A Thesis Title

Hernan Santiago Gonzalez Toral

Department of Computer Science

University College London

hernangt12re3@gmail.com

Supervisor

Dr. Jun Wang

jun.wang.l@cs.ucl.ac.uk

This report is submitted as part requirement for the

MSc in Web Science & Big Data Analytics

at

University College London.

It is substantially the result of my own work except

where explicitly indicated in the text.

Department of Computer Science

University College London

August 20, 2016

This page is purposely left blank.

Abstract

My research is about stuff.

It begins with a study of some stuff, and then some other stuff and things.

There is a 300-word limit on your abstract.

This side is purposely left blank.

Acknowledgements

Acknowledge all the things!

This side is purposely left blank.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Overview

The volume of information available on the internet have been increasing rapidly with the explosive growth of the World Wide Web, e-Commerce and other online services, making difficult for users to find relevant products in a short period of time. To avoid this problem, many websites use recommendation systems to help customers to find items that satisfies their needs[3]. Suggestions for books on Amazon, or movies on Netflix, are real-world examples of Recommender Systems (RS) that demonstrate that the evolution of RS and the web should go hand-in-hand. As a result, the final goal of a RS is to efficiently manage an ever growing set of items and generate meaningful recommendations to a collection of users with common interests.

The design of such recommendation engines depends on the domain and the particular characteristics of the data available. While *Collaborative Filtering* systems analyze historical interactions alone, *Content-based* RS systems are based on user profile attributes; and finally, *hybrid* techniques combine both of these designs. Latest studies in CF[1] showed that combining conceptual and usage information can improve the quality of web recommendation.

Current RS typically act in a *greedy* manner by recommending items

with the highest user ratings. However, greedy recommendations are suboptimal over the long term. This approach does not actively gather information on user preferences and fails to recommend novel songs that are potentially interesting. A successful recommender system must balance the need to explore user preferences and to exploit this information for recommendation.

Therefore, the architecture of recommender systems and their evaluation on real-world problems is an active area of research. In fact, RS have found an active application area for diverse Information Retrieval, Web Mining and Machine Learning techniques[1]. Reinforcement learning[2] for example, has been suited to solve the recommendation problem. In general, once a user makes her choice based on a list of recommendations given by a RS, a new list of recommended items is then presented. Thus, the recommendation process can be thought as a sequential process, where users choices are sequential by nature (e.g. a user buy a book by the author of a recent book we liked).

Indeed, reinforcement learning has demonstrated to be a potentially effective approach in the domain and particularly good to solve the cold-start problem and in some extent, the exploration/exploitation trade-off[4]. Nevertheless, it has received relatively little attention and found only limited application.

On the other hand, most recent research[5] have focused on making a generalization of recent advances in Deep Learning[6] from Identically and Independently Distributed (i.i.d.) input to the non-i.i.d. (CF-based) input, and propose hierarchical Bayesian models that jointly performs deep representation learning for the content information and CF for estimating the ratings matrix. Experiments carried out shown that Deep Learning approaches can significantly outperform the state of the art. As a result, current research has been getting involved on using deep representational approaches that can lead to better recommendations.

Furthermore, a recent work presented by Dulac-Arnold et al. on deep reinforcement learning in large discrete action spaces[7] presented a new policy architecture that not only allows reinforcement learning methods to be applied to large-scale learning problems, and to operate efficiently with a large number of actions, but also can generalize over the action set in logarithmic time. This algorithm showed to a promising convergence of the model among different continuous environments, where a simulated recommender system was also considered.

The main objective of this project is to show that a deep reinforcement learning model is able to perform well under a recommendation system environment and generate good recommendations to users over a large set of items. After defining a simulated hybrid recommendation system environment under the Open AI gym framework, the performance of agents based on the Deep Deterministic Policy Gradient algorithm with k-nearest neighbors (kNN) and matrix factorization variants, and the Deep Q-learning with model-based acceleration are compared. [***](Missing explanation of evaluations and conclusions)

1.2 Dissertation Objectives and Structure

The remaining chapters of this report are organized as follows. Chapter 2 defines the recommender problem and introduces Reinforcement Learning. Following, it reviews some related work that has been made for modeling recommender systems using reinforcement learning approaches and deep learning. The proposed model and its implementation details are presented in Chapter 3, while the experimental evaluation and results are analyzed in Chapter 4. Finally, we present a summary of lessons learned and conclusions, and propose some future work on the field.

Chapter 2

Background

2.1 Recommender Systems

In recent years, there has been growing focus on the study of Recommender Systems (RSs), where different techniques have been published in order to to address the problem of information overload on the Internet. Consequently, the evolution of RS and the web usually go hand-in hand. Resnick and Varian[3] defined RS as systems that help users limit their search by supplying a list of items that might interest them. Such systems has found an active application area for diverse Information Retrieval, Web Mining and Machine Learning techniques that help to solve typical recommendation problems. Additionally, different RS mechanisms have been categorized depending on the way they analyze the data sources to find affinities between users and items.

The most general setting in which recommender systems are studied is presented in Figure 2.1. Having a rating matrix of n users and m items representing the current user preferences, the task of a RS is to predict all missing ratings $r_{a,i}$ for the active user a, and then recommend the item(s) with the highest rate[8]. However the user ratings matrix is typically sparse, as most users do not rate most items. Different approaches to solve this task can be categorized into the following types.

		Items					
	1	2	...	i	...	m
1	5	3		2		1
2	3	4	3	4		
Users :		2			3	
u	5			1		5
:		4	2			1
n	4		1	1	5	
a	3	5		?	1	

Figure 2.1: Typical user ratings matrix, where each cell $r_{u,i}$ corresponds to the rating of user u for item i.

2.1.1 Collaborative Filtering

In Collaborative Filtering (CF) systems, items are recommended by exploiting the similarities amongst several users based on the feedback of previously consumed items. Usually, databases that fit into a CF approach, store the past users interactions in the form of explicit ratings (e.g., 1 to 5 range), or implicit ratings (e.g. a song played by a user, or an item bought by her). In general, CF methods are subdivided into: memory-based and model-based approaches.

- **Memory-based CF:** items are recommended using a weighted combination of ratings in a subset of users that are chosen based on their rating similarity to the target user. The most commonly used measure is the Pearson correlation coefficient[9]. Alternatively, the similarity between the ratings of two users (represented as a vector in an m-dimensional space) can be and computed using the Cosine similarity or the adjusted Cosine similarity measure which overcomes the issue of users with different rating behavior schemes[10]. Previous studies found that correlation[11] and adjusted cosine similarity[10] performs slightly better.

  Memory-based CF methods have been extended and improved over the years. Linden, Smith, and York[12] proposed an *Item-based* (or item-item) CF method to overcome the problem of dimensionality found when conventional memory-based algorithms cannot scale well when finding similarities between millions of users and items. This approach, which matches a user's rated items using Pearson

correlation, lead to faster online systems and improved recommendations.

- **Model-based CF:** provides recommendations by using statistical models for predicting user ratings. The most widely used models are Bayesian classifiers, neural networks, fuzzy systems, genetic algorithms, latent features and matrix factorization[1]. For instance, CF methods can be turned into a classification problem, where a classifier is built for each active user representing items as features over users and available ratings as labels. However, latent factor and matrix factorization models have emerged as a state-of-the-art methodology in this class of techniques[13].

2.1.2 Content-based RS

Content-based (CB) RS (compared to pure CF that only utilizes the user rating matrix) tries to make a better personalized recommendation by exploiting the knowledge about a user (e.g. demographic information), or the properties of items (e.g. genre of a movie). Several approaches have treated this problem as an information retrieval (IR) task, where the content associated with the user's preferences is treated as a query, and the unrated items are scored with relevance/similarity to this query[14]. Alternatively, CB-RS has also been treated as a classification task using algorithms such as k-Nearest Neighbors (k-NN), decision trees, and neural networks[15]

2.1.3 Hybrid approaches

In order to leverage the strengths of both CB-RS and CF methods, several hybrid approaches have been proposed. For instance, Melville et al. presented in [16] a general framework for content-boosted collaborative filtering, where content-based predictions are applied to convert a sparse user ratings matrix into a full ratings matrix, and then a CF method is used to provide recommendations. In essence, this approach

has been shown to perform better than pure CF, pure CB-RS, and a linear combination of the two.

2.1.4 The cold-start problem

The cold-start problem is a common issue that happens in recommender systems when it is not possible to make reliable recommendations due to an initial lack of ratings. There are three kinds of known cold-start problems: new community, new item and new user[1]. However, the latter represents one of the greatest difficulties faced by an RS in operation. Since new users have not yet provided any rating in the RS, they cannot receive any personalized recommendations when using memory-based CF. Usually, when the users enter their firsts ratings they expect the RS to offer them personalized recommendations, but there are not enough ratings yet to be able to make reliable predictions. As a consequence, new users feel that the RS does not offer the service they expected.

This problem is often faced using hybrid approaches such as CF-CB RS, CF-demographic based RS, or CF-social based RS[1]. However, these methods can be combined with clustering techniques over items in order to improve the prediction quality.

2.1.5 Trends in Recommendation Systems

Latest studies in CF showed that combining conceptual (explicit) and usage (implicit) information can improve the quality of web recommendations. Bobadilla et al.[1] presented a taxonomy for RS which unifies the current recommender methods and algorithms that can be applied to incorporate memory-based, social and content-based information into their CF system depending on the type of information available. The taxonomy depicted in Figure 2.2, also detail at its higher levels the current evaluation methods for RS in terms of quality measures, diversity and novelty.

Moreover, Bobadilla et al. argue that the most widely used algo-

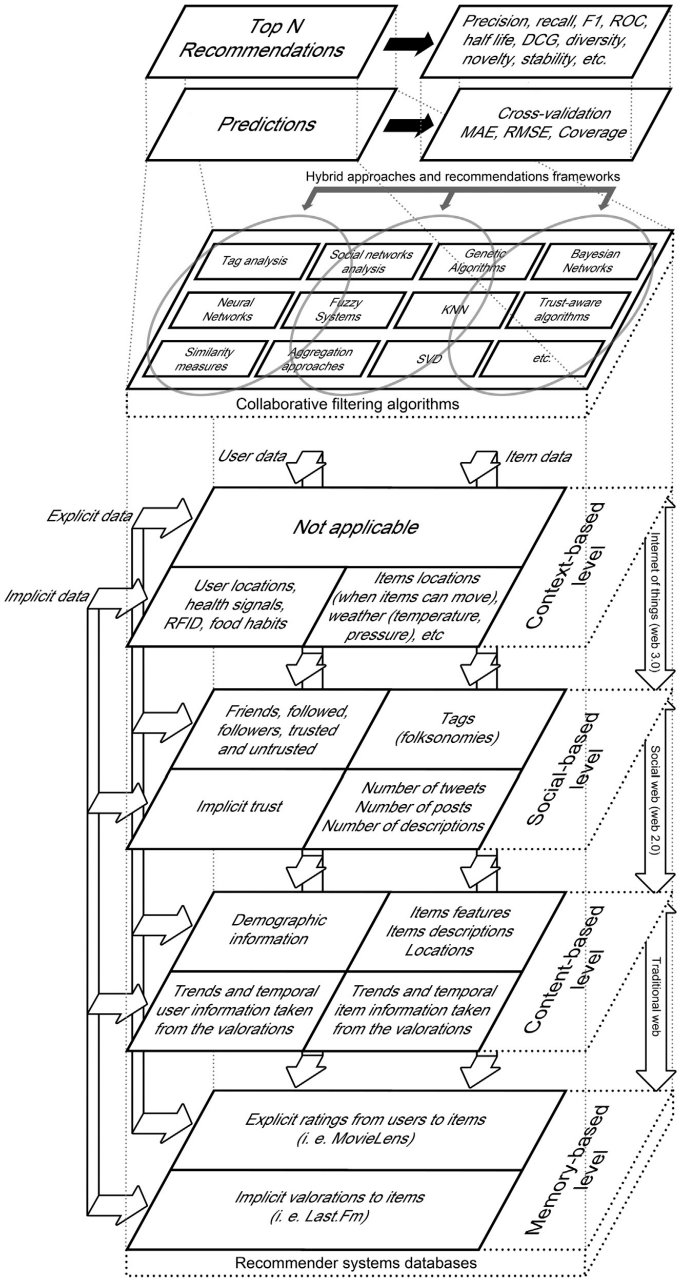

Figure 2.2: Recommender System taxonomy. Source: Recommender systems survey[1]

rithm for CF is the kNN. In general, kNN generates recommendations by executing the following tasks: (1) determine k users neighbors; (2) implement an aggregation approach with the ratings for the neighborhood in items not rated by a; and (3) select the top N recommendations from predictions obtained in step 2.

On the other hand, current research, like [5], made a generalization of recent advances in Deep Learning [6] of the Identically and Independently Distributed (i.i.d.) input and applied it to the non-i.i.d. (e.g. CF-based) input by presenting hierarchical Bayesian models that jointly performs deep representation learning for the content information and CF for the ratings matrix. Additionally, Wu Y. et al. in [17] proposed a Collaborative Deep Auto Encoders (CDAE) model which formulates the top-N recommendation problem using Denoising Auto-Encoders to learn the latent factors from corrupted inputs. Experiments carried out over different domains have shown that the two deep learning approaches can significantly outperform the state of the art. Therefore, recent research has started to focused on defining deep representational approaches that can lead to better recommendations.

2.1.6 A sequential nature of the recommender process

Recommender systems usually work in a sequential manner: the RS suggest items to the user who can then accept one of the recommendations. At the next stage a new list of recommended items is calculated based on user feedback and then presented to the user. This sequential nature allows the reformulation of the recommendation process as a sequential optimization process where optimal recommendations do not depend only on the previous items purchased, but also in the order in which those items were purchased. Zimdars et al. [18] suggested the use of a k-order Markov chain model (with k=3) to represent this behavior by dividing a sequence of transactions $X_1, ..., X_T$ into cases $(X_{t-k}, ..., X_{t-1}, X_t)$ for $t = 1, ..., T$. and then, build a model to predict the item at the time step t given the other columns. Thus, Recommender Systems can be modeled using reinforcement learning approaches.

2.2 Reinforcement Learning

Reinforcement learning (RL) [19] is a family of machine learning algorithms that optimize sequential decision making processes based on scalar evaluations or rewards. Similarly to an n-armed bandit model[20], RL considers the problem as a goal-directed agent interacting with an uncertain environment, where the main objective is to perform actions that maximize the expected sum of future reward for each state in the long term. The most important feature distinguishing RL from other types of learning is that it uses training information that evaluates the actions taken rather than instructs by giving correct actions (like supervised learning algorithms). In other words, an agent must be able to learn from its own experience, like a trial-and-error search.

RL uses a formal framework which defines the continuous interaction in terms of states, actions, and rewards, between the *agent* and an *environment*[2]. At each time step t, the agent receives some state representation of the environment $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of possible states. Then it selects an action $a_t \in \mathcal{A}(s_t)$, where $\mathcal{A}(s_t)$ is the set of actions available in the observed state. One time step later, the agent receives a numerical reward $r_{t+1} \in \mathcal{R}$, and the environment turns into a new state s_{t+1}. Figure **??** shows the whole interaction in an agent-environment interface.

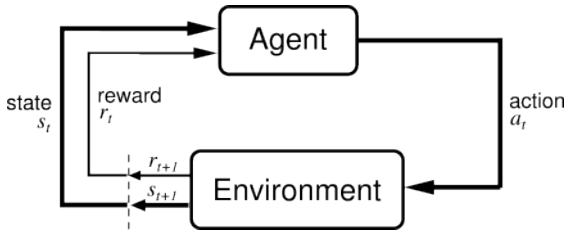

Figure 2.3: The Agent-Environment interface in a Reinforcement Learning problem. Source: Reinforcement Learning: a Survey[2]

This framework is intended to be a simple way of representing four essential features of the artificial intelligence problem: (1) a *policy* (commonly stochastic) defines a mapping from the perceived states of the

environment to actions to be taken when in those states; (2) a *reward* function maps each state-action pair to a single number that represents how good or bad the new state is for the environment; (3) a *value* function specifies the total amount of reward an agent can expect to accumulate over the future and starting from a given state. This function is considered the most important as it is used by the agent during decision-making and planning; and optionally (4) a model that mimics the behavior of the environment (transitions and rewards) and provides a way of deciding which action to perform considering possible future situations before they are actually experienced.

In general, agents are categorized depending on how the reinforcement learning problem needs to be modeled. *Value-based* and *Policy-based* agents are solely based on the value and policy functions respectively. *Actor-critic* agents use both policy and value functions; *Model-free* agents use policy and/or value function but no model; and *Model-based* agents have all properties mentioned above.

On the other hand, The *return* R_t is defined as the sum of the discounted future rewards over an episode of T time steps that an agent actually seeks to maximize:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

where $\gamma, 0 \leq \gamma \leq 1$ is a discount factor that determines the present value of future rewards

2.2.1 Markov Decision Processes

A Markov Decision Process (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[4]. It is defined by a tuple $\langle S, A, R, Pr \rangle$ representing a Markov Chain with values and decisions that follows the Markov property: *a state*

S is Markov if and only if $\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ...S_t]$*. In other words, in a MDP the future is independent of the past given the present.* Therefore, if a reinforcement learning task satisfies the Markov property and the state and action spaces are finite, then it can be modeled as a finite Markov Decision Process.

A particular finite MDP is defined by its state and action sets, a reward function R (equation 2.2) that assigns a real value to each state/action pair, and a state-transition function Pr (equation 2.3) which provides the probability of transitioning between every pair of states given each action. Altogether, these quantities completely specify the most important aspects of the dynamics of a finite MDP.

$$\mathcal{R}^a_{ss'} = \mathbb{E}r_{t+1}|s_t = s, a_t = a, s_{t+1} = s' \tag{2.2}$$

$$\mathcal{P}^a_{ss'} = Prs_{t+1} = s'|s_t = s, a_t = a \tag{2.3}$$

Therefore, the decision-maker's goal is to find an optimal policy π, such that at each stage of the decision process, the agent needs only to obtain the current state s and execute the action $a = \pi(s)$. Various exact and approximate algorithms have been proposed for estimating the optimal policy $\pi*$, such as policy iteration, value iteration, Monte-Carlo learning, temporal difference, Q-learning, SARSA, etc[19][2].

2.2.2 A standard model-free Reinforcement Learning Setup

A standard RL setup consists of an agent interacting with an environment E in discrete time steps. At each time step t, the agent receives an observation s_t, takes an action a_t and receives a reward r_t. Additionally, the environment E may be stochastic so it can be modeled as an MDP with a state space $\mathcal{S}$, action space $\mathcal{A}$, an initial state distribution $\rho(s_1)$, transition dynamics $\rho(s_{t+1}|s_t, a_t)$, and reward function $r(s_t, a_t)$. On the other hand, the agent's behavior is defined by a policy π, which maps

states to a probability distribution over the actions $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$. Finally, the return from a state is defined as the sum of the discounted future reward $R_t = \sum_{i=t}^{T} \gamma^{(i?t)} r(s_i, a_i)$. As the return depends on the actions chosen and therefore on π, it may be stochastic.

The goal in reinforcement learning is to learn a policy which maximizes the expected return from a start distribution $J = \mathbb{E}_{r_i, s_i \sim E, a_i \sim \pi}[R_1]$. The action-value function is used in many RL algorithms and describes the expected return after taking an action a_t in state s_t and thereafter following policy π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i>t} \sim \pi}[R_t | s_t, a_t] \tag{2.4}$$

Many approaches in RL opt to represent the action-value function as a recursive relationship using the Bellman equation (eq. 2.5). Nevertheless, If the target policy is deterministic we can describe it as a function $\mu : \mathcal{S} \leftarrow \mathcal{A}$ and avoid the inner expectation as shown in equation 2.6. As the expectation depends only on the environment, it is possible to learn Q^{μ} off-policy, using transitions which are generated from a different stochastic behavior policy μ.

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma \mathbb{E}_{a_{t+1} \sim \pi}[Q^{\pi}(s_{t+1}, a_{t+1})]] \tag{2.5}$$

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu_{t+1})] \tag{2.6}$$

Q-learning[21], is a commonly used off-policy algorithm which uses a *greedy* policy $\mu(s) = argmax_a Q(s, a)$ in order to estimate the action that gives the maximum reward. As the algorithm considers function approximators parameterized by θ^Q, it can be used as an optimizer that minimize the loss:

$$L(\theta^Q) = \mathbb{E}_{s_t \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[(Q(s_t, a_t | \theta_Q) - y_t)^2] \tag{2.7}$$

where $y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q)$.

2.2.3 Exploration/Exploitation Dilemma

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation[19][2]. To obtain a lot of reward, a reinforcement learning agent must prefer actions that has already tried in the past and found to be effective in terms of reward. In order to discover such patterns, it needs to try actions that it has not been selected before. so the agent has to be able to exploit actions that are already known as effective, but it also needs to explore in order to make better action selections in the future.

Naïve approaches use a greedy policy $\pi(s) = argmax_a Q(s, a)$ to get the action with maximum reward as a pure exploitative model. However, an ϵ-greedy policy can be considered instead to allow exploration in the environment and improve the estimation of the non-greedy action values. Therefore, the agent will select a random action with a probability ϵ, and will act greedily with probability $1 - \epsilon$. The advantage of ϵ-greedy policy over greedy policy is that the former continuously explore and improve the chances of recognizing possible actions that can lead to better rewards.

Nevertheless, this has still been an issue to solve in some reinforcement learning tasks, so research has continuously focused on trying different techniques to solve the exploration/exploitation trade-off that can lead to the discovery of better policies[19].

2.3 Recommender Systems and Reinforcement Learning

Although reinforcement learning seems to have great potential for improving the recommendation problem, it has received relatively little attention and found only limited application. The first experiments on using

reinforcement learning techniques were focused on developing recommendation systems for web content using implicit data from server files which stored the navigational logs of the users throughout their contents.

Ten Hagen et al. [22] considered RL approach with an exploration/exploitation trade-off for discovering unknown areas and automatically improve their recommendation policy to helping users navigate through an adaptive web site. They stated that a model-free algorithm like Q-Learning can become infeasible if the number of actions are relatively high and showed that a greedy policy can indeed lead to a suboptimal recommendation model as the use of such kind of policy function, where the b̈estäctions (e.g with higher probability) are taken, does not always improve the policy. They finally demonstrated that this problem is solved by starting the algorithm with a lot of exploration and gradually reduce the parameter ϵ as the policy is getting closer to an optimum. Results concluded that a recommender system without exploration potentially can get trapped into a local maxima.

Rojanavasu et al. in [23] presented a general framework for web recommendation that learns directly from customer's past behavior by applying an RL process based on the SARSA method and a ϵ-greedy policy. The system was composed of two models: a global model to keep track of customers trends as a whole, and a local model to record the user's individual browsing history. In general, the model treated pages as states of the system and links within a page as actions. To predict the next state, it used a ranking system that is also separated into 2 parts. i) a global ranking system using the data from a Q_{global}-matrix obtained from the action-value function and an ϵ-greedy policy that gave the chance for rank new items that have few clicks but may match a user's interest; and ii) a local ranking Q_{local} using an inverse ϵ-greedy policy. The system then finds the final total reward using $Q_{total} = Q_{local} + wQ_{global}$ where $w \in (0 - 1]$ is a weight hyper parameter of the model.

Overall, the system provided customers the chance to explore other products than the ones they already visited. Experimental results showed that a value of $\epsilon = 0.2$ preserves the balance between exploration and exploitation, meanwhile If $\epsilon < 0.2$ the system will gave small chances to users to explore new items, or otherwise may include products that do not match to the customer's interest ($\epsilon > 0.2$). On the other hand, results also prove that even if the purpose of Q_{local} was to discover new products, it appeared to be less effective than Q_{global}.

Shani et al. in [4] argued that it is more appropriate to formulate the problem of generating recommendations as a sequential optimization problem so they described a novel approach for a commercial web site based on MDPs together with a predictive model. The MDP-based recommender system took into account the expected utility and the long-time effect of a particular recommendation. Then it suggested items whose immediate reward is lower, but leads to more *profitable* rewards in the future. However, the benefits of this model are offset by the fact that the model parameters are unknown, randomly initialized and that they take considerable time to converge. So, they defined a strong initial model-based for collaborative filtering, which solves quickly, and that does not consume too much memory.

Before implementing the recommender, they initialize a predictive model of user behavior using data extracted from the web site, and then used it to provide the initial parameters for the MDP. They proposed to use maximum-likelihood estimation with three major enhancements: skipping, clustering, mixture modeling, so the predictive model can be thought of as a first-order Markov chain (MC) of user dynamics in which states correspond to sequences of k events (in this case: previous selections) representing relevant information about the users interaction. Then, the transition function described the probability that a user ,whose k recent selections were $x_1, ..., x_k$ will select the item x' next.

Authors experimented with small values of k ($k \in [1-5]$) in order to overcome data sparsity and MDP solution complexity issues. Moreover, enhancements on the maximum-likelihood n-gram formulation showed to be useful to find a good estimate of the transition function for the initial predictive model without suffering the problem of data sparsity and bad performance presented on other model variations.

On the other hand, the components of the reinforcement learning model were defined as follows: a) the states are the k-tuples of items purchased; b) actions correspond to a recommendation of an item; c) rewards depends only on the last item defining the current state, where its net profit was used; and d) the transition function as the stochastic element of the model that estimates the user's actual choice.

In order to solve the MDP problem, a policy iteration algorithm was used as the defined state space presented certain characteristics that lead to fast convergence. The transition function was carefully initialized in order to be fairly accurate when the system was first deployed to avoid the cold-start problem,causing states that are infrequently observed to be updated faster than already observed states. Moreover, when the first transition to a state is observed, its probability is initialized to 0.9. In this way, the system balances the need to explore unobserved items in order to improve its model by recommending non-optimal items occasionally until getting their actual counts. Finally, to update the model, the system used an off-line approach which keeps track of the recommendations and the user selections and build a new model at fixed time intervals (e.g. once a week).

Experiments demonstrated that a deployed system using three mixture components (combined using an equal weight), with history length $k \in [1,3]$ was able to generate 28% higher average user reward compared to pure MC model, while in terms of computational costs, the MDP-based model was built quickly and provided fastest recommendations at

the price of more memory use. All in all, the off-line predictive model approach provided an adequate initial performance that overcomes the cold-start problem, however, authors avoid using some form of reinforcement learning technique as they argued that at that time, its implementation requires many calls and computations by a recommender system online, which lead to slower responses and undesirable results for the web site owner.

A machine learning perspective, introduced by Taghipour et al. in [24], used an off-line reinforcement learning model to solve the recommendation problem using the Q-Learning algorithm while employing concepts and techniques commonly applied in the web usage mining domain. They argued that this method is appropriate for the nature of web page recommendation problem as it provides a mechanism which is constantly learning, does not need periodic updates, can be easily adapted to changes in the website structure and new trends in users behavior.

The RL problem formulation was considered as as a competition between different recommender systems to gather more points, together with a stochastic model with the following properties:

- *states:* showing the history of pages visited by the user so far. For this case, a notion of N-Grams was adopted and a sliding window of size w was set to limit the page visit sequences to a constant number and avoid large state spaces.

- *actions:* consisting of a single page recommendation at each state.

- *policy:* rewards actions positively if it recommends a page that will be visited in one of the consequent states

- *reward* function: defined as $r(s,a) += reward(Dist(Rs', p), Time(p_w^v))$, where $Dist(Rs', p)$ is the distance of page p from the end of the recommended pages list to state s', and $Time(p_w^v)$ indicates the time user has spent on the last page of the state. Finally, the

$reward(Dist, Time)$ function was defined as a linear combination of both values: $reward(Dist, Time) = \alpha \times dist + \beta \times Time$ with $\alpha + \beta = 1$.

The proposed off-line RL method used a simplified formulation of the recommendation problem but it obtained much better results compared with two baselines methods: association rules and item-based collaborative filtering (with probabilistic similarity measure). As the coverage increase, naturally accuracy decrease in all systems, but the RL approach outperformed the other two systems displaying a lower rate in which its accuracy decreased. Authors concluded that the algorithm is a good candidate for solving the recommendation problem as it does not rely on any previous assumptions regarding the probability distribution of visiting a page after having visited a sequence of pages, and that the nature of the problem matches perfectly with the notion of delayed reward (known as temporal difference). Additionally, they suggested that in order to produce better results it could be possible to use a more complicated formulation of the reward function such as a neural networks, rather than a linear combination of factors.

Later in [25] they exploited the previous RL framework and present a hybrid web recommendation method enriched with semantic knowledge about the usage behavior and thus obtain a more generalized solution regarding to the usage data it has. The new system used the incremental Document Conceptual Clustering[26] algorithm to map pages to higher level concepts, and thus exploit the hierarchical and conceptual document clustering to provide a semantic relationship between them in combination with the usage data in a user session.

Therefore, new states consist of a sequence of concepts visited by the user, while actions are recommendation of pages that belong to a specific concept. This definition resulted in a much smaller state-action space as the size of the state space is now dependent on the number of distinct page clusters. Finally, the new reward function takes into account the

content similarity of the recommended and visited pages along with the usage-based reward defined in the previous approach.

The modified algorithm do not make predictions based on weak usage patterns as the new states represented a generalized view of many single visit sequences. Furthermore, evaluation results showed the flexibility of the new RL approach to incorporate different sources of information in order to improve the quality of recommendations.

A model-based RL agent for music playlist recommendation proposed by Liebman et al. [27] modeled the preferences of both songs and song transitions as MDPs, and demonstrated to be potentially effective in the domain and particularly good to solve the cold-start problem as it is able to generate personalized song sequences within a single listening session and with no prior knowledge of the new user's preferences.

The adaptive playlist generation problem was defined as an episodic MDP $\langle S, A, P, R, T \rangle$ with the following components:

- a state space S composed by the ordered sequence of songs played, $S = \{(a_1, a_2, ..., a_i) | 1 \leq i \leq k; \forall j \leq i, a_j \in \mathcal{M}\}$
- an actions space A formed by all possible candidates for being the next song to play, $a_k \in A$ which means $A = \mathcal{M}$
- a deterministic transition function $P(s, a) = s'$ which indicates the existing transitions from state s to s' by taking action a
- an utility function R(s, a) derived from hearing song a when in state s
- $T = \{(a_1, a_2, ..., m_k)\}$: the set of playlists of length k.

In order to find an optimal policy $\pi*$ (and thus obtain the most pleasing sequence of songs to the listener), a model-based approach was chosen arguing that even though model-free approaches learn the value of

taking an action a from state s directly, it requires a lot of data to converge, and it is considerably scarce in the domain music recommendation (as well as in other kind of applications). On the other hand, model-based formulations often requires a lot of computation to find an approximate solution to the MDP, but the trade-off of computational expense for data efficiency makes the model-based approach a good option for this problem.

Since P is deterministic, only the listener's reward function R was required to be modeled. Therefore, the reward function defined as $R(s,a) = R_s(a) + R_t(s,a)$ represented the sum of two distinct components: (1) the listener's preference over songs, $R_s : A \to \mathcal{R}$, and (2) the preference over transitions from songs played to a new song, $R_t : S \times A \to \mathcal{R}$.

In essence, the model represents each song as a compacted vector of spectral auditory descriptors that capture meaningful differences in user's preferences, so the system is able to leverage knowledge using only a few transition examples to plan a future sequence of songs. On the other hand, the agent architecture was composed by a module for learning the listener parameters during initialization and learning on the fly processes; and an additional module for planning a sequence of songs in the playlist.

The initialization is divided in two parts: (1) initialization of song preferences polls the listener for her k_s favorite songs in the database and updates the user's preference vector using the feature descriptors of each of the selected items; and 2) initialization of the transition preferences by presenting different possible transitions that encapsulate the variety in the dataset, and directly asking which of a possible set of options the listener would prefer. Updates are carried out in the same manner as the initialization of song preferences.

After initialization, the agent begins to play songs for the listener, waiting for her feedback (reward), and updating the user's preferences

accordingly. This learning of the fly procedure is perceived as a temporal-difference update with an attenuating learning rate that balances the trust between the previous history of observations and the newly obtained signal. Then, a Monte Carlo Tree Search (MCTS) heuristic[28] is used for planning. The iterative process chooses a subset of 50% of the songs in the database with highest R_s score, and then at each point, it simulates a trajectory of future songs selected at random and calculate the expected payoff of the song trajectory using R_s and R_t. The process ends when it finds the trajectory which yields to the highest expected payoff, and finally, the first item in the trajectory is selected to be the next song to recommend.

Additionally, to mitigate the problem of high complexity during re-planning under large song spaces, the agent used the canonical K-means algorithm for clustering songs according to song types and reduce the search complexity drastically.

An evaluation of the cumulative reward distribution of the proposed approach showed that overall, the DJ-MC agent outperforms two baselines models (a random agent and a greedy agent), while in terms of transition reward preferences, the new system presented a small but significant boost in performance compared to a model based only on reasoning about song preferences. All in all, this approach demonstrated to be a good improvement as a reinforcement learning model for music recommendation, as it enables learning from relatively few examples, and provides quality on recommendation sequences.

2.4 Deep Reinforcement Learning

So far, the RL agents presented above have shown to achieve some success under the recommendation domain, nevertheless, their performance has been conditioned to the quality of the hand-crafted feature engineering made, as well as the custom definition of value functions or policy

representations. Moreover, to use RL successfully in situations approaching to real-world complexity, agents usually need to derive efficient representations of the environment from high-dimensional sensory inputs, and use these to generalize past experiences to new situations.

Recent advances in deep learning have made it possible to extract high-level features from raw sensory inputs. Even if it seems that this approach could fit in a RL task, its applicability to the RL domain may present several challenges from a deep learning perspective, For instance, successful deep learning applications have required large amounts of hand-labelled training data to obtain good predictions results. Additionally, most deep learning algorithms assume the data samples to be i.i.d., while in RL an agent typically encounters with non-i.i.d. sequences of highly correlated states.

On the other hand, agents must be able to learn from a scalar reward signal that is frequently sparse, noisy and delayed, while the RL data distribution changes as the algorithm learns new behavior. However, thanks to the increasing success of adapting different deep learning techniques to different domains, deep reinforcement learning model have started to been proposed and started to successfully learn control policies directly from high-dimensional sensory input.

Playing atari with deep reinforcement learning and **Human-level control through deep reinforcement learning** presented the first deep learning approach model which uses an end-to-end reinforcement learning approach to learn control policies directly from a high-dimensional raw input space. The model, applied to a range of Atari 2600 games, consists of convolutional neural network and an experience replay mechanism[***] to alleviate the problems of correlated data and non-stationary distributions in typical RL problems. Moreover, the model's architecture uses only the state representation as input, and a separate output unit for each possible action (target network), corre-

sponding to the predicted Q-values of each individual action for the input state. This configuration allows the computation of for all possible actions reward in a given state with only a single forward pass. Finally, the resulting neural network is trained with a variant of Q-learning which learns from raw pixels input and the underlying environment properties, and outputs a value function estimating the future rewards.

The formulation models the πenvironment $\mathcal{E}$ as a Markov Decision Process composed by a state space $\mathcal{S}$, an action space $\mathcal{A} = \{1...K\}$, a scalar reward function $r(s_t, a_t)$ and a transition dynamics $\rho(s_{t+1}|s_t, a_t)$. Moreover, the algorithm follows the basic idea behind many reinforcement learning tasks: given the agent's behavior defined by a policy π, they estimate the action-value function by using an approximator function of the Bellman equation (presented in eq. 2.5) as an iterative update. A Q-network function approximator with weights θ is then trained by minimising the sequence of loss functions $L_i(\theta_i)$ defined in equation 2.8

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}[(y_i - Q(s, a; \theta_i))^2] \tag{2.8}$$

where $y_i = \mathbb{E}_{s'\sim\mathcal{E}}[r + \gamma max_{a'} Q(s', a'; \theta_{i-1})|s, a]$ is the target for iteration i and $\rho(s, a)$ is a probability distribution over sequences of states s and actions a known as behavior.

Rather than computing the full expectations in the gradient of the loss function, they considered stochastic gradient mini-batch updates of the weight parameters, and a learning algorithm which is: a)*model-free*: replaces the expectations by directly using uniform samples from the behaviour distribution ρ, and b)*off-policy*: it follows an ϵ-greedy strategy of the behaviour distribution that ensures an adequate exploration of the state space.

Evaluation results of the proposed algorithm showed that this method is able to learn how the value function evolves for a reasonably complex sequence of events. Furthermore, a performance evalua-

tion compared this approach with the Sarsa[***] and Contingency[***] methods demonstrating that the deep reinforcement learning algorithm outperforms other methods which need to incorporate significant (hand-crafted) prior knowledge about the visual problem to get considerable performance. Finally, the proposed method also achieved better performance than an expert human player in 29 out of 49 games.

The main advantages of using this deep learning approach with an experience replay of size N are: i) each step of experience is potentially used in many weight updates, allowing greater data efficiency; ii) randomizing the samples breaks the correlation in a sequential event and therefore reduces the variance of the updates; iii) the experience replay mechanism allows the behavior distribution to be averaged over many of its previous states, smoothing out learning and avoiding instability or divergence in the parameters; and iv) the method showed that can be applied to a series of different reinforcement learning tasks with no adjustment of its architecture or learning algorithm.

However, this model is only able to handle discrete low-dimensional action spaces and cannot be directly applied to tasks under the continuous domain as the iterative process of finding the action that maximizes the action-value function will become intractable. Additionally, a naive discretization of the action space is not a good solution as it will throw away information about the structure of the action domain.

Continuous control with deep reinforcement learning adapted the ideas presented above to the continuous space and action domain and provided an actor-critic, model-free algorithm based on the Deterministic Policy Gradient[***] (**Deterministic Policy Gradient Algorithms**) that is able to find policies end-to-end whose performance is competitive compared to previous approaches based on a planning algorithm with access to the dynamics of the domain. To do so, they used the similar features of DQN (network architecture, experience replay memory, tar-

get outputs, etc), along with batch normalization[***], to overcome the internal covariate shift problem by normalizing the network layer inputs during mini-batches.

The Deep Deterministic Policy Gradient (DDPG) implementation uses an actor-critic approach based on the DPG algorithm which mainly maintains two functions: 1)the actor function $\mu(s|\theta_\mu)$ that obtains the current policy by deterministically mapping states to specific actions; and 2)the critic function $Q(s,a)$ learned by applying the Bellman equation. DPG then updates the actor by applying the chain rule to the expected return from a start distribution $J = \mathbb{E}_{r_i, si\sim E, a_i\sim\pi}[R_1]$ and with respect to the actor parameters:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}[\nabla_{\theta^\mu} Q(s,a|\theta_Q)|_{s=s_t,a=\mu(s_t|\theta_\mu)}] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}[\nabla_a Q(s,a|\theta_Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s=s_t}]\end{aligned} \tag{2.9}$$

Even if convergence is no longer guaranteed due to the non-linearity of the proposed approximation function, it produces a good generalization on large state spaces.

On the other hand, the DDPG algorithm also implement certain improvements that overcome the problems and challenges presented when using Deep Q-learning: instability of the learning process, generalization of the solution to multiple environments, and handling a mechanism to improve the exploration-exploitation trade-off. First, in order to solve the relatively unstable problem of learning the action-value function and optimize the gradient results significantly, target networks were modified to use soft target updates, rather than directly copying the weights. Then, the underlying weights are slowly updated by exponentially moving their average values using $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ with $\tau \ll 1$.

A second improvement solved the difficulty of finding hyperparameters that allow generalization across environments when the low dimensional feature vector observations have different physical units and

the ranges. This issue is addressed by adapting a batch normalization technique proposed by [***], that minimizes the covariance shift during training by normalizing each dimension across samples in a mini-batch to have unit mean and variance. As a result, DDPG ensures that each layer receives whitened input in favor of learning effectively across many different tasks with differing types of units.

The third improvement tries to overcome the common challenge on how to manage exploration in reinforcement learning tasks with continuous actions spaces. The exploration-explotaition trade-off was managed independently from the learning algorithm by adding noise sampled from a noise process $\mathcal{N}$ to the actor policy μ and thus create an exploration policy $\mu'(s_t) = \mu(s_t|\theta_t^\mu) + \mathcal{N}$. Hence, any sampling process $\mathcal{N}$ can be plugged in to suit exploration on specific environments.

To evaluate the algorithm under physical environments with different levels of difficulty, they used the the Ornstein-Uhlenbeck[***] sampling process for adding exploration to the policy. The results demonstrated that even when harder tasks obtain poor Q estimates, DDPG is able to learn good policies across a variety of domains with continuous action spaces and using both low-dimensional feature vector and high-dimensional pixel inputs. Additionally, all the experiments were solved using fewer steps of experience than the DQN algorithm, showing that DDPG may be able to solve even more difficult problems. However, it requires of a large number of training episodes to find solutions and only works in environments with a small set of actions, so more robust model-free approaches should be proposed in order to tackle these limitation.

Later, **Deep Reinforcement Learning in Large Discrete Action Spaces** presented a new policy architecture, under the same actor-critic framework used in [***], that not only allows reinforcement learning methods to be applied to large-scale learning problems, and to operate efficiently with a large number of actions, but also can generalize

over the action set in logarithmic time. The policy is then trained and optimized using DDPG. As a result, they obtained a more efficient algorithm that makes both learning and acting tractable in time and allows value-based policies to use the action features to reason about previously unseen actions.

The *Wolpertinger* architecture introduces two characteristics that previous approaches did not provide: 1)leverages prior information about the actions and embed them in a continuous space upon which the actor can generalize using a smooth function approximator, and 2)after the policy produces a continuous action within this space, it then uses an approximate nearest neighbor search to find the set of closest discrete actions in logarithmic time to finally select the highest valued action relative to a cost function. Overall, the algorithm is now able to handle large-scale reinforcement learning tasks by adding a sub-linear complexity relative to the action space and the ability to generalize over actions.

Moreover, the algorithm also avoids the heavy cost of evaluating all actions mainly by defining an efficient action-generating actor, and then using the critic to refine the actor?s choices for the full policy. The underlying function approximators are built using multi-layer neural networks and the resulting policy is trained with DDPG.

The action generation function $f_{\theta^\pi}(s) = \hat{a}$ provides a proto-action in $\mathbb{R}^n$ for a given state s. $\hat{a}$ is then mapped to the discrete action set $\mathcal{A}$ by applying the k-nearest-neighbor[1] function from equation 2.10, which returns the k closest actions in $\mathcal{A}$ by computing the L_2 distance between them. Even if g_k has the same complexity as the *argmax* in the Q action-value function, each step of evaluation of the L_2 distance is faster than a full value-function evaluation as the lookup is performed in logarithmic

[1]The size of the generated action set k is task specific, and allows for an explicit trade-off between policy quality and speed

time.

$$g_k(\hat{\mathbf{a}}) = \arg\min_{a \in \mathcal{A}}^{k} |\mathbf{a} - \hat{\mathbf{a}}|_2 \tag{2.10}$$

Following the selection of the of k closest actions to â, the choice of the actual action to be applied to the environment is set according to the highest-scoring action obtained according Q_{θ^Q} defined in equation 2.11. Consequently, the full Wolpertinger policy π_θ with parameter θ, representing both the parameters of the action generation element θ_π and of the critic θ_Q, makes the algorithm significantly more robust to imperfections in the choice of action representation.

$$\pi_\theta(s) = \arg\max_{a \in g_k \circ f_{\theta^\pi}(s)} Q_{\theta^Q}(s, a) \tag{2.11}$$

The goal of this approach is to perform policy iteration by alternatively performing policy evaluation on the current policy with Q-learning, and then improving upon the current policy by following the DDPG configuration introduced in [***](previous paper) over f_{θ^π}, considering that the effects of g are a deterministic aspect of the environment. In this case, the critic is trained from samples stored in the replay buffer (which are generated by the full policy $\pi_{\theta^\pi} = g \circ f_{\theta^\pi}(s)$), whereas the policy gradient $\nabla_a Q_{\theta^Q}(s, a)$ is taken at the actual output $\hat{\mathbf{a}} = f_{\theta^\pi}(s)$. Finally, the target action in que Q-update is generated by the full policy.

The agent was evaluated on three environment classes: discretized continuous control, multi-step planning, and recommender systems. Specifically, the latter used a simulated recommendation system to demonstrate how the agent would perform on a real world large action space problem. Results showed that a subset of the full set of actions is sufficient in many tasks to converge and provide significant speedups. Therefore, the Wolpertinger algorithm can scale to real-world MDPs with large number of actions, but with an still existing issue of exploration when the agent needs to learn from scratch.

Chapter 3

Models

3.1 Naïve Model using Matrix Factorization

3.1.1 Reinforcement Learning model

The model is defined as a Model-free approach which its main objective is to find the optimal policy which maximizes the total future reward

State space: is the set of items recommended to the user

Action space: are the single items recommended at each timestep

Value function: rating feedback received after giving a recommendation.

At current iteration h+1, we have gathered h observed recommendation history Dh = (vi, ti, ri)h i=1. Collect user rating rh and update p(? | Dh)

3.1.2 Evaluation

*For insights and arguments, follow **Recommender systems survey**

Chapter 4

Experiments

Chapter 5

General Conclusions

Appendix A

An Appendix About Stuff

(stuff)

Appendix B

Another Appendix About Things

(things)

Appendix C

Colophon

This is a description of the tools you used to make your thesis. It helps people make future documents, reminds you, and looks good.

(example) This document was set in the Times Roman typeface using LaTeX and BibTeX, composed with a text editor.

Bibliography

[1] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013.

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

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

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

[5] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1235–1244. ACM, 2015.

[6] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[7] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep Reinforcement Learning in Large Discrete Action Spaces. 2015.

[8] Prem Melville and Vikas Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2011.

[9] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

[10] Hilmi Yildirim and Mukkai S Krishnamoorthy. A random walk method for alleviating the sparsity problem in collaborative filtering. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 131–138. ACM, 2008.

[11] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 43–52. Morgan Kaufmann Publishers Inc., 1998.

[12] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

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

[14] Marko Balabanović and Yoav Shoham. Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72, 1997.

[15] Michael Pazzani and Daniel Billsus. Learning and revising user profiles: The identification of interesting web sites. *Machine learning*, 27(3):313–331, 1997.

[16] Prem Melville, Raymond J Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Aaai/iaai*, pages 187–192, 2002.

[17] Yao Wu, Christopher DuBois, Alice X Zheng, and Martin Ester. Collaborative denoising auto-encoders for top-n recommender systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, pages 153–162. ACM, 2016.

[18] Andrew Zimdars, David Maxwell Chickering, and Christopher Meek. Using temporal data for making recommendations. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, pages 580–588. Morgan Kaufmann Publishers Inc., 2001.

[19] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[20] Michael N Katehakis and Arthur F Veinott Jr. The multi-armed bandit problem: decomposition and computation. *Mathematics of Operations Research*, 12(2):262–268, 1987.

[21] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[22] Stephan Ten Hagen, Maarten Van Someren, and Vera Hollink. Exploration/exploitation in adaptive recommender systems. *proceedings of Eunite 2003*, 2003.

[23] Pornthep Rojanavasu, Phaitoon Srinil, and Ouen Pinngern. New recommendation system using reinforcement learning. *Special Issue of the Intl. J. Computer, the Internet and Management*, 13, 2005.

[24] Nima Taghipour, Ahmad Kardan, and Saeed Shiry Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 113–120. ACM, 2007.

[25] Nima Taghipour and Ahmad Kardan. A hybrid web recommender system based on q-learning. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1164–1168. ACM, 2008.

[26] Daniela Godoy and Analía Amandi. Modeling user interests by conceptual clustering. *Information Systems*, 31(4):247–265, 2006.

[27] Elad Liebman, Maytal Saar-Tsechansky, and Peter Stone. Dj-mc: A reinforcement-learning agent for music playlist recommendation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 591–599. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[28] Guillaume Chaslot. Monte-carlo tree search. *Maastricht: Universiteit Maastricht*, 2010.