An MDP-based Coupon Issuing System

Author: Travis Russell
Supervisor: Dr. Kumar Murty

A thesis submitted in conformity with the requirements for the degree of Master of Science

Graduate Department of Mathematics

University of Toronto

© Copyright by Travis Russell 2015
Abstract

We present a system based on the work of Shani et al. [“An MDP-based recommender system,” Journal of Machine Learning Research, vol. 6, pp. 1265–1295, 2005], who showed that the recommendation process could be modeled as a sequential decision process and that a Markov decision process (MDP) provided an adequate representation of the process. The major addition to our system is that of coupons. Given a set of coupons, and an integer \(n > 0\), our system will issue users a coupon every \(n\) purchases. This system determines the optimal coupon to issue each user by analyzing their purchase history and the potential profit made from issuing the coupon. We also present an additional method for determining transition probabilities, a method for updating the system in real-time, and a method for solving our system that is potentially more computationally efficient.
## Contents

1 Introduction 1

2 The Predictive Model 3
   2.1 Constructing the Predictive Model ........................................ 3
   2.2 Evaluating the Predictive Model ........................................... 8
      2.2.1 Decision Support Metrics ............................................. 8
      2.2.2 Recommendation Score ............................................... 9
      2.2.3 Exponential Decay Score ............................................. 9
      2.2.4 Results .................................................................. 10

3 The Coupon Issuing System 10
   3.1 The Markov Decision Process ................................................. 10
   3.2 Solving the Markov Decision Process ....................................... 14
      3.2.1 Policy Iteration ......................................................... 15
      3.2.2 Modified Policy Iteration ............................................. 16
      3.2.3 Approximate Modified Policy Iteration ......................... 17
   3.3 Updating the Model ........................................................... 20
   3.4 Evaluation of the Coupon Issuing System .................................. 22

4 Conclusion and Future Work 23

5 Acknowledgments 24

References 24
1 Introduction

Today, consumers are faced with a variety of products that they can choose from, and an enormous amount of information about them. In the past, people generally relied on the recommendations of friends, family, critics, and other resources to make these choices. While these resources are still important, many websites have implemented recommender systems (see Resnick and Varian [1]) to sift through this large amount of data and help consumers make their choices.

Many different methods and approaches have been taken for supplying meaningful recommendations to consumers. Methods that stem from the study of information retrieval are known as content based methods (see Mooney and Roy [2]). These methods rely on the characteristics of items in the system in order to find similar items to recommend to users.

Another method is known as collaborative filtering (see Su and Khoshgoftaar [3]), which focuses on relationships between users rather the similarity between items. Instead of using information regarding the content of items, these methods use historical data gathered about users. The main assumption of collaborative filtering is that, given two consumers \( A \) and \( B \) who agree about a certain issue, \( A \) is more likely to agree with \( B \) on a separate issue than with another random consumer.

As stated above, people generally rely on recommendations from people they know or trust when choosing products. Collaborative filtering exploits the fact that the internet has billions of users who can recommend products to each other. The main goal of collaborative filtering is to find out which of these users are similar so that they can predict the tastes of a certain user, and use these predictions to give recommendations.

In order to do this, users must interact with the website or system in some way in order to give data. Some websites give explicit rating scales, e.g., 1-5 stars. Others use binary rating systems, e.g., like/dislike. Many websites use implicit ratings rather than explicit ratings, the most common approach being that a user “likes” an item if they purchase it.

There are two main approaches to collaborative filtering. Memory based collaborative filtering works directly with stored user data (like the ratings described above) to identify similar users and recommend items to users based on this data. Model based collaborative filtering uses this data to create a predictive model using structures such as Bayesian networks or Markov chains. A survey describing both methods
can be found in [3]. In this paper, we will be focusing on model based collaborative filtering.

One major characteristic of the recommendation process is that it is sequential in nature. Once a user makes a purchase or rates an item, a new list of recommendations is generated. Moreover, purchases themselves are sequential in nature - if a user buys a book and enjoys it, they are more likely to buy the sequel to that book or another book by the same author.

Zimdars et al. [4] used the sequential nature of the recommendation process to implement an auto-regressive model (a $k$-order Markov chain) to represent it. Given a sequence of user transactions, this chain would predict what the next transaction would be.

While Zimdars et al. and many others approached this problem as a sequential prediction problem, Shani et al. [5] approached the recommendation process as a sequential decision problem. Instead of implementing a system that would predict what item a user would buy at each point, they implemented a system that would make a decision: which recommendation to issue the user. While this distinction might seem small, it leads to a much more sophisticated approach to recommender systems.

The decision made by the recommender system should take into account the exact sequential process involved and an optimization criteria suitable for the recommendation system, such as profit or user satisfaction. To do this, Shani et al. used the Markov decision process (MDP) model (see Puterman [6]), a well known stochastic model of sequential decisions.

Not only do MDPs model the sequential recommendation process well, but they also take into account the utility of a recommendation. For example, the MDP may choose to recommend an item that has a slightly lower probability of being bought but has a massive profit advantage when compared to other items. The MDP can also take into account future profits and purchases. For example, the MDP may choose to recommend a camera that gives little profit but has very profitable accessories. Similarly, the MDP may choose to recommend a movie with a sequel as the user will likely buy the sequel in the future. All of these characteristics make the MDP model a good choice for the recommendation process.

In collaboration with the Israeli bookstore Mitos, Shani et al. implemented their system and found that their system increased total profits by 16%, a 27% increase over a different recommender system [5]. This showed that the recommendation pro-
cess could indeed be modeled as a sequential decision problem, and that MDPs were in fact an adequate model for this view.

In this paper, we present a recommender system based on the methods used by Shani et al., but with some differences, the most major being the addition of coupons. With over 300 billion coupons being distributed last year in the United States and with over 2.84 million redeemed [7], coupons are a very important way for companies to entice consumers. With the explosion in popularity of mobile phones, consumers now have digital coupons at the touch of their finger tips wherever they travel. Juniper Research projects there to be over 1.05 billion users of mobile coupons by the year 2019 (see Holden [8]). With these growing numbers, coupons will be more important than ever for business. Given a set of coupons, and an integer \( n > 0 \), our MDP coupon issuing system will issue a coupon to a user after every \( n \)-th purchase they make. The coupon issued is based on what the user is likely to buy next, as well as a number of other factors. Our system will also act as a traditional recommender system between the issuing of coupons.

In addition to this change, we will also give another method of computing the probabilities for the predictive model presented by Shani et al. that uses a data mining technique known as sequential rule mining. We will also present an alternative algorithm for solving our MDP that has the potential to be more efficient than the standard algorithm presented in [5], as well as a different method of real-time learning so that the MDP model becomes updated after it is initialized.

\section{The Predictive Model}

\subsection{Constructing the Predictive Model}

Before we construct our coupon issuing system, we must first construct a predictive model that predicts which item a user will purchase next. This predictive model will model user behaviour without the presence of coupons. We will then extend this model to model user behaviour with coupons in the system. Because Markov decision processes are natural extensions of Markov chains, we use a Markov chain as our model. This model is based on already existing user data.

\textbf{Definition 1.} A Markov chain is defined as a sequence of random variables \( X_1, X_2, \ldots \) that, given the present state of the variables, the future and past states are independent. Formally, we say \( Pr(X_{n+1} = x | X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = Pr(X_{n+1} = x | x_n) \).
$x | X_n = x_n$) if both conditional probabilities are well defined.

A natural way to think of a Markov chain is as a directed graph where the vertices of the graph represent our variables (or states) and the edges of the graph are labeled with the probability of moving from one state at time $t$ to the adjacent state at time $t + 1$.

![Diagram of a two state Markov Chain](image)

Figure 1: An example of a two state Markov Chain.

To complete our predictive model we must define our state space as well as our transition function that will give us the probabilities of transitioning between states. These elements are taken from Shani et al. [5]:

- **States:** We model our state space using $N$-gram models. These models originated in the field of language modeling. Given the first $n - 1$ words in a sentence, $N$-gram models predict the $n$-th word of the sentence. These models are used extensively in the implementation of search engines. Instead of words, we deal with items. Specifically, our state space consists of all possible user purchase histories. To reduce the state space, we only consider sequences of $k$ items. We represent these states as vectors of size $k$. In particular, we use $< x_1, x_2, \ldots, x_k >$ to represent the state of a user whose last $k$ purchases were $x_1, \ldots, x_k$, with $x_k$ being the most recent purchase. For states representing users that have bought $l < k$ items, the values $x_1, \ldots, x_{k-l}$ will be missing, with the initial state being the state with all values missing. In this paper we set $k = 3$ in most cases. This will be discussed in section 2.1. We note that, because we
encapsulate the past $k$ user purchases into a user state, that we do satisfy the Markov principle.

- **The Transition Function:** The transition function for our Markov Chain describes the probability that a user whose past $k$ items bought were $x_1, \ldots, x_k$ will purchase item $x'$ next. We denote this by $P(< x_1, \ldots, x_k >, < x_2, \ldots, x_k, x' >)$. We must use user data to determine our function. The most naive approach to this is to use a maximum-likelihood estimation, i.e.,

$$P_{ML}(< x_1, x_2, x_3 >, < x_2, x_3, x_4 >) = \frac{\text{count}(< x_1, x_2, x_3, x_4 >)}{\text{count}(< x_1, x_2, x_3 >)}$$

where $\text{count}(< x_1, \ldots, x_k >)$ is the number of times the sequence $< x_1, \ldots, x_k >$ appears in the user data. Because the data that will be processed is often sparse, this approach will often give unreliable results. Shani et al. improved the transition function by incorporating traditional $N$-gram modeling techniques such as skipping, clustering via similarity, and smoothing [5]. Below we outline these improvements as well as introduce several new ones:

1. **Skipping:** Skipping (see Chen and Goodman [9]) refers to the observation that the occurrence of the sequence $x_1, x_2, x_3$ in the user data lends some likelihood to the appearance of the sequence $x_1, x_3$, i.e., if a user’s purchase history is $x_1, x_2, x_3$, it is likely somebody else will buy $x_3$ after $x_1$. The skipping model implemented by Shani et al. is known as the simple additive model. The model is initialized by setting the count for each state transition by counting the number of observed transitions in the user data. Then, given a sequence $x_1, \ldots, x_k$, the fractional count $\frac{1}{2;j-(i+3)}$ is added to the transition count corresponding to the transition between states $< x_i, x_{i+1}, x_{i+2} >$ and $< x_{i+1}, x_{i+2}, x_j >$ for all $i + 3 < j \leq k$. The more items that are skipped, the weaker the fractional count is. These new fractional counts are then normalized to get a probability:

$$P_{\text{skip}}(s, s') = \frac{\text{count}(s, s')}{\sum_{s''} \text{count}(s, s'')}.$$

2. **Similarity:** Similarity [5] refers to the fact that some states in our state space are similar. For example, the state $< x, y, z >$ is similar to the state $< w, y, z >$, as 2 of the items appearing in the former state appear in the
latter state. This approach uses the fact that the likelihood of a transition between states $s$ and $s'$ can be predicted by the likelihood of a transition between states $r$ and $s'$ where $s$ and $r$ are similar. Shani et al. defined a similarity function to determine how similar two states are by setting:

$$sim(s, s') = \sum_{m=1}^{k} \delta(s_m, s'_m)(m + 1)$$

where $s_m$ represents the $m$-th item in state $s$, and $\delta(s_m, s'_m) = 1$ if the two inputted items are the same and equals 0 otherwise. We note that the more recently purchased items are weighed heavier when computing similarity. A similarity count is then defined:

$$\text{simcount}(s, s') = \sum_{s''} sim(s, s'')P_{ML}(s'', s')$$

We then normalize this to get our probabilities:

$$P_{\text{sim}}(s, s') = \frac{\text{simcount}(s, s')}{\sum_{s''} \text{simcount}(s, s'')}.$$ 

3. **Sequential Pattern Mining:** The objective of sequential pattern mining is, given a set of sequences or transactions, to find all transactions that have a specified minimum support in the system (see Liu [10]). Each of these transactions is known as a sequential pattern. We define the support of a sequence by the fraction of total transactions in the data set that contain the sequence in question. By setting support low enough, we will be able to find rules for all observed sequences. In classical sequential pattern mining no rules are generated, so we define sequential rules that are analogous to association rules:

**Definition 2.** A **sequential rule** is an implication $X \implies Y$ where $X$ and $Y$ are sequences and $X$ is a proper subsequence of $Y$.

In our case, a sequential rule implies that a user whose purchase history contains the sequence $X$ is likely to buy the items in $Y$ not in $X$ (in the order given in $Y$). Our sequential patterns can easily be mined by implementing a mining algorithm such as the well known Apriori algorithm. We can then set a minimum confidence level and generate our
rules. Our confidence for a rule \( X \implies Y \) is defined as the fraction of total transactions in the data set that contain \( X \) and also contain \( Y \) as subsequences. Because these sequential rules are too restrictive for our use, we introduce wildcards that correspond to any sequence of items. These wildcards only appear in the first sequence in the implication. We denote these wildcards by \( \cdot \). After generating our rules, if we wish to compute the transition probabilities for the state \( < x_1, x_2, x_3 > \), we look for all rules of the form \( \cdot, x_1, \cdot, x_2, \cdot, x_3, \cdot \implies Y \), e.g., \( \cdot, x_1, \cdot, x_2, \cdot, x_3, \cdot \implies (x_2), x_1, (x_4, x_5), x_2, (x_3), x_3, (x_1, x_5) \). This rule predicts the item \( x_1 \) to be the next item bought, improving the transition probability between \( < x_1, x_2, x_3 > \), and \( < x_2, x_3, x_1 > \). We then weigh the transition probability for each predicted state using the confidence corresponding to the sequential rule giving the prediction. We then normalize these probabilities to get \( P_{SRM}(s, s') \).

4. **Smoothing:** Smoothing [5] is a general name for a method of improving probability estimates to achieve higher accuracy by adjusting zero or low probabilities upwards. The approach Shani et al. took to this problem is known as finite mixture modeling. In our state space, choosing different values for \( k \) gives different advantages. Higher values of \( k \) are more informative while lower values give easier statistics to work with. To balance this discrepancy, we calculate our probabilities using states containing 1, \ldots, \( k \) items. For example, for \( k = 3 \), we calculate our probabilities using states containing 1 item, for states containing 2 items, and states containing 3 items. We then weigh and sum these probabilities. We note that computing probabilities for states with a small number of items has little computational overhead.

Our final transition function is given by weighing \( P_{skip} \), \( P_{sim} \), \( P_{SRM} \), and applying finite mixture modeling. We have:

\[
P^m(s, s') = \lambda_1 P_{skip}(s, s') + \lambda_2 P_{sim}(s, s') + \lambda_3 P_{SRM}(s, s')
\]

where \( m \) is the number of items in a state and \( \lambda_1 + \lambda_2 + \lambda_3 = 1 \). The weights correspond to how sparse the data is; if the data is especially sparse, \( P_{sim} \) should be weighed especially high. If the data is less sparse, \( P_{SRM} \) should be weighed especially high as more rules will be generated. Setting \( k = 3 \), we now apply
finite mixture modeling to get our final transition function:

\[ P(s, s') = \frac{1}{3} P^1(s, s') + \frac{1}{3} P^2(s, s') + \frac{1}{3} P^3(s, s'). \]

We note that each \( P^i(s, s') \) is weighed the same for simplicity.

With our state space and transition function defined, we have now completely described our predictive model.

### 2.2 Evaluating the Predictive Model

The predictive model should be evaluated before being extended to the full coupon issuing system. Among the variants that should be tested are the different weightings for the transition function and the value for \( k \). We give a brief survey of evaluation methods for our predictive model:

#### 2.2.1 Decision Support Metrics

Decision support metrics are derived from statistical decision theory (see Salton [11]). Whenever a user buys an item, the predictive model predicts what item that user will purchase next. For some arbitrary time period, we count the number of true positives given by the model, as well as the false positives, true negatives, and false negatives. We can represent these numbers in a retrieval confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>Relevant</th>
<th>Irrelevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>Not retrieved</td>
<td>FN</td>
<td>TN</td>
</tr>
</tbody>
</table>

Figure 2: A retrieval confusion matrix.

The core metrics in the evaluation are precision (\( P \)) and recall (\( R \)):

\[ P = \frac{TP}{TP + FP} \quad R = \frac{TP}{TP + FN}. \]

These two metrics are often combined into a single metric known as the \( F_1 \) metric:

\[ F_1 = \frac{2PR}{P + R}. \]
We note that the $F_1$ metric is the harmonic mean of precision and recall. As the harmonic mean is the smallest of the three Pythagorean means (the others being the arithmetic mean $\frac{P+R}{2}$ and the geometric mean $\sqrt{PR}$), it tends to mitigate the impact of outliers, or in our case, large differences between our precision and recall values. We also note that precision and recall share a common numerator but have different denominators, so to average them it makes sense to average their reciprocals, thus we use the harmonic mean.

### 2.2.2 Recommendation Score

To calculate a recommendation score [12], some recommender system must be implemented for comparison purposes. For this metric, a prediction is deemed successful relative to the parameter $m$ if the predicted item appears in the top $m$ items recommended by the recommender system. The recommendation score of the model is simply the percentage of times that a prediction made is successful.

### 2.2.3 Exponential Decay Score

Similar to the recommendation score, a recommender system is needed for this evaluation. The exponential decay score (see Breese et al. [13]) is based on the assumption that a user is more likely to accept a recommendation that is at the top of a list of recommendations. In particular, it is assumed that a user will see the $m$-th item on a recommendation list with probability

$$p(m) = 2^{-\frac{(m-1)}{(m-1)}}$$

for $m \geq 1$, where $\alpha$ is the half-life parameter, i.e., the number of the item on the recommendation list with probability 0.5 of being seen. $\alpha$ is typically chosen to be the number of items in the recommendation list given by the recommender system divided by two, e.g., if the recommender system gives a list of ten items, then $\alpha = 5$. The exponential decay score is given by:

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

where $C$ is the set of transactions of the form $t_{i-j}, \ldots, t_{i-1}$, and $\text{pos}(t_i|c)$ is the position of the observed item $t_i$ is the recommendation list for $c \in C$. 

9
2.2.4 Results

Shani et al. tested their Markov chain predictive model using data from the online Israeli bookstore Mitos [5]. They performed tests using \( k = 1, \ldots, 5 \), and found \( k = 3 \) to be sufficient in most cases. They also tested a predictive model using unordered Markov chains in which sequences such as \( x, y, z \) and \( z, x, y \) are mapped to the same state, but it was found that the ordering of states made a substantial difference. Their predictive model was also compared to other predictive software such as Microsoft’s Predictor tool, and was found to be competitive.

3 The Coupon Issuing System

The predictive model outlined above does not account for short or long term effects of recommendations or coupons nor does it try to optimize any kind of behaviour. In this section we extend the Markov chain outlined in the previous section to a Markov decision process that will determine the optimal coupon or recommendation to issue to a user at fixed points based on what they are likely to buy next and the potential profit made from the sale.

3.1 The Markov Decision Process

A Markov decision process (MDP) is a model for sequential stochastic decision problems (see Bellman [14]). We can think of an MDP as an extension of a Markov chain. To get an MDP from a Markov chain we must add actions (choice), and reward (motivation). We give a formal definition:

**Definition 3.** A Markov Decision Process (MDP) is a 5-tuple \((S, A, R_a, P_a, \gamma)\), where

- \( S \) is a finite set of states,
- \( A \) is a finite set of actions,
- \( P_a(s, s') \) is the transition function, i.e., the probability that performing action \( a \) in state \( s \) will lead to the state \( s' \),
- \( R_a(s, s') \) is the immediate reward (or expected reward) for transitioning from state \( s \) to state \( s' \).
• $\gamma \in [0, 1]$ is the discount factor, which represents the difference in importance for future awards and present rewards.

The core problem of MDPs is to find a policy $\pi$ that specifies the action $\pi(s) = a$ to perform while in state $s$ that maximizes some cumulative function of the rewards. We will discuss this further in section 3.2. Similar to the Markov chain in our predictive model, we must first define the elements that compose our MDP:

• **States:** The states of the MDP correspond exactly to the states used in our predictive model.

• **Actions:** The actions of the MDP correspond to the issuing of a coupon. We represent our actions as coupons:

Definition 4. We define a coupon as a pair $(x, d)$ where $x$ is an item and $d \in [0, 1]$ is the discount given by the coupon (in percent), e.g., if the coupon is for 30% off item $x$, we denote it by $(x, 0.3)$.

We note that coupons in dollar amount can easily be written in this way, e.g., for a coupon giving $5 off item $x$ we simply set $d = \frac{5}{\text{price}(x)}$ where $\text{price}(x)$ is the total price of item $x$. When a coupon $(x', d)$ is issued to a user in state $<x_1, \ldots, x_k>$ the user has three options:

1. Use the coupon and purchase $x'$, thus transferring to state $<x_2, \ldots, x_k, x'>$.
2. Purchase the item $x''$ that the user has not received a coupon for and transfer to state $<x_2, \ldots, x_k, x''>$.
3. Purchase no items and remain in state $<x_1, \ldots, x_k>$.

Thus, the stochastic element in this model is the choice that the user actually makes.

We make a number of assumptions on coupons for our model. The first is that a user can only possess one coupon at once. There are two main reasons for this. The first is computational complexity, which will be discussed in section 3.2. The second is security. While there is a small potential for our model to be exploited, issuing multiple discount coupons makes the possibility of exploitation more complex as users will be able to stockpile coupons. Thus, we also assume that a user’s old coupon expires when that user receives a new coupon. We note that the set of coupons can change.
Because our coupons are only issued at fixed points (every $n$ transactions), our actions will represent the issuing of recommendations even when we are not issuing a coupon. We note that a recommendation can be represented as a coupon with discount $d = 0$, thus when we refer to coupons we include recommendations. For a coupon with $d > 0$, we will refer to it as a discount coupon. In our algorithms we consider two action spaces: our discount coupon action space will be the set of all available discount coupons, and our recommendation action space will consist of the coupons $(x, 0)$ for all $x$.

- **Rewards:** Rewards in this system encode the utility of selling an item. Because the state encodes the list of items purchased, the reward only depends on the last item defining the current state, e.g., the reward for transitioning to the state $< x_1, x_2, x_3 >$ only depends on $x_3$. In our model our reward corresponds to the net profit of a sale, but takes coupons into consideration. For example, if we let $a = (x, d)$, in our model we set $R_a(< x_1, x_2, x_3 >, < x_2, x_3, x >) = (1 - d) \cdot \text{price}(x) - \text{cost}(x)$, and $R_a(< x_1, x_2, x_3 >, < x_2, x_3, x' >) = \text{price}(x') - \text{cost}(x')$ for $x \neq x'$.

- **The Discount Factor:** The discount factor represents the difference in importance for future awards and present awards. Because most e-commerce sites are concerned with making a sale right away, we choose $\gamma$ to be small.

- **The Transition Function:** Here, our transition function is slightly different than in the predictive model because we incorporate actions (coupons). We must make two assumptions:

  1. A coupon being issued to a user will increase the probability that the user will buy the corresponding item. This probability is proportional to the probability that the user will buy this item in the absence of coupons, and is also proportional to the discount given by the coupon. We denote the proportionality constant for coupon $a$ in state $s$ by $\alpha_{s,a}$, where $\alpha_{s,a} > 1$.

  2. The probability that a user will buy an item that they did not receive a coupon for is lower than the probability that she will buy when the system issues no coupons at all, but is still proportional to it. We denote the proportionality constant for coupon $a$ in state $s$ by $\beta_{s,a}$ where $\beta_{s,a} < 1$.

Thus, if a user is in state $s = < x_1, \ldots, x_k >$ and receives a coupon $a = (x', d)$
for an item \( x' \), we set
\[
P_a(s, s') = \alpha_{s,a} P(s, s'),
\]
where \( s' = < x_2, \ldots, x_k, x' > \) and we recall \( P(s, s') \) is the transition function from our predictive model. Thus, \( P_a(s, s') \) is the probability the user will buy the item \( x' \) that they received a coupon for. Likewise, if a user is in state \( s = < x_1, \ldots, x_k > \) and receives a coupon \( a' = (x'', d) \) for an item \( x'' \neq x' \), we have
\[
P_{a'}(s, s') = \beta_{s,a'} P(s, s'),
\]
the probability a user will buy \( x' \) if they receive a coupon for a different item \( x'' \). The probability a user in state \( s = < x_1, \ldots, x_k > \) purchases nothing after receiving a recommendation for item \( x' \) is:
\[
P_a(s) = 1 - \alpha_{s,a} P(s, s') - \sum_{x'' \neq x'} \beta_{s,a} P(s, < x_2, \ldots, x_k, x'' >).
\]
The proportionality constants \( \alpha_{s,a} \) and \( \beta_{s,a} \) must be chosen carefully. Shani et al. implemented a similar system, but chose to make \( \alpha_{s,a} \) and \( \beta_{s,a} \) constant over all states and actions for simplicity [5]. In our case we cannot do this as our coupons have different discount values and our constants are proportional to these values.

It has been shown by Kitts et al. that users are more likely to follow a recommendation when the recommended item has high lift:
\[
lift(a) = \frac{pr(a|x)}{pr(a)}
\]
where \( a \) is an item and \( b \) is a user’s most recently bought item [15]. Thus, it is reasonable to assume that the probability of a user buying an item with an issued coupon is proportional to lift.

It has also been found that consumers are more likely to use a coupon with a high discount percentage rather than a coupon that will save them more money overall (see Thomas et al. [16]). For example, consumers are more likely to use a coupon that gives 20% of an item that is $5, saving them $1, than a coupon that gives 10% of an item that is $15, saving them $1.50. Thus, for a coupon \((x, d)\), \( d \) is the most relevant measure of how our probabilities change. Unfortunately, we do not have any results on the behaviour of consumers as the values of the coupons rise, so we model it as a linear relationship.

For a coupon \( a = (x, d) \), we set \( \gamma_a = \frac{d+1}{1000} \). We note that if \( a \) is a recom-
mendation, \( \gamma_a = \frac{1}{1000} \). We then calculate

\[
\alpha_{s,a} = \frac{\gamma_a + pr(x)}{pr(x)}
\]

where \( s = \langle x_1, \ldots, x_k \rangle \) is the current state of the user in question, \( s' = \langle x_2, \ldots, x_k, x \rangle \), and \( pr(x) \) is the prior probability that the user will buy \( x \). This gives us a proportionality constant slightly above 1. We can then solve for \( \beta_{s,a} \) using the following restriction:

\[
P_a(s, s') + \sum_{a \neq a' \in A} P_{a'}(s, s') < 1,
\]

recalling that we have two action spaces, i.e., the above equality is satisfied for all actions in our discount coupon space, and for our recommendation action space.

One element of our system that should be touched upon is the potential for customers to exploit the system to receive specific discount coupons. We note that customers will not have access to many of the elements in our system, including full user data, the cost of an item, and a list of all coupons. Thus, with proper security measures it will be virtually impossible for a user to recreate an optimal policy. Because of this, a user will most likely attempt to exploit the system by receiving the same discount coupon twice or receiving a discount coupon issued to someone they know. Both these scenarios can easily be worked around by frequently changing the set of discount coupons used and altering our system so that a user cannot receive the same discount coupon twice.

Now that all elements of our MDP have been addressed, we must discuss the solving of our MDP.

### 3.2 Solving the Markov Decision Process

As mentioned above, in an MDP we are trying to make a decision that maximizes some function of its reward stream. An optimal solution to an MDP is this maximizing behaviour. Formally, we define a policy for an MDP as a mapping from states to actions, which gives what action to perform in each state. We are looking for an optimal policy \( \pi \) that specifies the action \( a = \pi(s) \) to perform while in state \( s \) that maximizes this function of our reward stream. In our system, given a user’s purchase
history, an optimal policy tells us the optimal coupon to issue to that user.

Traditionally, the function we wish to maximize is known as the value function. The value function of a policy, \( V^\pi \), assigns a value to each state \( s \) corresponding to the expected infinite horizon discounted sum of rewards obtained when using \( \pi \) starting from the state \( s \). Basically, the value function accounts for all future purchases made while following \( \pi \) and starting from \( s \). The value function satisfies the recursive equation:

\[
V^\pi(s) = \sum_{s'} P_{\pi(s)}(s, s')(R_{\pi(s)}(s, s') + \gamma V^\pi(s')).
\]

Various algorithms exist for computing an optimal policy. Below we discuss the relevant methods:

### 3.2.1 Policy Iteration

Policy iteration (see Howard [17]) is one of the first and most basic algorithms for computing an optimal policy. It was the algorithm used by Shani et al. when implementing their MDP-based recommender system [5]. To perform policy iteration, we simply start with an arbitrary policy, for example expected reward:

\[
\pi_0(s) = \arg\max_{a \in A} \left( \sum_{s'} P_a(s, s') R_a(s, s') \right).
\]

At each step we then compute the value function based on the former policy (policy evaluation), then update the policy given the new value function (policy improvement):

\[
V_i(s) = \sum_{s'} P_{\pi_i(s)}(s, s')(R_{\pi_i(s)}(s, s') + \gamma V_i(s')),
\]

\[
\pi_{i+1} = \arg\max_{a \in A} \left( \sum_{s'} P_a(s, s') (R_a(s, s') + \gamma V_i(s')) \right).
\]

We stop when \( \pi_i = \pi_{i+1} \). It is known that policy iteration will always converge to an optimal policy.

Traditionally, solving MDPs via policy iteration is a polynomial problem in the number of states. In order to compute our value functions, we must solve a system of equations, in which there are \( O(|S|) \) equations. Using linear programming, this gives us a complexity of \( O(|S|^3) \) for each iteration, although certain aspects of the coupon issuing system state space reduces this. These aspects include [5]:

---

15
• **Directionality:** Transitions in our space have built in directionality. It is clear that some states cannot follow other states. Also, if item \( x \) is likely to be bought after item \( y \), item \( y \) is less likely to be bought after item \( x \) (in most cases). Thus, loops are very unlikely to occur when calculating our optimal policy.

• **Indifference to \( k \)-value:** It has been found that policy iteration is not very sensitive to the number of past purchases listed in a state. For example, encapsulating the past 2 purchases and the past 3 purchases will often give the same policy.

• **Unobserved States:** In practice, many states will not appear in our user data. When using policy iteration, ignoring these states will still give an optimal policy. Not only this, but there is no need to compute a policy choice for a state that was unobserved in user data. If a new state that has not been observed before does occur in practice, its transition function value can be estimated using the techniques such as skipping described in section 2 and its policy value can quickly be computed.

Unfortunately, due to our action space (switching between discount coupons and recommendations), policy iteration does not work for our application. Fortunately, there is a modified version of policy iteration that will work, and actually has the potential to improve the computational complexity of our system.

### 3.2.2 Modified Policy Iteration

Value iteration is another traditional method of finding an optimal policy (see Bellman [18]). In value iteration, the policy \( \pi \) is not used. Instead, the value of \( \pi(s) \) is calculated within the value function whenever it is needed. Starting with an arbitrary value function (usually \( V_0 = 0 \)), we compute:

\[
V_{i+1}(s) = \max_{a \in A} \left( \sum_{s'} P_a(s, s')(R_a(s, s') + \gamma V_i(s')) \right).
\]

Value iteration does not usually converge; instead, one chooses an arbitrary stop point. Traditionally, value iteration has complexity \( O(|S|^2|A|) \).

Modified policy iteration combines both policy iteration and value iteration so to get the best aspects of both algorithms, i.e., the fast convergence of policy iteration and the complexity of value iteration. In our case, we also receive the benefits given
by policy iteration due to our state space.

In traditional policy iteration, at each step we evaluate our policy and update our value function by solving a system of equations. Modified policy iteration replaces this step with value iteration iterating several times. We then update our policy and repeat.

Each iteration of the value iteration can be thought of as a step in our process. Thus, when we are taking the value at a step in which we issue a discount coupon, we can use our discount coupon action space. When are taking the value at a step issuing a recommendation, we can use our recommendation action space. When calculating our policy using this method, we need to use our input parameter $n$ to determine how many recommendations are issued between the issuing of our discount coupons, i.e., how many value iterations in the recommendation action space do we perform before using the coupon action space.

We recall that in our application, we issue a coupon at fixed points (every $n$ transactions). Then, when we are issuing a discount coupon, our first iteration uses our discount coupon action space, followed by $n$ iterations using our recommendation action space, followed by a last iteration using our discount coupon action space. Our policy improvement step then gives a policy that issues discount coupons, and we repeat. When issuing a recommendation, if $n$ is even, we compute $\frac{n}{2}$ iterations using our recommendation action space, followed by one with our discount coupon action space, followed by $\frac{n}{2}$ in our recommendation action space. If $n$ is odd, we simply replace the first $\frac{n}{2}$ iterations with $\frac{n-1}{2}$ iterations and the last $\frac{n}{2}$ iterations with $\frac{n+1}{2}$ iterations.

This method replaces the $O(|S|^3)$ complexity of each policy evaluation iteration with a complexity of $O(n|S|^2|A|)$. In practice, $|A|$ will be significantly smaller than $|S|$, and $n$ will be relatively small, thus this is an improvement. Because we are dealing with a finite number of policies, we know our modified version of the algorithm will converge.

### 3.2.3 Approximate Modified Policy Iteration

Because of the way our state space is designed, an increase in the number of items in our system would exponentially increase the number of states in our state space. This is known as the “curse of dimensionality”. Because the amount of data in our world is growing at an alarming rate, much work has been done recently on reducing the impact of this “curse” while solving MDPs. This is often done by using
approximations and estimations for the value function.

One such algorithm is known as the AMPI-V algorithm (see Scherrer et al. [19]). In this algorithm, we start with an arbitrary value function $V_0$. For this algorithm, we assume that the values given by $V_i$ are represented in a function space $\mathcal{F} \subseteq \mathbb{R}^{\left|\mathcal{S}\right|}$. We can then estimate $\pi_{i+1}(s)$ as follows:

$$
\pi_{i+1}(s) = \arg\max_{a \in A} \left( \frac{1}{M} \sum_{j=1}^{M} P_a(s, s_j)(R_a(s, s_j) + \gamma V_i(s_j)) \right)
$$

where, for $1 \leq j \leq M$, and all $a \in A$, $s_j$ is a sample state taken from the possible next state when action $a$ is performed in state $s$. Thus, each iteration of our policy improvement step needs $M|A|$ samples. In our case, we sample from our predictive model.

The algorithm works as follows. At each iteration, we sample $N$ states from our state space using the simple distribution given from our user data. For each sampled state $s$, it generates a rollout of size $m$, i.e., $(s, a_0, r_0, s_1, a_1, \ldots, r_{m-1}, s_m)$, where $a_t$ is the action suggested by the estimated policy, i.e., $a_t = \pi_{i+1}(s_t)$, $r_t$ is the expected reward induced by this action, and $s_{t+1}$ is the next state induced by this action, given to us by our transition function. For each of these $N$ rollouts, we then compute a rollout estimation:

$$
\hat{V}_{i+1}(s) = \left( \sum_{t=0}^{m-1} \gamma^t \left( \sum_{s'} P_{a_t}(s, s') R_{a_t}(s, s') \right) \right) + \gamma^m V_i(s_m),
$$

where $s$ is the starting state of the rollout. Finally, $V_{k+1}$ is computed as the best fit in our function space $\mathcal{F}$ using these estimates, i.e., it is a function $v \in \mathcal{F}$ that minimizes the empirical error:

$$
\frac{1}{N} \sum_{i=1}^{N} (\hat{V}_{i+1}(s_i) - v(s_i))^2,
$$

where each $s_i$ is a starting state of a rollout.

Each iteration of this algorithm requires $N$ rollouts of size $m$, and for each rollout any of the $|A|$ actions need $M$ samples to calculate the estimation of the policy. Thus, a total of $Nm(M|A| + 1)$ samples are needed.

Choosing the values for $N$, $m$, and $M$ is a matter of how accurate one wants the algorithm to be. In each iteration of the algorithm there are two places where errors arise; we let $\epsilon_k$ denote the error when fitting the value function $V_k$, and we let $\epsilon'_k$
denote the error due to using $M$ samples to improve our policy. We note that $\epsilon_k$ can further be broken down into two parts: one related to the finite number of samples when computing our rollouts, and one related to finding the best fit for $V_k$ in $F$.

Scherrer et al. found upper bounds for these errors by viewing $F$ as a linear function space and framing the problem in a linear regression setting [19]. They considered a linear architecture with parameters $\alpha \in \mathbb{R}^d$ and bounded basis functions $\{\varphi_j\}^d_{j=1}$, $||\varphi_j||_{\infty} \leq L$. Letting $\phi(\cdot) = (\varphi_1(\cdot), \ldots, \varphi_d(\cdot))^T$ be the feature vector, we can then think of $F$ as the span of the features, i.e., $F = \{f_\alpha(\cdot) = \phi(\cdot)^T \alpha : \alpha \in \mathbb{R}^d\}$. Defining $V_k$ as the truncation (by the maximum value of the value function, $V_{\text{max}}$) of the solution of this linear regression problem, the error $\epsilon_k$ can then be bounded:

**Proposition 1.** [19] Consider the linear regression problem described above, and let $\mu$ be the probability distribution from which we sample the starting states for our $N$ rollouts. Then for any $\delta > 0$,

$$
||\epsilon_k||_{2,\mu} \leq 4 \cdot \inf_{f \in F} ||V_k - f||_{2,\mu} + \epsilon_1(N, \delta) + \epsilon_2(N, \delta),
$$

with probability at least $1 - \delta$, where

$$
\epsilon_1(N, \delta) = 32V_{\text{max}} \sqrt{\frac{2}{N} \log \left( \frac{27(12e^2N)^2d+1}{\delta} \right)},
$$

$$
\epsilon_2(N, \delta) = 24(V_{\text{max}} + ||\alpha||_2 \cdot \sup_x ||\phi(x)||_2) \sqrt{\frac{2}{N} \log \left( \frac{9}{\delta} \right)},
$$

where $\alpha$ is such that $f_\alpha$ is the best approximation of $V_k$ in $F$.

For bounding $\epsilon'_k$, Scherrer et al. assumed that the action space in question only had 2 actions, i.e., $|A| = 2$ [19]. We recall that the VC-dimension of a classification model is the maximum number of points that can be arranged so that the model shatters them. A model with a parameter vector $\alpha$ is said to shatter a set of data points $(x_1, \ldots, x_n)$ if there exists an $\alpha$ such that the model makes no errors when evaluating that set of data points. For example, consider a straight line that classifies positive and negative points by separating them; any 3 points that are not collinear can be shattered, but it is impossible to separate any 4 points by Radon’s theorem. Thus, the VC-dimension of this classifier is 3. Using VC-dimension, we can bound $\epsilon'_k$ in the case when $|A| = 2$:

**Proposition 2.** [19] Let $h$ be the VC-dimension of of the policy space obtained by
the estimation of $\pi_{k+1}$ from the truncation (by $V_{\text{max}}$) of the function space $\mathcal{F}$, and let $\mu$ be the probability distribution from which we sampled $N$ states for our rollout. Then, for each of those $N$ sampled states, and any $\delta > 0$,

$$||e'(s)||_{1,\mu} \leq e_3'(N, \delta) + e_4'(N, M, \delta) + e_5'(N, M, \delta),$$

with a probability of at least $1 - \delta$, where

$$e_3'(N, \delta) = 16V_{\text{max}}\sqrt{\frac{2}{N}(h \cdot \log\left(\frac{eN}{h}\right) + \log(\frac{24}{\delta}))},$$

$$e_4'(N, M, \delta) = 8V_{\text{max}}\sqrt{\frac{2}{NM}(h \cdot \log\left(\frac{eNM}{h}\right) + \log(\frac{24}{\delta}))},$$

$$e_5'(N, M, \delta) = V_{\text{max}}\sqrt{\frac{2\log(3N/\delta)}{M}}.$$

The extension of this proposition to more than two actions is fairly straightforward, with the main difference between the two cases being the replacement of VC-dimension with Natarajan dimension, which is a generalized version of the VC-dimension for multiclass classifiers. This generalization is achieved by generalizing the definition of shattering a set. Other methods of approximate modified policy iteration include AMPI-Q, and CBMPI [19].

### 3.3 Updating the Model

Once the coupon issuing system is up and running, the model has to be updated using actual observed data showing user behaviour with implemented coupons. There are two main approaches to this. The first is to perform offline updates during fixed intervals. The second is to perform updates online using reinforcement learning. Shani et al. used the former method, updating the transition function and model weekly. They claimed that user reinforced learning was too computationally expensive and that it slowed their recommendations [5].

Like Shani et al., we issue recommendations every time a user goes to purchase an item, but we are not working in our coupon action space very frequently, thus we have the computational freedom to implement reinforcement learning so that the probabilities related to our coupons improve in real time. We take a hybrid approach to updating our model, updating our transition function offline at fixed time intervals for our recommendations and using reinforcement learning to update the probabilities.
related to coupons in real time.

We use the method given by Shani et al. to update our transition function for recommendations [5]. Letting \( s \) be a state and \( s' \) be the state transitioned to from \( s \) after the purchase of item \( x \), we take the following counts:

- \( c_{in}(s, s') \) is the number of times a user in state \( s \) bought item \( x \) after receiving a recommendation for item \( x \).
- \( c_{out}(s, s') \) is the number of times a user in state \( s \) bought item \( x \) after receiving a recommendation for a different item.
- \( c_{total}(s, s') \) is the total number of times a user in state \( s \) bought the item \( x \).

We initialize these counts at time \( t=0 \):

\[
\begin{align*}
c_{in}^0(s, s') &= \zeta_{s} P_{a}(s, s') \\
c_{out}^0(s, s') &= \zeta_{s} P_{a}(s, s') \\
c_{total}^0(s, s') &= \zeta_{s}
\end{align*}
\]

where \( a \) is a coupon for the item \( x \), \( P \) is the initial transition function, and \( \zeta_{s} \) is a constant proportional to the number of times the state \( s \) appears in the original data. Shani et al. chose \( \zeta_{s} = 10 \cdot \text{count}(s) \) [5]. At time \( t+1 \) we compute:

\[
\begin{align*}
c_{in}^{t+1}(s, s') &= c_{in}^{t}(s, s') + \text{count}(s, a, s') \\
c_{out}^{t+1}(s, s') &= c_{out}^{t}(s, s') + \text{count}(s, s') \\
c_{out}^{t+1}(s, s') &= c_{out}^{t}(s, s') + \text{count}(s, s') - \text{count}(s, a, s')
\end{align*}
\]

\[
\begin{align*}
P_{a}(s, s') &= \frac{c_{in}^{t+1}(s, s')}{c_{total}^{t+1}(s, s')} \\
P_{a'}(s, s') &= \frac{c_{out}^{t+1}(s, s')}{c_{total}^{t+1}(s, s')}
\end{align*}
\]

where \( a' \) is the coupon for any item \( x' \neq x \), \( \text{count}(s, a, s') \) is the number of times people bought \( x \) after being given a coupon for it between times \( t \) and \( t + 1 \), \( \text{count}(s, s') \) is
the total number of people who bought $x$ while in state $s$, and all other variables are the same as outlined above. The probability that a user stays in the same state is computed in the same way it was in section 3.1. These are the probabilities we will use while working in the recommendation action space.

We use real time learning to update the probabilities used with our discount coupons. Because of the nature of coupons, we can not take a similar approach to that used by Shani et al. for recommendations. When a user is issued a coupon but buys another item, they still possess this coupon and can use it in the future, thus we do not have a value analogous to $c_{out}$ as defined above. The way we get around this is by using our proportionality constants from section 2.1. Letting $a = (x, d)$, we let $count(a)$ be the number of times a user has used the coupon $a$, and $count(x)$ be the number of people who bought item $x$ (this is analogous to $c_{total}$ above), updating our counts each time a transaction takes place. We then set

$$\gamma_a = \frac{d + 1 + (count(a)/count(x))}{1000},$$

recalling that

$$\alpha_{s,a} = \frac{\gamma_a + pr(x)}{pr(x)}.$$

We can then solve for an appropriate $\beta_{s,a}$ and compute our new probabilities by updating the counts in our predictive model and using the equations found in section 3.1.

### 3.4 Evaluation of the Coupon Issuing System

In collaboration with the Israeli bookstore Mitos, Shani et al. implemented their MDP-based recommender system (using basic policy iteration) as well as three other systems based on the predictive model and compared the profit gained by using each system. While all four systems gave similar results, the system that implemented the MDP’s optimal policy gave the highest average discounted profit. Shani et al. also compared it to the Markov chain based Microsoft Predictor model, and found that the MDP model gave a 27% higher increase in profits over the Predictor model, and a 16% increase in total profits [5].

Shani et al. also compared the computational run-time of their system to the run-time of the Predictor model, finding that the MDP system generates models and gives recommendations at a much faster rate than the Predictor model [5]. This high
speed comes at the price of memory, as the MDP model will have a significantly higher memory footprint when dealing with large state spaces.

These results show that the recommendation process can in fact be modeled as a sequential decision problem and that MDPs provide an adequate model for this view. Though their have been no results for MDP recommender systems that issue coupons, we expect our results to be similar to those given above. One particularly useful study would be the comparison of run-times for MDP-based recommender systems implementing basic policy iteration, modified policy iteration, and approximate policy iteration. Scherrer et al. showed that approximate policy iteration can reduce computational run-time in a number of different applications [19].

4 Conclusion and Future Work

In conclusion, we have presented a coupon dispensing system based on the MDP recommender system first presented by Shani et al. [5]. Specifically, given a set of discount coupons, and an integer \( n \), our system decides which coupon to issue to a specific user at fixed points by predicting their future purchases and optimizing future profit from the issuing of the coupon. In our system, a user is issued a coupon every \( n \) transactions they make, and specific recommendations are issued to the user between the issuing of coupons.

Beyond coupons, we presented some additional changes to the original model. Specifically, we presented a new method for calculating the probabilities for our predictive model using sequential rule mining, adapted modified policy iteration and approximate modified policy iteration to our state space, and gave a new learning method for updating the MDP model. All these changes should be tested and implemented using real user data. In particular, different weightings for the transition function of our predictive model should be tested; the computational run-time of modified policy iteration and approximate modified policy iteration should be compared; and the profit increase due to coupons should be tested. The security of our system should also be tested, i.e., how easy it is to exploit the system to get specific discount coupons? As mentioned in section 3.1, this will most likely not be a big concern, and easy fixes are available.

Beyond testing our model, studies relating to other behaviour involving coupons would be helpful, particularly consumer’s behaviour as the discount of a coupon rises. It would also be helpful to incorporate user information such as gender and age into
our decisions, and to incorporate some kind of hierarchical system for items so that our probability estimations involving items such as sequels improve. Although this information is embedded in the user data, it could be helpful in making decisions when user data is especially sparse.

Lastly, it would be interesting to see if a partially observable Markov decision process (POMDP) could be implemented for our sequential decision problem. In this model we do not ignore unobserved states, but rather use additional probability estimations based on these states to make decisions. This model would be more accurate, but unfortunately current algorithms may be too computationally complex for practical use.

5 Acknowledgments

I wish to thank my supervisor, Dr. Kumar Murty for his guidance and help throughout this project. I would also like to thank my family, friends, and colleagues who made my time at the University of Toronto a memorable experience.

References


