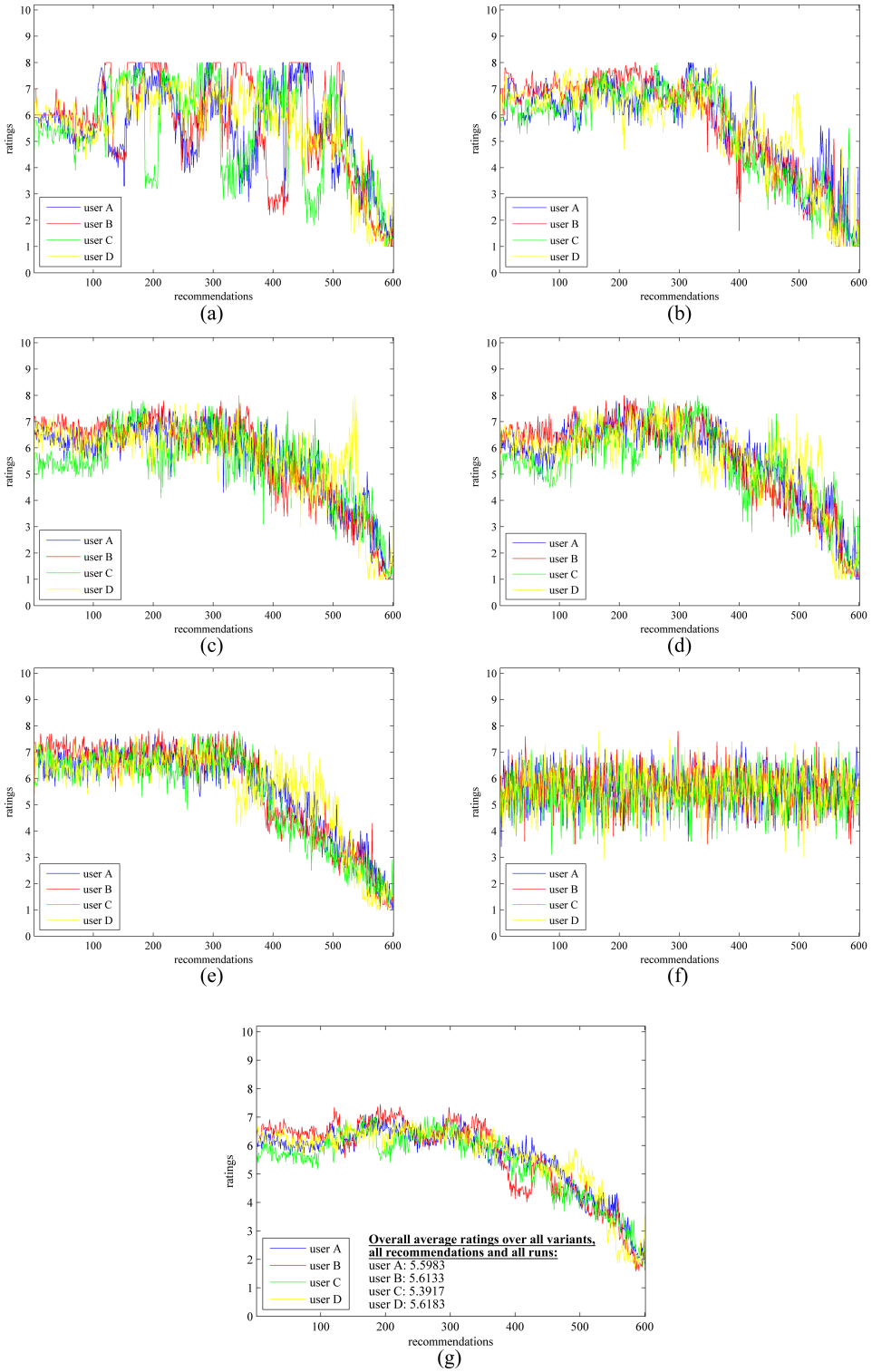


FIGURE 5.12: Comparison of users' *average per recommendation ratings* over all runs: (a) BYLI, (b) BYLI *without trailers*, (c) BYLI - *VPI on trailers*, (d) BYLI - *VPI on trailers & movies*, (e) BYLI - *VPI on movies without trailers*, (f) Random, (g) *Average per recommendation ratings* over all variants and all runs for the 4 simulated users.

In Figure 5.12, we show how the methods perform for all 4 users on all variants. It is observable that our agent treats all users in a similar way during the execution of each variant, since all plots follow similar "trajectories" in each figure; but she also detects the differences between users' preferences resulting unique plots for each one. Additionally, in Figure 5.12(g), we can see the average per recommendation ratings of each user over all runs and all variants, where there exist significant alterations between the average ratings of the first and the last recommendations — this is an evidence that the good movies were recommended first.

5.7 On the usefulness of trailers

In the experiment above, the VPI method *without trailers* seemed to be quite successful. For interest, we executed additional experiments in order to obtain more accurate conclusions about the value of using trailers. For these experiments, we relied on the setup of those in Sections 5.1 and 5.2. In particular, for the *simulated* database and users of Section 5.1, we ran the basic BYLI algorithm *with mood alternations* without the projection of trailers. We also ran the basic BYLI algorithm on the MovieLens dataset using 2 modifications; *(1)* without the trailers projection phase and, *(2)* with the projection of trailers that the user had never seen before during the whole recommendation process. At this point, we describe how our algorithm works without the use of trailers: there are no major changes; simply, the *trailer-based* user type no longer exists, and the movies are recommended using the overall user type. The rest of the recommendation process remains unaffected. We mention again, that the setup of the experiments is similar to the corresponding experimental setup in Sections 5.1 and 5.2.

Figures 5.13 and 5.14, enable us to compare the results of the basic algorithm without trailers, with those of Section 5.1. We observe that the algorithm without trailers begins very well; however, after the first mood changes, the average per recommendation ratings fall below the ratings of the algorithms that exploit trailers. The most important observation is that this algorithm is not able to follow mood changes; this fact is observable in the first 80 recommendations, where the mood-change points are visible (Fig. 5.13(b)). Further, the results from experiments on individuals (Figure 5.14) verify those conclusions and give us a better insight to agent's behavior *with* and *without* trailers.

On the other hand, Figure 5.15 presents the results in the MovieLens dataset. In this case, there are no remarkable differences between the algorithms, except for the lag of the *BYLI with projected-only-once trailers* (trailers that user had never seen before) in the last 10 recommendations. The sudden drop in performance for this algorithm was caused

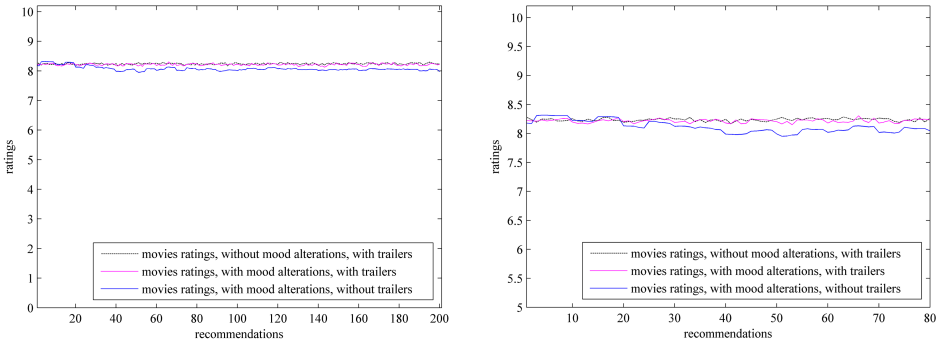


FIGURE 5.13: (a) Comparison of *average per recommendation ratings* of 50 *simulated* users on 200 movies for 10 runs in 3 different cases: (*i*) exploiting trailers without mood alterations simulation, (*ii*) exploiting trailers with mood alterations simulation, (*iii*) with mood alterations simulation without exploiting trailers. (b) Zoomed depiction of (a).

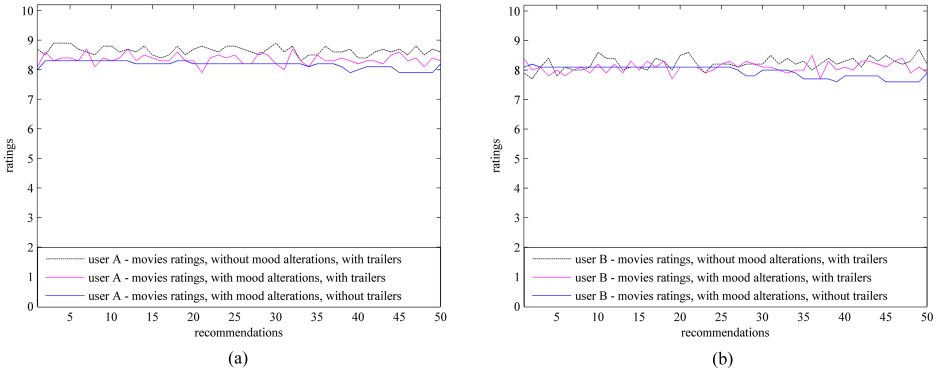


FIGURE 5.14: Comparison of *average per recommendation ratings* of 2 *simulated* users on 50 movies for 10 runs in 3 different cases: (*i*) exploiting trailers without mood alterations simulation, (*ii*) exploiting trailers with mood alterations simulation, (*iii*) with mood alterations simulation without exploiting trailers. (a) *user A*, (b) *user B*.

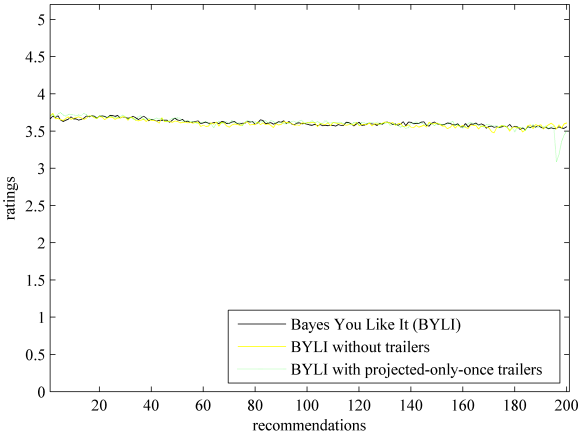


FIGURE 5.15: *Average per recommendation ratings* over all users and all runs; MovieLens dataset.

by the fact that the remaining trailers and movies did not match the current user type. Nonetheless, it is worth mentioning that the agent coped well with that situation and adapted to the new data receiving high ratings again. Generally, our algorithm receives high ratings whether it exploits trailers or not, so it can be easily applied without using them.

To conclude, it seems that the main usefulness of showing trailers is in capturing the user's mood alterations. Thus, no time consuming trailers are required after the first recommendations, when the system possesses an accurate user model.

5.8 Overview of experimental results

We ran a large number of experiments, aiming to examine all the aspects of our work. First, we tested the *scalability* of our method applying the basic *BayesYouLikeIt* on large-scale simulated databases. That experimental setup allowed us to also test the ability of our algorithm to capture the changes of user's *mood*, simulating mood alternations during the recommendation process. The results show that there exist neither ratings reduction nor abrupt rating changes, so our agent can successfully face those challenges. Afterwards, we executed all the variants of our algorithm using the MovieLens dataset. Specifically, we ran the basic *BYLI* algorithm, and the *VPI* and *Boltzmann* variants; and compared them with the established *LSMF* method. In that case, the results demonstrate that our method is quite competitive and successfully incorporates the VPI and Boltzmann explorations, since the average ratings of all variants differ only for few decimals.

Furthermore, we tried to observe our method's behavior on databases which contain only a few "good" movies for recommendation. For those experiments, we used: *(a)* user datasets from MovieLens containing users that had rated few movies with high ratings, and *(b)* the MovieLens dataset modified in such a way that each user's movies ratings follow a Beta distribution. The purpose of the specific settings was to test if our agent can quickly "learn" a user and recommend her all the available "preferred" movies. The results testify that the agent can deal with these objectives very well. Moreover, we examined the way our method treats a specific user. For this, we randomly selected 3 users from the MovieLens dataset and tested their response to all BYLI variants. Moreover, we examined the adaptation abilities of our method regarding possible changes in user preferences. To test these, we created 4 simulated users, their movies databases and their modified preferences, and executed some BYLI variants on these data. All variants respond very well to this challenge, but the basic BYLI algorithm lags behind the others regarding its learning pace.

Finally, we tested the usefulness of employing trailers during the recommendation process. Based on the experimental setups of Sections 5.1 and 5.2, we modified the basic BYLI algorithm removing the trailers projection phase, or projecting only trailers that the user had never seen before. Through those experiments, we concluded that the use of trailers does not immediately lead to increased effectiveness, but they are crucial for capturing a user's mood changes. Summarizing, our method achieves all the experimental goals, since it gives promising results, and displays stable behavior and significant learning abilities. The basic BYLI algorithm has the best overall average rating compared to the rest of BYLI variants. However, the VPI variants were demonstrated to be able to learn a user model quite fast; to exhibit a smooth performance; and to be more adaptive than others to changes in the user preferences.

Chapter 6

CONCLUSIONS & FUTURE WORK

In this work, we presented a novel approach for making entirely personalized recommendations, and applied it in the movie recommendations domain. Our approach uses Bayesian updating to infer a *user model*, which is a *distribution of the same form* with the models of the items under recommendation. This is, we believe, key to enabling the implicit inference of an individual's latent or otherwise unrepresentable, complicated preferences; and enables us to devise and employ a kind of *Bayesian exploration* in this domain. We presented and tested several variants of our method, and evaluated their performance, with very promising results. Futher, we strongly tested the *exploration* and *adaptation* abilities of our agent deriving remarkable conclusions; and we first ever exploited the *value of perfect information* theory in the recommendations domain receiving successful outcomes. In a sense, what we exploit in this work is the fact that modern-world datasets contain much "annotated" information regarding the nature of contained data (e.g., in the form of "genre"-specifying labels). This information allows us to properly define the dimensions of multivariate Gaussians representing users (according to our "you are what you consume" idea), without relying on the tastes of others—and without attempting to explicitly predict user ratings. Note that the main ideas of this work have already appeared in a *RecSys '13* paper of ours [5].

Summarizing our main contributions in this thesis, we propose a completely *personalized* approach producing recommendations based only on previous observations of a specific user, and introduce a new Bayesian *Reinforcement Learning* technique without the exploitation or combination of any established recommendation method. We do not perform *Collaborative Filtering*, *Preference Elicitation* or *Content-aware* techniques, and this is the first non-PE method that does not need to predict ratings or probability

distributions over ratings. More, *BayesYouLikeIt* focuses on user *preferences* and *mood* in combination with item features, modeling both users and items with the same form, and is able to capture *latent features* and *temporal effects* [52]. We are also the first who integrate *Bayesian reinforcement learning* theory in a recommendation system and exploit *VPI* to "learn" a user faster and more efficiently, and tackle the problem of "cold-start".

Furthermore, we differ significantly from all aforementioned approaches. All those approaches need to predict ratings either as a simple numeric value or as a probability distribution over ratings, while we do not have this need. We provide personalized recommendations, unlike others who combine data from many users in order to recommend to an individual. Also, the modeling of both users and items by a probability distribution of the same form is novel, as is the use of trailers for user's mood capturing. The recommendation with emphasis on the understanding of user preferences and the adaptation on them, is another difference between our approach and the others.

We remark that our method is generic, and not "fine-tuned" for movie recommendations. Nonetheless, our *personalized* method's performance almost matches that of a state-of-the-art movie recommendations method, which is able to exploit preference data originating from thousands of users. As our method proved to be competitive against algorithms that do have this "advantage", it is most probably especially well-suited for environments where user preferences data is scarse. In sparse datasets, our approach is expected to perform better than methods which require other users' ratings. Therefore, it could be used as a "bootstrapping" tool, generating recommendations until more data is available; moreover, it can be readily employed as a "training" component used during a more sophisticated system's initial operation period.

Regarding future work, we plan to conduct a user satisfaction survey based on a number of users testing our system, after developing a functional system that implements the main aspects of our approach. We also intend to run experiments on larger movies datasets in order to further test the scalability and the memory capabilities of our method, and to combine the *BayesYouLikeIt* algorithm with other recommendation techniques, e.g., *Collaborative Filtering* [23, 24] and *Content-based* [21, 22] techniques. With this, we can either use BYLI as a "bootstrapping" tool or enhance it using preference acquisition techniques and item's content descriptions that those methods use.

It is true that users usually rate only items that they like, otherwise they probably do not bother. This fact inserts additional bias in the recommendation process, and the question is how a recommender system can tackle this bias. Similar problem is the interpretation of *implicit ratings*, i.e., inferring a user rating based on user's behavior, e.g., the fact that a user doesn't watch the whole movie, can be translated as a very

low rating. In the future, we plan to take it into account modifying accordingly our algorithm. Based on that, it is worth to be mentioned that a completely personalized approach like ours has an advantage over others, since it can follow user behaviors of that kind and reduce the bias. It is also worth testing our approach in other domains, since it is generic and allows us to model and recommend items regardless of the domain without changing anything in our system—e.g., use it for recommending scientific papers; studying nutritional habits; or employ it in story-telling environments, being influenced by works like the one in [32].

Finally, we aim to incorporate multiagent systems techniques in our model, allowing us to exploit user-specific information coming from other types of online agents; incorporating knowledge from other agents, we can build a more accurate and complete user profile or "prepare" our agent for a user that is a newcomer into the system. For instance, we could extend our model to allow for recommendations based on (implicit) "voting" among several *"recommenders"*. The recommenders would be allowed to suggest one out of several candidate recommendations each, and a recommendation would emerge as a result of the "vote". Then, the outcome of this vote would be rated by the user, essentially determining the actual success or failure of the recommendation. This process could help determine the *recommendation power* of the various recommenders', computed given some *power index* (like, e.g., Banzhaf index or Rae index [59, 60]); and this revealed power could potentially be used both for determining an updated weight for the recommender, and for guiding the type-determining process of the user—especially when he is a "newcomer", with a prior that is still highly uninformative.

Indeed, we believe that recommendation systems can benefit from research conducted within the broader AI and multiagent systems communities; and we view our work here as a small step towards bringing these communities closer together.

Appendix A

DESCRIPTION OF SYSTEM MODELS

All the available data is stored in the system using suitable *models*. As mentioned many times throughout the text, both *items* and *users* are modeled by multivariate Gaussians. In this appendix, we try to provide a more detailed description of these models and describe the way the item information is transformed to a model. Hereafter, we refer to the movies recommendation domain. The only movie features we exploit during the recommendation procedure, are the available movie genres.

As already stated, a movie model (or *movie type*) is a k-dimensional multivariate Gaussian distribution. Such a distribution consists of an 1×k *mean vector* and a k×k *covariance matrix*, and each dimension corresponds to a movie genre. The mean contains equal values to all dimensions. The covariance matrix is *diagonal* and each diagonal element takes value equal to 1 or 20, according to the underlying uncertainty, and depending whether the corresponding movie genre characterizes the movie; we consider movie genres to be independent. The value 20 of the not relevant genres was decided empirically, since a Gaussian distribution with $\sigma^2 = 20$ does not influence the updated model. On the other hand, the relevant movie genres are associated with $\sigma^2 = 1$ because of their importance.

In our experiments, we used the *MovieLens* 1 *million* dataset. In order to create the model of a movie, we need to know the genres which characterize it, and its overall average rating. The specific dataset provides the genres of each movie and the rating that each user has assigned to it; by this, we compute the average rating of each movie over all users. Thus, we have what we need for creating all movies' models. Note that, the available movie genres in the MovieLens dataset are 18, the following: *Action*, *Adventure*,

Action	x
Adventure	x
Animation	x
Children's	x
Comedy	x
Crime	x
Documentary	x
Drama	x
Fantasy	x
Film-Noir	x
Horror	x
Musical	x
Mystery	x
Romance	x
Sci-Fi	x
Thriller	x
War	x
Western	x

(a)

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
Action	y_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Adventure	0	y_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Animation	0	0	y_3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Children's	0	0	0	y_4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Comedy	0	0	0	0	y_5	0	0	0	0	0	0	0	0	0	0	0	0	0
Crime	0	0	0	0	0	y_6	0	0	0	0	0	0	0	0	0	0	0	0
Documentary	0	0	0	0	0	0	y_7	0	0	0	0	0	0	0	0	0	0	0
Drama	0	0	0	0	0	0	0	y_8	0	0	0	0	0	0	0	0	0	0
Fantasy	0	0	0	0	0	0	0	0	y_9	0	0	0	0	0	0	0	0	0
Film-Noir	0	0	0	0	0	0	0	0	0	y_{10}	0	0	0	0	0	0	0	0
Horror	0	0	0	0	0	0	0	0	0	0	y_{11}	0	0	0	0	0	0	0
Musical	0	0	0	0	0	0	0	0	0	0	0	y_{12}	0	0	0	0	0	0
Mystery	0	0	0	0	0	0	0	0	0	0	0	0	y_{13}	0	0	0	0	0
Romance	0	0	0	0	0	0	0	0	0	0	0	0	0	y_{14}	0	0	0	0
Sci-Fi	0	0	0	0	0	0	0	0	0	0	0	0	0	0	y_{15}	0	0	0
Thriller	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	y_{16}	0	0
War	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	y_{17}	0
Western	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	y_{18}

(b)

FIGURE A.1: The forms of (a) the *mean* and (b) the *covariance matrix* of a movie type.

Animation, *Children's*, *Comedy*, *Crime*, *Documentary*, *Drama*, *Fantasy*, *Film-Noir*, *Horror*, *Musical*, *Mystery*, *Romance*, *Sci-Fi*, *Thriller*, *War*, *Western*; therefore, the models are 18-dimensional Gaussians.

In Figure A.1, we depict the forms of both the mean and the covariance matrix. Let us consider a movie with average MovieLens rating equal to 3.45 and genres *action, adventure, sci-fi, thriller*. Therefore, the x value of the mean becomes equal to 3.45 ($x = 3.45$) and the y_1, y_2, y_{15}, y_{16} elements of the covariance matrix take the value 1, while all other diagonal elements have a value of 20. Here, we should note that the models of the simulated database (Section 5.1) exploit exactly the same form, but with 16 movie genres.

With these movie models at hand, we can build a representative user model, called *user type*, after receiving a user rating on each movie. This user type is continuously updated through Bayesian updating (Section 2.3), and its form is similar to that of movies. That is, the user type is a 18-dimensional Gaussian whose values are determined by the updating procedure.

Appendix B

BAYESIAN RANDOM EXPLORATION

Bayesian Random Exploration (BRX) has the *same functionality as our main implementation*, i.e., it is a Bayesian exploration algorithm, but it is extended with *randomness*. We simply replace uniformly at random only one demo and the selected item with another unused one. That is, we select 1 of the 5 to be projected demos, following a uniform probability distribution, and we randomly replace it with another (unused in current session). Similarly, the agent uses a uniform probability to decide if she has to replace or not the recommended item with another random one. Specifically, BRX has similar behaviour with the Bayesian exploration, but it initially introduces an explicit exploration factor, which drops with time.

When it is applied to demos selection, the agent picks 5 demos as usual, and selects one of them with uniform probability. After that, the selected demo is replaced with a new one based on the *exploration factor* (this is a probability factor with an initial value of 0.5, decreasing with a step of 0.1 after every recommendation), which denotes if this demo should be replaced:

$$e_0 = 0.5,\ with\ e_{t+1} = e_t - 0.1\ until\ e_t = 0 \tag{B.1}$$

When BRX is applied to demos selection only, the way the actually recommended item is picked stays unchanged. On the other hand, when BRX is applied both to demos and item selection, the recommendation process is augmented with the replacement of the selected-for-recommendation item based on the exploration factor, as above. This item is replaced by an unused one, picked with uniform probability, which is recommended to the user. Experiments with this algorithm show that randomness causes a slight reduction of user ratings — i.e., a slight drop in the system's performance.

Bibliography

[1] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. *Recommender Systems Handbook*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010. ISBN 0387858199, 9780387858197.

[2] R. Dearden, N. Friedman, and D. Andre. Model based Bayesian Exploration. In *Proceedings of Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 150–159, 1999.

[3] Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: a bayesian approach. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, 2003.

[4] Shlomo Berkovsky, Tsvi Kuflik, and Francesco Ricci. Distributed Collaborative Filtering with Domain Specialization. In *Proceedings of the 1st ACM Conference on Recommender systems*, RecSys '07, 2007.

[5] Konstantinos Babas, Georgios Chalkiadakis, and Evangelos Tripolitakis. You are what you consume: A bayesian method for personalized recommendations. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 221–228, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2409-0. doi: 10.1145/2507157.2507158. URL http://doi.acm.org/10.1145/2507157.2507158.

[6] Michael Schrage. *Who Do You Want Your Customers to Become?* Harvard Business Press, 2013.

[7] Jonathan L. Herlocker, Joseph A. Konstan, and John Riedl. Explaining collaborative filtering recommendations. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, CSCW '00, pages 241–250, New York, NY, USA, 2000. ACM. ISBN 1-58113-222-0. doi: 10.1145/358916.358995. URL http://doi.acm.org/10.1145/358916.358995.

[8] Alan Lewis. *The Cambridge handbook of psychology and economic behaviour*. Cambridge University Press, 2008.

[9] Richard H. Thaler. Mental accounting and consumer choice. *Marketing Science*, 27(1):15–25, January 2008. ISSN 1526-548X. doi: 10.1287/mksc.1070.0330. URL http://dx.doi.org/10.1287/mksc.1070.0330.

[10] Richard Thaler. Toward a positive theory of consumer choice. *Journal of Economic Behavior & Organization*, 1(1):39–60, 1980.

[11] James R Bettman, Mary Frances Luce, and John W Payne. Constructive consumer choice processes. *Journal of consumer research*, 25(3):187–217, 1998.

[12] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521899435.

[13] Meltem Öztürk and Alexis Tsoukiàs. Preference modelling. In *State of the Art in Multiple Criteria Decision Analysis*, pages 27–72. Springer-Verlag, 2005.

[14] M.H. DeGroot and J. Schervish. *Probability and Statistics*. 2002.

[15] Daniel Fink. A compendium of conjugate priors, 1997.

[16] Kevin P. Murphy. Conjugate bayesian analysis of the gaussian distribution. Technical report, 2007.

[17] T.W. Anderson. *An introduction to multivariate statistical analysis*. John Wiley and Sons, 2003.

[18] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[19] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009. ISBN 0136042597, 9780136042594.

[20] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian q-learning. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 761–768, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence. ISBN 0-262-51098-7. URL http://dl.acm.org/citation.cfm?id=295240.295801.

[21] Pasquale Lops, Marco Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer US, 2011. ISBN 978-0-387-85819-7. doi: 10.1007/978-0-387-85820-3_3. URL http://dx.doi.org/10.1007/978-0-387-85820-3_3.

[22] MichaelJ. Pazzani and Daniel Billsus. Content-based recommendation systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321 of *Lecture Notes in Computer Science*, pages 325–341. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-72078-2. doi: 10.1007/978-3-540-72079-9_10. URL http://dx.doi.org/10.1007/978-3-540-72079-9_10.

[23] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004. ISSN 1046-8188. doi: 10.1145/963770.963772. URL http://doi.acm.org/10.1145/963770.963772.

[24] Michael D. Ekstrand, John T. Riedl, and Joseph A. Konstan. Collaborative filtering recommender systems. *Found. Trends Hum.-Comput. Interact.*, 4(2):81–173, February 2011. ISSN 1551-3955. doi: 10.1561/1100000009. URL http://dx.doi.org/10.1561/1100000009.

[25] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society. ISBN 978-0-7695-3502-9. doi: 10.1109/ICDM.2008.22. URL http://dx.doi.org/10.1109/ICDM.2008.22.

[26] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th Int. Conf. on Algor. Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.

[27] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth National Conference on Artificial Intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence. ISBN 0-262-51129-0. URL http://dl.acm.org/citation.cfm?id=777092.777124.

[28] Souvik Debnath, Niloy Ganguly, and Pabitra Mitra. Feature weighting in content based recommendation system using social network analysis. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 1041–1042, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-085-2. doi: 10.1145/1367497.1367646. URL http://doi.acm.org/10.1145/1367497.1367646.

[29] Eyrun A. Eyjolfsdottir, Gaurangi Tilak, and Nan Li. MovieGEN: A Movie Recommendation System. 2008.

[30] Chumki Basu, Haym Hirsh, and William Cohen. Recommendation as classification: Using social and content-based information in recommendation. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 714–720. AAAI Press, 1998.

[31] Nicola Barbieri, Gianni Costa, Giuseppe Manco, and Riccardo Ortale. Modeling Item Selection and Relevance for Accurate Recommendations: a Bayesian Approach. In *Proceedings of the 5th ACM Conference on Recommender systems*, RecSys '11, 2011.

[32] Hong Yu and Mark O. Riedl. A sequential recommendation approach for interactive personalized story generation. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 71–78, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0-9817381-1-7, 978-0-9817381-1-6. URL http://dl.acm.org/citation.cfm?id=2343576.2343586.

[33] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. Tfmap: Optimizing map for top-n context-aware recommendation. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 155–164, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1472-5. doi: 10.1145/2348283.2348308. URL http://doi.acm.org/10.1145/2348283.2348308.

[34] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: Learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 139–146, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1270-7. doi: 10.1145/2365952.2365981. URL http://doi.acm.org/10.1145/2365952.2365981.

[35] Yehuda Koren and Joe Sill. Ordrec: An ordinal model for predicting personalized item rating distributions. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 117–124, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0683-6. doi: 10.1145/2043932.2043956. URL http://doi.acm.org/10.1145/2043932.2043956.

[36] Nathan N. Liu and Qiang Yang. Eigenrank: A ranking-oriented approach to collaborative filtering. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 83–90, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-164-4. doi: 10.1145/1390334.1390351. URL http://doi.acm.org/10.1145/1390334.1390351.

[37] Nathan N. Liu, Min Zhao, and Qiang Yang. Probabilistic latent preference analysis for collaborative filtering. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, pages 759–766, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-512-3. doi: 10.1145/1645953.1646050. URL http://doi.acm.org/10.1145/1645953.1646050.

[38] Urszula Chajewska, Daphne Koller, and Ronald Parr. Making rational decisions using adaptive utility elicitation. In *Proceedings of AAAI-2000*, 2000.

[39] Sybil Shearin and Henry Lieberman. Intelligent profiling by example. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, IUI '01, pages 145–151, New York, NY, USA, 2001. ACM. ISBN 1-58113-325-1. doi: 10.1145/359784.360325. URL http://doi.acm.org/10.1145/359784.360325.

[40] Henry Blanco and Francesco Ricci. Inferring user utility for query revision recommendation. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 245–252, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1656-9. doi: 10.1145/2480362.2480416. URL http://doi.acm.org/10.1145/2480362.2480416.

[41] Chihiro Ono, Mori Kurokawa, Yoichi Motomura, and Hideki Asoh. A context-aware movie preference model using a bayesian network for recommendation and promotion. In *Proceedings of the 11th International Conference on User Modeling*, UM '07, pages 247–257, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-73077-4. doi: 10.1007/978-3-540-73078-1_28. URL http://dx.doi.org/10.1007/978-3-540-73078-1_28.

[42] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 452–461, Arlington, Virginia, United States, 2009. AUAI Press. ISBN 978-0-9749039-5-8. URL http://dl.acm.org/citation.cfm?id=1795114.1795167.

[43] David H. Stern, Ralf Herbrich, and Thore Graepel. Matchbox: Large scale online bayesian recommendations. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 111–120, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4. doi: 10.1145/1526709.1526725. URL http://doi.acm.org/10.1145/1526709.1526725.

[44] Helge Langseth and Thomas Dyhre Nielsen. A latent model for collaborative filtering. *Int. J. Approx. Reasoning*, 53(4):447–466, June 2012.

[45] Yi Zhang and Jonathan Koren. Efficient bayesian hierarchical user modeling for recommendation system. In *Proceedings of the 30th Annual intern. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 47–54. ACM, 2007.

[46] Martin Szomszor, Ciro Cattuto, Harith Alani, Kieron O'Hara, Andrea Baldassarri, Vittorio Loreto, and Vito D.P. Servedio. Folksonomies, the semantic web, and movie recommendation. In *4th European Semantic Web Conference, Bridging the Gap between Semantic Web and Web 2.0*, 2007. URL http://eprints.ecs.soton.ac.uk/14007/.

[47] Rajatish Mukherjee, Partha Sarathi Dutta, Sandip Sen, and Ip Sen. Movies2go - a new approach to online movie recommendation. In *In Proceedings of the IJCAI Workshop on Intelligent Techniques for Web Personalization*, 2001.

[48] Jennifer Golbeck. Generating predictive movie recommendations from trust in social networks. In *Proceedings of the 4th International Conference on Trust Management*, iTrust'06, pages 93–104, Berlin, Heidelberg, 2006. Springer-Verlag. ISBN 3-540-34295-8, 978-3-540-34295-3. doi: 10.1007/11755593_8. URL http://dx.doi.org/10.1007/11755593_8.

[49] Anan Liu, Yongdong Zhang, and Jintao Li. Personalized movie recommendation. In *Proceedings of the 17th ACM International Conference on Multimedia*, MM '09, pages 845–848, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-608-3. doi: 10.1145/1631272.1631429. URL http://doi.acm.org/10.1145/1631272.1631429.

[50] Mehdi Elahi, Matthias Braunhofer, Francesco Ricci, and Marko Tkalcic. Personality-based active learning for collaborative filtering recommender systems. In *AI*IA*, pages 360–371, 2013.

[51] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL http://dx.doi.org/10.1109/MC.2009.263.

[52] Robert M Bell, Yehuda Koren, and Chris Volinsky. The bellkor solution to the netflix prize. *KorBell Team's Report to Netflix*, 2007.

[53] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

[54] Kenneth E. Kendall and Julie E. Kendall. *Systems Analysis and Design (8th Edition)*. Prentice-Hall, Inc., 2011. ISBN 013608916X.

[55] Gordon P. Kurtenbach, Abigail J. Sellen, and William A. S. Buxton. An empirical evaluation of some articulatory and cognitive aspects of marking menus. *Hum.-Comput. Interact.*, 8(1):1–23, March 1993. ISSN 0737-0024. doi: 10.1207/s15327051hci0801_1. URL http://dx.doi.org/10.1207/s15327051hci0801_1.

[56] S. Kullback. *Information Theory and Statistics*. Wiley series in probability and mathematical statistics. Probability and mathematical statistics. John Wiley &Sons, 1959. URL http://books.google.gr/books?id=XeRQAAAAMAAJ.

[57] Frank Nielsen and Richard Nock. Emerging trends in visual computing. pages 164–174, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-00825-2. doi: 10.1007/978-3-642-00826-9_7. URL http://dx.doi.org/10.1007/978-3-642-00826-9_7.

[58] David Carmel and Shaul Markovitch. Exploration strategies for model-based learning in multiagent systems. *Autonomous Agents and Multi-agent Systems*, 2 (2):141–172, 1999. URL http://www.cs.technion.ac.il/~shaulm/papers/pdf/Carmel-Markovitch-aamas1999.pdf.

[59] G. Chalkiadakis, E. Elkind, and M.J. Wooldridge. *Computational Aspects of Cooperative Game Theory*. Synthesis digital library of engineering and computer science. Morgan & Claypool Publishers, 2012. ISBN 9781608456529. URL http://books.google.gr/books?id=bN9aC0uabBAC.

[60] Josep Freixas and Montserrat Pons. Success and decisiveness on proper symmetric games. *Central European Journal of Operations Research*, pages 1–16, 2013. ISSN 1435-246X. doi: 10.1007/s10100-013-0332-5. URL http://dx.doi.org/10.1007/s10100-013-0332-5.