# Automatic Representation for Life-Time Value Recommender Systems

**Assaf Hallak** ifogph@gmail.com
**Elad Yom-Tov** eladyt@microsoft.com
**Yishay Mansour** mansour@microsoft.com
*Microsoft ILDC*
*Hertzelya, Israel*

## Abstract

Recommender systems are embedded in almost every commercial site, proposing users items which are likely to draw their interest. While most systems maximize the immediate gain, a better notion of success would be the lifetime value (LTV) of the user-system interaction. The LTV approach instead considers the future implications of the item recommendations, and seeks to maximize over the cumulative gain over time. Reinforcement Learning (RL) framework is the standard formulation for optimizing the cumulative successes over time, but RL is rarely used in practice due to its complicated representation, optimization and validation techniques. In this paper we propose a new architecture for combining RL with recommendation systems which obviates the need for hand-tuned features, thus automating the state-space representation construction process. We analyze the practical difficulties in this formulation and test our solutions on real-world recommendation data.

**Keywords:** Recommender Systems, Reinforcement Learning, Feature Selection

## 1. Introduction

Recommender systems (RS) study the problem of optimizing the user interaction with the item catalog. In each session, the user is recommended an item by the system and either accepts the recommendation or rejects it. Recommender systems often interact with the same user repeatedly, and seek to improve the recommendations through personalization and cross-user inference. RS formulation have been applied to several commercial domains including movies, music and games recommendations, as well as some marketing schemes (Resnick and Varian, 1997; Adomavicius and Tuzhilin, 2005; Ricci et al., 2011).

Traditional approaches for recommender systems learn the preference of each user from offline batch data. The learned model is used to recommend additional items when the user queries the system. As more data is collected and time passes, the model is refined according to the new samples. One specific approach related to our work is Matrix Factorization (MF; (Koren et al., 2009)) which learns the user-item preference matrix through partial or noisy observations. These suggested solutions' aim is to optimize the success probability of the current interaction with the user. However, for systems that interact with the same users repeatedly over time, a more suitable metric would be the cumulative successes over time, also known as lifetime value (LTV) (Theocharous and Hallak, 2013; Pfeifer and Carraway, 2000). The LTV perspective can be beneficial in many scenarios that myopic policies (optimizing the current interaction result) might find challenging: Expanding the user's taste, avoid insulting sensitive users, not recommending a competitor and more. The alternative view which considers the dynamics of the interaction and optimizes over the LTV is usually formalized through the Reinforcement Learning (RL) framework (Sutton and Barto, 1998).

While the idea of optimizing interaction with the user over time has been around for many years (Jonker et al., 2004; Pednault et al., 2002; Pfeifer and Carraway, 2000), the formulation is usually content based which leads to hand-tuned features and dynamics. A more relevant solution was proposed by Shani et al. (2005), where the aggregated past $k$ recommendations are used to define the current state, this approach has scalability issues and cannot cope with high $k$ values or with a large recommendations set.

In this paper we propose the automatic construction of features for RL which aggregates all past recommendations to one feature vector using the MF framework. We follow a common scenario in which maximizing the LTV is considered as an improvement for a given recommender, and lay down the process for answering whether it should be incorporated.

## 2. Matrix Factorization

Our automatic feature construction is based on the MF technique commonly used for RS. Assume there are $m$ users and $n$ items, MF-algorithms receive as input the sampled rating matrix $\mathcal{R} \in \mathbb{R}^{m,n}$ which contains in the $(i,j)$ coordinate the rating user $i$ gave item $j$ (or 0/1 if the item was rejected/accepted), or 0 if the item was not rated by the user. The algorithm outputs $k$-dimensional latent vectors for each user and item written in matrix form by $U \in \mathbb{R}^{m,k}$ and $V \in \mathbb{R}^{k,n}$ correspondingly. These vectors are then used to reconstruct the rating matrix $F$ through various models. For example, the simplest method solves: $UV = \mathcal{R}, U \in \mathbb{R}^{m,k}, V \in \mathbb{R}^{k,n}$, where regularization and bias terms can be added to improve the model or reduce over-fitting (Koren et al., 2009). In this paper we propose an architecture in which any MF algorithm can be plugged in as a black-box.

## 3. Reinforcement Learning

In this section we provide the key notations and definitions required for the paper: An MDP is a tuple $M = (\mathcal{S}, \mathcal{A}, P, R, \nu)$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P$ is the transition probability kernel, $R$ is the reward function and $\nu$ is the initial state distribution. The process iterates as follows: first an initial state is sampled from $\mathcal{S}$: $s_0 \sim \nu$. Afterwards, each time step $t = 0, 1, ...$ the agent chooses an action $a_t \in \mathcal{A}$, receives a stochastic reward $r_t \sim R(s_t, a_t)$ and the state transitions to $s_{t+1} \sim P(s_t, a_t)$. A strategy for choosing actions given the current state $s$ is called a policy, we denote by $\pi(a|s)$ the probability to choose the action $a$ in state $s$ according to policy $\pi$.

Due to the everlasting nature of recommender systems, we consider the $\gamma$-discounted infinite horizon setup. The LTV of a specific state $s$ (also known as the value function) when following the policy $\pi$ is the expected cumulative discounted reward: $V^\pi(s) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right]$, and the mean value of the policy is $J^\pi = \sum_{s \in \mathcal{S}} \nu(s) V^\pi(s)$. The problem of finding the policy maximizing $J^\mu$ is called planning. Another useful notation is the $Q$-function: $Q^\mu(s,a) = \mathbb{E}\left[R(s,a)\right] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\pi(s')$, which is the cumulative expected value of first taking action $a$ in state $s$, and following with the policy $\pi$ for the rest of the actions. The Q-function is often used to make a one-step improvement of the current policy by maximizing its value over the actions.

For problems with an excessively large state space, the common practice is to find features for every state $\phi(s) \in \mathbb{R}^d$ where $d$ is the number of features, and then use a linear function approximation for the value function: $V(s) \approx \theta^\top \phi(s)$ for some weight vector $\theta \in \mathbb{R}^d$. Similarly, if the action space is large the $Q$-function can be approximated through $Q(s,a) \approx \theta^\top \phi(s,a)$ for some other feature vector $\phi(s,a) \in \mathbb{R}^d$.

One major problem in the scenario described in Section 5 is called policy evaluation: Given batch data, find the value $V^\pi(s)$ for a specific policy $\pi$ which may have generated the data (on-line policy evaluation), or not (off-line policy evaluation). If we use linear function approximation, this problem translates to finding an appropriate weight vector $\theta$.

## 4. Model Construction

In sequential recommender systems the state features are often composed of engineered parameters such as the time steps since the last click, the cumulative reward or the user's demographics (Pednault et al., 2002; Theocharous et al., 2015). These suffer from scaling and tuning issues, redundancy and clunky dynamics. In addition, it is not uncommon for a recommender system to provide recommendations from a large item catalog corresponding to a large action space. Hence a feature vector for a state-action pair is needed. Subsequently, our main goal in this paper is to automate the feature generating process.

Ossume we are given such an MF black-box which given batch data of the form (User ID, Item ID, Rating, Timestamp), learns a model fitting every item and every user with their latent vectors of dimension $k$. In addition, we expect this model to support queries composed from a user's history of system interactions: $u = \mathrm{MF}(h)$, which returns the corresponding latent vector for this user. We propose the following architecture:

**Input**: Off-line batch data (User ID, Item ID, Rating, Time stamp), dimension $k$.

1. Apply an MF algorithm on the data such that each user $i$ and item recommendation $j$ are coupled with the corresponding vectors $u_i, v_j \in \mathbb{R}^k$.

2. For each user $i$, aggregate the data to trajectories $u_i$: $(u_i, v_{i,t}, r_{i,t})_{t=0}^{T_i}$ using the resulting representation and the timestamp, where $T_i$ is the number of interactions with the user. Denote the $i$'th user's history at each time step $t = 0..T_i$: $h_{i,t} = (u_i, v_{i,l}, r_{i,l})_{l=0}^{t}$.

3. For each user, generate the trajectory: $s_{i,t} = \mathrm{MF}(h_{i,t})$ for every $t = 0..T_i$.

4. Apply any RL technique on the resulting trajectories $\{(s_{i,t}, v_{i,t}, r_{i,t})_{t=0}^{T_i}\}_i$.

## 5. Off-line Setup

In this scenario, suppose we are providing recommendation services and we are given data which was acquired through some existing recommender system. The company is looking for ways to increase its revenue over time, and an RL based solution is suggested instead of the solution currently in place. Alas, without reasonable confidence in its improved performance the RL based solution will not be tried out (or A/B tested). This is due to the fact that experiments require many samples for the following reasons: low success rate, large number of recommendations, and high dimensional information vector available each user. Following a sub-optimal policy reduces revenue in the short term, and affects the perception of the system by its users, losing credibility and reputation.

Hence, our goal is to suggest a better policy $\pi_t$ (the target policy) and prove its superiority over the existing behavior policy $\pi_b$, **without** deploying it on-line. To do so, we pursue the following practice: Perform **(1) on-policy evaluation** to estimate the value of the behavior policy, **(2) Estimate the Q-function**, then propose a better policy by **(3) optimizing over the Q-function**, and finally use **(4) off-policy evaluation** to evaluate the new proposed policy. We point out that the behavior policy may be unknown, which may be an additional obstacle for the some of these steps.

## 5.1 On-policy Evaluation

First, we need is to estimate the value of the current policy. To do so we use LSTD(0) (Bradtke and Barto, 1996), which essentially finds $\theta_{\pi_b}$ such that $V^{\pi_b}(s) \approx \theta_{V,\pi_b}^{\top}\phi(s)$ through least squares. After finding $\theta_{\pi_b}$, we should ask how to obtain a single value as an estimate. One approach could be to simply take the value of the cold-user, since it represents the LTV expected from a new user. However, when the user interacts with the system many times, the cold start is hardly representative of the average state of users in the system, and indeed the optimization is done over the entire trajectory. Our proposal is to instead sample several states from each trajectory and average over their values. In order to estimate the variance of this process, we could use bootstrapping for repeated sampling.

## 5.2 Q-function Estimation

We use LSTDQ(0) Peters et al. (2003) that finds a linear function approximation for the Q-function of the behavior policy. To use LSTDQ(0), we must specify the features of each state-action pair. We suggest to simply concatenate the state and action feature vectors, with an additional constant feature. Since the scalar product between these feature vectors is related to the success probability, we suggest concatenating the coordinate-wise product of these vectors as well. In summary: $\phi(s,a) = (s, v, s \odot v, 1), \qquad [s \odot v]_i = [s]_i \cdot [v]_i$.

## 5.3 Q-function Optimization

Once we have the an estimate of the Q-function, we use one-step policy improvement through softmac. Since the behavior policy is unknown, we suggest using the same parametric form for both policies, and optimize over the corresponding parameters. Specifically, we propose: $\pi(v|s) \propto \exp(w_{\pi}^{\top}\phi(s,v))$ which also assures every item has positive probability by both policies. For the target policy, we can take $w_{\pi_t} = \alpha\theta_Q$ for some $\alpha > 0$ and thus obtain the softmax policy which is commonly used in RL. As for $w_{\pi_b}$, we need to learn it from the data. Maximizing the likelihood is too computationally expensive due to the denominator: $\sum_a \exp(w_{\pi}^{\top}\phi(s,a))$, so instead we suggest dropping the denominator and adding a regularization term weighted by the regularization parameter $\eta$:

$$w_{\pi_b} = \operatorname{argmax}_w \left[\exp(-\eta\|w\|^2)\prod_{h=1}^{H}\prod_{t=0}^{T_h-1}\exp(w^{\top}\phi(s_{t,h}, v_{t,h}))\right], \tag{1}$$

which leads to a simple analytical solution $w_{\pi_b} = C\sum_{h=1}^{H}\sum_{t=0}^{T_h-1}\phi(s_{t,h},a_{t,h})$, for some constant $C>0$. It is now easier to maximize the original likelihood over a scalar parameter $C$, for example through grid/binary search.

### 5.4 Off-policy Evaluation

Following the previous sections, we chose off-policy weighted LSTD Mahmood et al. (2014), where in order to get bounds we use bootstrapping . Other methods are included in Theocharous et al. (2015) - Improved concentration inequalities as suggested in Thomas et al. (2015), Student's $t$-test and Bias Corrected and accelerated (BCa) bootstrap Efron (1987), though these are only fitted to the Monte-Carlo estimate.

### 5.5 On-line Setup

A follow up scenario to the off-line setup is deployment of the RL solution to the recommender system. In this case, the system can benefit from the vast research in on-line RL for efficient learning and planning under uncertainty - actor-critic schemes with TD-learning, policy search etc. The main advantage of the RL framework in this scenario is the fast adaptation to changes and its computational efficiency.

## 6. Experiments

In order to test our approach we applied our architecture on 2 datasets: The "Coupon Purchase Prediction" challenge publicly available through Kaggle (Kaggle) and the publicly available Movielens's 1M data set (Harper and Konstan, 2016). We compared 2 basic MF algorithms - (1) Solving the basic MF through Alternating Least Squares (ALS) with random initialization and regularization $\lambda = 0.1$, and (2) SVD applied on the sampled matrix where only the top singular values were taken; Our representation is of size $k = 20$ for both methods. To construct the states trajectory for each user, for both methods we solved the regularized equation which can be computed using a single ALS iteration: $u_t = \left(V\mathrm{diag}(w_t)V^\top + \lambda I_{k\times k}\right)^{-1} V\mathrm{diag}(w_t)q_t$, where $u_t$ denotes the state at time $t$, $V$ is the item vectors' matrix, $w_t$ is a binary weight vector with 1's in elements corresponding to catalog items consumed by the user, and $q_t$ is a vector with the observed ratings. The discount parameter $\gamma$ was chosen for every set as the maximum likelihood estimate of the drop-out probability: $\gamma = 1 - \frac{\#\text{ Users}}{\#\text{ Samples}}$.

We perform bootstrapping sampling for the on-policy and off-policy evaluation. For the SVD representation which showed better prediction results, we have also used off-policy evaluation to find the estimate of the myopic policy $\pi_m$ (which can be found similarly to $\pi_t$ using LSTDQ with $\gamma = 0$). To further evaluate our results, we used one-sided Wilcoxon signed rank test (Gibbons and Chakraborti, 2011) on the SVD results to test which approach yields higher value, the p-values are reported on the same table for every two policies (all values but one were lower than $10^{-16}$).

### 6.1 Coupon Purchase Prediction

| | Coupon | MovieLens |
|---|---|---|
| $V^{\pi_b}_{\text{ALS}}$ | $9.66 \pm 0.18$ | $16.82 \pm 2.34$ |
| $V^{\pi_t}_{\text{ALS}}$ | $11.48 \pm 0.26$ | $17.35 \pm 2.56$ |
| $V^{\pi_b}_{\text{SVD}}$ | $9.39 \pm 0.21$ | $21.98 \pm 2.49$ |
| $V^{\pi_m}_{\text{SVD}}$ | $9.21 \pm 0.19$ | $22.93 \pm 2.69$ |
| $V^{\pi_t}_{\text{SVD}}$ | $10.78 \pm 0.64$ | $26.54 \pm 3.24$ |
| $p(\pi_b, \pi_m)$ | 1 | 0 |
| $p(\pi_m, \pi_t)$ | 0 | 0 |
| $p(\pi_b, \pi_t)$ | 0 | 0 |

This data set relates to a monetary competition published in the Kaggle website `https://www.kaggle.com` in July 2015. The goal of the competition was to improve the recommendations given in the Ponpare coupon site which offers discounts in various markets. The data set is composed of ∼3M samples which contain the user ID, coupon ID, whether or not it was purchased and a timestamp. The samples were gathered over a time interval of roughly one year, and include ∼23K anonymized users and ∼33K items. Before handling the data, we have removed users that had less then 20 interactions with the system or have zero clicks, and we were left with only ∼13K users and an average success rate of 0.042. The discount factor is $\gamma = 0.9952$. The policy improvement step significantly improves the cumulative discounted reward over the behavior policy when SVD is used. Notice that $\pi_t$ also improves over the myopic policy which exhibits worse results than the behavior policy.

### 6.2 MovieLens

This data set relates to ratings gathered from the MovieLens website `http://movielens.org` and made publicly available by GroupLens Research. They offer various data sizes, where we chose to use the 1M dataset which contains 1 million ratings from 6,000 anonymous users on 4,000 movies. Note that in this dataset the order of the ratings may have been decided by the users (that alternate between movies on a whim) or the system (which recommends the user movies to rate and see). Roughly 0.042 of the samples were rated, and the ratings were scaled to be between 0 and 1, the discount factor is $\gamma = 0.994$.

## 7. Summary

Our paper discusses the practical aspects of enhancing a currently operating Recommender Systems with the look-ahead qualities enabled by the field of reinforcement learning. While adding some complexity to the problem, we have shown there can be improvement over time even with the most basic algorithms in the RS field on relatively small sample datasets.

Practically, there is much that can be done to improve the shown empirical results - better MF or RL algorithms with parameters sweeping, more data, larger latent dimension, smart embedding of side information and so on. However, our goal in this paper was not perfecting results over one specific data set, but offering an entirely new architecture that enables easy modular hands-free LTV optimization.

The suggested framework was designed from an industrial point of view since most recommender systems that deal with large quantities of data belong to data companies. Thus, any change in the production has to be justified, as working systems are rarely tampered with. We took this line of thinking into account and the new policy proposed by our algorithm builds upon the existing system, can be chosen arbitrarily close to it, and its improved performance can be proved statistically, or even bounded using the recent literature on off-policy evaluation.

## References

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749, 2005.

Dimitri P Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

Steven J Bradtke and Andrew G Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1-3):33–57, 1996.

Bradley Efron. Better bootstrap confidence intervals. *Journal of the American statistical Association*, 82(397):171–185, 1987.

Jean Dickinson Gibbons and Subhabrata Chakraborti. *Nonparametric statistical inference*. Springer, 2011.

Anupriya Gogna and Angshul Majumdar. Svd free matrix completion with online bias correction for recommender systems. In *Advances in Pattern Recognition (ICAPR), 2015 Eighth International Conference on*, pages 1–5. IEEE, 2015.

Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Prem Gopalan, Francisco J Ruiz, Rajesh Ranganath, and David M Blei. Bayesian nonparametric poisson factorization for recommendation systems. In *AISTATS*, pages 275–283, 2014.

William W Hager. Updating the inverse of a matrix. *SIAM review*, 31(2):221–239, 1989.

F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

Jedid-Jah Jonker, Nanda Piersma, and Dirk Van den Poel. Joint optimization of customer segmentation and marketing policy to maximize long-term profitability. *Expert Systems with Applications*, 27(2):159–168, 2004.

Kaggle.

Yehuda Koren, Robert Bell, Chris Volinsky, et al. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

A Rupam Mahmood, Hado P van Hasselt, and Richard S Sutton. Weighted importance sampling for off-policy learning with linear function approximation. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2014.

Edwin Pednault, Naoki Abe, and Bianca Zadrozny. Sequential cost-sensitive decision making with reinforcement learning. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 259–268. ACM, 2002.

Jan Peters, Sethu Vijayakumar, and Stefan Schaal. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots*, pages 1–20, 2003.

Phillip E Pfeifer and Robert L Carraway. Modeling customer relationships as markov chains. *Journal of interactive marketing*, 14(2):43, 2000.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the twenty-fifth conference on uncertainty in artificial intelligence*, pages 452–461. AUAI Press, 2009.

Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40 (3):56–58, 1997.

Francesco Ricci, Lior Rokach, and Bracha Shapira. *Introduction to recommender systems handbook*. Springer, 2011.

Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005.

Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Nuria Oliver, and Alan Hanjalic. Climf: learning to maximize reciprocal rank with collaborative less-is-more filtering. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 139–146. ACM, 2012.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999a.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112 (1):181–211, 1999b.

Georgios Theocharous and Assaf Hallak. Lifetime value marketing using reinforcement learning. *RLDM 2013*, page 19, 2013.

Georgios Theocharous, Philip S Thomas, and Mohammad Ghavamzadeh. Ad recommendation systems for life-time value optimization. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1305–1310. ACM, 2015.

Philip S Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. In *AAAI*, pages 3000–3006, 2015.