# Markov Chain Monte Carlo for Effective Personalized Recommendations

Michail-Angelos Papilaris and Georgios Chalkiadakis

School of Electrical and Computer Engineering, Technical University of Crete, Greece
mixpapilaris@gmail.com, gehalk@intelligence.tuc.gr

**Abstract.** This paper adopts a Bayesian approach for finding top recommendations. The approach is entirely personalized, and consists of learning a utility function over user preferences via employing a sampling-based, non-intrusive preference elicitation framework. We explicitly model the uncertainty over the utility function and learn it through passive user feedback, provided in the form of clicks on previously recommended items. The utility function is a linear combination of weighted features, and beliefs are maintained using a Markov Chain Monte Carlo algorithm. Our approach overcomes the problem of having conflicting user constraints by identifying a convex region within a user's preferences model. Additionally, it handles situations where not enough data about the user is available, by exploiting the information from clusters of (feature) weight vectors created by observing other users' behavior. We evaluate our system's performance by applying it in the online hotel booking recommendations domain using a real-world dataset, with very encouraging results.

**Keywords:** Adaptation and Learning · Recommender Systems

## 1 Introduction

Personalized recommender systems aim to help users access and retrieve relevant information or items from large collections, by automatically finding and suggesting products or services of potential interest [20]. User preferences could be self-contradicting; are often next to impossible for the user to specify; and are notoriously difficult to infer, while doing so often requires a tedious elicitation process relying on evidence of others' behavior [15, 20].

In this work we propose a complete model for learning the user preferences and use it to make targeted recommendations. Our system: *(a)* does not rely on user-specified hard constraints; and *(b)* does not require an explicit user preferences elicitation process: rather, it learns a user model through passive observation of her actions. In particular, we follow a Bayesian approach that operates as follows. We model the user utility over items as a linear function of the item features, governed by a weight vector $\boldsymbol{w}$. Our goal is to learn this vector in order to provide personalized recommendations to our user. We capture the uncertainty of the weight vector $\boldsymbol{w}$, which parameterize the utility function, through a distribution $P_w$ over the space of possible weight vectors.

Every time a user enters our system, we propose a number of items that comply with feedback we had received from her in previous interactions. The feedback is in the

form of clicks, and it can be translated to preferences. This feedback could in principle be used to update the prior weight distribution via Bayes rule. However, even for the simplest case of a uniform prior, the posterior could become very complex [19]. Thus, we adopt a different approach: instead of trying to refit the prior through an algorithm (such as *Expectation-Maximization (EM)* [4]), we keep the constraints infered from previous user interactions, and employ a *Markov Chain Monte Carlo (MCMC)* algorithm (specifically, Metropolis-Hastings [14]), in order to condition the prior and derive efficiently samples from the posterior [2, 8, 25].

In our work, we show how to find effectively and efficiently a "good" starting point for the MCMC algorithm, a challenging task in a multidimensional space like ours. Moreover, we take into consideration the problem of having conflicting constraints. This situation could be encountered when the user changes radically her preferences, or if we have gathered too many constraints and we could not find a weight vector that could satisfy all of them at the same time. We overcome this by using a linear programming algorithm that finds, effectively and efficiently, a convex region of all the possible weight vectors that satisfy all the user constraints. In addition, we show that the problem of having insufficient information about users can be alleviated by performing clustering over other users' preferred items, and exploiting the clusters to recommend items to the "high uncertainty" users.

As explained above, we have no need to set questions to the user, nor use textual information regarding an item in order to elicit user preferences. Moreover, we do not rely on any user ratings of the recommended items. We tested our system using a real world dataset that consist of 5000 hotels, each of which is being characterized by five main features; and exploited our knowledge of the preferences of synthetic users we constructed, in order to verify the effectiveness of our algorithm. Our simulations indeed confirm that the approach is able to quickly focus on a users' true preferences, and produce top personalized recommendations when operatiing in a realistically large recommendations space.

## 2 Background and Related Work

MCMC techniques estimate by simulation the expectation of a statistic in a complex model. Successive random selections form a Markov chain, the stationary distribution of which is the target distribution. Thus, MCMC is particularly useful for the estimation of posterior distributions in complex Bayesian models like ours [1].

Our work was to some extent inspired by work on *inverse reinforcement learning (IRL)*. The usual *reinforcement learning (RL)* problem is concerned with how agents ought to take actions in an environment so as to maximize some notion of cumulative reward, while the goal of IRL is to learn the model's reward function (which guides optimal behaviour). For instance, [18] views agent decisions as a set of linear constraints on the space of possible reward functions. Bayesian IRL [19], on the other hand, assumes that a distribution over possible reword functions exists and has to be inferred. In our case, the goal is also to learn the utility each item has for a user. We model this function as being linearly additive, governed by a weight vector. Conceptually, there-

fore, our approach is similar to IRL; however, we do not use an MDP to model our problem, as we do not explicitly view it as a sequential decision making one.

### 2.1 Related Work

In a recent work focused on the movies' recommendations domain, Tripolitakis and Chalkiadakis [24] proposed the use of *probabilistic topic modeling* algorithms to learn user preferences.That work modeled items *and* users as mixtures of latent topics, and was to an extent able to capture changes in user tastes or mood shifts, via the incorporation of the "Win Or Learn Fast" principle [6] found in the reinforcement learning literature. However, their work relies on "crowdsourced" or otherwise gathered textual information about the items (movies in their case); and it was able to exhibit only marginal improvement in recommendations' performance, when tested against other algorithms used in the movie recommendations literature. [1] By contrast, our work here does not attempt to build preference models relying on crowdsourced information about the items, and uses a rather simple exploration technique. Still, via the combination of MCMC and linear programming techniques, it is able to achieve an outstanding recommendations performance, as demonstrated in our experiments.

Also recently, Nasery *et al.* [17] present a recommender that introduces a novel prediction model that generates item recommendations using a combination of feature-based and item based preferences. More specifically, they propose a novel matrix factorization method (called FPMF), which incorporates user feature preferences to predict user likes, and evaluate it in the movie recommendations domain.

There is naturally much previous work on hotel or travel booking recommenders. However, these typically involve lengthy preference elicitation processes; require the use of textual information; or are "collaborative filtering" in nature, relying on choices made by other users in the past. For instance, Dong and Smyth [12] propose a personalized hotel booking recommendation system. In their work, however, they are attempting to mine features from the user reviews in order to create a user profile and exploit it to recommend items that comply with that information. Then, [22] considers an interactive way of ranking travel packages. However, for each iteration, the user is asked to rank a set of items, and the system requires several iterations of explicit preference elicitation before providing recommendations to the user.

One interesting work is that of [8], which considers the broad task of predicting the future decisions of an agent based on her past decisions. To account for the uncertainty regarding the agent's utility function, the authors consider the utility to be a random quantity that is governed by a prior probability distribution. When a new agent is encountered, this estimate serves as a prior distribution over her utility function. The constraints implied by the agent's observed choices are then used to condition the prior, obtaining a posterior distribution through an MCMC algorithm, as we do here.

The work whose model we adopt to a large extent in this paper, however, is that of [25], which generates top-$k$ *packages* for users by capturing personalized preferences

[1] That was also the case for the work of [3], which was modeling items *and* users as multivariate normals; like [24], and in contrast to our work here, [3] required users to actually *rate* a (small) number of items.

over packages using a linear utility function, which the system learns through feedback provided by the user. In their work, the authors of [25] propose several sampling-based methods to capture the updated utility function. They develop an efficient algorithm for generating top-$k$ packages using the learned utility function, where the rank ordering respects any of a variety of ranking semantics. Nevertheless, they do not tackle conflicts in the user constraints, do not employ clustering to reduce uncertainty for new users, and do not deal with the problem of appropriately initializing their MCMC algorithm.

## 3 A Personalized Recommender System

An overview of our system is the following (Figure 1). When a user enters, we retrieve the constraints derived from her previous interactions, and check if there are any conflicting constraints in order to handle them. Next we update beliefs by conditioning the prior distribution with the aforementioned constrains. Subsequently, we rank our items according to the posterior samples, and present the top $k$ items to the user. We record which of those items are being clicked on by the user, and assume that clicked items are more appealing to her than un-clicked ones. We store the new constraints derived from the feedback received, and progress to the next interaction. Finally, we form clusters with the posterior samples from the already registered users on our system, and use these with new users so as to reduce our initial uncertainty about them.

**Model settings** We assume that we are given a set of $n$ items $S_I$. Each item is being described by $m$ features $(f_1, f_2, ..., f_m)$. For example, a feature of an item could be the price, rating etc. Every item $i \in I$ is described by an $m$-dimensional vector $(v_1^i, v_2^i, ..., v_m^i)$, where $v_j^i$ corresponds to the value of the respective feature $f_j$. For every user we have a *constraint set* based on the feedback that we have received from her. Every constraint has the following form: $i_k \succ i_t$ where $i_k$, $i_t$ are items in $S_I$.

Our proposed model needs all of our items $i \in S_I$ in our database to be normalized. To achieve that, we found the maximum and the minimum values of each item feature, and then for each item feature value $v_j^i$ we normalize its value: $v_j^i \leftarrow \frac{v_j^i - min(v_j)}{max(v_j) - min(v_j)}$, where $min(v_j)$ and $max(v_j)$ are the min and max values of the respective $j$ feature in $S_I$. After normalization, all items' feature values lie in [0, 1].

**Utility Function** The user-specific utility function $U$ over items $i \in S_I$, that we wish to learn, directly depends on its feature vector. The space of all mappings between possible combinations of the feature values and utility values is uncountable, making this task challenging. Since we need to express the structure of an item concisely, we assume that the utility function is linearly additive and governed by a weight vector. This is a structure for the utility function commonly assumed in practice [8, 15]. For each item, the utility function is computed as follows:

$$U(i) = \sum_{j=1}^{m} w_j v_j^i \tag{1}$$

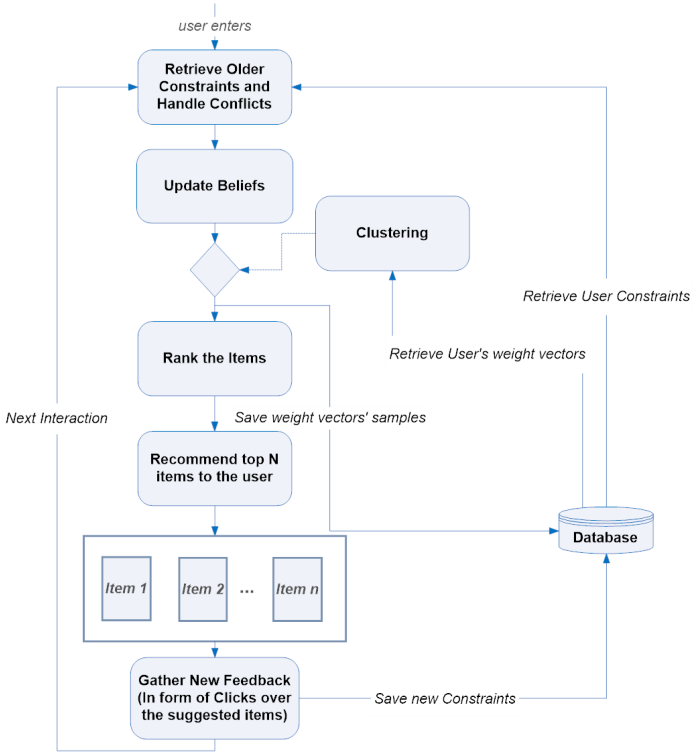


Fig. 1: Model Overview.

where $v_j^i$ is the value of the respective feature, and $w_j$ is the weight value associated with this feature. The value for $m$ depends on the number of features that each item has. Each $w_j$ takes values in [-1, 1]. A positive $w_j$ weight means that larger rather than smaller values are preferred for the corresponding feature, a negative $w_j$ weight means that smaller rather than larger values are preferred for the corresponding feature, while a zero weight means that the user is indifferent about the specific feature.[2] Our final goal is to learn the weight vector and suggest items to the user that have the maximum utility taking into consideration this weight vector.

Users often only have a rough idea of what they want in a desirable item, and also find such preference very difficult to quantify. For this reason, users are not able to specify or even know the exact values of the $w_j$ weights that drive utility function $U$. We model this uncertainty in a Bayesian manner, assuming that the vector of weights, $\boldsymbol{w}$, is not known in advance, but it can be described by a probability distribution $P_w$. We model this distribution as a *multivariate normal*. Such a distribution is often used to describe, at least approximately, any set of (possibly) correlated real-valued random

[2] To illustrate, assume a "hotel price" is "low", say $0.1$. If the user prefers really cheap hotels, she might have a weight of $-0.9$ for "hotel price", thus deriving a higher utility for this hotel compared to that derived for an expensive hotel of price, say, $0.9$ (since $-0.09 > -0.81$).

variables, each of which clusters around a mean value (see also the work of [3]). The $P_w$ can be learned by the feedback received.

### 3.1 Constraints

We now discuss how to generate constraints to reduce our uncertainty over weight vectors. To begin, our system suggests to the user a number of items, and we record which of them are being selected (clicked on) by the user. We make the assumption that clicked items are more appealing to the user than the un-clicked ones.[3] Thus, the feedback in form of new clicks, produces a set of pairwise preferences as follows. Assume that we present three items, $(i_1, i_2, i_3)$, to a user, each of which has (the same) three features $(f_1, f_2, f_3)$. If a user selects the second item, we can derive the following pairwise preferences: $i_2 \succ i_1$ and $i_2 \succ i_3$ (to put this otherwise, the user's selection is translated to $U(i_2) > U(i_1)$ and $U(i_2) > U(i_1)$). We can use, e.g., $i_2 \succ i_1$ to get:

$$U(i_2) > U(i_1) \tag{2}$$

$$\Rightarrow \sum_{j=1}^{m} w_j * v_j^{i_2} > \sum_{j=1}^{m} w_j * v_j^{i_1} \tag{3}$$

$$\begin{aligned} \Rightarrow w_1 * v_1^{i_2} + w_2 * v_2^{i_2} + w_3 * v_3^{i_2} > \\ w_1 * v_1^{i_1} + w_2 * v_2^{i_1} + w_3 * v_3^{i_1} \end{aligned} \tag{4}$$

The feature values of each item are known in advance, but we do not know the weights. We can focus on possible weight vectors by employing the following lemmas:

**Lemma 1** *[25]: Given feedback $i_1 \succ i_2$, if $i_1 \succ i_2$ does not hold under the weight vector $\boldsymbol{w}$, $P_w(\boldsymbol{w}|i_1 \succ i_2) = 0$.*

That is, every feedback received in terms of clicks, rules out all the weight vectors that do not satisfy the generated constraint. Additionally,

**Lemma 2** *[25]: The set of valid weight vectors which satisfy a set of preferences forms a convex set. So valid weight vectors form a continuous and convex region.*

**Handling Conflicting Constraints** A major challenge encountered is that, after a few interactions with our system, some of the generated constraints may have conflicts with newer ones *(a)* when the user changed radically his preferences; or *(b)* if we have gathered too many constraints, and cannot find a weight vector that satisfies all of them at the same time. To deal with this problem we used a well-known linear programming algorithm, Simplex [10]. We employ Simplex in order to find whether a feasible weight vector is available. In the case Simplex returns that there is no feasible weight vector which satisfies all constraints, we remove the oldest constraint we have gathered and rerun the Simplex algorithm. We do this as many times required to get a valid weight vector; notice that this process always leaves the set of constraints with at least one member, thus Simplex never faces the degenerate case of dealing with an empty constraints set. In this way, the set of constraints we have stored is always free of conflicts.

[3] Moreover, all clicked items are originally equally appealing. However, as interactions with the system increase, beliefs about the desirability of the items get updated.

### 3.2 Beliefs updating

Even if the prior distribution is a "nice" one with a compact closed form representation, the posterior distribution can be quite complex [19, 8]. Refitting the weight distribution $P_w$ after each feedback from the user would be a very inefficient and time-consuming task for a real-time recommender. We avoid this by employing the sampling-based framework of [25]: items preferences resulting from user feedback can be readily translated into constraints on the samples drawn from $P_w$. In our approach we follow a technique that is also used in the work of [8] and [25]: we condition our weights prior distribution with the constraints derived from each user in order to acquire samples from the posterior via employing MCMC, as we detail below. Note that if the amount of the feedback received is small, simple sampling techniques like the popular rejection sampling [13] could be effective. But as the feedback increases, those sampling methods prove to be inefficient, because the valid convex region drastically shrinks. Sampling techniques like MCMC are more suitable for cases like this, because they are 'aware' of the feedback received, and can handle cases with higher dimensionality [25].

A major problem encountered in the sampling process was that in order for our MCMC algorithm to start collecting samples from the posterior distribution, it needs to start from a point that is located inside the valid convex region. Otherwise, the Markov chain remains stuck to the initial point. To overcome this problem we used Simplex algorithm. More specifically, for each constraint $i_j > i_k$, we introduce a variable $\epsilon_t$ so that we have:

$$U(i_j) > U(i_k) + \epsilon_t \tag{5}$$

and the objective function that we want to maximize is

$$max \sum_{t=1}^{T} \epsilon_t \tag{6}$$

where $T$ is the number of the constraints. This results in finding a weight vector that is closer to the center of the convex region. Otherwise, if we start from a 'bad' location in the distribution (the limits of the convex region), the sampler spends the first iterations to slowly move towards the main body of the distribution. With Simplex we were able to find a starting point that does not violate our constraints, which was necessary in order to iterate and start collecting posterior samples. After initiating the algorithm with a Simplex sample, we are ready to carry out the main sequence of its MCMC steps.

**Metropolis Hastings Algorithm** We first construct a regular grid in the $m$-dimensional hypercube, where $m$ is the number of features that each item has. Each $w_i$ takes values in [-1, 1] and our grid interval is set to 0.02. The main loop of our Metropolis-Hastings[4] is known Algorithm 1 consists of three components:

1. Generate a proposal (or a candidate) sample $x_{cand}$.

[4] We note that ours is essentially a standard version of the Metropolis-Hastings algorithm, which is known to be almost always convergent [21].

**ALGORITHM 1:** Our Metropolis Hastings Algorithm

Initialize x(0) with the sample from Simplex Algorithm;
$i$ = 1;
**repeat**
  With $p = 0.5$ accept the current sample $x_i$; continue;
  Propose: $x_i \sim$ Stochastic walk (described in text) ;
  Acceptance Prob: $\alpha(x_i|x_{i-1})$ = $\min(1, \frac{\pi(x_i)}{\pi(x_{i-1})})$;
  $p\sim$Uniform$(0, 1)$;
  **if** $x_i$ *satisfies all constraints and* $p < \alpha$ **then**
    accept $x_i$,
  **else**
    accept $x_{i-1}$,
  **end**
**until** $i >= numberofsamples$;

2. Compute an "acceptance probability" $\alpha$ for the candidate sample, via a function based on the candidates' distribution (usually termed *the proposal distribution*), and the full joint density. In fact, we use an estimate of the probability density function that corresponds to a clustering of the prior samples we have gathered from previous interactions.[5]
3. Accept the candidate sample with probability $\alpha$, the acceptance probability, or reject it with probability $1 - \alpha$.

Specifically, we choose a candidate as follows. With probability $0.5$ we keep the old sample $x_{i-1}$, and with the remaining probability, $x_{cand}$ (or $x_i$ as in Algorithm 1) is chosen with a stochastic walk from among $x_{i-1}$'s $2m$ neighbours as follows. After choosing (uniformly) the neighbour that we will visit first, we move forward with $0.5$ probability; and move backwards with $0.5$ probability also. After we have chosen the direction, we choose the number of steps that we are going to move in the grid. With $0.9$ probability we move one step in the grid. With probability $0.1$, we choose to move, with equal probability, either 2*step, 3*step or 4*step. We observed empirically that making, with small probability, relatively bigger "jumps" in the grid, helps in preventing the Markov chain from getting stuck in a particular part of the distribution. All probability values above were chosen empirically.

The next step after choosing $x_{cand}$, is to check if it satisfies all generated constraints. If this sample violates one or more constraints, then we reject it and we keep the old one; otherwise we decide whether to keep it based on the acceptance function. After we

[5] In some detail, we divide *each* of the $m$ dimensions into a fixed number of "segments"—ten (10) in our implementation—and use this segmentation to generate "buckets" to place our samples into. In this way, we create $10^m$ buckets in total: for instance, if we had only two dimensions, e.g. "price" and "distance to city center", we would be creating 100 buckets. Then, each prior sample is allocated to its corresponding bucket, based on Euclidean distance. When Algorithm 1 picks a sample, it checks which bucket it belongs to, and uses the number of samples in the buckets to estimate the $\pi(\cdot)$ density in Eq. 7. Thus, this method uses the prior samples to estimate the posterior joint density.

start collecting samples from the posterior, to avoid collecting samples that are highly correlated, it is common to pick a subset of them. So we introduce a "lag" parameter, set to 100 iterations in our implementation.

Now, there are mainly two kinds of "proposal distributions", symmetric and asymmetric. A proposal distribution is a symmetric distribution if: $q(x_i|x_{i-1}) = q(x_{i-1}|x_i)$ Standard choices of symmetric proposals include Gaussians or uniform distributions centered at the current state of the chain. Here we work with a symmetric (uniform) proposal distribution, as it makes the algorithm more straightforward, both conceptually and computationally. In cases with symmetric distributions like ours, the latter simplifies to [9]:

$$\alpha(x_i|x_{i-1}) = \min(1, \frac{\pi(x_i)}{\pi(x_{i-1})}) \tag{7}$$

where $\pi(\cdot)$ is the full joint density. Simply put, when the proposal distribution is symmetric, the probability $\alpha$ becomes proportional to how likely each of the current state $x_{i-1}$ and the proposed state $x_i$ are under the full joint density.

### 3.3 Ranking the Items

The ultimate goal of a recommender system is to suggest a short ranked list of items, namely the top-$k$ recommendations that are ideally the most appealing for the end user. A framework based on utility function, like ours, essentially defines a total order over all items. Moreover, a recommender naturally faces the dilemma of recommending items that best match its current beliefs about the user, or items that could improve user satisfaction and help form more accurate beliefs. This corresponds to the typical exploration vs exploitation problem in learning environments [23]. Although several ranking methods exist, there is no universally accepted ranking semantics given the uncertainty in the utility function. In our work, we use a ranking method based on expectation, used widely in the AI literature (see, e.g., [5, 7, 23]).

**Ranking based on the Expectation algorithm** The expectation algorithm *(Exp)* is defined as follows [25]. Given an item space I and probability distribution $P_w$ over weight vectors $w$, find the set of top-$k$ items with respect to their expected utility value. More specifically, for each item and every (sample) weight vector, we calculate its utility $U_{\boldsymbol{w}_l}(i)$ under each sample $\boldsymbol{w}_l$ vector, using Equation 1. Subsequently, the expected utility value for each item $i$ can be calculated given $U_{\boldsymbol{w}_l}(i)$ , and the probability of the corresponding $\boldsymbol{w}_l$ weight vector:

$$EU(i) = \sum_{l=1}^{L} U_{\boldsymbol{w}_l}(i) * Prob(\boldsymbol{w}_l) \tag{8}$$

where $L$ is the number of sample weight vectors. Finally, we sort the dataset in ascending order according to the items expected utility, and present the top $k$ ranked items to the user, after adding an explicit exploration component as follows.

**Suggesting Items** It is very important to introduce an exploration component in the system. User preferences are in many cases very complex and difficult to map completely. We confront this challenge by presenting some[6] random items along with those that the *Exp* algorithm returns. Those items serve the purpose of correcting the bias introduced from the initial distribution of $P_w$ and combating mistakes and noise from user feedback. Moreover, the uncertainty inherent in the utility function aids exploration, in a true Bayesian manner.

Now, a recommender makes suggestions to the user based on knowledge acquired through user's previous interactions with the system. A major problem occurs when we have gathered little or no data on what the user prefers. In those situations, the suggestions made, could be almost "blind". Such a process would converge slowly, and especially the initial beliefs could be far from the user's true preferences. In what follows, we propose a solution to address this problem.

### 3.4 Clustering

It is often the case in recommender systems that we have no knowledge of the preferences of a user, or that we have not gathered enough constraints in order to limit the uncertainty that the weight distribution $P_w$ has. This is the notorious "cold start" problem. The most common approach to tackle this problem is to randomly suggest items to the user in order to receive feedback and update the distribution. However, this has the disadvantage that suggested items in the initial interactions, might be far from the users' true preferences and this could result in producing suboptimal recommendations and therefore slow convergence of the weight distribution. Thus, instead of recommending random items in the initials interactions, we make use of a novel method that employs"clustered" weight vectors from other, existing users (essentially recommending items matching the preferences of corresponding formed user categories).

**Employing the Clusters** Every time we run *Metropolis-Hastings* to derive posterior samples for a user, we save those samples to our database. After having collected several weight vector samples from the posteriors of existing users, we cluster them into $k$ groups ($k$ being the number of recommendations made to a user in every interaction) using the popular $k$*-means* algorithm [16]. As mentioned, we use them to address scenarios of limited or no information regarding a new user. Instead of considering all weight vectors as candidates, we use the clusters' centroids, and suggest to the new user the most "appropriate" items given the preferences of each group. Specifically, the $k$ clucster centroids are used in the ranking algorithm, to rank and present to new users the items that are best fits for the clusters' centroids. By suggesting items based on the cluster centroids, and receiving feedback from the user, we drastically decrease the uncertainty that we have about their preferences, as demonstrated in our simulations.[7]

[6] Specifically, 2 out of 7 items presented to the user are chosen randomly; see section 4.2 below.

[7] The idea of employing clustering to address the "cold start" problem has also appeared in [26]. However, that work uses averaging over *user ratings* to produce recommendations that are appropriate for each cluster. In our work, we make no use of user ratings over items, and make

## 4 System Evaluation

In our work we used a real-world (anonymized) dataset consisting of 5000 hotels in the city of Paris, retrieved from a popular international hotel booking website in JSON format. In order to compose a user's preferences, we focus on the following hotel features: the price per day, the number of hotel 'stars', the rating it has received from clients[8], the distance from the city center, and the amenities that it has (e.g., breakfast, pool, spa etc). The features we focused on were chosen or made quantifiable. In our experiments, we assume that the user has chosen her destination, and we try to learn her preferences in order to make personalized recommendations, so that she should not need to search manually through the thousands of hotels available. Our users are synthetic: this allows us to measure our model's effectiveness more accurately, since we can compare recommendations to the "true" user preferences (which we have perfect knowledge of).

### 4.1 Building the Synthetic Users

We generated 100 simulated users that was used in order to evaluate our system's performance. Their preferences values were sampled from distributions. It was important that the simulated users represented as best as possible the preferences of the real users, in order to have a realistic assessment of the model performance. By creating the user preferences by sampling distributions, we introduce an uncertainty necessary to account even for users with unconventional preferences. The preferences of each user consist of five values.

After sampling the distributions we normalize all preferences values to lie in the $[0, 1]$ interval. In order to determine the price per day that a user is willing to pay for a hotel, we were inspired by the income distribution of the UK citizens for the year 2011.[9] Thus, for extracting the price preference, we used a Burr type XII distribution ($a = 25007, c = 2.0, k = 2.0$). To derive the hotel's stars preference, we assume that if the 'can pay' price is high, then it is expected that a hotel with a high number of stars (4-5) is preferred, so we sample a Beta distribution with the following parameters a = 8, b = 2. Respectively, we use a Beta with parameters $a = 2, b = 8$ if the price value is low. Otherwise, we take sample from a Beta distribution with $a = 8, b = 8$. Here, we choose a Beta distribution, as we want the number of stars to have mean values between zero and one. Additionally, such distributions are more appropriate for modelling uncertainties in the real world, as they can concentrate probability in a desired range [11]. Subsequently we choose to sample a Beta distribution with parameters $a = 1, b = 3$ in order to derive the proximity to the city center preference. Using this distribution we want to simulate most people's tendency to prefer a hotel near the city center rather than in the countryside. A sampled value closer to zero corresponds to a preference for a hotel close to the center, while value closer to 1 means the opposite. In order to specify the

recommendations based on the clusters' centroids rather than employing some averaging-over-cluster-contents process.

[8] Note that "clients' rating" is just an item's (a hotel's) feature. We stress that we do not ask our system's users to rate the items, and it is not the system's aim to produce recommendations based on such ratings.

[9] *https://www.gov.uk/government/statistics*

value of 'ratings by previous guests' value a user demands, we used a Beta with parameters $a = 6$ and $b = 2$. We choose this distribution in order to simulate the tendency that most users prefer hotels with high ratings. Finally, the last feature that characterizes the user preferences was the required hotel amenities. To derive the "amenities" preference, we used a uniform distribution in $[0, 1]$, with a higher number meaning that a hotel with more amenities is preferred. All amenities are assumed to have equal impact on a user.

### 4.2 Experiments and Results

In our experiments we used the synthetic users described above, each of whom had twenty (20) interactions with our system. In each interaction, seven (7) hotels are recommended to the user, and she chooses one. Five (5) of those are the top ranked ones according to our current beliefs regarding user preferences, while two (2) are picked completely randomly from our dataset. The choice made by the user is based on Euclidean distance:

$$d(h, u) = \sqrt{(h_{F1} - u_{F1})^2 + \cdots + (h_{Fn} - u_{Fn})^2} \tag{9}$$

where $h_{F1}, h_{F2}, ..., h_{Fn}$ the values of the corresponding features of the hotel, and $u_{F1}, u_{F2}, ..., u_{Fn}$ are the actual user preferences. Thus the user will always choose the hotel with the minimum Euclidean distance from her preferences.

$$Selection = \min(d(h_1, u), d(h_2, u), ..., d(h_k, u)) \tag{10}$$

where $k$ is the number of suggestions made to the user. In all experiments, our ranking algorithm uses 50 samples randomly picked from 10000 samples returned by the MCMC algorithm. This allows us *to keep response time under 2 sec*, on a PC with an i7@4.5GHz processor and 16GB of RAM.

Regarding our results, we first report the (average) mean square error (MSE) for each user interaction, calculated given the user's true value function and her "ideal" choices. This helps us assess the accuracy of the recommendation made as the interactions increases. As seen in Figure 2, the MSE of our method drops quite sharply, after only a few interactions.

The use of MSE does provide us with an insight of the overall method performance. We can provide further testimony to the effectiveness of our approach, however, by using the following "metric". For each simulated user, we found and saved the "top 200" ranked hotels for the 5000 hotels of our dataset, based on the minimum Euclidean distance from the user's true preferences. Then, we were able to compare the suggestions our model made with the top 200 hotels for each user. As shown in Figure 3, after only five interactions, our system was able to recommend hotels with 38% of them being among the best 200 hotels of each user (38% of the hotels suggested during the last 15 interactions were in the "top-200"). Note that the "top-200" constitutes only 4% of our dataset of 5000 hotels. Similarly, about 20% of the recommended hotels belong in the "top 50" (1% of our dataset), and 8% belong in the "top 10" (0.2% of our dataset).

We also report on the observed performance when using clustering to reduce our initial uncertainty regarding a user. In Figure 2, we observe a faster reduction of the

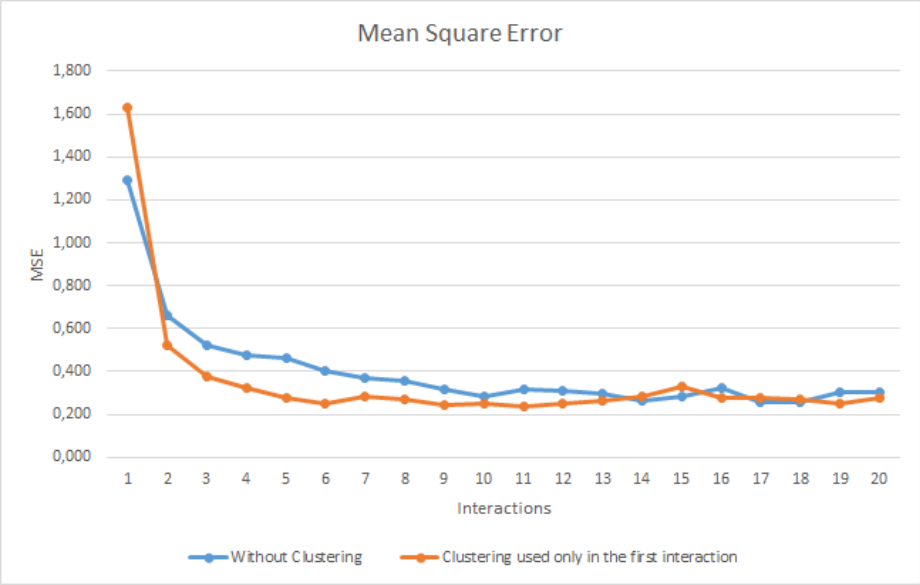


Fig. 2: Mean Squared Error with and without clustering.

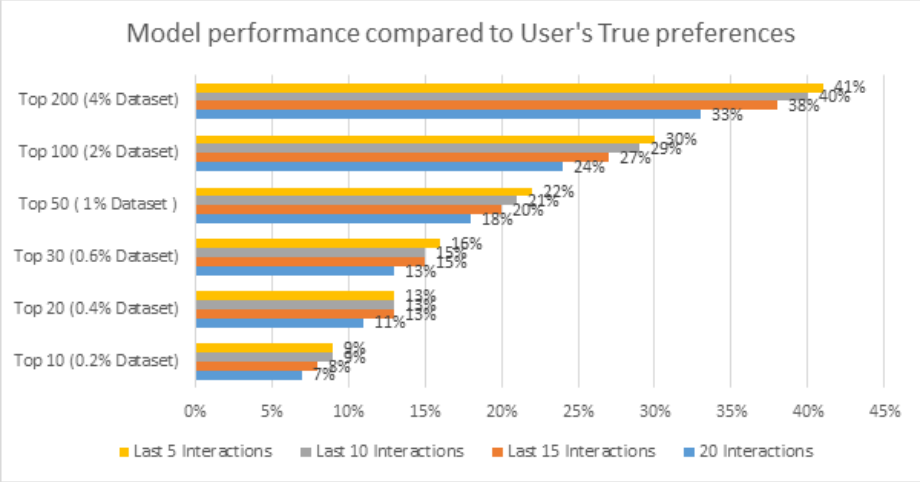


Fig. 3: Performance wrt to a user's true preferences (averages).

MSE when clusters are used. Naturally, in the long run the MSE converges to the same low value, with and without clustering (the latter loses its importance once enough information is available).

## 5 Conclusions and Future work

In this work, we presented a lightweight recommendation system which uses a Bayesian preference elicitation framework, and applied it in the online hotel booking recommendations domain. The algorithm makes personalized recommendations, is simple and fast, but still generic and effective. Our system is non-intrusive, and does not ask the user to rate items. It tackles conflicting user preferences and the problem of having scarse information about a user, while it possesses a simple exploration component. Our simulation experiments confirm the effectiveness of our method.

There are many possible extensions to this work. An obvious future step is to directly compare its effectiveness against other hotel booking recommender systems. This is however a non-trivial task, given that most existing systems are not open-source or fully described in research articles, and thus a direct comparison with the same dataset and user pool is next to impossible to conduct. Another direction we aim to take is experimenting with real users, employing questionnaires in order to evaluate and measure the system's performance. Additionally, it is very common that a user is affected by unexpected factors or unquantifiable features such as the presentation of the item or the available pictures. For instance, in a booking recommendation scenario like ours, the user may not choose based solely on features: on the contrary, she may get influenced by the beautiful photos of a hotel. There exist numerous works in the marketing and advertising domains that describe how the presentation of an item can strongly influence the choices of a user. Thus, in order to take cases like these into account, we aim to incorporate a noise model to simulate unexpected and unquantifiable factors affecting user choices, and thus obtain a more realistic user preferences model.

*Acknowledgements* The authors would like to thank Professor Michail Lagoudakis for extremely useful suggestions for improving an earlier version of this work.

## References

1. Andrieu, C., de Freitas, N., Doucet, A., Jordan, M.I.: An introduction to mcmc for machine learning. Machine Learning **50**(1), 5–43 (Jan 2003). https://doi.org/10.1023/A:1020281327116, https://doi.org/10.1023/A:1020281327116
2. Applegate, D., Kannan, R.: Sampling and integration of near log-concave functions. In: Proceedings of the twenty-third annual ACM symposium on Theory of computing. pp. 156–163 (1991)
3. Babas, K., Chalkiadakis, G., Tripolitakis, E.: You Are What You Consume: A Bayesian Method For Personalized Recommendations. In: Proceedings of the 7th ACM Conference on Recommender Systems (ACM RecSys 2013), Hong Kong, China (2013)
4. Bishop, C.M.: Neural networks for pattern recognition (1995)
5. Boutilier, C.: A POMDP Formulation of Preference Elicitation Problems. In: Proceedings of the 18th AAAI Conference (AAAI-02) (2002)
6. Bowling, M.H., Veloso, M.M.: Multiagent learning using a variable learning rate. Artificial Intelligence (AIJ) **136**(2), 215–250 (2002)
7. Chajewska, U., Koller, D., Parr, R.: Making rational decisions using adaptive utility elicitation. In: Proceedings of the 17th AAAI Conference (AAAI-00)). pp. 363–369 (2000)

8. Chajewska, U., Koller, D., Ormoneit, D.: Learning an agent's utility function by observing behavior. In: Proceedings of the International Conference on Machine Learning (ICML) (2001)
9. Chib, S., Greenberg, E.: Understanding the Metropolis-Hastings algorithm. pp. 327–335 (2012)
10. Dantzig, G.B., Orden, A., Wolfe, P.: The generalized simplex method for minimizing a linear form under linear inequality restraints. Pacific Journal of Mathematics **5**(2) (1955)
11. DeGroot, M.: Probability and statistics. Addison-Wesley series in behavioral science, Addison-Wesley Pub. Co. (1975), https://books.google.gr/books?id=fxPvAAAAMAAJ
12. Dong, R., Smyth, B.: From more-like-this to better-than-this: Hotel recommendations from user generated reviews. In: Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization (UMAP'16). pp. 309–310 (2016)
13. Gilks, W.R., Wild, P.: Adaptive Rejection Sampling for Gibbs Sampling. In: Journal of the Royal Statistical Society. Series C (Applied Statistics) Vol. 41, No. 2. pp. 337–348 (1992)
14. Hastings, W.K.: Monte carlo sampling methods using markov chains and their applications. Biometrika **57**(1), 97–109 (1970), http://www.jstor.org/stable/2334940
15. Keeney, R.L., Raiffa, H.: Decisions with Multiple Objectives: Decisions with Preferences and Value Tradeoffs. Cambridge University Press (1993)
16. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability (1967)
17. Nasery, M., Braunhofer, M., Ricci, F.: Recommendations with optimal combination of feature-based and item-based preferences. In: Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization (UMAP'16). pp. 269–273 (2016)
18. Ng, A.Y., Russell, S.J.: Algorithms for inverse reinforcement learning. In: Proceedings of the Seventeenth International Conference on Machine Learning. pp. 663–670. ICML (2000)
19. Ramachandran, D., Amir, E.: Bayesian Inverse Reinforcement Learning. In: Proceedings of IJCAI-2007. pp. 2586–2591 (2007)
20. Ricci, F., Rokach, L., Shapira, B., Kantor, P.B.: Recommender Systems Handbook. Springer-Verlag New York, Inc., New York, NY, USA, 1st edn. (2010)
21. Robert, C.P., Casella, G.: Monte Carlo Statistical Methods. Springer Science and Business Media (2004)
22. Roy, S.B., Das, G., Amer-Yahia, S., Yu, C.: Interactive itinerary planning. In: Proceedings of the IEEE International Conference on Data Engineering (ICDE) (2011)
23. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 3rd edn. (2009)
24. Tripolitakis, E., Chalkiadakis, G.: Probabilistic topic modeling, reinforcement learning, and crowdsourcing for personalized recommendations. In: 14th European Conference on Multi-Agent Systems (EUMAS 2016), Valencia, Spain, December 15-16, 2016, Revised Selected Papers. pp. 157–171 (2016)
25. Xie, M., Lakshmanan, L.V., Wood, P.T.: Generating Top-k Packages via Preference Elicitation. In: Proceedings of the VLDB Endowment Volume 7 Issue 14. pp. 1941–1952 (2014)
26. Yanxiang, L., Deke, G., Fei, C., Honghui, C.: User-based clustering with top-n recommendation on cold-start problem. In: 2013 Third International Conference on Intelligent System Design and Engineering Applications (2013)