Ben-Gurion University of the Negev
Department of Compute Science

An MDP-Based Recommender System

Thesis submitted as part of the requirements for the
M.Sc. degree of Ben-Gurion University of the Negev

by

Guy Shani

The research work for this thesis has been carried out at
Ben-Gurion University of the Negev
under the supervision of Dr. Ronen Brafman

July 2002

Subject: **An MDP-Based Recommender System**

This thesis is submitted as part of the requirements for the M.Sc. degree
Written by: **Guy Shani**
Advisor: **Dr. Ronen Brafman**
Department: **Computer Science**
Faculty: **Natural Sciences**
Ben-Gurion University of the Negev

Author signature: _____ Date: _____

Advisor signature: _____ Date: _____

Dept. Committee Chairman signature: _____ Date: _____

Abstract

Recommender systems — systems that suggest to users in e-commerce sites items that might interest them — adopt a static view of the recommendation process and treat it as a prediction problem. We argue that it is more appropriate to view the problem of generating recommendations as a sequential decision problem and, consequently, that Markov decision processes (MDP) — a well known stochastic model — provide a more appropriate approach for recommender systems. MDPs introduce two benefits: they take into account the long-term effects of each recommendation, as well as the expected value of each recommendation. A fundamental problem in using MDPs is the weak performance of the model before its parameters are learned online. Therefore, to succeed in practice, an MDP-based recommender system must employ a strong initial model; we suggest an approach for the generation of such a model. In particular, we suggest the use of an n-gram predictive model — models that are used in the field of speach recognition — for generating the initial MDP. Our n-gram model induces a Markov-chain model of user behavior whose predictive accuracy is greater than that of existing predictive models. We describe our predictive model in detail and evaluate its performance on real data. In addition, we show how the model can be used in an MDP-based recommender system.

Acknowledgements

I would like to thank Dr. Ronen Brafman who guided me through this work, supplied many deep insights, and patently endured many debates where I tried to prove myself right even though I knew little of the subject at hand.

This work would not have been written without the help of Dr. David Heckerman who introduced me to the field of collaborative filtering and online recommendations, and provided willing help each time I was stuck with a question I was too lazy to find an answer to myself. David also was a main figure in writing the paper that stands at the basis of this thesis.

The Israeli online bookstore Mitos (www.mitos.co.il) provided us with input data on which to base our experiments and a test bed for our recommender system. I would like to thank Amitay Dobo and especially Ruthi Bendor (Senior Developers) who tolerated my endless nagging and dedicated their valuable time to this project.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In many markets, consumers are faced with a wealth of products and information from which they can choose. To alleviate this problem, many web sites attempt to help users by incorporating a *Recommender system* (Resnick & Varian, 1997) that provides users with a list of items and/or web-pages that are likely to interest them. Once the user makes her choice, a new list of recommended items is presented. Thus, the recommendation process is a sequential process. Moreover, in many domains, user choices are sequential in nature, e.g., we buy a book by the author of a recent book we liked.

The sequential nature of the recommendation process was noticed in the past (Zimdars, Chickering, & Meek, 2001). Taking this idea one step farther, we suggest that recommending is not simply a sequential prediction problem, but rather, a sequential decision problem. At each point the Recommender system makes a decision: which recommendation to issue. This decision should be optimized taking into account the sequential process involved and the optimization criteria suitable for the Recommender system. Thus, we suggest the use of Markov Decision Processes (MDP) (e.g., Puterman, 1994), a well known stochastic model of sequential decisions. With this view in mind, a more sophisticated approach to Recommender systems emerges. First, one can take into account the utility of a particular recommendation – e.g., we might want to recommend a product that has a slightly lower probability of being bought, but generates higher profits. Second, we might suggest an item whose immediate reward is lower, but leads to more likely or more profitable rewards in the future. These considerations are taken into account automatically by any good or optimal policy generated for an MDP model of the recommendation process. In particular, an optimal policy will take into account the likelihood of a recommendation to be accepted by the user, the immediate value to the site of such an acceptance, and the long-term implications of this on the user's future choices. And these considerations are taken with the appropriate balance to ensure the generation of maximal expected reward stream. For instance, consider a site selling electronic appliances faced with the option to suggest a video camera with a success probability of 0.5, or a VCR with a probability of 0.6. The site may choose the camera, which is less profitable, because the camera has accessories that are likely to be purchased, where as the VCR does not. If a video-game console is another option with a smaller success probability, the large profit from the likely future event of selling game cartridges may tip the balance toward this latter choice.

Keeping these benefits in mind, we suggest an approach for the construction of an MDP-based Recommender system. The first, and most crucial step in the construction of our model is the generation of a powerful predictive model. We believe that in any real environment, it is essential to start with a powerful initial model for the MDP—commercial sites will not accept a model that generates poor recommendations initially with the promise of better performance in the future. Use of standard reinforcement learning techniques for MDPs would perform poorly at first due to the amount of needed explorations before converging into a near-optimal policy. Using the MDP framework for supplying mall navigation recommendations was suggested before by Bohnenberger and Jameson (2001), but not for a collaborative filtering based recommender system.

The predictive model we describe is motivated by our sequential view of the recommendation process, but constitutes an independent contribution. The model can be thought of as an n-gram model (e.g., Chen & Goodman, 1996) or, equivalently, a (first-order) Markov chain in which states correspond to sequences of events. In this thesis, we emphasize the latter interpretation due to its natural relationship with an MDP. We note that Su, Yang, and Zhang (2000) have described the use of simple n-gram models for predicting web pages. Their methods, however, yield poor performance on our data.

Validating our MDP approach is not simple. Most Recommender systems, such as dependency networks (Heckerman, Chickering, Meek, Rounthwaite, & Kadie, 2000), are tested on historical data for their predictive accuracy. That is, the system is trained using historical data from sites that do not provide recommendations, and tested to see whether the recommendations conform to actual user behavior. This test provides some indication of the system's abilities, but it does not test how user behavior is influenced by the system's suggestions or what percentage of recommendations are accepted by users. To obtain this data, one must employ the system in a real site with real users, and compare the performance of this site with and without the system (or with this and other systems). We also note that testing our MDP model using standard techniques would yield worse results than using the Markov- chain model since the MDP would recommend items by their expected value to the site, rather than by their probability of being interesting to the user. For this reason, testing the MDP model on a test bed that doesn't allow interactions with users is useless. This issue has been a problem for past work as well as for this work – it is difficult to convince commercial sites to use experimental systems, and nearly impossible to convince them to compare two different systems. Thus, like most previous work, we evaluate only the predictive component of our model.

In this paper we view the recommendation process as a sequential optimization problem, suggest the MDP model as an adequate representation and offer a smart way to initialize, update and solve the MDP. For the initialization of the MDP we suggest the use of an n-gram based Markov chain model using some improvements that rely heavily on the field of n-gram models, yet differ in their adaption to our domain. We evaluate our approach using standard recommender system metrics and conclude that our Markov chain model is superior to state of the art algorithms.

The main contributions of this work are:

1. We suggest viewing the recommendation process as a sequential optimization problem.

2. We show how this problem can be formalized and modeled using a Markov Decision Process.

3. We show how the MDP can be implemented, initialized using a prediction model and solved efficiently in practice. We hope our approach might be applicable to other MDP implementations as well.

4. We suggest a new n-gram based Markov Chain prediction model that performs better than state of the art recommendation algorithms.

5. We suggest a new method to evaluate the performance of recommender systems by estimating the utility the site assigns to the recommendations rather than the click through.

The thesis is structured as follows:

- Chapter 2 supplies background on recommender systems in general and collaborative filtering algorithms in particular.
- Chapter 3 shortly overviews the origins of n-grams and focuses on some of the common improvements to the original model such as skipping, clustering and mixture models.
- Chapter 4 supplies definitions for MDPs and shows how they can be solved using Policy Iteration.
- Chapter 5 explains the structure of the n-gram based Markov Chain model.
- Chapter 6 describes the MDP based model, explaining how it can be initialized and updated online using the unique attributes of the problem.
- Chapter 7 explains the evaluation criteria, comparison models and the results of the experiments we have conducted.
- Chapter 8 details performance issues including model build time, model size and number of recommendations per second.
- Chapter 9 describes the implementation of our system in the online book store Mitos.
- Chapter 10 summarizes this work and suggests a few possible paths to expand the research.

Chapter 2

Recommender Systems

Early in the 90's, when the internet became widely used as a source of information, *information explosion* became an issue that needed addressing. Many web sites presenting a wide variety of content (such as articles, news stories or items to purchase) discovered that users had difficulties finding the items that interest them out of the total selection.

When the user knows what to look for (title, author or some other information about the item), the problem can be solved using smart search engines. The problem becomes even more difficult though, when the user is browsing, say, a site of news stories, looking for some interesting story without any specific subject in mind.

So called *recommender systems* (Resnick & Varian, 1997), help users limit their search by supplying a list of items that might interest a specific user. Different approaches were suggested for supplying meaningful recommendations to users and some where implemented in modern sites (Schafer, Konstan, & Riedi, 1999). Traditional data mining techniques such as association rules have been tried at the early stages of the development of recommender systems, but proved to be insufficient for the task. Approaches originating from the field of *information retrieval (IR)* rely on the *content* of the items (such as description, category, title, author) and therefore are known as *content-based recommendations* (Mooney & Roy, 2000). These methods use some similarity score to match items based on their contents, then a list of similar items to the ones the user previously selected can be supplied. Another possibility would be to avoid using information about the content, but rather use historical data gathered from other users in order to make a recommendation. These methods are widely known as *collaborative filtering (CF)* (Resnick, Iacovou, Suchak, Bergstorm, & Riedl, 1994). Many new systems try to create hybrid models that combine collaborative filtering and content-based recommendations together (Balabanovic & Shoham, 1997).

2.1 Collaborative Filtering

The collaborative filtering approach originates in human behaviour; people trying to find an interesting item they know little of (e.g. when trying to decide which movie to take from the video store), tend to rely on friends to recommend items they tried and liked (good movies they saw). The person asking for advice is using a (small) community of friends that know her taste and can therefore make good predictions as to whether she will like a certain

item. Over the net however, a larger community that can recommend items to our user is available, but the persons in this gigantic community know little or nothing about each other. The goal of a collaborative filtering engine is to identify those users whose taste in items is predictive of the taste of a certain person (usually called a *neighborhood*), and use their recommendations to construct a list of items interesting for her.

To build a user's neighborhood, these methods rely on a database of past users interactions with the system. Originally, the database consisted of items and some type of score given by the users who experienced those items (e.g. 5 stars to a great movie, 1 star to a horrible one). The user had to enter the site, grade a few items, and then a list of items she did not grade, but that the system predicts she will score highly, can be presented to her [1]. This approach is called *explicit ratings*, since users explicitly rate the items. Recommender systems that used explicit ratings were found in sites that recommended movies, books, music cds and so forth, but also in news sites, that tried to help the users find interesting news articles. GroupLens (Resnick et al., 1994) was one of the pioneer systems in which users were asked to rate stories after reading them and then received a list of stories graded highly by similar users. Recommendation engines that used explicit ratings needed to address the notion of positive and negative votes. Users who usually rate items highly (e.g. give every movie no less than three stars) should be handled differently than users who use all the score range uniformly. A neutral vote is therefore computed for each user and her actual votes are considered by their distance from the neutral vote.

It soon became clear however that in many cases, and especially in e-commerce sites, users do not want to spend their time grading items. E-commerce sites developed great interest in supplying their clients with good recommendations, causing users to buy more items thus raising the site profits. In order to make collaborative filtering methods applicable for these sites, we make an assumption that users liked an item that they bought. This assumption is often false, but for the lack of a better method for deciding which items interest a user, current recommender systems are using it nonetheless. Claypool et al. (Claypool, Le, Wased, & Brown, 2001) suggested using other methods through a special web browser that kept track of user behavior such as the time spent looking at the web page, the scrolling of the page by the user, and movements of the mouse over the page, but failed to establish a method of rating that gave consistent better results. This rating system is often called *implicit ratings* since an assumption about the user's ratings is made without her actually grading items. Implicit ratings are also binary: *liked, didn't like*, eliminating the problem of the neutral vote discussed above.

At first, CF methods viewed the database as a set of user vectors $V = \{v_a\}$ when $v_{a,j} = r$ if user a gave a grade of r to item j. A vector representing the historical activity of a user is called a *case*. It is clear that in many cases, most items would not be graded by the user, due to the amount of items (e.g. millions of movie titles). The user ratings vector is therefore rather sparse as most values are missing. Generally speaking, it is reasonable to assume that users like items they have rated, and rate items they like. Missing values can therefore be treated not as neutral votes, but rather as negative votes. A sparsity problem also emerges from the viewpoint of items that might be rated by only a small amount of users. It is reasonable to assume that, whether we use implicit or explicit rating system,

[1] An example of such a system can be found at http://www.movielens.umn.edu/.

many items will not be rated by enough users to yield a reasonable statistical sample for them. Many different methods have been suggested to handle the sparsity problem, including using automatic agents with a specific "taste" (e.g. a horror agent, a comedy agent and a drama agent, in a movie site) going through the items and grading them (Good, Schafer, Konstan, Borchers, Sarwar, Herlocker, & Riedl, 1999), and decomposing the user-item rating matrix to smaller matrices , thus both lowering the complexity and increasing the density of the item ratings (Sarwar, Karypis, Konstan, & Riedl, 2000a).

2.2 Memory-based and Model-based CF Algorithms

Systems using the database directly are known as *user-based* or *memory-based* methods. Those methods predict the ratings of a user based on her known data and a set of weights calculated from the user database, using the following method: if I_i is the set of items that user i has rated, let us define the mean vote for user i as:

$$mean_i = \frac{1}{|I_i|} \sum_{j \in I_i} v_{i,j}$$

The predicted rate user a would assign to item j, $p_{a,j}$, is a weighted sum of the ratings assigned by the other users.

$$p_{a,j} = mean_a + k \sum_{i=0}^{n} w(a,i)(v_{i,j} - mean_i)$$

where user a is the *active user* - the user currently logged onto the site, n is the number of users, and k is a normalizing factor. It is clear that the actual value of $p_{a,j}$ depends heavily on the weights $w(a,i)$. Different memory-based algorithms, such as correlation and vector similarity use different methods to calculate those weights. Algorithms using memory-based techniques are considered quite accurate (Breese, Heckerman, & Kadie, 1998), but introduce a new problem - scalability. It is reasonable for modern sites to have millions of users, and millions of items. The database used to generate the recommendations can easily become too big to allow for efficient calculations. E-commerce sites however demand the fastest performance available - recommender systems that force users to wait a long time for the web page to show, might drive users to other sites, thus decreasing the site revenues. It soon became clear that faster, more scalable algorithms are needed.

Model-based or *item-based* algorithms use the database to construct a model that specifies dependencies between ratings of items, rather than user vectors. The predicted rate on unobserved items is defined as the expected value the user will assign to an item given what we know about the user rating history:

$$p_{a,j} = E(v_{a,j}) = \sum_{i=0}^{m} pr(v_{a,j} | v_{a,k}, k \in I_a) i$$

where possible values (ratings) range from 0 to m, and the probability expression is the probability that the active user will assign value i to item j given her previous ratings on

other items. These models are pre-calculated and optimized so that they will be as fast as possible when used online (Karypis, 2001). Many statistical models have been suggested over time to specify the relationships between items in an efficient way, including clustering models (Ungar & Foster, 1998), Baysesian models (Chen & George, 1999) and dependency networks with decision trees (Heckerman et al., 2000).

2.3 Dependency Networks and Decision Trees

Bayesian networks have shown good abilities when used for collaborative filtering. They are however not clear enough to people lacking the technical background necessary to understand the relationships described by the model. This model tries to capture the ideas described by a Bayesian network, using a simpler graphical representation, more understandable to common people. The probabilistic dependencies between variables are specified by probabilistic decision trees. Heckerman et al. showed this model to be computationally efficient and quite accurate comparing to current state of the art algorithms. Dependency networks are used by Microsoft as part of the Predictor component of Commerce Server 2002 as a tool for supplying recommendations in online sites.

2.4 Sequential Recommendations

It has been suggested before (Zimdars et al., 2001) that one can look at the input data not as sets of ratings but also consider the order in which they were rated. Most recommender systems work in a sequential form: they suggest items to the user who can then take one of the recommendations. At the next stage a new list of recommended items is calculated and presented to the user. This process is clearly sequential by its nature - at each stage a new list is calculated based on the user past ratings.

Case ID	X_{-2}	X_{-1}	X_T
1	–	–	x_1
2	–	x_1	x_2
3	x_1	x_2	x_3
4	x_2	x_3	x_4

Table 2.1: Transforming the case x_1, x_2, x_3, x_4 using data expansion into 4 new cases

Zimdars et al. suggested using standard algorithms such as dependency networks by transforming the input data into a sequential form. The set of cases in the input data set for the training algorithm is transformed into a new data set, representing sequential structure. The data transformation method that showed the best performance is the *data expansion* method, originating in the n-grams approach (see below). It divides the data into two variable groups: source variables $X_{-k}, X_{-k+1}, ..., X_{-1}$ representing the last k items that were rated, and a target variable X_T that represents the next item in the sequence. Table 2.4 shows how we convert a case into a sequential form. A predictive model is then built to

predict X_T given $X_{-k}, ..., X_{-1}$. Dependency networks using a data set that was transformed using data expansion showed superior performance to non-sequential dependency networks.

Chapter 3

N-Grams Models

N-grams models originate from the field of *language modeling*, where they are used to calculate the probabilities of the next word w_i given the sequence of previous words - $p(w_i|w_{i-k}, ..., w_{i-1})$. This is highly useful in many applications, but especially in *speech recognition*, when attempting to decide what the next word should be. Translating the sequence of sounds into words is far from trivial, and usually this translation is inconclusive. At this stage it is possible to figure out the next word from the possible translations, using a model that assigns a probability to each option based on the words that were translated earlier. An n-gram model truncates the sequence of previous words to a length of $n-1$. The basic model is the *tri-gram* - a model that looks back two words in order to predict the next one. Even though this basic model is quite simple, it performs well in many applications. Many improvements were studied for this simple model including higher order n-grams, smoothing, clustering and skipping (Chen & Goodman, 1996).

3.1 Smoothing

In speech recognition, the system might work on the sounds corresponding to the sequence of words $w_{i-n}, ..., w_i$ using an n-gram model to decide between the possible alternatives. Now, it might be the case that there is only one alternative for w_i given the input sounds, since the speaker has pronounced w_i very clearly, or w_i is easy to understand. In that case we would not want the system to assign a zero probability for the sequence $w_{i-n}, ..., w_i$ even if this sequence was never observed in the training set. Smoothing techniques try to eliminate zero probabilities by taking some probability from certain sequences and redistributing it to other word sequences. Different methods for smoothing exist including Katz (Katz, 1987), Kneser-Ney (Kneser & Ney, 1995) and Jelinek-Mercer (Jelinek & Mercer, 1980).

3.2 Higher Order N-Grams

The decision to use tri-grams seems rather arbitrary. In many cases it might make sense to look farther into the history, or to look less than two words back. It was therefore suggested to build k n-gram models $M_1, ..., M_k$, with the n value ranging from 1 to k, and

then use *backoff* or *interpolation* to combine them together. When building higher order n-grams, it is less likely that a certain sequence of words will appear enough times to collect meaningful statistics for it. The backoff technique tries to use the highest order n-gram for which the current sequence has been observed a sufficient number of times. If for example the sequence w_1, w_2, w_3 (for the four-gram model) was not observed, the system will try the sequence w_2, w_3 (for the tri-gram model) instead. A more complex approach would be to use all the n-gram models together using interpolation. A series of weights $\lambda_1, \lambda_2, ..., \lambda_k$ will be defined such that $\sum_{i=1}^{k} \lambda_i = 1$ and for all i, $0 <= \lambda_i <= 1$, and

$$p(w_i|w_{i-k}, ..., w_{i-1}) = \sum_{j=1}^{k} \lambda_j p_{M_j}(w_i|w_{i-j}, ..., w_{i-1})$$

where p_{M_i} is the probability that i-gram model assigns to the sequence of words. Note that it does not make sense to use the same weights for all different sequences. Intuitively, sequences that were observed many times would probably tend to be more accurately predicted by a higher order n-gram, while sequences that were observed less, might be better predicted by a lower order n-gram. Sequence-dependent weights might not be practical though, due to the computational complexity of learning them. A reasonable approach would therefore be to decide on a (small) amount of buckets - sets of sequences that can use the same mixture weights, and learn both the division of word sequences into the buckets, and the proper weights of each bucket.

3.3 Skipping

When using larger n-grams, the chances of viewing the exact same context many times decreases. Similar sequences might have been viewed many times though. For instance, the phrase "playing soccer with a ball" might never have been seen, but the phrase "playing basketball with a ball" is very frequent. We would be interested in giving a probability to the sequence "playing — with a ball". More formally we would define a set of n weights $\lambda_1, ..., \lambda_n$ and then use:

$$p(w_i|w_{i-n}, ..., w_{i-1}) =$$

$$\sum_{j=1}^{n} \lambda_j p(w_i|w_{i-n}, ..., w_{i-j-1}, w_{i-j+1}, ..., w_{i-1})$$

where $p(w_i|w_{i-n}, ..., w_{i-j-1}, w_{i-j+1}, ..., w_{i-1})$ is the probability that w_i will be the next word given the former n words excluding the $j-th$ word from the sequence, and λ_j defines the importance of the $j-th$ word, a higher value for λ_j means that we consider the word at that position to be somewhat irrelevant and thus replaceable by other words, a smaller value denotes a high importance to the appearance of the word in the sequence.

3.4 Clustering

A similar, yet more advanced approach is to use clusters of words. It might be the case that some sequences of words were never seen, but their probability is still quite high. For

example the sequences "playing soccer" and "playing basketball" might be quite frequent in the training set, but the sequence "playing volleyball" was never observed. The probability for such a sequence should be similar to the two former sequences. We would be interested in defining a probability for the sequence "playing GAME" and then use $p(game|GAME)$ to come up with right probabilities. GAME is sometimes viewed as a cluster of words, and this method is therefore known as clustering. If we assume that a word w_i can belong to only one cluster W_i (hard clustering) we could use:

$$p(w_i|w_{i-n}, ..., w_{i-1}) = p(W_i|W_{i-n}, ..., W_{i-1})p(w_i|W_i)$$

The clusters of words need to be learned from the input training set. Many variations exist on how to use the clusters and interpolate clustering with other methods, such as combining word-based and cluster-based estimations (Chen & Goodman, 1996).

Chapter 4

Markov Decision Processes (MDPs)

An MDP is a model for sequential stochastic decision problems. As such, it is widely used in applications where an autonomous agent is influencing its surrounding environment through actions (e.g. a navigating robot). MDPs have been known in literature for a while now (e.g., Howard, 1960) but due to some fundamental problems discussed below not many commercial applications have been implemented.

4.1 MDP Definitions

An MDP is defined by a four-tuple: $\langle S, A, R, tr \rangle$, where S is a set of states, A is a set of actions, R is a reward function, and tr is the state-transition function.

4.1.1 States

A state $s \in S$ encapsulates all the relevant information about the world. We usually define a set of state variables v_i associated with a range of values R_{v_i} (different variables might have different ranges). A state is therefore defined by some assignment of values to the variables $s = (v_0 = r_0, v_1 = r_1, ..., v_n = r_n)$. The state space size is hence exponential in the number of state variables causing us to try and limit the state variables as much as possible. Some work has been done in order to learn the proper set of state variables given all possible variables (McCallum, 1996) but we note that this approach requires more initial experience in order to converge to an optimal representation and hence to an optimal solution.

4.1.2 Actions

The actions influence the world (moving cubes using a robot's arm) or the location of the agent in the world (making the wheels of the agent turn), thus triggering state changes. We say that action a transitions us from state s to state s' if an agent that is initially in state s uses action a and reaches state s'. Actions usually have stochastic effects — the cube might fall from the arm in mid-air, the engine spinning the wheels is imprecise, stopping the spinning earlier than expected or failing to reach the designed speed, or the robot hand might grab the item too softly, causing in to fall in mid air.

4.1.3 The Transition Function

The stochastic nature of actions is captured by the transition function tr. The transition function assigns a probability distribution to every (state, action) pair. Thus, $tr(s, a, s')$ is the probability of making a transition from state s to state s' when a is performed. In many cases the transition function is unknown to begin with and needs to be learned.

4.1.4 Rewards

Finally, the reward function $R(s, a)$ assigns a real value to each (state, action) pair describing the immediate reward (or cost) of executing this action in that state. Often, as in our case, the reward is only a function of the state $R(s)$, and thus, is a measure of the desirability of reaching each state. Even though the rewards are sometimes unknown and need to be learned, for our application rewards are predefined.

4.2 Policies

In an MDP, the decision-maker's goal is to behave so that some function of its reward stream is maximized – typically the average or discounted sum of future rewards the agent shall collect while acting in the world. This sum is usually discounted so that the value of rewards collected in the near future is higher than rewards collected later. An optimal solution to the MDP is such a maximizing behavior. Formally, a stationary policy for an MDP π is a mapping from states to actions, specifying which action we should perform at each state. The policy can be stationary since we assume an infinite horizon of actions, making the current action indifferent to the time at which it was executed, and since the world is assumed to change only by the actions of the agent. Given such an optimal policy π and the current state s where the agent is located, the agent needs only to execute the action $a = \pi(s)$ on every step of the decision process.

Various exact and approximate algorithms exist for computing an optimal policy, and the best known are policy-iteration(Howard, 1960) and value-iteration (Bellman, 1962).

In value-iteration we define a value for a state as the expected discounted sum of future rewards when executing the policy from the current state. We initialize $V_0(s) = \max_{a \in A} R(s, a)$ and define

$$V_i(s) = \max_{a \in A} R(s, a) + \gamma \sum_{s_j \in S} tr(s, a, s_j) V_{i-1}(s_j)$$

where $0 < \gamma < 1$ is a discounting factor representing the higher value of closer rewards compared to rewards that will be earned in the far future, and ensuring convergence of the algorithm. Once V_i converges to V^* we define

$$\pi(s) = argmax_{a \in A} R(s, a) + \sum_{s_j \in S} tr(s, a, s_j) V^*(s_j)$$

as the optimal policy for the MDP. Value iteration ensures us that we converge towards an optimal value function.

In policy-iteration we search the space of possible policies rather than the space of possible value functions. The algorithm initializes $V_0(s) = \max_{a \in A} R(s,a)$ (as in value-iteration). On each step we optimize the new policy π_i with respect to the former value function V_{i-1}, and compute the new value function $V_i + 1$ based on the policy π_i.

$$\pi_i(s) = argmax_{a \in A} R(s,a) + \sum_{s_j \in S} tr(s,a,s_j) V_i(s_j)$$

$$V_{i+1}(s) = R(s, \pi_i(s)) + \gamma \sum_{s_j \in S} tr(s, \pi_i(s), s_j) V_i(s_j)$$

Once the policy stops changing we have reached the optimal policy, even though the values might still be just approximations of the optimal values. In practice, policy-iteration converges faster than value-iteration.

Solving MDPs is known to be a polynomial problem in the number of states, and therefore exponential in the number of state variables. This is known as the "curse of dimensionality" — the fundamental problem with MDPs — in many cases the straight forward implementation can not be solved efficiently. The latest advances in the MDP field focus on computing an optimal policy in a tractable manner using factored representations of the state space and other techniques (e.g. Poupart, Boutilier, Schuurmans, & Patrascu, 2002), but we take a different approach — exploiting the special attributes of the problem to compute a near-optimal solution. Using those features (e.g. the observation that most states — item sequences — are unlikely to occur) we manage to compute an approximation of the optimal policy very fast.

4.3 Partialy Observable MDPs

A well known extension to the MDP model is the Partially Observable Markov Decision Process (POMDP) model (e.g. Cassandra, Kaelbling, & Littman, 1994). In a POMDP the agent is unable to tell the current state for certain and is therefore forced to estimate the current state given the set of current observations (e.g. output of the robot sensors). More formally, a POMDP is a six-tuple $\langle S, A, R, tr, \Omega, O \rangle$, where S, A, R, tr define an MDP, Ω is a finite set of possible observations of the world and $O(s,a,o)$ is the probability of observing o after executing action a and landing in state s. In most applications the formalization of the problem as a POMDP is more natural and complete, but POMDPs increase the difficulty of computing an optimal solution and are therefore undesirable if the problem can be reasonably formalized as an MDP.

4.4 Markov Chains (MC)

A less sophisticated model for dynamic systems, where the system moves from state to state without any action, is the Markov-Chain model. A Markov-Chain is constituted of a set of states S where on each step the system moves from the current state to one of its successors, without performing any action. The transition function $tr_{MC}(s,s')$ describes the probability

of moving from state s to state s' and depends solely on the current state s. Hence the Markovian nature of the model. The MC model has an initial state where the system assumes to be when it is started. N-gram models can be viewed as a Markov-Chain where states correspond to sequences of $N-1$ words and $tr_{n-gram}(\langle w_{n-k}, .., w_{n-1}\rangle, \langle w_{n-k+1}, ..., w_n\rangle)$ is the probability $pr(w_n|w_{n-k}, ..., w_{n-1})$ calculated by the n-gram model. A Markov-Chain model can be easily used to initialize an MDP (as we will show below) and is therefore an appealing approach for an initial predictive model that will provide an informed initialization for the MDP based recommender system.

Chapter 5

An N-Gram Motivated Markov-Chain Prediction Model

5.1 States

The states in our Markov-Chain model represent the relevant information that we have about the user. This information corresponds to previous choices made by users in the form of a set of ordered sequences of selections. Many recommender systems use user attributes such as age or gender in order to generate stronger models, but this approach is problematic since users rarely bother to fill web forms with such data and when forced to, usually supply unreliable information. We have decided to ignore such data, although it could be beneficial but we note that it can be incorporated into our model. Thus, the set of states contains all possible sequences of user selections. This formulation leads to a gigantic state space in our Markov-Chain, an undesirable outcome since our MDP model will have corresponding states with the usual associated problems—data sparsity (e.g. many states will not have sufficient data to compute a good estimation of the transition function) and MDP solution complexity. To reduce the size of the state space, we consider only sequences of at most k items, for some relatively small value of k. We note that this approach is consistent with the intuition that the near history (e.g. the current user session) often is more relevant than selections made less recently (e.g. in past user sessions). The restriction to k items seems sensible for many commercial domains where distant history has little or no impact on user choices.This is much the same as the notion in tri-gram models that the n'th word depends on the previous two words, rather than on the whole sequence of words, or even on the last sentence. These sequences are represented as vectors of size k. In particular, we use $\langle x_1, \ldots, x_k \rangle$ to denote the state in which the user's last k selected items were $x_1, \ldots, x_k$. Selection sequences with $l < k$ items are transformed into a vector in which x_1 through x_{k-l} have the special value *missing*. The initial state in the Markov chain is the state in which every entry has the value *missing* standing for a user who has made no activity in the site. As mentioned before, some recommender systems have better information over the current user gathered through explicit ratings (grades for items that the user supplies). Note that this framework also accommodates such systems, by either mapping such ratings into the values "preferred" and "not preferred", or enriching the state space with the rating values.

In our experiments we used values of k ranging from 1 to 5, but our description below typically fixes k to 3. We also note that k is equivalent to $n+1$ in the n-gram model definitions.

5.2 The Transition Function.

The transition function for our Markov chain describes the probability that a user whose k recent selections were $x_1, \ldots, x_k$ will select the item x' next. This is denoted

$$tr_{MC}(\langle x_1, \ldots, x_k \rangle, \langle x_2, \ldots, x_k, x' \rangle)$$

Initially, this transition function is unknown to us; and we would like to estimate it based on user data.

The maximum-likelihood estimation seems like a good approach for calculating the transition function as described in the next section. This approach, however, still suffers from the problem of data sparsity (e.g., Sarwar et al., 2000a) and performs poorly in practice. In the next section, we describe several techniques for improving the estimation by using methods drawing from the n-gram field.

5.3 Improvement to the Prediction Model

We experimented with several enhancements to the maximum-likelihood n-gram model using data different from that used in our formal evaluation. The improvements described and used here are those that were found to work well in practice.

5.3.1 Skipping

One enhancement is a form of *skipping* (Chen & Goodman, 1996), and is based on the observation that the occurrence of the sequence x_1, x_2, x_3 lends some likelihood to the sequence x_1, x_3. That is, if a person bought x_1, x_2, x_3, then it is likely that someone will buy x_3 after x_1. The intuition behind this approach is that sometimes, the structure of the web site does not allow the user to reach her (sometimes unknown) goal directly, but rather forces her to execute a sequence of actions before reaching that goal. The particular skipping model that we found to work well is a simple additive model. First, each state transition is initialized to zero. Then, given a user sequence $x_1, x_2, ..., x_n$, we add the fractional count $1/2^{(j-(i+3))}$ to the transition from $\langle x_i, x_{i+1}, x_{i+2} \rangle$ to $\langle x_{i+1}, x_{i+2}, x_j \rangle$, for all $i+3 < j \leq n$. This fractional count corresponds to a diminishing probability of skipping a large number of transactions in the sequence. Finally, the counts are normalized to obtain the transition probabilities:

$$tr_{MC}(s, s') = \frac{count(s, s')}{\sum_{s'} count(s, s')}$$

where $count(s, s')$ is the (fractional) count associated with the transition from s to s'.

This approach is problematic in that it generates many states (sequences) that were not actually observed while looking at the ordered set of transactions, but exist solely as successors of observed states. For instance, it might be that many users bought $\langle x_1, x_2, x_3, x_4 \rangle$, but none ever bought $\langle x_1, x_3, x_4 \rangle$. Even though the latter sequence was never observed in the data set, our skipping approach would still create a state for it. For efficiency, we will not generate a state until it is needed, and assume that states that were not created have a fixed value equal to their reward (as opposed to the values calculated by the policy iteration for the "real" states).

5.3.2 Clustering

A second enhancement is a form of clustering that we have not found in the literature. Motivated by properties of our domain, the approach exploits similarity of sequences. For example, the state $\langle x, y, z \rangle$ and the state $\langle w, y, z \rangle$ are similar because some of the items appearing in the former appear in the latter as well. The essence of our approach is that the likelihood of the transition from s to s' is influenced by occurrences from t to s', where s and t are similar. In particular, we define the similarity of states s_i and s_j to be

$$sim(s_i, s_j) = \sum_{m=1}^{k} \delta(s_i^m, s_j^m) \cdot (m+1)$$

where $\delta(\cdot, \cdot)$ is the Kroneker delta function assigning a value of 1 if $x = y$ and 0 otherwise, and s_i^m is the mth item in state s_i. In addition, we define the *similarity count* from state s to s' to be

$$simcount(s, s') = \sum_{s_i} sim(s, s_i) \cdot tr_{MC}^{base}(s_i, s')$$

where $tr_{MC}^{base}(s_i, s')$ is the original transition function (with or without skipping). The new transition probability from s' to s is then given by

$$tr_{MC}(s, s') = \frac{1}{2} tr_{MC}^{base}(s, s') + \frac{1}{2} \frac{simcount(s, s')}{\sum_{s'} simcount(s, s')} \tag{5.1}$$

This definition of states similarity is rather arbitrary in its definition. We have experimented with many different similarity methods, all originating from vector similarity and emphasizing the higher importance of latter items in the sequence to items observed farther in the past, but failed to come up with a method that generates higher performance.

Figure 5.1 shows the algorithm for building the a Markov-Chain model MC_k using skipping and clustering.

5.3.3 Mixture modeling

A third enhancement is the use of finite mixture modeling.[1] Finite mixture modeling is a method for combining a finite number of statistical components together into a single model

[1] Note that Equation 5.1 is also a simple mixture model.

BuildModel $(T,\ k)$
Input:
T — A set of transactions sequences,
k — the maximal length of a sequence

Let $s_0 = \langle missing, ..., missing \rangle$
Initialize the states $S = \{s_0\}$
foreach $t = \langle t_0, t_1, ..., t_n \rangle$ in T **do**
  for $(i = 0; i < n - 1; i++)$ **do**
    Let $s = \langle t_{i-k+1}, .., t_i \rangle$
    (if $i < k - 1$ then fill the state with $missing$ values)
      for $(j = i + 1; j < n; j++)$ **do**
        Let $s' = \langle t_{j-k+1}, .., t_j \rangle$
        (if $j < k - 1$ then fill the state with $missing$ values)
        Let $tr^{base}(s, s') += \frac{1}{2^{j-i}}$
foreach $s, s' \in S$
  $tr(s, s') = \frac{1}{2} tr^{base}(s, s') + \frac{1}{2} \frac{simcount(s,s')}{\sum_{s'} simcount(s,s')}$

Figure 5.1: Building an approximation for tr.

— a model that supplies output based on the output of the components. Most data sets will not have a uniform distribution of the item sequences which will lead to various levels of accuracy of the transition function in different states. Our mixture model is motivated by the fact that larger values of k lead to states that are more informative whereas smaller values of k lead to states that are observed more frequently and thus states where more statistics were gathered. To balance these conflicting properties we mix k models, where the ith model looks at the last i transactions. Thus, for $k = 3$, we mix three models that predict the next transaction based on the last transaction (bi-gram), the last two transactions (tri-gram), and the last three transactions (quad-gram). Similar methods are used in n-gram models, where—for example—a trigram, a bigram, and a unigram are combined into a single model. N-gram researchers have developed several methods for combining the components including backoff and interpolation using mixture weights. We experimented with backoff but interpolating using mixture weights led to superior results. In general, we can learn the mixture weights from data. As mentioned above, it is reasonable to use a small number of weights sets and learn to classify a state for the set of weights that is most beneficial for it using an hold out set of transactions. Because our primary model is based on the k last items, the generation of the models for smaller values entail little computational overhead. Figure 5.2 shows the algorithm we used to learn the mixture weights.

LearnMixtureWeights (M, T, n)
Input:
M — A set of k model components $\langle M_1, ..., M_k \rangle$
T — A set of transactions sequences,
n — The number of mixture weights sets
Output:
W – Sets of mixture weights $(W_1, ..., W_k)$
$Bucket$ — A mapping of states to mixture weights $Bucket : S- > W$

Initialize weights sets $W_i = \langle w_{i,1}, ..., w_{i,k} \rangle$ randomly such that $\sum_j w_{i,j} = 1$
do
  foreach sequence of user transactions $t \in T$
    for $i = k$ to $|t| - 1$
      Let $Bucket(s) = argmax_j(Predict(M, \langle t_{i-k}, ..., ti - 1 \rangle, W_j))$
    for $i = 1$ to k
      improve W_i using hill climbing on all states where $Bucket(s) = i$
until all W_i can not be improved and states stop moving between buckets

Predict (M, $case$, W)
Input: A set of model components $M = \langle M_1, ..., M_k \rangle$
A sequence of items $case$
A set of mixture weights W
Output: A set of predictions ordered by their likelihood P

for $i = 1$ to k
  Let s be the state in M corresponding to $case$
  Let P_i be the set of predictions generated by s
Let $P = \langle p_1, ..., p_n \rangle$ where $p_i = \sum j P_{j,i} W_j$
sort P by increasing order
return P

Figure 5.2: Learning mixture weights for model components.

Chapter 6

An MDP-Based Recommendation Strategy

Our ultimate goal is to construct a recommender system—a system that chooses a link, product, or other item to recommend to the user at all times. In this section, we describe how such a system can be based on an MDP.

6.1 Defining The MDP

The states of the MDP for our recommender system correspond to the states of the predictive (MC) model. This correspondence may not be optimal, especially in light of our experimental results showing the benefits of skipping and clustering. Nonetheless our approach yields reasonable results and as other, more sophisticated, approaches might prove computationally difficult we adopt this simple definition.

The actions of the MDP correspond to a recommendation of an item. One can consider multiple recommendations but, to keep our model simple and computationally tractable, we will start with a single recommendation and discuss multiple recommendations later. When we recommend an item x', the user has two options:

- accept this recommendation, thus transferring from state $\langle x_1, x_2, x_3 \rangle$ into $\langle x_2, x_3, x' \rangle$
- select a non-recommended item.

Since item recommendations are generated only after the user has picked a new item, we do not model the case where a user has requested a different set of recommendations without moving to another state. The rewards in our MDP encode the utilities of selling items (or showing the web pages) as defined by the site. For example, the reward for state $\langle x_1, x_2, x_3 \rangle$ can be the profit generated for the site from the sale of item x_3—the last item in the transaction sequence.

Notice that the stochastic element in our model is the user actual choice given all possible options. The transition function for the MDP model:

$$tr^1_{MDP}(\langle x_1, x_2, x_3 \rangle, x', \langle x_2, x_3, x'' \rangle)$$

is the probability that the user will select item x'' given that item x' is recommended. We write tr^1_{MDP} to denote that only single item recommendations are used.

Unlike traditional MDP implementations that learn the proper values for the transition function and hence the optimal policy online, our system needs to be fairly accurate when it is first deployed. A for profit e-commerce site is unlikely to use a recommender system that generates irrelevant recommendations for a long period, waiting for it to converge to an optimal policy. We therefore need to initialize the transition function carefully. We can do so based on our predictive model, making the following assumptions:

- A recommendation increases the probability that a user will buy an item. This probability is proportional to the probability that the user will buy this item in the absence of recommendations. This assumption is made by most CF models dealing with e-commerce sites. We denote the proportionality constant by α, where $\alpha > 1$.
- The probability that a user will buy an item that was not recommended is lower than the probability that she will buy it if it was recommended, but still proportional to it. We denote the proportionality constant by β, where $\beta < 1$.
- A third option is that the user will decide not to buy any item, thus remaining in the same state.

To allow for a simpler representation of the equations, for a state $s = \langle x_1, ..., x_k \rangle$ and a recommendation r let us denote by $s \cdot r$ the state $s' = \langle x_2, ..., x_k, r \rangle$.

$$tr^1_{MDP}(s, r, s \cdot r) = \alpha \cdot tr_{MC}(s, s \cdot r)$$

$$tr^1_{MDP}(s, r', s \cdot r) = \beta \cdot tr_{MC}(s, s \cdot r),$$
$$r' \neq r$$
$$tr^1_{MDP}(s, r, s) = 1 - tr^1_{MDP}(s, r, s \cdot r) - \sum_{r' \neq r} tr^1_{MDP}(s, r', s \cdot r)$$

where α and β are constant for all initial states and r such that $tr^1_{MDP}(s, r, s{\cdot}r) + \sum_{r' \neq r} tr^1_{MDP}(s, r', s{\cdot}r) < 1$. Note that $tr^1_{MDP}(s, r', s \cdot r)$ does not depend on r' since our calculations are based on data that was collected without the benefit of recommendations. This representation of the transition function allows us to keep at most two values for every pair of states. We note that many states are not actually initialized since we use a "lazy initialization" technique for states as described above, so we actually maintain only two values for every $\langle state, item \rangle$ pair, with the number of items being much smaller than the number of states.

When moving to multiple recommendations we make the assumption that recommendations are not mutually dependent. Namely we assume:

$$\forall R, R' \; tr_{MDP}(s, \{r\} \cup R, s \cdot r) = tr_{MDP}(s, \{r\} \cup R', s \cdot r)$$

This assumption might prove false. For example consider the case where the system "thinks" that the user is interested in an economic cooking book. It can then recommend a few cooking books where most are very expensive and one is reasonably priced (but in no way cheap).

The reasonably priced book will seem economic compared to the expensive ones, thus making the user more likely to take it.

Nevertheless, we make this assumption so as not to be forced to create a larger action space where actions are ordered combinations of recommendations. Taking the simple approach for representing the transition function we defined above we still keep only two values for every state, item pair:

$$tr_{MDP}(s, r \in R, s \cdot r) = tr^1_{MDP}(s, r, s \cdot r)$$

$$tr_{MDP}(s, r \notin R, s \cdot r) = tr^1_{MDP}(s, r', s \cdot r),$$
$$\forall r' \neq r$$

As before $tr_{MDP}(s, r \in R, s \cdot r)$ does not depend on r, and will not depend on R in the discussion that follows. These values are no more than smart initial values and would most likely require adjustments based on actual user behavior.

We note that the MDP can be initialized using any probabilistic predictive algorithm. In particular, to initialize the transition probability from $\langle x_1, x_2, x_3 \rangle$ to $\langle x_2, x_3, x_4 \rangle$, we simply need the predictive algorithm's probability for x_4, given that a user purchased x_1, x_2, and x_3.

The formalization of the MDP as described above causes the system to optimize the site profits. The model is rewarded for items that were purchased no matter whether the system recommended them or not. This formalization directs the system not to recommend items that are likely to be bought whether recommended or not, but rather to recommend items whose likelihood of being purchased is maximized if they are recommended. This approach might lead to a low acceptance rate of recommendations. Another possible approach is to reward the system only if an item was bought after it was recommended. This will lead to recommendations that are likely to be received enhancing the acceptance rate of recommended items, but site profits might not increase since the system might usually recommend items that would have been bought anyhow.

6.2 Solving The MDP

Given an MDP $\langle S, A, R, tr \rangle$, we need to solve it—that is, determine the best course of action for every possible state — an optimal policy. For the domains we have studied, we have found policy iteration (Howard, 1960)—with a few approximations to be described—to be a tractable solution method. In fact, on tests using real data, we have found that policy iteration terminates after a handful of iterations. Fast convergence occurs because we are iterating from the state that was initialized last to the first and, if there is a non-zero transition probability from s to s', then usually s' was initialized later than s due to the sequential nature of the domain. Also, we have seen that the computation of the optimal policy is not heavily sensitive to variations in k — the number of past transactions we encapsulate in a state. As k increases, so does the number of states, but the number of positive entries in our transition matrix remains similar. Note that, at most, a state can have

as many successors as there are items. When k is small, the number of observed successors for a state is indeed close to the number of items. When k is larger, however, the number of successors decreases considerably.

Although the number of iterations required is small, each iteration requires the computation of the expected rewards for every state, given the current policy. Even though we have reduced the state space by limiting each state to hold the last k transactions, the state space is quite large even for $k = 3$. Thus, the computation of an optimal policy without approximation is extremely time consuming. We reduce run time using the following approximation. As explained above, the vast majority of states in our models do not correspond to sequences that were observed in our training set. This fact holds because most combinations of items are extremely unlikely. For example, it is unlikely to find adjacent purchases of a science-fiction and a gardening book. In our approximation, we do not compute a policy for a state that was not encountered in our training data. We use the immediate reward for such states as an approximation of their long-term expected value for the purpose of computing the value of a state that appeared in our training data. Of course, if a policy is explicitly required for an unencountered state, such as when the model is queried for a prediction given an unobserved sequence, we compute its value in the same manner as states that appeared in our training data using our clustering technique. This approximation, although risky in general MDPs, is motivated by the fact that in our initial model, the probability of making a transition into an unencountered state is very low.

We use some special features of our representation of the MDP to allow for fast computations. As we have shown earlier the computation of policy iteration requires the computation of the value function at each stage:

$$V_{i+1}(s) = Reward(s) + \gamma \max_R \sum_{s' \in S} tr(s, R, s') V_i(s') =$$

$$\begin{gathered} Reward(s) + \\ \gamma \max_R (\sum_{r \in R} tr_{MDP}(s, r \in R, s \cdot r) V_i(s \cdot r) + \\ \sum_{r \notin R} tr_{MDP}(s, r \notin R, s \cdot r) V_i(s \cdot r)) \end{gathered}$$

where $tr(s, r \in R, s \cdot r)$ and $tr(s, r \notin R, s \cdot r)$ follow the definitions above.

The standard computation of the optimal policy requires us to compare every subset of recommendations for each state in order to establish the best action (e.g. the recommendations list). We will show how special attributes of our domain can be used to simplify this process. Let us define $\Delta(s, r)$ — the additional value of recommending r in state s:

$$\Delta(s, r) = (tr(s, r \in R, s \cdot r) - tr(s, r \notin R, s \cdot r)) V(s \cdot r)$$

Now we can define

$$R^{s,k}_{max\Delta} = \{r_1, ..., r_k | \Delta(s, r_1) \geq ... \geq \Delta(s, r_k) \wedge \forall r \neq r_i, \Delta(s, r_k) \geq \Delta(s, r)\}$$

the set of k items that have the maximal $\Delta(s, r)$ values.

Theorem 1 *$R^{s,k}_{max\Delta}$ is the set that maximizes $V_{i+1}(s)$*

$$\begin{gathered} V_{i+1}(s) = Reward(s) + \\ \gamma (\sum_{r \in R^{s,k}_{max\Delta}} tr(s, r \in R, s \cdot r) V_i(s \cdot r) + \\ \sum_{r \notin R^{s,k}_{max\Delta}} tr(s, r \notin R, s \cdot r) V_i(s \cdot r)) \end{gathered}$$

Proof: Let us assume that there exists some other set of k recommendations $R \neq R^{s,k}_{max\Delta}$ that maximizes $V_{i+1}(s)$. For simplicity, we will assume that all Δ values are different. If that is not the case than R should be a set of recommendations not equivalent to $R^{s,k}_{max\Delta}$. Let r be an item in R but not in $R^{s,k}_{max\Delta}$ and r' be an item in $R^{s,k}_{max\Delta}$ but not in R (we can find such r and r' since $R \neq R^{s,k}_{max\Delta}$ are both sets of size k). Let R' be the set we get when we replace r with r' in R. We need only show that $V_{i+1}(s,R) < V_{i+1}(s,R')$.

$$V_{i+1}(s,R') - V_{i+1}(s,R) =$$

$$Reward(s) + \sum_{s'} tr(s,R,s')V_i(s') -$$
$$(Reward(s) + \sum_{s'} tr(s,R',s')V_i(s')) =$$

$$\sum_{r'' \in R} tr(s, r'' \in R, s \cdot r'')V_i(s \cdot r) + \sum_{r'' \notin R} tr(s, r'' \notin R, s \cdot r'')V_i(s \cdot r'') -$$
$$\sum_{r'' \in R'} tr(s, r'' \in R', s \cdot r'')V_i(s \cdot r) - \sum_{r'' \notin R'} tr(s, r'' \notin R', s \cdot r'')V_i(s \cdot r'')$$
$$=$$

$$tr(s, r \in R, s \cdot r)V_i(s \cdot r) - tr(s, r' \notin R, s \cdot r')V_i(s \cdot r') -$$
$$(tr(s, r' \in R', s \cdot r)V_i(s \cdot r) - tr(s, r \notin R', s \cdot r')V_i(s \cdot r)) =$$

$$\Delta(s,r) - \Delta(s,r') > 0$$

■

In order to compute $V_{i+1}(s)$ we therefore need to compute all $\Delta(s,r)$ and find $R^{s,k}_{max\Delta}$, making the computation of $V_{i+1}(s)$ independent of the number of subsets (or even worse — ordered subsets) of k items and even independent of k itself. The complexity of finding the optimal policy when recommending multiple items at each stage under the assumptions mentioned above remains the same as the complexity of computing an optimal policy for single item recommendations.

Using this approach assumes the best recommendation to be that of an item where appearance in the recommendations list enhances the likelihood of it being bought, known as the $lift$ of an item. Another possible approach would be to recommend items that the user is most likely to buy. The first approach maximizes the total site profits (profits generated with and without the recommender system) but the latter maximizes the profits generated by the recommender system. A combination of the two approaches could be to use the lift to order the items that are ϵ likely to be bought.

6.3 Updating The Model Online

The initialization of the model was done using a data set of user behavior in the site without the benefit of recommendations. Even if we had the true probabilities $pr(x_4|x_1,x_2,x_3)$ the MDP model built using these values would still be imperfect, due to the assumption that recommendations modify user behavior. The optimal model could therefore only be built using data of the form $pr(x_4|x_1,x_2,x_3,R)$ which is the probability that the user will take x_4 given her former purchases and the supplied recommendations. This data could only be

gathered by keeping track of the system recommendations and of the user selections given those recommended choices.

Once the Recommender system is deployed with its initial model, we will need to update the model according to the actual observations. One approach would be to use a form of reinforcement learning — methods that improve the model after each recommendation is made. Although this idea is appealing, since such models would need little administration to improve, the implementation requires more calls and computations by the recommender system online, which will lead to slower responses and is therefore undesirable. A simpler approach is to perform off-line updates at fixed time intervals. The site need only keep track of the recommendations and the user selections and, say, once a week use those statistics to build a new model and replace it with the old one.

In order to re-estimate the transition function the following values need recording:

- $count(s = \langle x_1, x_2, x_3 \rangle, s' = \langle x_2, x_3, x_4 \rangle)$ — the amount of times the user selected x_4 after previously selecting x_1, x_2, x_3.
- $count(s = \langle x_1, x_2, x_3 \rangle, x_4, s' = \langle x_2, x_3, x_4 \rangle)$ — the amount of times the user selected x_4 after is was recommended.

Using these two values we can compute a new approximation for the transition function:

$$c_{in}^{new}(s, r, s \cdot r) = c_{in}^{old}(s, r, s \cdot r) + count(s, r, s \cdot r)$$

$$c_{total}^{new}(s, s \cdot r) = c_{total}^{old}(s, r, s \cdot r) + count(s, s \cdot r)$$

$$c_{out}^{new}(s, r, s \cdot r) = c_{out}^{old}(s, r, s \cdot r) + count(s, s \cdot r) - count(s, r, s \cdot r)$$

$$tr(s, r \in R, s \cdot r) = \frac{c_{in}(s, r, s \cdot r)}{c_{total}(s, s \cdot r)}$$

$$tr(s, r \notin R, s \cdot r) = \frac{c_{out}(s, r, s \cdot r)}{c_{total}(s, s \cdot r)}$$

where $c_{in}(s, r, s \cdot r)$ is the number of times the r recommendations was taken, $c_{out}(s, r, s \cdot r)$ is the number of times the user took item r even though it wasn't recommended and $c_{total}(s, s \cdot r)$ is the number of times the user took item r while being in state s regardless of whether it was recommended or not.

Given the new estimation of tr we need to run the policy iteration algorithm again to calculate a new policy. As more such iterations will be performed, the tr values will converge to the transition function and the optimal policy could be calculated. It is quite likely though that before this will happen new items will appear in the site, introducing new states, making the process of improvement infinite.

In order to ensure convergence to an optimal solution, the system requires every path in the model to be passed a large number of times. If the system always returns the best recommendations only, then most counts $count(s, r, s \cdot r)$ would be 0 since most items will not appear in the list of the best recommendations available. Therefore, the system needs to recommend non-optimal items occasionally, in order to get counts for those items too. This problem is widely known in computational learning as the tradeoff between exploration and

exploitation problem. The system needs to decide when to explore unobserved options and when to exploit the data it has gathered in order to get rewards. Thus, it seems appropriate to select some constant ϵ, such that recommendations whose expected value is ϵ-close to optimal will be allowed—for example, by following a Boltzmann distribution:

$$Pr(choose(r_i)) = \frac{\exp \frac{V(s \cdot r_i)}{\tau}}{\sum_{j=1}^{n} exp \frac{V(s \cdot r_j)}{\tau}}$$

with an ϵ cutoff — meaning that only items whose value is within ϵ of the optimal value will be allowed. The exact value of ϵ would be determined by the site operators. The price we pay for this conservative exploration policy is that we are not guaranteed convergence to an optimal policy. We note that commercial sites would probably dislike a system that returns irrelevant recommendations in order to improve itself in the future. It is unlikely that significant exploration would be allowed by site operators in practice.

Since the system recommends more than one item at each state, a reasonable solution is to allow larger values of ϵ for the recommendations near the bottom of the recommendation list. For example, the system could always return the best recommendation first, but show items less likely to be purchased as the second and third items on the list.

Another problem that arises in recommender systems in general is adjusting to changes in the site. In e-commerce sites items get frequently added and removed, and user trends shift making the model inaccurate after a while. Gathering the statistics shown above and improving the model once enough data has been gathered should readjust the transition function to follow the changes in users tastes. Once new items are added, users will start buying them and positive counts for them will appear. The counts for c_{in} will however remain zero until the system starts recommending the new item. It is therefore useful at first to initialize the value of c_{in} for new items not to be zero. Making the value of c_{in} and c_{out} equal at first seems reasonable. Using our *Delta* method for finding the best list of recommendations will make those items less likely to appear. As we observe more evidence for c_{Out}, the probability for buying the item if it wasn't recommended should decrease making the *Delat* value for the item higher, thus making it more likely to be recommended. At this stage the system should add new states for these new items, and the transition function will expand to express the transitions for these new states. The system will not be able at first to recommend those new items, which is known as the "cold start" (Good et al., 1999) problem in recommender systems. Removed items however need to be explicitly removed by the site administrator from the model. Even though a removed item transition probabilities will slowly diminish it will be better to set it to 0 manually to avoid getting recommendations for items that cannot be bought.

The ideas previously presented for learning the n-gram model could also be adjusted for faster learning of the MDP transition probabilities discussed above. It is possible to use skipping and clustering for example in much the same way they were used for the predictive model. It would however be reasonable to use diminishing weights for the generalized data, starting with high weight for the data gathered using skipping and clustering and as more and more evidence of the transitions is gathered for the specific sequence lower the weight to allow higher influence of the "real" transition counts. It is also possible to use the value of

c_{out} to learn c_{in}. We can increase the value of c_{in} every time evidence for c_{out} is observed by a fraction. This fraction can again be slowly diminished as more evidence of c_{in} is observed.

Chapter 7

Testing The Predictive Model

Below, we describe an evaluation of the predictive model. The evaluations were done using data from real user behavior on a web site, and follow the footsteps of the evaluation metrics commonly used in the collaborative filtering literature.

7.1 Data Sets

For our tests, we used real data from the Israeli online bookstore *Mitos* (www.mitos.co.il). We used two data sets, one containing user transactions (purchases) and the other containing user browsing paths obtained from web logs. We filtered out items that were bought/visited less than 100 times and users who bought/browsed no more than one item as is commonly done when evaluating predictive models (e.g. Zimdars et al., 2001). We were left with 116 items and 10820 users in the transactions data set, and 65 items and 6678 users in the browsing data set. Items that were rarely bought can not be reliably predicted, and our MDP model should learn the transition function for those states online as previously discussed. In our browsing data, no cookies were used by the site. If the same user visited the site with new IP address, then we would treat her as a new user. Also, activity on the same IP address was attributed to a new user whenever there were no requests for two hours. These data sets were randomly split into a training set (0.9 of the users) and a test set (0.1 of the users).

We evaluated predictions as follows. For every user sequence $t_1, t_2, .., t_n$ in the test set, we generated the following test cases:

$$\langle t_1 \rangle, \langle t_1, t_2 \rangle, ..., \langle t_{n-k}, t_{n-k+1}, ..., t_{n-1} \rangle$$

closely following tests done by Zimdars et al. (2001). For each case, we then used our various models to determine the probability distribution for t_i given $t_{i-k}, t_{i-k+1}, ..., t_{i-1}$ and ordered the items by this distribution. Finally, we used the t_i actually observed in conjunction with the recommendation list to compute a score for the list.

7.2 Evaluation Metrics

We used two scores: Recommendation Score (RC) (Microsoft, 2002) and Exponential Decay Score (ED) (Breese et al., 1998) with slight modifications to fit into our sequential domain.

7.2.1 Recommendation Score

For this measure of accuracy, a recommendation is deemed successful if the observed item t_i is among the top m recommended items (m is varied in the experiments). The score RC is the percentage of cases in which the prediction is successful. A score of 100 means that the recommendation was successful in all cases. This score is more meaningful for commerce sites that require a short list of recommendations and therefore care little for the ordering of the items in the list.

7.2.2 Exponential Decay Score

This measure of accuracy is based on the position of the observed t_i on the recommendation list, thus evaluating not only the content of the list but rather the order of items in it. The underlying assumption is that users are more likely to select a recommendation near the top of the list. In particular, it is assumed that a user will see the mth recommendation with probability

$$p(m) = 2^{-(m-1)/(\alpha-1)}, (m \geq 1)$$

where α is the half-life parameter—the index of the item in the list we assume having probability of 0.5 being seen. The score is given by

$$100 \cdot \frac{\sum_{c \in C} p(m = pos(t_i|c))}{|C|}$$

where C is the set of all cases, $c = t_{i-k}, t_{i-k+1}, ..., t_{i-1}$ is a case, and $pos(t_i|c)$ is the position of the observed item t_i in the list of recommended items for c. We used $\alpha = 5$ in our experiments in order to be consistent with the experiments of Breese et al. (1998) and Zimdars et al. (2001). The relative performance of the models was not sensitive to α.

7.3 Comparison Models

7.3.1 Commerce Server 2002 Predictor

The main model to which we compared our results is the Predictor tool developed by Microsoft as a part of Microsoft Commerce Server 2002, based on the models of (Heckerman et al., 2000). This tool builds dependency- network models in which the local distributions are probabilistic decision trees. We used these models in both a non-sequential and sequential form. These two approaches are described in Heckerman et al. (2000) and Zimdars et al. (2001), respectively. In the non-sequential approach, for every item, a decision tree is built that predicts whether the item will be selected based on whether the remaining items were

or were not selected. In the sequential approach, for every item, a decision tree is built that predicts whether the item will be selected next, based on the previous k items that were selected. The predictions are normalized to account for the fact that only one item can be predicted next. Zimdars et al. (2001) also uses a "cache" variable, but preliminary experiments showed it to decrease predictive accuracy. Consequently, we did not use the cache variable in our formal evaluation.

These algorithms appear to be the most competitive among published work. The combined results of Breese et al. (1998) and Heckerman et al. (2000) show that (non-sequential) dependency networks are no less accurate than Bayesian-network or clustering models, and about as accurate as *Correlation*, the most accurate (but computationally expensive) memory-based method. Sarwar, Karypis, Konstan, and Riedl (2000b) apply dimensionality reduction techniques to the user rating matrix, but their approach fails to be consistently more accurate than Correlation. Only the sequential algorithm of Zimdars et al. (2001) is more accurate than the non-sequential dependency network to our knowledge.

We built five sequential models $1 \leq k \leq 5$ for each of the data sets. We refer to the non-sequential Predictor models as Predictor-NS, and to the Predictor models built using the data expansion methods with a history of length k as Predictor-k.

7.3.2 Unordered MCs.

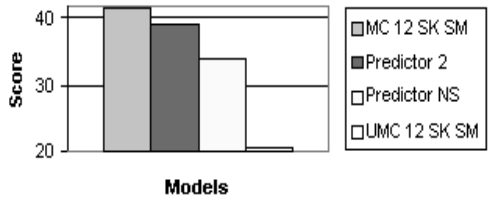
We also evaluated a non-sequential version of our predictive model, where (e.g.) the sequences $\langle x, y, z \rangle$ and $\langle y, z, x \rangle$ are mapped to the same state. If our assumption over the sequential nature of recommendations is void we should expect this model to perform better than our MC model. Skipping, clustering, and mixture modeling were included as described in section 2. We call this model UMC (Unordered Markov chain).

7.4 Variations Of The MC Model

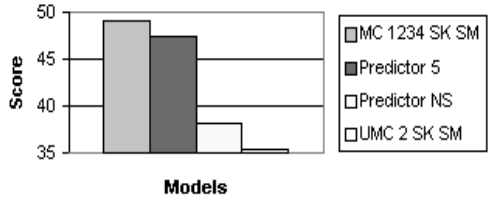
In order to measure how each n-gram enhancement influenced predictive accuracy, we also evaluated models that excluded some of the enhancements. In reporting our results, we refer to a model that uses skipping and similarity clustering with the terms SK and SM, respectively. In addition, we use numbers to denote which mixture components are used. Thus, for example, we use MC 123 SK to denote a Markov-chain model learned with three mixture components—a bigram, trigram, and quadgram—where each component employs skipping but not clustering.

7.5 Experimental Results

Figure 7.1(a) and figure 7.1(b) show the exponential decay score for the best models of each type (Markov chain, Unordered Markov chain, Non-Sequential Predictor model, and Sequential Predictor Model). It is important to note that *all* the MC models using skipping, clustering, and mixture modeling yielded better results than *every one of* the Predictor-k models and the non-sequential Predictor model. Thus, to simplify the graphs, only the best



(a) Transactions data set.

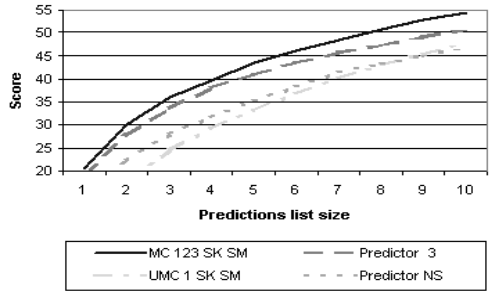


(b) Browsing data set.

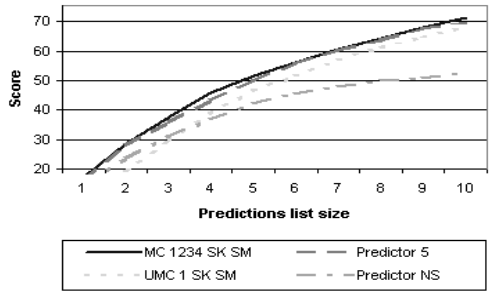
Figure 7.1: Exponential decay score for different models.

models of each class are presented. We see that the sequence-sensitive models are better predictors than those that ignore sequence information. Furthermore, the Markov chain predicts best for both data sets.

Figure 7.2(a) and Figure 7.2(b) show the recommend score as a function of list length (m). Once again, sequential models are superior to non-sequential models, and the Markov-chain models are superior to the Predictor models.



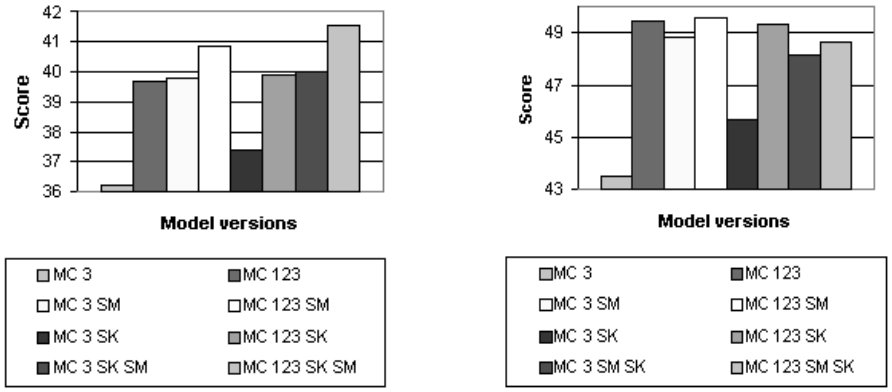
(a) Transactions data set.



(b) Browsing data set.

Figure 7.2: Recommend score for different models.

Figure 7.3(a) and Figure 7.3(b) show how different versions of the Markov chain performed under the exponential decay score in both data sets. We see that multi-component models out-perform single-component models, and that similarity clustering is beneficial. In contrast, we find that skipping is only beneficial for the transactions data set. Perhaps users tend to follow the same paths in a rather conservative manner, or site structure does not allow users to "jump ahead". In either case, once recommendations are available in the site (thus changing the site structure), skipping may prove beneficial.



(a) Transactions data set.

(b) Browsing data set.

Figure 7.3: Exponential decay score for different Markov chain versions.

Chapter 8

Performance Issues

Most modern model-based algorithms implement some type of optimization mechanism making the worst case analysis uninteresting. The Microsoft Commerce Server Predictor, for example, allows you to build models based on a sample of the input data and limit the number of items used during the build process. We allowed similar options such as building a model for the top-n observed items and limiting the number of initialized states on each mixture model and finally — optimizing the model for online recommendations, removing all the irrelevant data for that task.

It is however interesting to examine the amount of time required to build a model and the memory needed to hold such a model, as well as the number of predictions generated per second, as these parameters effect the performance of the recommender system in real e-commerce sites. We ran experiments on models differing by the number of items in the test data set thus generating different model sizes, build time and latency for predictions.

We compared our model to five non sequential Predictor models. The Predictor enables you to limit the number of items for which decision trees are built. Using this feature, the number of items used in the models was set; we did not change the data set, just modified the "Input attribute fraction" and "Output attribute fraction" parameters to achieve the proper number of elements. Setting the "Input attribute fraction" parameter to 0.1, notifies the model build process to use only the top 10 percent observed items for building the model. The "Output attribute fraction" parameter sets the number of decision trees to be built. Similar methods are used by our MC models to filter a portion of the items used when the model is built. Those models do not have the exact number of items as the MC models since the Predictor method for selecting the number of items is different from our method, making it difficult to adjust the exact number. We allowed a maximal difference of 0.1 between the two models always in favor of the Predictor. We used the non-sequential data since models built from sequential data are slower to build, require more memory and supply less predictions per second.

All tests were done on a 1Ghz Pentium III processor with 512MB memory, running Windows Professional 2000 with Service Pack 3.

Table 8.1 shows the time needed to build a new model based on the transactions data set used in our experiments for models using all improvements discussed above and different numbers of mixture components. We note that build time is the least important parameter

number of items	15231	2661	1142	354	86
MC-1-SK-SM	26	24	19	19	7
MC-12-SK-SM	79	50	41	31	16
MC-123-SK-SM	112	63	58	41	16
MC-1234-SK-SM	119	69	62	49	18
MC-12345-SK-SM	122	78	64	50	20
Predictor-NS	3504	631	177	80	25

Table 8.1: Required time (seconds) for model building.

when selecting a recommender system, as model building is an offline task executed at long time intervals (say once a week at most) on a machine that does not affect the performance of the site. That being said, we note that our models are built faster than the corresponding Predictor models.

number of items	15231	2661	1142	354	86
MC-1-SK-SM	77.5	30.9	19.8	11.2	7.6
MC-12-SK-SM	125	63.3	47	29.8	9.8
MC-123-SK-SM	138	74	55.7	33.3	11.4
MC-1234-SK-SM	138.3	74.8	58.7	35.2	11.9
MC-12345-SK-SM	144.3	79.2	58.9	35.6	12.2
Predictor-NS	50.1	26	25	22.3	18

Table 8.2: Required memory (MB) for building a model and generating recommendations.

Table 8.2 shows the amount of memory needed to build and store a model in MB. This parameter is more important since using a large amount of memory causes the system to slow down considerably. This is less important while building the model but is crucial on the web server holding the model for predictions where free space is usually used for caching ASP pages making the site responses quicker. Our models perform worst from this aspect. This is because Predictor models have a decision tree built for each item, while we have a state for sequences of items. Even though we used several optimization techniques mentioned above (e.g. not holding all states explicitly) the number of states far exceeds the number of decision trees. Many papers were written discussing smart representations of the state and action space and the transition and value functions that could be leveraged for decreasing the needed system space (e.g. Koller & Parr, 1999).

Table 8.3 shows the number of recommendations generated per second by the recommender system. This measure is crucial for online sites as it encapsulates the latency caused by waiting for the system to add recommendations to the page. If this latency is noticeable, no reasonable site administrator will use the recommender system. Modern sites are visited by large numbers of users every day and generating a recommendation list for every user multiple times (say, every time she enters the shopping cart page or viewing an item description page) can cause the system to slow down. The Commerce Server Predictor sets

number of items	15231	2661	1142	354	86
MC-1-SK-SM	34483	26316	28571	30303	30303
MC-12-SK-SM	434	454	588	769	1923
MC-123-SK-SM	250	277	322	384	1030
MC-1234-SK-SM	208	250	243	357	813
MC-12345-SK-SM	206	227	263	312	769
Predictor-NS	23	74	175	322	1000

Table 8.3: Recommendations per second.

the boundary at a minimum of 30 recommendations per second and an average of 50 recommendations but our experiments show them to get far superior results. Still, our models perform better than the equivalent Predictor ones on most cases since we do almost no computations online; while predicting, the model simply finds the proper state and returns the state's precalculated list of recommendations. We view this as one of the advantages of the MC model over other model-based approaches. We note, though, that in order to decrease the memory requirements we could be forced to move to a different representation of the policy that might lead to some computational cost and hence, a decreased performance.

Overall we conclude that our approach is competitive with commercial implementations of recommender systems — it builds faster, supplies fast recommendations and can be adapted to require a reasonable amount of memory, making it usable in real life e-commerce sites. We also note that even though our experiments were conducted using the MC model, our MDP model does not require more memory, a longer model building time or causes growth in latency of predictions.

Chapter 9

Implementing the System in Mitos

The system is currently incorporated into the online book store Mitos. Users can receive recommendations in two different locations. Users looking at the description of a book are presented with a list of recommendations based solely on this book. Users adding items to the shopping cart receive recommendations based on the last k items added to the cart ordered by the time they were added. Every time a user is presented with a list of recommendations on either page, the system stores the recommendations that were presented and whether the user took a recommended item. Once a week, an automated process is run to update the model given the data that was collected during the week as described above.

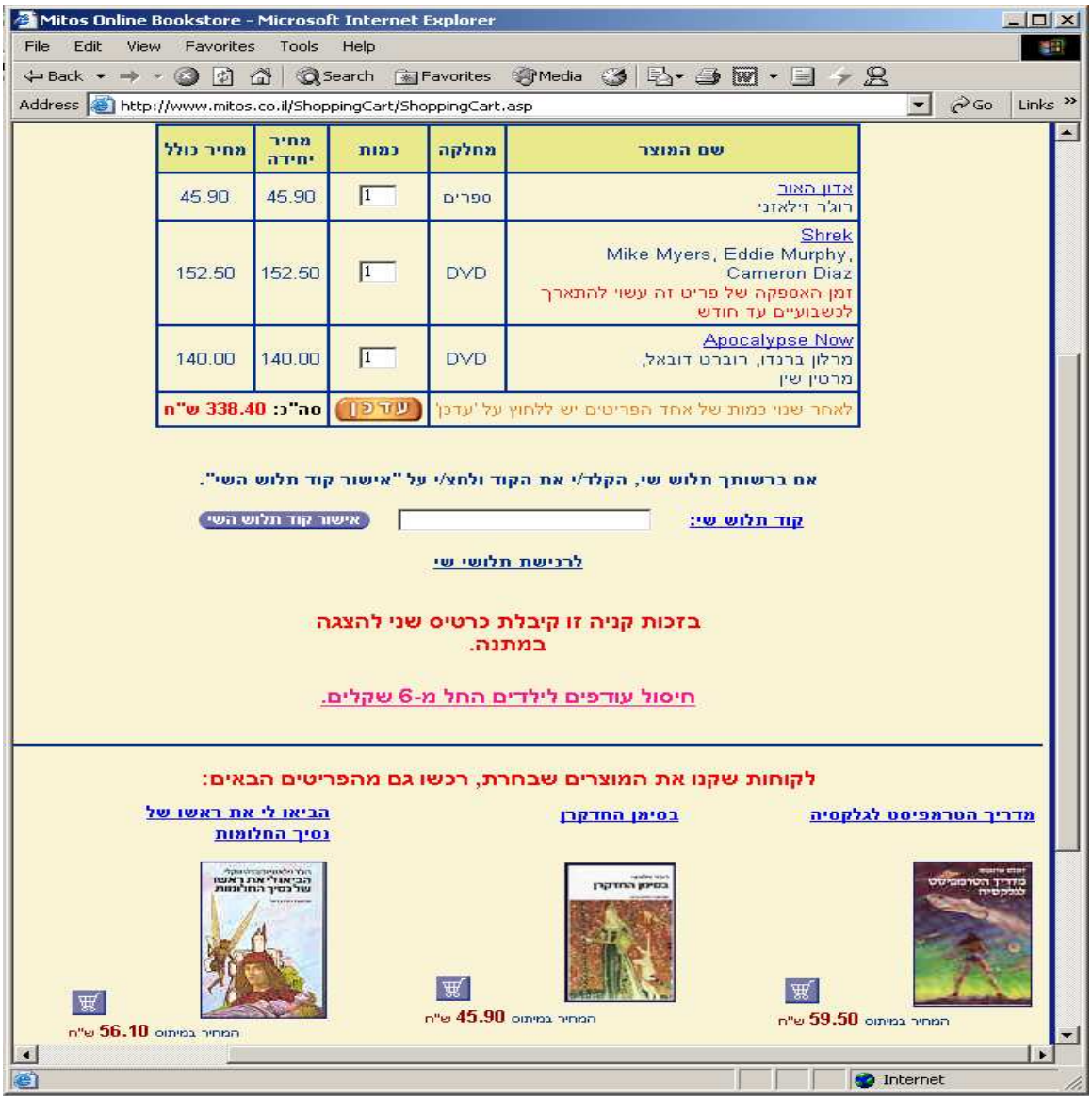


Figure 9.1: Recommendations in the shopping cart web page.

Figure 9.2: Recommendations in the book focus web page.

Chapter 10

Conclusions and Future Work

This thesis describes a new model for recommender systems based on an MDP. Our approach makes two main contributions: a conceptual contribution that stems from this new view of Recommender systems, and a technical contribution in the form of a new n-gram based Markov-chain predictive model.

Our work also suggests a novel approach to learning MDPs. It takes into account the fact that a deployed MDP-based recommender system cannot count on standard reinforcement learning techniques for its initialization, as these would lead to initial behavior that would not be tolerated by a site owner. Rather, we use the behavior of customers in the site prior to the deployment of the recommender system to train a Markov chain that can then be used to initialize the MDP in an informed manner.

Our particular approach to the generation of the n-gram models is motivated by our domain of interest, and our clustering approach seems much more powerful for this domain than standard n-gram clustering techniques.

Weaknesses of our predictive (Markov chain) model include the use of *ad hoc* weighting functions for skipping and similarity functions. Although the recommendations that result from our current model are (empirically) useful for ranking items, we have noticed that the probability distributions they produce are not calibrated. Learning the weighting functions from data should improve calibration.

Our predictive model should also make use of relations between items that can be explicitly specified. For example, most sites that sell items have a large catalog with hierarchical structure such as categories or subjects, a carefully constructed web structure, and item properties such as author name. Finally, our models should incorporate information about users such as age and gender.

Bibliography

Balabanovic, M., & Shoham, Y. (1997). Combining content-based and collaborative recommendation. *Communications of the ACM*, *40*(3).

Bellman, R. E. (1962). *Dynamic Programming*. Princeton University Press.

Bohnenberger, T., & Jameson, A. (2001). When policies are better than plans: Decision-theoretic planning of recommendation sequences. In Lester, J. (Ed.), *IUI 2001: International Conference on Intelligent User Interfaces*, pp. 21–24. ACM, New York. Available from http://dfki.de/~jameson/abs/BohnenbergerJ01.html.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th conference on Uncertainity in Artificial Inteligence*, pp. 43–52.

Cassandra, A. R., Kaelbling, L. P., & Littman, M. L. (1994). Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Vol. 2, pp. 1023–1028 Seattle, Washington, USA. AAAI Press/MIT Press.

Chen, S. F., & Goodman, J. (1996). An empirical study of smoothing techniques for language modeling. In Joshi, A., & Palmer, M. (Eds.), *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pp. 310–318 San Francisco. Morgan Kaufmann Publishers.

Chen, Y. H., & George, E. I. (1999). A bayesian model for collaborative filtering. *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*.

Claypool, M., Le, P., Wased, M., & Brown, D. (2001). Implicit interest indicators. In *Intelligent User Interfaces*, pp. 33–40.

Good, N., Schafer, J. B., Konstan, J. A., Borchers, A., Sarwar, B. M., Herlocker, J. L., & Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. In *AAAI/IAAI*, pp. 439–446.

Heckerman, D., Chickering, D. M., Meek, C., Rounthwaite, R., & Kadie, C. M. (2000). Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, *1*, 49–75.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes.* MIT Press.

Jelinek, F., & Mercer, R. L. (1980). Interpolated estimation of markov source parameters from sparse data. *Proceedings of the Workshop on Pattern Recognition in Practice*, 381–397.

Karypis, G. (2001). Evaluation of item-based top-n recommendation algorithms. In *CIKM*, pp. 247–254.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 400–401.

Kneser, R., & Ney, H. (1995). Improved backing-off for n-gram language modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, Volume 1, 181–184.

Koller, D., & Parr, R. (1999). Computing factored value functions for policies in structured MDPs. In *IJCAI*, pp. 1332–1339.

McCallum, A. K. (1996). Reinforcement learning with selective perception and hidden state. *PhD Thesis, Department of Computer Science, University of Rochester.*

Microsoft (2002). Recommendation score. *Microsoft Commerce Server 2002 Documentation.*

Mooney, R. J., & Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of DL-00, 5th ACM Conference on Digital Libraries*, pp. 195–204 San Antonio, US. ACM Press, New York, US.

Poupart, P., Boutilier, C., Schuurmans, D., & Patrascu, R. (2002). Piecewise linear value function approximation for factored MDPs. *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI02).*

Puterman, M. (1994). *Markov Decision Processes.* Wiley, New York.

Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., & Riedl, J. (1994). GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pp. 175–186 Chapel Hill, North Carolina. ACM.

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Special issue of Communications of the ACM*, 56–58.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000a). Application of dimensionality reduction in recommender systems–a case study. In *ACM WebKDD Workshop.*

Sarwar, B. M., Karypis, G., Konstan, J. A., & Riedl, J. (2000b). Analysis of recommendation algorithms for e-commerce. In *ACM Conference on Electronic Commerce*, pp. 158–167.

Schafer, J. B., Konstan, J. A., & Riedi, J. (1999). Recommender systems in e-commerce. In *ACM Conference on Electronic Commerce*, pp. 158–166.

Su, Z., Yang, Q., & Zhang, H. J. (2000). A prediction system for multimedia pre-fetching in internet. *ACM Multimedia.*

Ungar, L., & Foster, D. (1998). Clustering methods for collaborative filtering. In *Proceedings of the Workshop on Recommendation Systems, AAAI Press, Menlo Park California.*

Zimdars, A., Chickering, D. M., & Meek, C. (2001). Using temporal data for making recommendations. In *Proceedings of Seventeenth Conference on Uncertainty in Artificial Intelligence,* Seattle, WA, pp. 580–588.