Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Evaluating the Impact of Recommender Systems to Society

Relatore:
Chiar.mo Prof.
UGO DAL LAGO

Presentata da:
VALENTINA FERRAIOLI

Correlatore:
Chiar.ma
ASIA J. BIEGA

Sessione II
Anno Accademico 2021/2022

Abstract

Nowadays, Recommender systems play a key role in managing information overload, particularly in areas such as e-commerce, music and cinema. However, despite their good-natured goal, in recent years there has been a growing awareness of their involvement in creating unwanted effects on society, such as creating biases of popularity or filter bubble. This thesis is an attempt to investigate the role of RS and its stakeholders in creating such effects. A simulation study will be performed using EcoAgent, an RL-based multi-stakeholder recommendation system, in a simulation environment that captures key user interactions, suppliers and the recommender system in order to identify possible unhealthy scenarios for stakeholders. In particular, we focus on analyzing the document catalog to see how the diversity of topics that users have access to varies during interactions. Finally, some post-processing methods will be defined on EcoAgent, one reactive and one proactive, which allows us to manipulate the agent's behavior in order to study whether and how the topic distribution of documents is affected by content providers and by the fairness of the system.

Sommario

Negli ultimi decenni, con l'ascesa di YouTube, Amazon, Netflix e molti altri servizi web di questo tipo, i sistemi di raccomandazione hanno preso sempre più posto nella nostra vita. L'enorme volume di informazioni disponibili sul web porta al problema del sovraccarico di informazioni, che rende difficile per un decisore fare le scelte giuste. Questo diventa ovvio quando nella vita di tutti i giorni ci troviamo di fronte a una lunga lista di elementi in un negozio di shopping online; più elementi sulla lista, più difficile diventa fare una selezione. I Recommender System (rss) sono strumenti sviluppati con l'idea di aiutare gli utenti a trovare gli articoli relativi a loro, con la previsione delle loro preferenze o valutazione sugli articoli. Tuttavia, nel mondo reale, l'applicazione è un po' più complessa in quanto gli utenti non sono l'unica parte coinvolta nel processo di raccomandazione. Un altro gruppo di attori chiave, ad esempio, sono i fornitori di contenuti. Questi, infatti, manipolando il catalogo di contenuti disponibili per la raccomandazione esercitano una grande influenza sulla piattaforma e indirettamente sulla soddisfazione degli utenti. Inoltre, nonostante il loro obiettivo benevolo, negli ultimi anni c'è stata una crescente consapevolezza del loro coinvolgimento nella creazione di effetti indesiderati sulla società. Infatti, per natura del loro design, il sistema di raccomandazione è soggetto alla creazione di bias a seguito dell'interazione tra i vari componenti. È proprio la presenza di questi bias che riduce drasticamente la qualità delle raccomandazioni e mette a rischio il benessere della società.

Questa tesi è un tentativo d'indagare il ruolo di RS e dei suoi stakeholder

nella creazione di tali effetti. Sarà effettuato uno studio di simulazione utilizzando EcoAgent, un sistema di raccomandazione multi-stakeholder basato su RL, in un ambiente di simulazione che cattura le interazioni chiave tra gli utenti, i fornitori e il sistema di raccomandazione al fine di identificare possibili scenari malsani per le parti coinvolte. In particolare, ci concentreremo sull'analisi del catalogo dei documenti per vedere come la diversità degli argomenti a cui gli utenti hanno accesso varia durante le interazioni. Nello specifico, abbiamo individuato la presenza di un bias sugli argomenti disponibili, in quanto durante la simulazione un argomento viene creato con una maggiore frequenza rispetto agli altri. Abbiamo quindi implementato due tecniche di post-processing che modificano il comportamento dell'agente nei confronti dei fornitori di contenuti al fine di definire sia il ruolo dei content providers che quello della fairness del sistema nella creazione o amplificazione di questi effetti. Le tecniche utilizzate creerannno prima una piattaforma che massimizza il numero di fornitori disponibili e successivamente una che si comporta correttamente nei loro confronti.

I risultati ottenuti mostrano che fornire un ambiente sano per i fornitori di contenuti, dove tutti sono trattati in modo equo, si riflette nel mitigare il bias di popolarità trovato sugli argomenti. Infatti, ricevendo raccomandazioni, i fornitori di contenuti sono incoraggiati a creare nuovi documenti in base alle loro preferenze che includono argomenti più di nicchia, aumentando così la diversità degli argomenti proposti. Inoltre, poiché più fornitori creano documenti sugli stessi argomenti la possibilità che i documenti su un particolare argomento siano polarizzati diminuisce, consentendo all'utente di accedere a diversi punti di vista e quindi riducendo il rischio di restare intrappolato in delle echo chambers. Invece, come risultato della mitigazione dei bias di popolarità sugli argomenti, abbiamo un catalogo più diversificato tra cui scegliere gli elementi da raccomandare agli utenti. Questo ci permette di soddisfare le preferenze degli utenti di cui siamo a conoscenza, ma anche di verificare la loro posizione su quelli non ancora menzionati, aumentando così la probabilità di avere raccomandazioni *nuove* o *serendipite*.

Introduction

In recent decades, with the rise of YouTube, Amazon, Netflix and many other web services of this type, recommendation systems have taken more and more place in our lives. The huge volume of information available on the web leads to the problem of information overload, which makes it tough for a decision maker to make the right decisions. It becomes obvious when in everyday life we face a long list of items in an online shopping store; the more items on the list, the more difficult it becomes to make a selection. Recommender systems (RSs) are tools developed with the idea of helping users to find items relevant to them, through the prediction of their preferences or rating on items. Aligning the goal of a recommendation system with the user utility is the most natural thing, since users are first-hand consumers of recommendation services. However, in the real world, the application is a bit more complex and users are not the only stakeholders involved in the recommendation process. Another group of key players are content providers that have a great influence on the platform and indirectly on user satisfaction, through the manipulation of the content pool available for recommendation.

Therefore, depending on the stakeholders involved in the recommendation issue, the data used to generate a recommendation, such as online user history or preferences of people similar to them, and the technique used to generate recommendations, we can distinguish different types of RSs. Among the most important methods, we find traditional methods such as collaborative filtering, content-based or knowledge-based methods in addition to the most recently developed methods involving the use of machine learning techniques.

Worthy of particular note, lately, are the methods that make use of reinforcement learning. These are more and more frequently adopted in real life scenarios where it is necessary to model the complex interaction between several stakeholders. The problem of recommendation, in fact, is well-suited to be solved using reinforcement learning, where the recommendation system represents a decision-maker that interacts with an environment composed of users and content providers, it collects data through interaction and learns to suggest the element that maximizes the objective of all stakeholders.

However, despite their good-natured goal, in recent years there has been a growing awareness of their involvement in the creation of unwanted effects. In fact, by nature of their design, the recommendation system is subject to biases that are created during the interaction among the various components. The presence of these biases drastically lowers the quality of recommendations and leads to the creation of some undesirable effects on society. For instance, to name a few, it has been shown that they have reduced the diversity between elements consumed by a user, which intensifies the homogenization of users, resulting in the creation of filter bubbles, in which the user is only shown content similar to those he had already interacted with, resulting in intellectual isolation, political polarization, and eco chambers. Yet, these effects are hard to analyze since the items' consumption is governed by a complex interaction between the users' preferences, the content provider's intent, and the platform nature.

This thesis is an attempt to analyze these effects of RSs. To proceed with this study, we used **EcoAgent** [Zha+21], a multi-stakeholder recommendation system based on RL, which captures the dynamics between users, agent and content providers. Once the agent is trained and validated, we studied how it behaves in a simulated environment and how its behavior affects other elements of the environment, such as users, content provider, and the document catalog. In particular, we focus on analyzing the document catalog to see how the diversity of topics that users have access to varies. Once we identify the presence of a bias over the topics available on the platform, we see

how applying post-processing techniques that modify the agent's behavior towards content providers reflects in the mitigation of the bias detected.

Structure of the thesis:

- **Chapter 1** contains a theoretical introduction to RL where the main ideas and elements are described, such as the concepts of reward, policy and value function and its definition in relation to Markov's decision-making processes (MDP). Then the fundamental algorithmic techniques used to solve it are explored, considering tabular, approximated and policy gradient methods. Finally, the differences with the variant of Multi-Agent Reinforcement Learning (MARL) are explored.

- **Chapter 2** contains the definition of the context of application of recommendation systems and explore some of the most important types of recommendation systems: collaborative filtering, content-based system, systems using learning approaches and multi-stakeholder system.

- **Chapter 3** discusses the impact of recommendation systems on society, by analyzing some of the underlying problems in the design of RS and how these lead to the creation of biases that negatively affect society, i.g creating filter bubble and echo chamber.

- **Chapter 4** contains a simulation study on the effect of RS on the diversity of topics to which users are exposed. EcoAgent, a multi-stakeholder recommendation system based on RL, and the environment in which it works will be introduced first. Later, we will further investigate how the environment is affected by the behavior of the recommendation system during simulation, focusing on identifying biases on topics. Finally, will be proposed two post-processing techniques that will allow us to assess the role of content providers and the fairness of the system in creating or amplifying such biases.

Contents

List of Figures

Chapter 1

Reinforcement Learning

Nowadays, whether we realize it or not, machine learning is everywhere – automated translation, image recognition, fraud detection, self-driving cars, and beyond. Lots of fields have benefitted from the application of machine learning algorithms that make the most of the big data they have access to, bringing a significant improvement in accuracy. In this thesis, we will talk in depth about one field that has benefited from the use of machine learning: recommendation systems. Recommendation system aims to help users deal with a big amount of information by filtering only items that could be relevant for them, based on various properties, such as their historical preferences, similarity to what they are currently consuming or other similar users have consumed and so on. Traditional recommendation methods have evolved from simple matrix-based methods that only capture simple linear interactions between users and items, to very complex methods that capture higher orders and more sophisticated interactions. One method which has been widely used recently is reinforcement learning.

In this chapter, we are going to briefly introduce machine learning and its categories, and then we are going to deeply explore reinforcement-learning, a computational approach to goal-directed learning by interaction, that focuses on providing solutions to teach machines and artificial intelligence to act and learn in an environment, exactly as a human being. First, we provide

an overview of all its meaningful elements and its mathematical definition in terms of a Markov Decision Process, finally we explore some of the approaches studied in literature to solve the RL problem, such as **Tabular** and **approximated methods** and **policy gradient methods**. Everything we will see in this chapter will help us to have a good theoretical understanding of reinforcement-learning to see in the next chapters its application to the recommendation problem.

1.1 Introduction to ML

Nowadays, whether we realize it or not, machine learning is everywhere – automated translation, image recognition, fraud detection, self-driving cars, and beyond. The term **machine learning** was first coined in the 1950s by the Artificial Intelligence pioneer Arthur Samuel [Mah20; SB18], who built the first self-learning system for playing checkers. He defined ML as the field of study that gives computers the ability to learn without being explicitly programmed. Machine learning is used to handle a high amount of data efficiently because it is able to automatically find the valuable underlying patterns among complex data that would otherwise be hard to discover. Then, the hidden patterns and knowledge about a problem can be used to predict future events and perform all kinds of complex decision-making.

However, the ML field is wide, indeed there is no single one-size-fits-all type of algorithm that is best to solve every problem. The algorithms to be used are chosen depending on the kind of problem you wish to solve, the available data, the type of feedback it receives while learning, and many other variables. We can classify ML algorithms into three broad categories [Mah20]:

- supervised learning,
- unsupervised learning,
- reinforcement learning

Moreover, we must point out the existence of an important sub-field of ML, *Deep learning*, which learns by mimic the mechanism of the human brain to interpret complex data such as images, sounds and texts capturing hidden features.

Supervised Learning

In supervised learning, the system learns from a set of input-output pairs, which are called *labeled examples*. Each example consists of a pair *(feature, label)* where the features match the input and provide the *description* while the label matches the output and gives prior knowledge about the features, for example, if the features are describing an object, the label could represent the category it belongs. Therefore, given a training set, SL systems learn a function that best approximates the relationship between input and output observable in the data. Throughout the learning process, the learner looks at the training data example and makes a guess about the output, a *supervisor* - the label - which knows the right answer corrects its guess if required. Learning ends when the learner achieves acceptable performances.Arrived here, the system has acquired some kind of patterns that allows him to generalize its responses so that it replies correctly also when it encounters a description for the first time. This approach is mainly used for **classification** and **regression tasks**.

Unsupervised Learning

In unsupervised learning, the system does not have any prior knowledge about the data it is analyzing and there is not any supervision. The system is fed with a set of examples composed of features only, and it is left alone to discover and present any interesting hidden structures it detects in the data through similarity detection and pattern recognition. When new data is introduced, it uses the previously learned hidden structures to recognize the class the data belongs to. The main tasks supervised learning tasks is used for are: **clustering**, **anomaly detection**, **association rules**, **feature**

reduction.

Reinforcement Learning

Reinforcement learning is fundamentally different from supervised and unsupervised learning in the sense that data are not provided as a fixed set of examples. Rather, there is a learner or decision-maker, called **agent**, that through interaction with an external world, called **environment**, collects data and learns what to do by mapping situations to actions that maximize a numerical reward. This approach, according to Sutton and Barto [SB18], suits particularly well the so-called **RL problem**, which presents the following characteristics:

1. the problem is closed-loop, which means that the learner's action influences its later system
2. the learner does not have a tutor to teach it what to do, but it should figure out what to do through trial-and-error,
3. actions influence not only the short-term results, but also the long-term ones

Another peculiarity of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment, while other approaches consider general sub-problems which could lead to uncertainty when fitted in real-time decision-making problems.

Deep Learning

Deep learning can be generally considered to be a sub-field of machine learning. It mimics the mechanism of the human brain to interpret data such as images, sounds, and texts. Neural networks are inspired by the biological neurons within the human body which activate under certain circumstances resulting in an action performed by the body in response. Neural nets consist of various layers of interconnected artificial neurons powered by activation

functions that help in switching them ON/OFF. Briefly, each neuron receives a part of the inputs and random weights, which are then added with a static bias value; this is then passed to an appropriate activation function which returns the neuron output. Once the output is generated from the final neural net layer, the loss function (distance between the actual value and the predicted value) is calculated, and backpropagation is performed where the weights are adjusted to make the loss minimum. Finding optimal values of weights is what the overall operation focuses around.

Neural networks can be of different types, depending on their structure. For example, Multilayer Perceptron (MLP) is the simplest form of neural network where input data travels in one direction only - forward — from the input nodes, through the hidden nodes (if any), and to the output nodes. A more complex variation of MLP is called Convolutional Neural Network (CNN), which is a special kind of feedforward neural network with convolution layers and pooling operations; it suits particularly well for images processing. Instead, Recurrent Neural Network (RNN) is different from feedforward nets since presents loops and memories in RNN to remember former computations. [Zha+19]

1.2 Reinforcement Learning

To begin, it can be useful to observe how RL works using an example, which we now describe: A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. At each such time, the robot chooses the action to execute:
- (1) search - actively search for a can,
- (2) wait - remain stationary and wait for someone to bring it a can,
- (3) recharge - go back to home base to recharge its battery

We know that finding a can produces high reward and the best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted and it will need to be rescued, producing low reward. The robot act considering the level of the battery.

It is easy to detect the **agent** - the robot - and **the environment** - any aspect the agent can sense, such as information about the room or its energy level. Moreover, we confirm that the example above presents all the characteristics of a RL problem, indeed the agent is not told which actions to take, and actions may influence not only immediate rewards, but also the consequent situations and therefore the subsequent rewards. So far, we can state that we have all the elements to solve our RL problem using reinforcement learning. On the other hand, there are four other main sub-elements that can be identified in a reinforcement learning system and that help to understand how the agent-environment interaction works:

- **policy**: it is generally indicated as π and defines how an agent behave in a particular situation. To do so, it maps perceived states of the environment to actions to be taken when in those states.

- **reward signal**: is an immediate feedback, in numeric form, on how good or bad was the action taken by the agent according to the policy. We must point out that the *reward* for a given action is non-deterministic because it depends also on the current state of the agent's environment. In our example above, the agent deciding to go after a can 5 m away can generate a high reward over this run, but lower in a situation where it risks shutting down because of low battery. Furthermore, the reward directly affects the policy changing the probability of taking an action, as a result of the reward obtained.

- **value function**: indicates the long-term desirability of states, it considers the rewards generated by the current states plus those generated by the states which are likely to follow. For example, a state might

always yield a low immediate reward but still has a high value because it is regularly followed by other states that yield high rewards, and vice versa. For example, picking one more can when the robot has low energy may yield a high reward, but bring to a state with low value since it may not be able to reach the charging station, producing low reward until it is rescued. Thus, the goal is to select actions that bring the agent to states of the highest value and reward *over the long-run.*

- **model**: provides dynamics about the behavior of the environment. For instance, the model can predict next state and next reward in a given state and action.

To sum up, in a reinforcement learning system the agent and the environment interact in the following way:

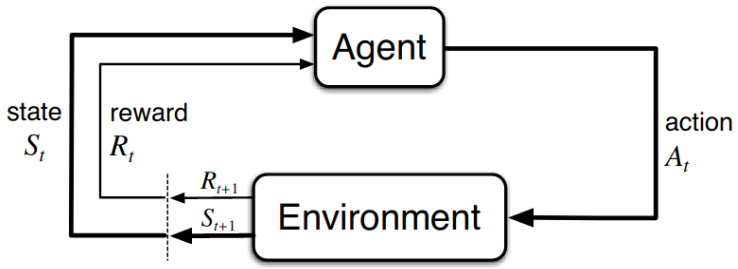


Figure 1.1: The agent-environment interaction in RL

Formally, the **agent** and **environment** interact at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, ...$ where at each time step t, the agent observes the environment, receiving some representation of its state $S_t \in S$, and upon that, selects an action, $A_t \in A(s)$, from the set of all possible actions. At the following time step, the agent receives a numerical reward $R_{t+1} \in R$ and enters a new state, S_{t+1}, both determined by the environment dynamics. In order to pick an action, the agent refers to a policy π_t that gives the probability of taking action $a \in A(s)$ when the agent is in state s, $\pi(a|s)$. The policy changes over time as a result of the agent's experience, which means that if an action got a good reward, its likelihood of being selected in

the next step will be higher. The agent's goal, roughly speaking, is to find the optimal policy that maximize the total amount of reward it receives over the long-run, by choosing actions that maximizes the value function.

Anyway, by simply choosing the action that maximizes the value function, we face one of the main challenges of RL, namely the trade-off between **exploration and exploitation**. Indeed, to obtain a high reward, a reinforcement learning agent must prefer actions that it has tried and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain the reward, but he must also explore to have the best action selected in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be the best. In the next sections, we will see how to concretely solve the RL problem framed, and we will introduce some of the most famous algorithms used in RL.

1.2.1 Markov Decision Processes

Markov decision processes (MDPs) are a tool that can allow us to describe the reinforcement learning problem in a mathematical way. In particular, it describes the environment and its behavior through a tuple (S, A, R, P, γ), and the agent behavior - the policy - as the probability distribution over actions given states $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$. The only condition required to formalize the problem as an MDP is that every environment states satisfy the *Markov Property*, meaning that states contain, other than current information, a summary of past sensation, in such a way that all relevant information is retained. Formally, an MDP is defined as a tuple (S, A, R, P, γ), where

- S is the set of all possible states,
- A is the set of available actions in all states,

- R is the reward function $R : S \times A \to \mathbb{R}$, $r(s,a) = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$

- P is the transition probability that guide the environment behavior, also called *dynamics*

$$p(s'|s,a) = Pr\{S_{t+1} = s'|S_t = s, A_t = a\} \tag{1.1}$$

- and γ is the discount factor.

The problem describes above can be formalized as an MDP (S, A, R, P, γ) where, S = {high, low}, the agent's action sets are A(high) = {search, wait}; A(low) = {search, wait, recharge} and the transition probability and expected reward are collected in Figure 1.1. The example can be summarized in the transition graph in Figure 1.2.

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	0.7	3
high	low	search	0.3	2
low	high	search	0.2	- 3
low	low	search	0.8	-1
high	high	wait	1	-2
high	low	wait	0	-
low	high	wait	0	-
low	low	wait	1	0
low	high	recharge	1	0
low	low	recharge	β	0

Table 1.1: Dynamics' table.

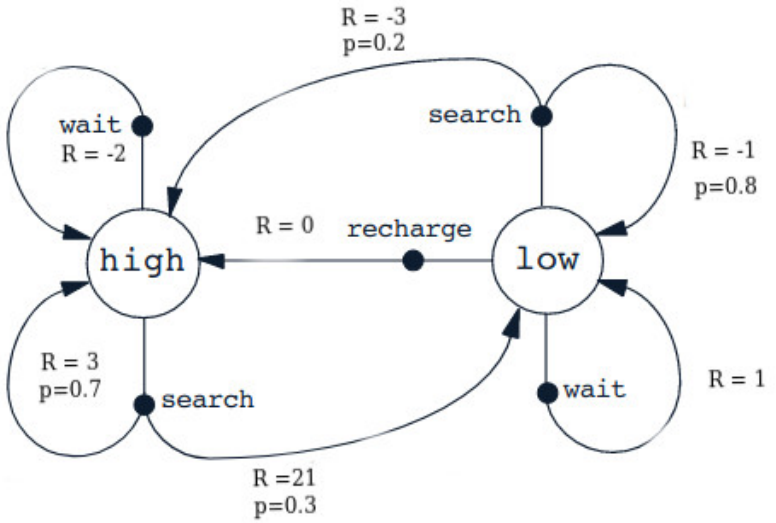


Figure 1.2: Transition graph

Once the problem is formalized as MDP, we can focus on solving it by finding the best path that will maximize the sum of rewards over the long-run. The solution involves estimating **value functions** of states that evaluate how good it is for the agent to be in a given state, defined in terms of future expected return when starting in s and following the policy π thereafter. The value function is defined in terms of the **expected future reward**, $G_t = R_{t+1} + .. + \gamma R_{t+2}$, which represents the total amount of reward the agent could receive over the long-run, which it seeks to maximize. Its peculiarity of satisfying the **Bellman equation**, which says that *the value function can be decomposed into two parts, the immediate reward plus the discounted future values*, allows decomposing a complex problem, into simpler, recursive sub-problems and finding their optimal solutions. It can be of two kinds:

- *State-value function*, indicated as v_π, evaluate "how good" is the state you are in and is defined as

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(s_{t+1}, A_{t+1})|S_t = s, A_t = a] \quad (1.2)$$

- *Action-value function*, indicated as q_π, evaluate "how good" is it to

take a particular action in a particular state and is defined as

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(s_{t+1})|S_t = s] \tag{1.3}$$

The maximization of the value function over all policies gets the **optimal value functions**, q_* and v_*, that specify the best possible performance we can obtain from the MDP. Then, it is easy to find the **optimal policy** just by picking action that maximize $q_*(s,a)$:

$$\begin{aligned} q_*(s,a) &= max_\pi q_\pi(s,a) \\ v_*(s) &= max_\pi v_\pi(s) \end{aligned} \tag{1.4}$$

$$\pi_*(a|s) = \begin{cases} 1, & \text{if } a = argmax_{a\epsilon A} q_*(s,a) \\ 0, & otherwise \end{cases}$$

Therefore, once we formalize the RL problem as MDP, we dispose of all the elements required to find the optimal value function, and thus the optimal policy that maximize expected reward. The solution of the former example will be:

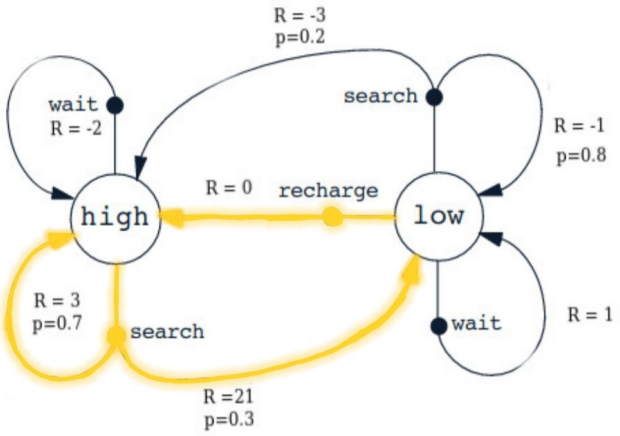


Figure 1.3: still to be modified

Reinforcement-Learning Algorithms

In the previous section, we demonstrated that it is possible to solve an RL problem by solving the *Bellman optimality equation*, however, there are tasks where this approach is too costly. This is due to the high computational cost they could require, given the necessity of an exhaustive search, looking ahead at all possibilities, computing their probabilities of occurrence and their desirabilities in terms of expected rewards. In tasks with small, finite state sets like the one we have analyzed, it is possible to reduce the computational cost by storing the value functions, avoiding useless re-computation. This approach is known as **tabular method**. Whereas, in tasks where the state space is too large to be stored, the value functions must be approximated using so-called **approximated methods**. Another method used very often is **policy gradient methods**, which learns directly to optimize a parameterized policy without consulting the value function. The rest of the chapter will cover the main algorithms we can use to solve RL problems, considering all methods named above: tabular methods, approximated methods, and policy-gradient methods.

1.3 Tabular Methods

Tabular methods are well-suited to solve RL problem with a small state and action space so that the approximate values are stored in arrays or tables. The three most important tabular methods are:

- Dynamic Programming, which are well-developed mathematically, but require a complete and accurate model of the environment.

- Monte Carlo methods, which do not require a model and are conceptually simple, but are not suited for step-by-step *incremental computation*, which means they do not allow improving the policy step-by-step as new information is available.

- Temporal-Difference learning, methods require no model and are fully incremental, but are more complex to analyze.

1.3.1 Dynamic Programming

Dynamic programming (DP) is a collection of algorithms that can be used to compute optimal policies assuming to have access to a perfect model of the environment in MDP form. On one hand, DP methods are well-developed mathematically to the point that they provide the theoretical foundation on which other methods, such as MC and TD, are built. On the other hand, the assumption of having access to a perfect model is extremely limiting. After all, MC and TD are attempts to achieve the same results as DP, but with lighter computation and with a loosen assumption about the environment.

Solving an MDP consists in - according to the **prediction problem (or policy evaluation)** - estimate the value function given a policy and - according to the **control problem (or policy improvement)** - in finding an optimal value function or optimal policy. The general idea of running multiple times **policy evaluation** and **policy improvement** that work together to find an optimal value function and an optimal policy, is known as **Generalized Policy Iteration**.

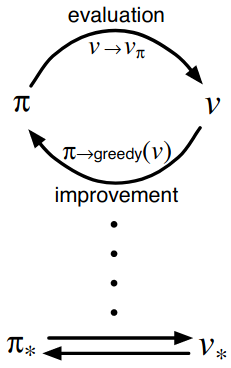


Figure 1.4: Generalized Policy Iteration (GPI)

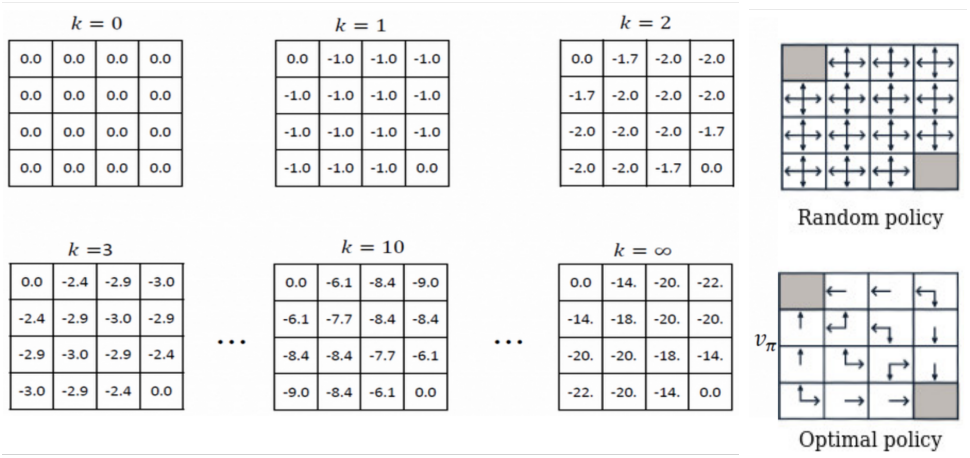
Assuming to have access to a perfect model and a policy π_0, DP algorithms can solve an MDP using two methods: **Policy Iteration** and **Value Iteration**. In **policy iteration,** the two phases are dependent. So, at each iteration, the *policy evaluatio*n is done considering a sequence of approximated value functions $v_0, v_1, v_2, ...$ where the initial v_0 is chosen arbitrarily, and each following approximation is obtained applying the equation 1.5 as an update rule until it converges to the optimal value function v_π. We must recall

that all previously computed estimations are stored in tables, and note that Eq. 1.5 updates v_{k+1} using an existing estimate, v_k, this mechanism is called *bootstrapping*. Only once the policy evaluation has converged, the *policy improvement* phase begins, which will greedily updates the policy taking, in every state, the action that maximize the value function, $\pi_1 = argmax v_*(s)$. Multiple iterations of evaluation and improvement may be required before the optimal policy can be achieved. We can summarize this process in Eq. 1.6 where $\xrightarrow{E}$ denotes a policy evaluation, $\xrightarrow{I}$ denotes a policy improvement, and -↠ denotes policy evaluation's iteration to converge.

$$v_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_k(s_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{1.5}$$

$$\pi_0 \xrightarrow{E} v_{\pi_0} \text{-↠} v_{*\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \text{-↠} v_{*\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \ldots \xrightarrow{I} \pi_* \xrightarrow{E} v_* \tag{1.6}$$

May be useful to look at an example to have a better understanding of the algorithm: suppose we have a grid world with 14 states, where each block represents a state and the top left and bottom right state are terminal. We can move up, down, left and right with each transition and we get an immediate reward of -1 until a terminal state is reached. We consider a discount factor γ=1 and a uniform random policy $\pi(up|S) = \pi(down|S) = \pi(right|S) = \pi(left|S)$ followed by our agent, as in Fig. 1.5b. To solve the prediction problem, we need to evaluate our policy. At the beginning v_0 is initialized at 0, then we compute v_1 using 1.5, and then v_3, v_4.. until it eventually converges to v_π. Then, in the policy improvement, first we extract $q(s, a)$, then we update the policy taking $\pi_1 = argmax q_*(s, a)$, which will result in Figure 1.5b. If it is different from the previous one, we do again policy iteration until we find the optimal policy.



(a) Policy evaluation

(b) Policy improvement

Figure 1.5: Policy Iteration

In **value iteration**, instead, policy evaluation and policy improvement are combined together taking, for each state, the maximum action-value as the estimated state value. Once state-values have converged to the optimal state-values, can be extracted the optimal policy. using the following update:

$$v_{k+1}(s) = max_a \mathbb{E}_\pi[R_{t+1} + \gamma v_k(s_{t+1}, A_{t+1}) | S_t = s, A_t = a]$$

A major limitation in these DP approaches is that they require to perform operations over the whole state set, that can result in a greatly expensive task when the state is large.

1.3.2 Monte Carlo Methods

The Monte Carlo methods (MC), unlike DP, do not require complete knowledge of the environment. For instance, they do not have access to transition probability or to the reward structure, so the only way to understand how the model evolves is by trying things out. This makes Monte Carlo methods learn directly from *experience*, but they are limited to be applied to episodic tasks.

As for *prediction problem*, MC methods are more interested in estimating the value of actions $q_\pi(s, a)$, than the states-value v_π, because the lack of a model does not allow having information about the next states. An obvious way to estimate the action-values of a state s from experience, composed of a set of episodes obtained following π then, is simply to average the returns observed after visits to that state in each episode. MC differentiates between a *first-visit method* and an *every-visit method*: in the former, v_π is estimated by averaging on the returns following first visits to s, while the latter averages the returns following all visits to s. The idea is that as more returns are observed, the average should converge to the expected value.

As for *control problem*, the policy improvement is done by taking greedy actions in each state. However, this may lead to a deterministic policy where low reward actions are never selected, compromising the achievement of maximum reward given the impossibility of exploring all possible paths. In fact, as described so far, MC *exploit* actions that it has tried and found to be effective, but it never *explores* new actions. So, to encourage *exploration*, it is necessary to introduce the definitions of *on-policy* and *off-policy* methods.

On-policy learning evaluates and improves the same policy which is being used to select actions. The mechanism used to encourage exploration considers $\epsilon - greedy$ policies, meaning that generally they select action greedily, but with probability ϵ they select a random action. The *off-policy* approach, instead, solves the dilemma of a policy trying to learn action values conditionally on optimal behavior while still behaving non-optimally to allow exploration. This is done through decoupling of the policy in two: the policy being learned about, called *target policy*, and the policy used to generate the data, called *behavior policy*. One possible way to achieve this is using an **importance sampling** mechanism, that is, weighting returns by the ratio of the probabilities of taking the observed actions under the two policies.

Also in this case, many evaluation and improvement iterations may be needed to reach the optimal policy. MC methods do not allow alternating policy evaluation and improvement on a step-by-step basis, though they al-

ternate on an episode-by-episode basis, allowing to compute $q_\pi(a, s)$ using episodes right away. This approach is known as **incremental method** and suits particularly well in non-stationary problems, where the true value is going to vary over time, so we do not care about old episodes. The idea derive directly from GPI, and we can summarize what happen as:

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} q_*$$

1.3.3 Temporal-Difference Learning

Temporal-Difference Learning combine both MC and DP approaches. Similarly to MC, it does not access a perfect model and learn from experience, whereas like DP, they update state-value estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap)[SB18].

Regarding the prediction problem, TD methods wait only until the next time step to update the state-value. At time $t+1$, once they have information about the reward gained with the action selected at time t, they immediately form a target and make a useful update using the observed reward R_{t+1} and the estimate $V(S_{t+1})$. The simplest TD method, known as TD(0), is

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

As for the control problem, both on-policy and off-policy methods, can be used. The on-policy methods make use of action-value since, as for MC, it is more informative in situation where the environment dynamics are unknown. Then it updates the policy using $\epsilon-greedy$ policy improvement. This method is known as Sarsa, and takes its name from the elements used in its update rule ($\mathbf{S}_t$, $\mathbf{A}_t$,$\mathbf{R}_{t+1}$, $\mathbf{S}_{t+1}$, $\mathbf{A}_{t+1}$):

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(A_t, S_t)]$$

The off-policy method is called **Q-learning**. It uses the behavior policy to explore the environment and to define the next state, S', that has to be visited. Then, it evaluates the action-value as in Eq. 1.7 considering $Q(S', a')$,

where a' is the action chosen by π, to reach the state S' decided by the behavior-policy. The target policy is updated greedily $\pi = max_a Q(S_{t+1}, a')$.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma max_a Q(S_{t+1}, a') - Q(S_t, A_t)] \quad (1.7)$$

1.4 Approximate Methods

So far, we have assumed that our estimates of value function could be represented as a table, with one entry for each state/state-action pair. Even though this method is easy to explain, it is usable only for tasks with small numbers of states and actions. For example, using a tabular method for chess is impossible, because we would need to evaluate and store the results of each state. In fact, a board has 64 possible positions and 32 pieces, so the state space consists of all possible board configurations, about 10120 states, which makes the calculation extremely expensive in terms of time and space.

Therefore, large state space RL problems, such as chess, must be solved with a different approach. Assuming that we have access to past-experience generated by following a policy π, we must estimate the value function through a *function approximator*, $\hat{v}(s, w)$ that learns the function that best fit the experience adjusting a weight vector **w** using **gradient-descent** methods. We will write $\hat{v}(s, w) \approx v_\pi(s)$ for the approximated value of state s given weight vector w. This method, known as **function approximation**, is an instance of Supervised Learning. It allows generalizing from seen states to unseen states and considerably reduces the memory cost, since it only needs the weight vector to estimates all the state-values, which otherwise would have been stored with the tabular approach.

1.4.1 Linear Function Approximator

Linear Function Approximator is one of the most important special cases of gradient-descent function approximation that approximate the function $\hat{v}$, as a linear function of the parameter vector, w. It needs states to

be represented as a features vector $x(S) = (x_1(s), x_2(s), ..., x_n(s))^\top$, where each feature x($S_i$) is giving some information about the state. As always, we begin from the prediction problem, the approximate state-value function is given by a linear combination of features $\hat{v}(S, w) = w^\top x(S) =$. Supervised learning is applied to training data, to find the weight vector w that best approximate the value-function. It aims to minimizing the loss between the approximated value-function and the true value-function, to do so it uses stochastic gradient descent that indicates the direction towards w's weight need to be moved to minimize the loss function. The control method, also in this case, uses an action-value function. Afterwards, policy improvement is done by changing the target policy to the greedy policy - in off-policy methods- or to an $\epsilon - greedy$ policy, in on-policy methods.

1.4.2 Nonlinear Function Approximator

RL application that use linear approximation have relied on hand-crafted features combined with linear value functions. Clearly, the performance of such systems heavily relies on the quality of the features' representation. However, in certain cases, we can have access to poor quality features. Fortunately, recent advances in deep learning have made it possible to extract high-level features from raw sensory data, leading to breakthroughs in computer vision, speech recognition etc. The same idea has been used in RL, using neural network trained on raw data, that learn better representation than linear approximator. This class of algorithm is known as **deep reinforcement learning**.

One of the most common algorithm of this class is *DQN*, which makes use of two different techniques to enable relatively stable learning: *experience replay* and *target networks*. Experience replay stores samples $< s, a, r, s' >$ in a buffer, randomly samples a mini-batch, and performs the above update over that mini-batch. This approach helps reduce the correlation between consecutive samples, which can otherwise negatively effect gradient-based methods. The second element, the target network, maintains a separate

weight vector θ to create a temporal gap between the target action-value function and the action-value function that updates continually. The separate weight vector. $\hat{\theta}$, is synchronized with θ after some period of time chosen as a hyperparameter. The algorithms at each time step t selects an action $\epsilon - greedily$ with respect to the action values, then, saves the tuple of experience $(S_t, A_t, R_{t+1}, \gamma, S_{t+1})$ into a replay memory buffer that can store up to one million transitions. Subsequently, the weights of the network are optimized using stochastic gradient descent $(R_{t+1} + \gamma max'_a q_\theta(S_{t+1}, a') - q_\theta(S_t, A_t))$ to minimize the loss. Back-propagation through gradient descent, however, is made only into the parameter θ of the Q-network - the network used to estimate actions' values - whereas the T-network parameters θ is updated only after a certain number of time steps as copy of the Q-network, and it is not directly optimized.

1.5 Policy Gradient

So far, all methods considered follow the same approach: learning the value function and then selecting actions based on estimated values.
In this section we consider methods that, instead, learn directly to optimize a parameterized policy without consulting a value function. The key idea underlying this method is reinforcing good actions: to push up the probabilities of actions that lead to higher return, and push down the probabilities of actions that lead to a lower return, until we arrive at the optimal policy.
Policy gradient methods focus on learning a parameterized policy $\pi(a|s;\theta)$, where $\theta \in R^d$ is the policy's parameter vector, and $\pi(a|s;\theta)$ is the probability that action a is taken at time t given that the environment is in state s at time t with the parameter θ. Policy gradient methods involve performing stochastic gradient ascent on some performance measure $J(\theta)$, so that every update step has the form $\theta_{t+1} = \theta_t + \alpha \Delta \hat{J}(\theta_t)$ where α is a learning rate and $\Delta \hat{J}(\theta_t)$ is the gradient of J.

1.5.1 Monte Carlo Policy Gradient

REINFORCE is a MonteCarlo method that uses episode samples in order to update the policy parameter θ. We consider a policy (here a neural network) and initialize it with some random weights. Then we play for one episode and after that, we calculate discounted reward from each time step towards the end of the episode. This discounted reward (G in the code below) will be multiplied by the gradient.

REINFORCE: Monte-Carlo Policy-Gradient Control (episodic) for π_*

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Algorithm parameter: step size $\alpha > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \gamma^t G \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 1.6: REINFORCE algorithm

REINFORCE with Baseline (episodic), for estimating $\pi_{\boldsymbol{\theta}} \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Algorithm parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)

Loop forever (for each episode):
  Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot,\boldsymbol{\theta})$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$:
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$ (G_t)
    $\delta \leftarrow G - \hat{v}(S_t,\mathbf{w})$
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t,\mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} \gamma^t \delta \nabla \ln \pi(A_t|S_t,\boldsymbol{\theta})$

Figure 1.7: REINFORCE-wb algorithm

A major problem with REINFORCE is the high variance in the estimation of the gradient it suffers from, which it is due to its MonteCarlo nature

that implies taking multiple random actions. To overcome this problem, an estimate of a state-value function $\hat{q}(A_t, S_t, w)$ can be added to the update rule as a **baseline** to reduce variance. The action-value $\hat{q}(A_t, S_t, w)$ is learned using one of the methods presented in the previous sections, e.g. MC methods or approximated methods. However, We must point out, that the action value acts just as a baseline, and it is not used for bootstrapping in order to improve the policy. Its only purpose is to stabilize REINFORCE variance in order to speed up the learning process.

1.5.2 Actor-critic methods

Actor-Critic is a class of algorithms that estimate both a parameterized policy and an action-value function, using the latter estimate to learn the former, they are actually among the first algorithms studied in the literature. Actor-Critic methods are composed of two processes. A *critic*, that inform the actor about how good was the action taken through the evaluation of the state is in, and an *actor*, that updates the policy in the direction suggested by the critic, using policy gradient.

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^{d}$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
  Initialize S (first state of episode)
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $A \sim \pi(\cdot|S, \boldsymbol{\theta})$
    Take action A, observe S', R
    $\delta \leftarrow R + \gamma \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$ (if S' is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S, \mathbf{w})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I \delta \nabla \ln \pi(A|S, \boldsymbol{\theta})$
    $I \leftarrow \gamma I$
    $S \leftarrow S'$

Figure 1.8: Actor-Critic algorithm

1.6 Multi-Agent RL

In a multi-agent system (MAS) multiple agents interact sharing the same environment. In this domain, MDPs are generalized to stochastic games or Markov games. MARL introduces a series of challenges to those already present in single agent RL, such as the curse of dimensionality that becomes even more problematic given the exponential growth of the state-action space, the problem of specifying a suitable goal, since agents' returns are correlated and cannot be maximized independently - from this the difficulty in shaping the reward, both in cooperative settings, where agents have a common goal, competitive and mixed. The exploration-exploitation dilemma is made more complex since agents need to explore not only to obtain more knowledge about the environment, but also on other agents. On the other hand, too much exploration can lead to destabilization of the other agents that are concurrently learning from the environment and the agent as well.

Chapter 2

Recommender Systems

In recent decades, with the rise of YouTube, Amazon, Netflix and many others similar web services, recommendation systems have taken more and more place in our lives. From e-commerce (suggesting buyers items that might interest them) to online advertising (suggesting users the right content, corresponding to their preferences), recommendation systems are now inevitable in our daily online journey. In a very general way, all recommendation systems have a common goal: to help people choose the most relevant element for them, among the huge amount of options available. However, the real-world application is a bit more complex and users are not the only stakeholders involved in the recommendation process. In fact, there are other parties that benefit from a good recommendation whose perspective should be integrated into the design of recommendation systems, such as suppliers of goods or services for sale and the system behind the platform. Therefore, based on the stakeholders involved in the recommendation issue and the data used to generate a recommendation, such as online user history or preferences of people similar to them, and the technique used to generate recommendations, we can distinguish different types of RSs.

Moreover, despite their good-natured aim, in recent years has grown awareness of their involvement in the generation of some negative effects to which society is subject, e.g. the spread of disinformation, homogeniza-

tion of users, the creation of filter bubbles and echo chambers. Given our interest in understanding the impact of the recommendation system on society, it is crucial to first have a solid understanding of how recommendation systems work thus, in the rest of this chapter, we will explore different types of recommendation systems.

2.1 Introduction

Recommender systems play a vital role in dealing with information overload, especially in domains such as e-commerce, music and film industries. They are a set of technologies used to sort information or goods a user might be interested in among everything provided by the website. For instance, the Netflix movies catalog is too big to be displayed to a user. Therefore, it is important to detect user preferences, and display just items that are most likely to be relevant for the user. This, of course, is beneficial to the individual user as well as the e-commerce owner. In fact, suggesting an element that matches the user's preferences increase his satisfaction, which increases the likelihood that the user will use the website again and consume other elements of the platform, consequently generating more income for the e-commerce owner. The first recommendation system implemented was Tapestry [Gol+92]. It was based on the simple observation that people often rely on the recommendations of others in their daily decisions, for example, when selecting a book to read. Thus, it tried to simulate this behavior, collecting users' opinions about an item to deliver useful recommendations to an active user looking for suggestions, and the author termed it *collaborative filtering* [RRS11]. However, later this term was integrated into the more generic *recommender systems*, which refers to any system that guides a user in a personalized way to interesting or useful objects in a large space of possible options or that produces such objects as output [FB08].

Recommendation systems, indeed, can use different approaches depending on the context they are working on, the type of data available and how

they produce the recommendation. Among the most important methods, we find the already mentioned **collaborative filtering** [Res+94], but also **content-based** or **knowledge-based** methods in addition to the methods developed more recently that involve the use of machine learning techniques, as **Reinforcement Learning Recommender System** (RLRS) methods.

2.1.1 Goals

Recommendation systems are designed to let the needs of users meet those of the platform. In the case of e-service providers, we have seen that the obvious reason to use recommender systems is to increase the number of users that accept the recommendation and consume an item, generating profit. Whereas, user's primary motivation is to find useful items. Both these goals can be reached by achieving less obvious sub-goals [Agg+16; RRS11]:

- **Relevance**: The main goal of a recommender system is to recommend items that are relevant to the user at hand.
- **Novelty**: There are items that the user has not seen in the past. For instance, in a movie RS such as Netflix, the service provider is interested in renting all the movies in the catalog, not just the most popular ones. Therefore, a RS suggests or advertises unpopular movies to the right user.
- **Serendipity**: The goal is to recommend items that are unexpected and yet useful. Serendipity is different from novelty because they are re not just recommending something he did not know about, but it is outside the user's expectations.
- **Increasing Diversity**: if more than a single item is recommended in each session, having several different items increases the chances that the user might like at least one of them. Moreover, diversity has the benefit of ensuring that the user does not get bored by repeated recommendation of similar items.

2.2 Types of Recommender Systems

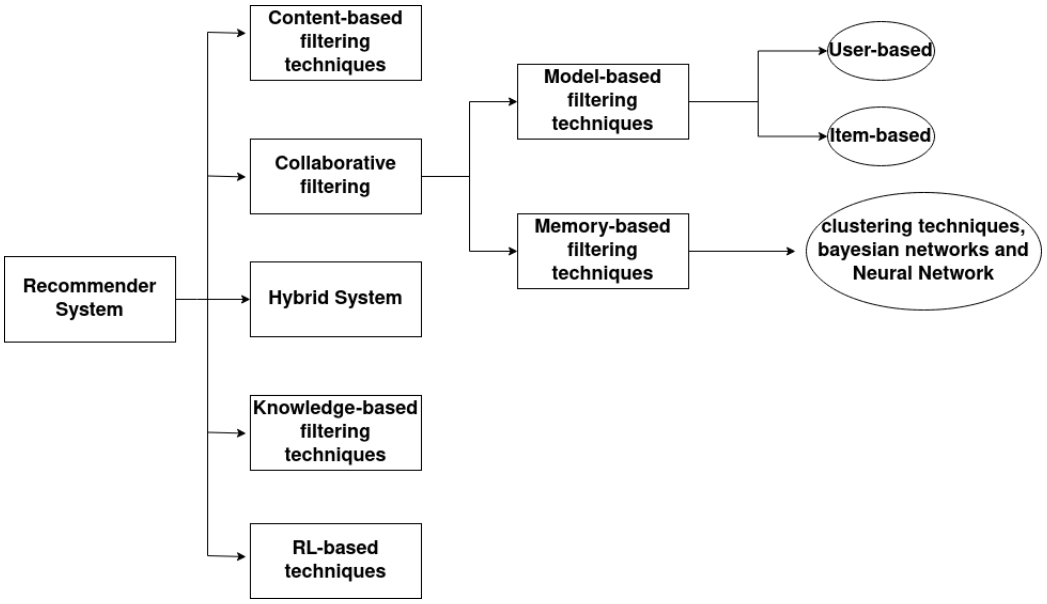


Figure 2.1: Recommendation Techniques

In most of the common formulation, the recommendation problem is reduced to the problem of estimating *ratings*, where rating are seen as the utility of an item for the user, that we want to maximize [AT05]. Intuitively, this estimation is usually based on the ratings given by this user to other items. Once we can estimate ratings for the yet unrated items, we can recommend to the user the items with the highest estimated ratings. Recommendations are make using different techniques, that are characterized by the **knowledge-source** they have access to and the type of algorithm used to estimate the utility. Specifically, knowledge-source is composed of [Bur02; RRS11]:

- *background data* the system have access to before the recommendation process begins. It could include a single user's **past-experience**, every user's past-experience, **items' features** or in some case some deeper knowledge about the domain they are operating in.

- *input data*, represents the information that **the user** must communicate to the system in order to generate a recommendation.

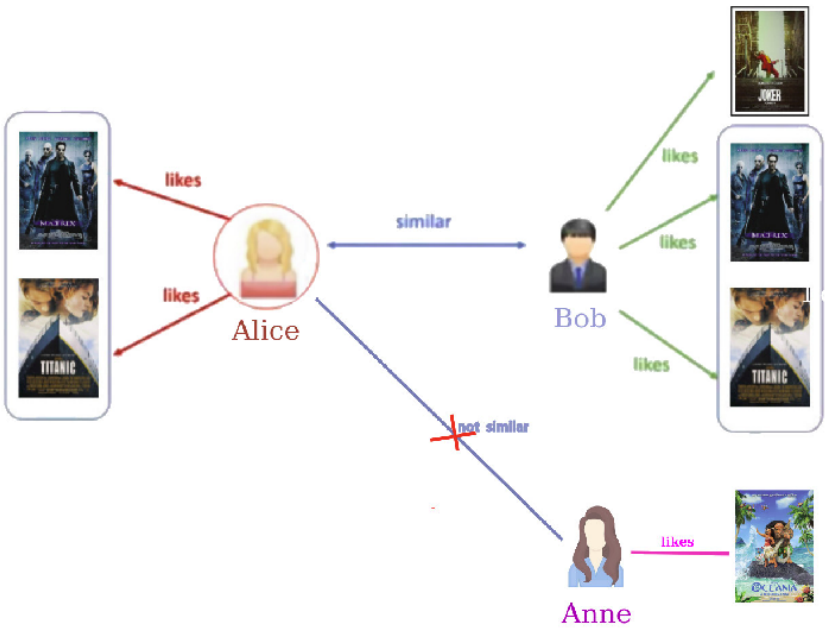
On this basis, we can have different techniques of recommendation, which combine different sources of knowledge and algorithmic approaches. In Fig. 2.1 we see some of the most important RS types, which we will analyze in detail in the next sections.

2.2.1 Collaborative Filtering

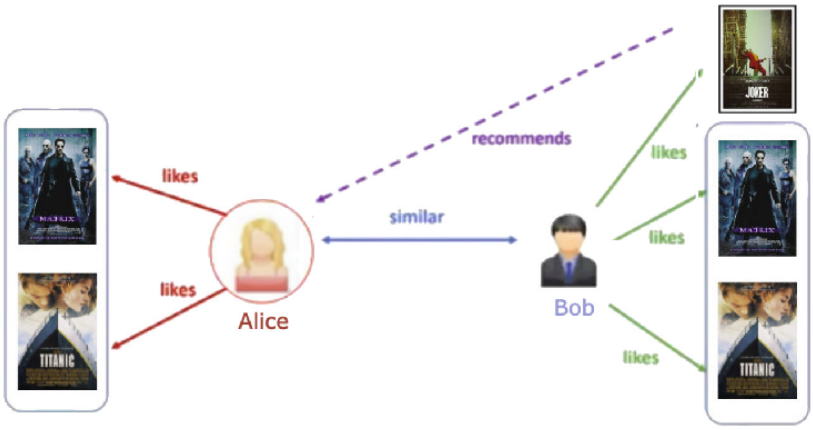
Collaborative filtering is the first technique developed in the recommender system field. The knowledge sources here are community and user's opinions. The system matches the profile of the active user, who is looking for a recommendation, with those in the community with similar taste by taking into account past rating history. Then, it suggests to the active user new items which have not yet been seen, but that have been liked by its peers in the past [Bob+13]. To do so, many algorithmic approaches can be used; the two more important are:

- **memory-based methods:** [Agg+16] They are based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings. Therefore, identifying similar users or items can be exploited to make recommendations. In particular, we can distinguish two techniques:
  - *user-based:* To catch the intuition behind these systems, we introduce this algorithm with the following example. Supposing to have access to the opinions of a small community composed of Bob and Anne, and to the active user's history, Alice. The first thing the system needs to do is to find users who share Alice's rating patterns. In the situation described in Figure, for instance, we can say that Alice and Bob are similar, because they both liked Titanic and Matrix. So, Bob and Alice are neighbors in terms of movie interest. Instead, Anne does not share Alice's taste. Therefore,

it makes sense to recommend to Alice a movie that her neighbor, Bob, has already seen and appreciated, meaning Joker.



(a) Step 1: defining Alice's neighborhood in terms of movie interests.



(b) Step 2: choose the most interesting movie for Alice, among those appreciated by her neighbors.

Figure 2.2: User-based example

Technically speaking, in order to decide if an unseen item X would be appreciated by user A, we use similarity measures and apply

the *k-nearest neighbor* algorithm. In other words, we:

1. compute the similarity between the target user and all other users
2. define the neighborhood of the target user, denoted $N(u)$, composed of a subset of similar users whom have rated the unseen item.
3. compute the utility of X for the active user, averaging the ratings given by users in the set of $N(u)$. Only if the resulting rating is above a certain threshold, X will be recommended.

– *item-based*: In the item-based approach, we look for the neighborhood to which an object belongs based on users' behavior. For example, Oceania and Matrix are considered neighbors, as they were positively rated by both Jack and Alice. So, Oceania can be recommended to Bob, as he has already shown interest in Matrix. We must highlight that here items' content (features) are not taken into account for recommendation generation.

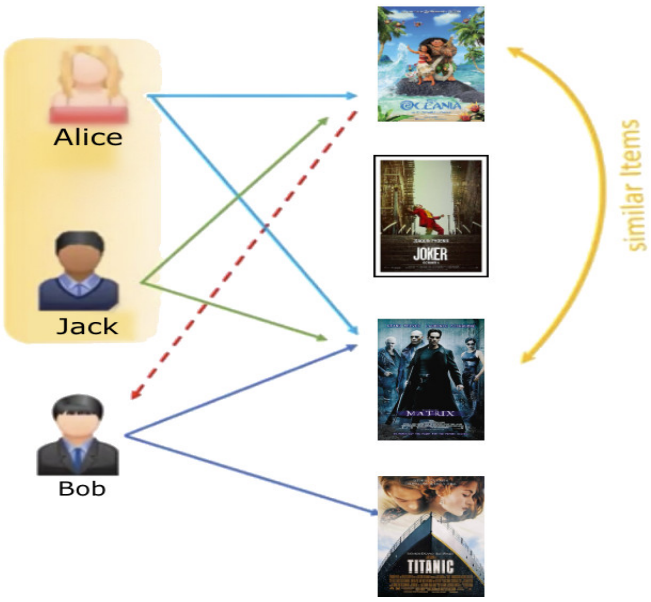


Figure 2.3: Item-based example

Technically speaking, we act similarly to the user-based approach. This time, we compute the similarity between the target item and all other items by comparing their ratings among users. Then we define the neighborhood of the target item i, composed of it similar items which have also been rated by the user u, denoted $N_u(i)$, . Finally, we rate i, as the average of the ratings from users in the set $N_u(i)$.

- **model-based:** a model is derived from the historical rating data and used to make recommendations. Some of the techniques used in model-based recommender system are *clustering* and *neural networks*.

The main advantage of CF is that it can perform well in environments where there is not much content associated with items or where items are too complex. Furthermore, it has proved to provide serendipitous recommendations [IFO15]. On the other hand, CF methods suffer in situations where there is a lack of previous ratings. On the contrary, the main problems they encounter are **data sparsity** and **cold start** problems, where data sparsity means that few ratings of the active users are available, and it is then hard to find *reliable similar users*, while the cold start problem is encountered when a new user or item has joined the system, so the system does not have enough interactions to make a useful prediction.

2.2.2 Content-Based Systems

Besides collaborative filtering, which tries to predict the target user behavior based on the community's rating, there is another important class of recommender systems, called *content-based*. Content-based systems, have access to items' features (or content) and to the active user's history. They analyze the target user's history to learn the profile of the user's interests, based on the features presented by the items the user has rated [Bur02]. This allows to predict the user's behavior towards an item and to suggest items that better satisfy his interests.

To catch the intuition behind this system, we can see how it behaves in the following situation: we have a system that has access to Alice's past history and to a catalog. The catalog is composed of just six movies, where the item features represent the genres they belong to. From Figure 2.4, we can see that Alice has already watched and rated three movies from the catalog. Now, the task of the recommender engine is to pick the movie that she would enjoy most among the three movies left. To do so, the system has to extract Alice's interests from her past interactions. For example, it notices that she gave a high rating to Guardians of the Galaxy and Captain America, which are both Comedy and Superheroes genres, hence, it concludes that she enjoys these genres. By analyzing the features of the three movies she has not watched yet, the engine ends up suggesting Spider-Man because it belongs to both the comedy genre and superheroes, and is therefore the one that best suits her interests.

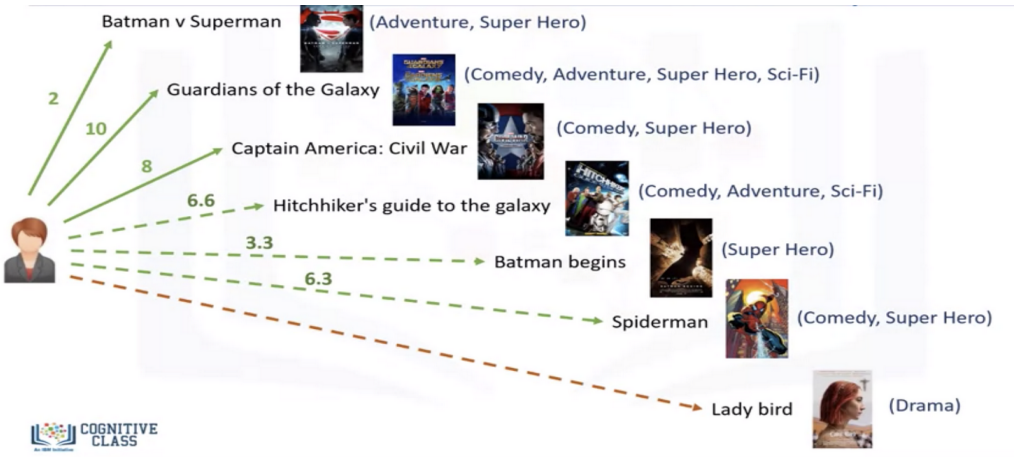


Figure 2.4: Content-based example

On the one hand, CB systems overcome some of the issues of CF systems. They are able to recommend new items even if there are no other users' ratings, as long as they match user interest. Therefore, they do not suffer from the *new item problem* encountered by the CF system. Moreover, they capture changes in the user's interests, adjusting recommendations in a short time. On the other hand, they still suffer from *new user* problem, because the

lack of user's history, leads to a lack of knowledge about user's interests on which to base recommendations. Moreover, they were criticized for reducing the diversity and novelty of recommended items, given that if a user has never consumed items with a particular feature, these items have little or no chance of being recommended.

2.2.3 Knowledge-Based Systems

Knowledge-based recommender systems are often considered to be closely related to content-based recommender systems. Here the knowledge source is composed of some domain-knowledge about the items it is recommending and the uses that they may serve, plus explicitly specified user requirements. This approach suits particularly well to contexts where the items are highly customized and are not bought frequently, such as real estate, automobiles, and luxury goods. Both content-based and collaborative systems have difficulties because of the lack of ratings that characterize these contexts. The former, indeed, has difficulties finding enough ratings in the user's history that match a specific instantiation of the item, from which to extract the user's behavior. The latter is facing the cold start problem, few ratings per user and few ratings per item make it hard to make useful predictions.

The idea behind a knowledge-based system, consists of **explicitly asking** users about their needs and providing items that fulfill their requirements, instead of trying to predict what they would like based on past experience. However, in order to provide personalized recommendations, these systems exploit *domain-knowledge*, which helps to map user requirements to the item's attributes. Therefore, a knowledge-based system first filters items that satisfy user's requirements, called anchor points. Then, domain-knowledge is applied to the anchor points to retrieve similar items that could be interesting for the user. Once, the system has returned its recommendations, the user has the opportunity to change his initial requirements on the fly, which is seen as a *critique* in response to recommended items. The process is then repeated with the updated requirements until the user gets

the desired result.

Figure 2.5: Knowledge-based approach example

One example is taken from Burke [Bur00], where it uses a knowledge-based system to help the users finding restaurants. The user is first asked explicitly his requirements. As shown in Figure 2.5a he wants to find a restaurant similar to Wolfgang Puck's "Chinois on Main" in Los Angeles. As shown in Figure 2.5b, the system finds a similar Chicago restaurant that combines Asian and French influences, "Yoshi's Café". The user, however, is interested in a cheaper meal and selects the "Less $$" button. The result in

Figure 2.5c is a creative Asian restaurant in a cheaper price bracket: "Lulu's". Note, however, that the French influence has been lost as a consequence of the move to a lower price bracket. The user-system interactions, play an important role in the process, because they encourage exploration, and allow to better identify user's needs in complex domain, where the user can not even be aware of various options trade-off in item's domain. [Agg+16; FB08]

2.2.4 Hybrid Systems

Hybrid systems can be seen as combinations of recommendation techniques. As discussed above, each technique relies on different knowledge sources, and all of them have strengths and weaknesses. Thus, the idea behind hybrid recommender systems is to combine two or more recommendation techniques to take advantage of all the knowledge available and get the best from every world. According to Burke [Bur02], hybrid recommender systems can be classified into the following categories:

- *Weighted*: in this case, the system implements two different recommendation techniques that work independently, i.g. CF and CB. The final result is given by a weighted combination of the output of the two systems;

- *Switching*: the system implements several recommendation techniques, and selects the one that best suits the current needs. For example, could use a knowledge-based system at the beginning to avoid the cold-start problem and then switch to CB or CF once more ratings are available

- *Mixed*: Recommendations from several different recommenders are presented at the same time;

- *Feature Combination*: the features from different data sources are combined and used in the context of a single recommender system.

- *Feature Augmentation*: one technique is employed to produce a rating

or classification of an item, and that information is then incorporated into the processing of the next recommendation technique;

- *Cascade*: one recommender system refines the recommendations given by another. For example, first using the KB system to make recommendations based on user interests, and then use to further ranking the suggestion in each bucket;

- *Meta-Level*: The model used by one recommender system is used as input to another system.

2.2.5 Learning Approaches:

Deep Learning Based Recommender Systems

The past few decades have witnessed the tremendous success of deep learning (DL). It has been applied to a wider range of applications due to its capability in solving many complex tasks while providing start-of-the-art results, from computer vision to speech recognition. The use of such techniques allowed to fix some of the obstacles encountered by traditional models, such as cold start (i.e., the system cannot provide useful recommendations when the user or the item is new), lack of novelty and diversity, scalability, low quality recommendation, and great computational expense. Lately, the data available has become more and more complex and the use of conventional methods, which are essentially linear models, can limit the model's expressiveness. Therefore, the use of deep learning models which are able to capture non-linearity in data allows capturing intricate user-items patterns, as well as efficiently represent input data to improve recommendation quality, and last, capturing sequential patterns in the user behavior. [Zha+19]

Deep-learning based systems can be classified into two categories: models that use deep learning to *integrate* traditional techniques, and models based solely on deep learning technique. For example, Van den Oord [VDS13] use a content-based approach integrated with the use of deep convolution neural

networks to predict latent factors from music audio when they cannot be obtained from usage data. Whereas, Devooght [DB16] used RNN to reframe collaborative filtering as a sequence prediction problem. Instead of rating an item, he takes into account the order in which items are consumed by a user and try to predict what the active user is likely to consume next given his history.

Reinforcement-Learning Based Recommender Systems

Lately, reinforcement-learning-based approach have been adopted in real-life scenarios to capture the users' preferences from the interactions between the user and the recommendation agent, instead of relying on past history. In fact, all the above presented techniques suffer from two serious limitations. Firstly, most of them consider the recommendation procedure as a static process, i.e., they assume the user's underlying preferences keeps unchanged. However, it is very common that a user's preference is dynamic w.r.t. time, i.e., a user's preference on previous items will affect her choice on the next items. Thus, it would be more reasonable to model the recommendation as a sequential decision-making process. Secondly, the aforementioned studies are trained by maximizing the immediate rewards of recommendations, which merely concentrates on whether the recommended items are clicked or consumed, but ignores the long-term utility of the items. However, the items with small immediate rewards, but large long-term benefits are also crucial. [Liu+18]

Hence, the sequential nature of user interaction with RS suggests that the recommendation problem could be modeled as an MDP and solved using Reinforcement-learning algorithms. We can think of the recommender system as the RL agent, while everything outside, as the user and items of the system, can be considered as the agent's environment and it can be formalized as an MDP where:

- **State** S: a state $s_t \in S$ is defined as the user preferences and their past history with the system;

- **Action** A: contains items available for recommendation; $a_t \in A$ means that items a has been recommended to the user at time step t;

- **Reward** R: the RL agent receives reward $r(s_t, a_t) \in R$ based on the user feedback on the recommendation provided;

- **Transition probability** P: transition probability $p(s'|s, a) \in P$ is the probability of transition from $s = s_t$ to $s' = s_{t+1}$ if action a is taken by the agent

- **Discount factor** γ: $\gamma \in [0, 1]$ is the discount factor for future rewards. With $\gamma = 0$, the agent becomes myopic, i.e., it only focuses on immediate reward. On the contrary, if $\gamma = 1$, the agent becomes farsighted and focuses more on future rewards [SB18].

The environment described by the MDP can be built using three different methods: **offline, simulation**, and **online**. In *offline* method, the environment is a static dataset containing the ratings of some users on some items. A common practice in offline methods is to train the agent on the training data (usually 70-80% of the data) and then test it on the remaining data. In *simulation* studies, usually, first a user model is built and then the recommendation agent is evaluated while interacting with this user model. This user model could be as simple as a user with some pre-defined behavior, or it could be more complex and be learnt using available data. In *online* method, the algorithm is evaluated while interacting with real users and in real-time [ACF21].

Figure 2.6 illustrates the agent-user interactions in MDP. With the notations and definitions above, the problem of item recommendation can be formally defined as follows: *Given the historical MDP, i.e.,* (S, A, P, R, γ)*, the goal is to find a recommendation policy* $\pi : S \rightarrow A$*, which can maximize the cumulative reward for the recommender system* [Zha+18].

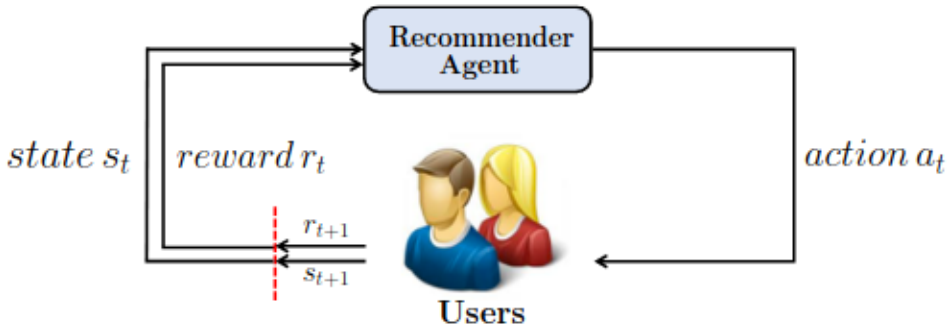


Figure 2.6: Agent-user interaction

2.2.6 Multi-stakeholders Recommendations Systems

All the recommendation techniques seen so far are able to satisfy the end user's needs. However, in many real-world applications, they are not the only stakeholder involved. There are other parties that benefit from good recommendations, and the integration of their perspectives into the design of recommender systems is the goal underlying the sub-field of *multi-stakeholder recommendation*. In this work, we are going to consider only key stakeholders that play an active role in the recommendation process which are the *users*, who consume recommendations that satisfy their needs; *providers* of goods or services for sale, and the *system* behind the platform, which suggests users documents whose features best suit their interests and which can possibly earn from the service offered [Abd+20]. An example of a multi-sided platform is Expedia which matches users with travel related-services (e.g hotels, car rentals, airlines), but also Airbnb or Uber.

Multi-stakeholder recommender system (MRS) should be designed to incorporate the interests of all parties. Therefore, MRS should consider not only the needs of users but also those of providers, as suppliers whose items are not recommended may experience poor user engagement and lose interest in staying on such a platform. Finally, if present, they should consider the

needs of the platform itself, which can have the goal of maximizing some income, or goals that concern the social welfare of the user, such as fairness and balance of items consumed by the user, which can go against users' preferences.

Ideally, all these stakeholder interests and factors should then feature in the learning algorithm, but in practice this requires both a precise mathematical specification of each objective as well as a learning algorithm that is able to solve for multiple (and potentially opposing) objectives [NDK17]. For this reason, in the literature, a variety of methodological approaches of different complexities have been explored to incorporate profit information into recommenders and to balance relevance and profitability. Chen [Che+08] compares existing recommendation algorithms such as CF with a new system that considers both product profitability and the purchases probability of the item by the user and shows that higher overall profitability can be achieved without a loss of accuracy for personalized recommendations. Das et al. [DMR09], implement an approach that supplements any traditional recommendation system and allows the vendor to control how much the profit-based recommendation should deviate from the traditional recommendation. Nguyen et al.[NDK17], instead, implemented a learning-to-rerank algorithm that, given an initial ranking of item recommendations built for the consumer, aims to re-rank it such that the new ranking is also optimized for the secondary objectives while staying close to the initial ranking. In the next chapter, will be seen in detail a different implementation proposed by Zhan [Zha+21] which uses a reinforcement learning (RL)-based recommendation approach to optimize the combination of both user and provider objectives.

Chapter 3

Recommender Systems and Society

Recommendation systems are seen as systems of good nature, which help users find interesting content among the information overload to which they are subjected, helping content providers to make more profit. On the other hand, in recent years awareness has grown about their possible negative effects on society. For instance, it has been proven that in order to maximize the platform's profit they can amplify the spread of misinformation, which obviously has an impact on the user's well-being.

In this chapter we are going to talk about the impact of RS on society, first analyzing some of the underlying problems in RS design, which lead to the creation of unwanted effects on society, and then analyzing in detail these effects. This chapter will give us the tools needed to identify the presence of biases due to the interaction with a recommender system, and therefore will allow us in the next chapters to analyze the impact on society of a particular recommendation system, EcoAgent.

3.1 Recommender Systems and Biases

In recent years there has been a growing awareness of the possible negative effects that recommendation systems can have on society: advertise items that maximize the platform profit, or gain user engagement through the spread of misinformation. Indeed, by nature of their design, the recommender system can lead to a self-reinforcing pattern by leveraging the updated data since the lifecycle of the recommendation has a **feedback loop** among three components: Users, Data, and RSs. These components are in a process of mutual dynamic evolution where users' profiles get updated over time via recommendations generated by the recommender system and the effectiveness of the recommender system is also affected by the profile of users, as shown in Figure 3.1 [Che+20; Man+20].

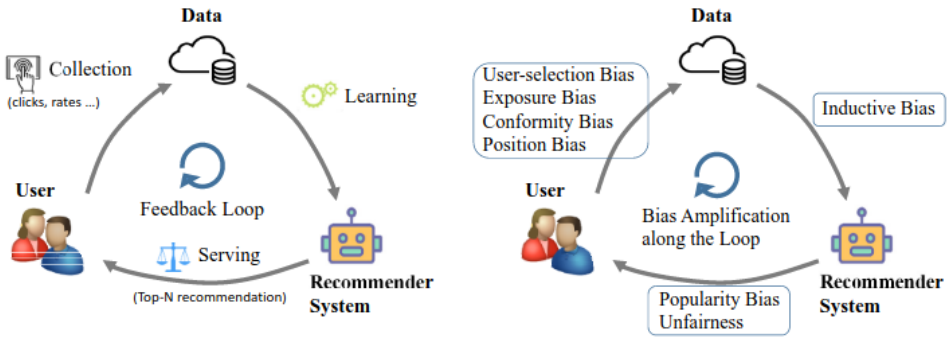


Figure 3.1: Recommendation Techniques

Throughout the feedback loop, we can identify different biases due to the algorithm used to make recommendation or to the data it uses, such as [Che+20]:

- **Selection Bias**: happens as users are free to choose which items to rate, and generally they tend to rate items they liked or those particularly bad. So the missing rating are not completely random.;
- **Exposure bias**: happens as users are only exposed to a part of specific items so that unobserved interactions do not always represent negative

preference

- **Conformity bias** happens as users tend to behave similarly to the others in a group, even if doing so goes against their own judgment, making the feedback do not always signify user true preference;
- **Position bias**: happens as users tend to interact with items in higher position of the recommendation list regardless of the items' actual relevance so that the interacted items might not be highly relevant;
- **Inductive bias**: denotes the assumptions made by the model to better learn the target function and to generalize beyond training data.
- **Popularity bias** Popular items are recommended even more frequently than their popularity would warrant;
- **Unfairness**: The system systematically and unfairly discriminates against certain individuals or groups of individuals in favor of others [Che+20];

Recent research studies indicate how the presence of biases deeply affects recommendations quality, and how they can lead to some undesired effects on society. For example, it has been proved that they *decreased diversity* among items consumed by a user, which intensify the homogenization of users, resulting in the creation of **filter bubbles**, where the user is only shown contents similar to those he had already interacted with, resulting in intellectual isolation, political **polarization**, and **echo chambers**. Furthermore, they may exhibit a bias towards popular items, resulting in homogenization of user behavior and a concentration of the market for content in the hands of a few creators[Luc+21; Ela+21a]. These are just some of the possible negative effects due to recommendation systems and considering the potentially significant effects of recommendations on different stakeholders, researchers increasingly argue that providing recommendations should be done in a *responsible* way, avoiding or at least mitigating negative effects.

To have a better understanding of what it means to provide a responsible recommendation, in the next section we are going to briefly introduce some of the undesired effects.

3.1.1 Undesired Effects to Society

Filter bubbles is one of the most studied effects of personalization and recommendations. Parisier [Par11], argued that the root of human intelligence is the ability to survive in a changing world, adapting and adopting new information. Simultaneously, recommender systems trap a user into an unchanging environment, called a *filter bubble,* where he is surrounded by things he is already familiar with and he likes, while removing key elements that trigger the necessity to see something different or to study different viewpoints. In the long-term, the filter bubble will be over-personalized and users will be isolated by the world outside their "bubble", and this may affect their creativity, other than their learning and thinking capabilities.

Echo chambers are a particular case of the filter bubble, where the user environment is other than personalized, also *polarized*, meaning that only certain viewpoints, information, and beliefs the users agree with are available [Ela+21b]. The recommender system amplifies this effect by repeatedly exposing the user to media contents that confirms their beliefs and viewpoints, decreasing their exposure to diverse opinions which could trigger the necessity to question their beliefs. The amplification of the echo chamber hinders mutual understanding and could lead to a situation where people are so far apart from those outside their chamber, that they have no common ground. It can be seen how they actually live in different realities and would hardly try to understand different points of view from their own.

Popularity bias is the tendency of emphasizing a "rich-get-richer" effect in favor of popular items. This bias generally derives from the data used to make recommendations, which is generally composed of a small set of popular items - also called *short-head* - and a big number of unpopular or niche items - called *long-tail*. The RS seems to focus on popular items, while

niche items do not get the deserved attention. On the one hand, this can be seen as a safe strategy, since users will be likely to enjoy popular items. On the other hand, it leads to some limitations since a platform that suffers from popularity bias will hardly promote niche items, and will push users to consume mainstream items, creating a market dominated by a few brands. In addition, they would miss the opportunity to improve the user's profile, as long-tail articles are unlikely to have been seen by the user already and will probably show different features than mainstream items. This would help to adjust the user preferences seen so far, and eventually find out their positions on those not yet met [ABM17].

Unfairness has several definitions in different contexts. One popular definition characterizes it *as the absence of any bias, prejudice, favoritism, mistreatment toward individuals, groups, classes, or social categories based on their inherent or acquired characteristics.* [Che+20]. However, in the context of RS there are different notions of fairness, for example, Burke [Bur17] defines fairness according to different stakeholders. In C-Fairness, the concern is that different users or user groups receive different types of recommendations, such as recommendations of lower quality or on particular themes, and that this increases the **discrimination** level of the algorithm. If we consider fairness toward providers, P-Fairness, instead they care that the benefit of being in the recommendation system is evenly distributed among providers, so that items of each provider are fairly recommended in order to avoid having providers leaving the platform. Having a fair system with respect to providers reduces the risk of having few providers **manipulating** the users' behavior.

Chapter 4

EcoAgent: a Case Study

In the previous chapters, we have seen the important role recommender systems play in helping consumers deal with the information abundance of a platform, reducing the sense of frustration and maximizing their satisfaction with the platform. However, we have also seen that user's satisfaction is strongly influenced by the content available on the platform, and therefore by content providers. Like for other digital technologies, there are some concerns about the effect of RS on society. There are those who argue that recommenders help consumers discover new products and, thus, increase recommendation diversity. Others, instead, believe recommenders only reinforce the popularity of already popular products, resulting in a reduction in the diversity of content to which the user is exposed, with the subsequent creation of a *filter bubble* or *echo chamber*. These effects are hard to analyze since the items' consumption is governed by a complex interaction between the users' preferences, the content provider's intent (that translates into increasing their own satisfaction, which means increasing their popularity and/or income), and the webpage-nature of the medium.

This thesis is an attempt to analyze these effects of RS. In particular, we focus on analyzing the diversity of topics that users have access to, where some biases that lead to the undesirable effects mentioned in section 2 may be recognized. In order to do so, we used the content provider-aware rec-

ommender system proposed by Zhan [Zha+21], **EcoAgent**. It is a multi-stakeholder recommendation system based on RL, that captures the dynamics that interplay among user-agent-content providers. In addition, it provides us with a simulated environment to study the effectiveness of the approach used and, as far as we are concerned, to study how RS behavior is reflected in other elements of the environment, such as documents available to the user that could create or amplify some bias.

Therefore, in the rest of the chapter, we will first see how EcoAgent is designed to meet the interest of all stakeholders involved, including a description of the simulation environment in which it can be applied in order to demonstrate its effectiveness towards the users and the content providers. Next, we will further study how the environment is affected by the behavior of the recommendation system throughout the simulation, focusing in particular on the effects it has on the document catalog regarding the distribution of topics. Finally, we will propose post-processing techniques that will allow us to evaluate the role of content providers and the fairness of the system, in the creation or amplification of some bias.

4.1 Problem Formulation

In this section, we will define the multi-stakeholder recommendation problem solved by **EcoAgent**. Stakeholders involved in this issue are *users*, who consume recommendation that meet their needs; *providers* of available content on the platform, in this case documents, and the *agent* which suggests users documents whose features best suit their interests. Figure 4.1 provides an intuitive scheme that captures stylized but real-world-inspired interactions between these stakeholders. In the rest of the section, we will analyze all stakeholders, defining their goals and how they interact with the agent to finally formulate the whole problem as an RL problem.

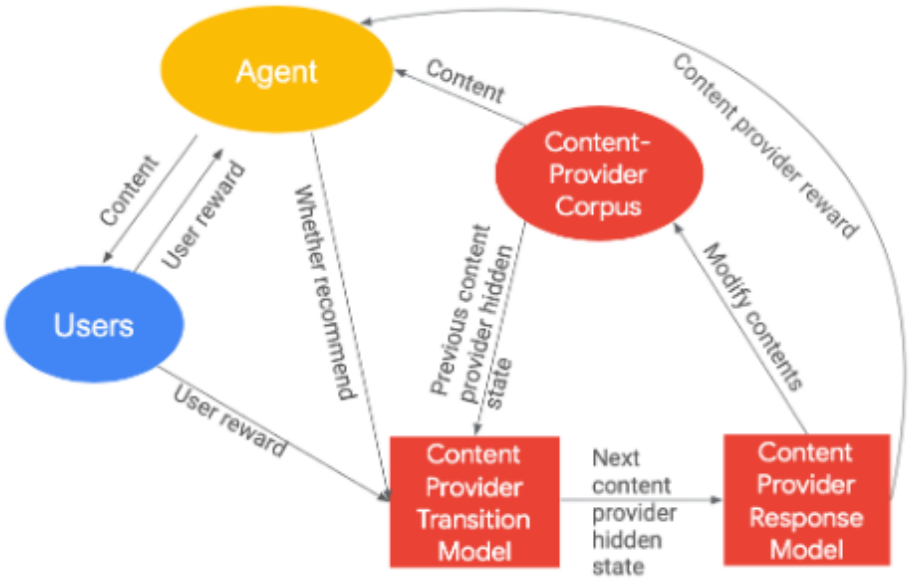


Figure 4.1: Interactions among Recommender Stakeholders

4.1.1 The User ↔ Agent Interaction

A user issues a (possibly implicit) query to the recommendation system. The recommender agent serves a slate of one or more items of content; the user interacts with the recommended content according to their preferences and emits implicit rewards, in the form of likes, clicks, and dwell times, which are used to evaluate their affinity with the item suggested and their overall engagement and satisfaction. Throughout this interaction, the agent and the users influence each other; the agent changes based on user feedback, and the users modify their preferences based on the elements the agent suggested.

The interaction between the user and the recommender agent follows the typical setup described in Section 2.2.5, so it can be formulated as an MDP $(S^u, A, P^u, R^u, \rho_0^u, \gamma^u)$ where S^u encodes the user topic preferences and satisfaction at time t; A is the action space composed of the content available on the platform; $P^u : S'^u \times A \to \Delta(S^u)$ captures the state transition shifting user topic preferences toward the topic of the recommended content, weighted by how much the user liked it, r^u and incrementing its satisfaction by the

reward obtained; R^u is the user reward to a recommendation and depends on the content's relevance for the items (how much the content topic align with the user features) and content quality; γ^u the discount factor. The MDP is solved following the *Generalized Policy Interaction* introduced in Section 1.3 to find the optimal state-value and policy. The state-value function, which we refer to as *user utility*, is estimated with an RNN that encodes user interaction histories, and predict observed user utility $Q_u(s^u, a)$ for any state s and action a. The *user agent*, then, aims to learn a policy $\pi(\cdot|s)$ that maximize the user utility by greedily taking:

$$\begin{aligned} &max_\pi \sum_{t=0}^{T} \mathbb{E}_{s_t^u \sim d_{\pi,t}^u(S_t^u), a_t \sim \pi(\cdot|s_t^u)}[Q_t^u(s_t^u, a_t)] \\ &\text{where } Q_t^u(s_t^u, a_t) = \sum_{t'=t}^{|} \tau^u|(\gamma^u)^{t'-t} r^u(s_{t'}^u, a_{t'}) \end{aligned} \tag{4.1}$$

Here T is the maximum trajectory length, and τ^u is the user trajectory sampled according to the policy π under the user MDP. We use $d_{\pi,t}^u(\cdot)$ to denote the average user state visitation distribution at time t, and $Q^u(s^u t, a_t)$ is the user utility calculated from time step t according to the policy π .

4.1.2 The Content Providers ↔ Agent Interaction

The recommender agent provides a slate of one or more items of content; depending on how many CP's contents are in the suggested slate and the user feedback to those contents, content providers decide what to do next: create more content, change their topic focus, and or even leave the platform if their satisfaction falls below a certain threshold. A CP leaving the platform affects the recommender agent since all its contents are removed from the content corpus the agent has access to. Again, each content provider can be formulated as an MDP $(S^c, A, P^c, R^c, \rho_0^c, \gamma^c)$ where:

- S^c encodes preferences for content topics, plus possible incentives for future content creation, and CP's current satisfaction with the platform.

- A is the action space composed of the content available on the platform,
- $P^c : S^c \times A \to S'^c$: captures the state transition, and defines the next states depending on the feedback from the recommender and the reaction of users to its content. For instance, if some of its contents are popular, in the next state it will continue to create content with the same topic, otherwise, it may change topics of future creation. Its satisfaction is incremented by the number of recommendations and summed to the user reward signals acquired from the current time step.
- R^c is the CP's reward to the recommendation, and is used to evaluate the CP engagement and satisfaction with the platform. It manifests as viability (deciding to stay or leave) and uploading new content. However, CPs do not react all in the same way to recommendations, in fact less established content providers who receive little attention or negative feedback from users are more likely to change their next content topics or decide to leave the platform, while established content providers are less affected by small changes in the recommendation and user feedback.
- γ^c the discount factor.

The content provider's MDP is solved in the same way as the user one; the *content provider's utility* $Q_c(s^c, a)$ is estimated using an RNN and the agent aims to learn a policy $\pi(\cdot|s)$ that maximizes the content provider utility.

4.1.3 EcoAgent: a Provider-Aware Agent

From the definition of users and content providers we have established that recommendations affect the future state and utility of both users and content providers, so the agent's recommendation problem is the following [Zha+21]:

> *Decide which content (from which content provider) to recommend to a user so that a combined metric of both content provider*

and user utilities are maximized, given the current state of the user and the providers of candidate content recommendations.

The system, called **EcoAgent** is defined as an MDP $(S, A, P, R, \rho_0, \gamma)$ where S is the concatenation of user state and states of all the content providers on the platform, A content available on the platform, $P : S \times A \to S'$ captures the transitions, R is the concatenation of proxy rewards of the user and all content providers, ρ_0 the initial state distribution,γ as a concatenation of user and content provider discount factors. The goal here is to learn the policy that maximizes two objectives: the user's utility and CPs' utility. The obvious objective definition is the following

$$max_\pi \sum_{t=0}^{T} \mathbb{E}_{s_t^c \sim d^u_{\pi,t}(S_t^c), a_t \sim \pi(\cdot|s_t^c)} [(1-\lambda) Q_t^u(s_t^u, a_t) + \lambda \sum_{c \in CP} Q_t^c(s_t^c, a_t)] \tag{4.2}$$

where is introduced a new parameter $\lambda \in [0, 1]$ that allows to interpolate between the two objectives. If $lambda = 0$, it means that only the user utility is maximized while $lambda = 1$ maximizes only the CP utility. One problem of this definition concerns its scalability in situations with many CPs since it requires the computation of all CPs' utility. The solution adopted requires to simplify the goal as in Eq. 4.3 where it is considered the utility obtained by a content provider recommending the item a_t minus the utility we could have obtained by recommending the item of a different CP, which is known as *utility uplift*. Here c_{a_t} denotes the content provider associated with the chosen content a_t and $b_t^{c_{a_t}}$ indicates any chosen content not associated with the content provider c_{a_t} at time t.

$$max_\pi \sum_{t=0}^{T} \mathbb{E}[(1-\lambda) Q_t^u(s_t^u, a_t) + \lambda \sum_{c \in CP} \underbrace{Q_t^{c_{a_t}}(s_t^{c_{a_t}}, a_t) - Q_t^{c_{a_t}}(s_t^{c_{a_t}}, b_t^{c_{a_t}})}_{\mathrm{utilityuplift}}] \tag{4.3}$$

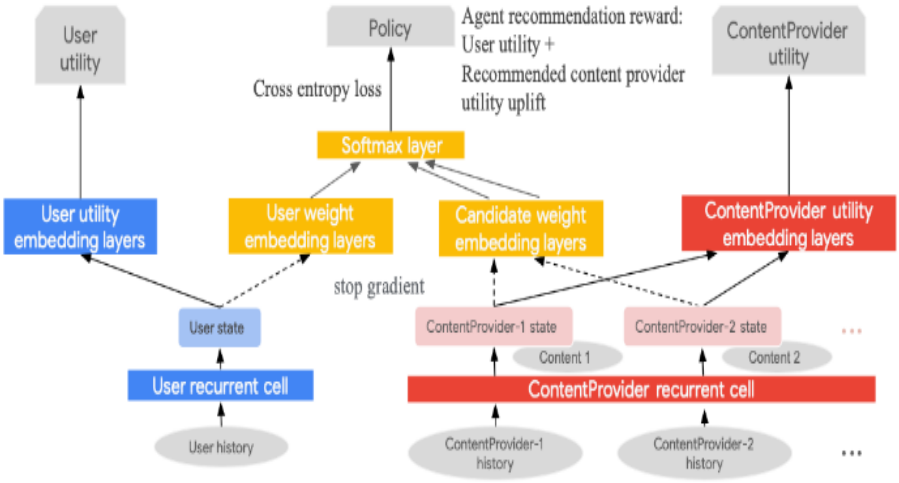


Figure 4.2: Illustration of EcoAgent structure. EcoAgent consists of three components: (i) a user RNN utility model that embeds user history into user hidden states and predicts user utility; (ii) a content provider RNN utility model that embeds content provider history into content provider hidden states and predicts content provider utility; (iii) an actor model that inputs user hidden state and candidates (content, content provider hidden state) to generate policy. Actor model is optimized using REINFORCE with recommendation reward being a linear combination of user utility and content provider utility uplift [Zha+21]

To sum up, the multi-stakeholder problem has been formulated as an RL problem that will be solved using the policy-gradient based algorithm, **REINFORCE**. EcoAgent captures the users-recommender-providers inter-dependence and in Figure 4.2 we have an overview of how the three components ties together.

4.2 Simulated Environment

EcoAgent has been evaluated by Zhan et al. [Zha+21] in a simulated setup to analyze its impact on the environment. However, in order to fully evaluate EcoAgent they developed a new Gym [Bro+16] environment able

to capture CP's dynamic on top of RecSim [Ie+19], which is a configurable platform to build simulation environments for recommender systems (RSs) that naturally supports sequential interaction with users. The environment captures the interaction illustrated in figure 4.1, and consists of a *user model*, *cp model* and *document model*. In this section, we are going to see in detail the configuration of these components and their dynamics in the environment.

The documents are drawn from the document corpus D. Each *document* $d \in D$ has an observable one-hot vector $v^d \in [0,1]^K$, where K is the number of possible content topics, and an inherent quality q^d, that is perceived by the user but is hidden to the agent.

The *user mode*l assumes users $u \in U$ have various degrees of interest in topics ranging from -1 (completely uninterested) to 1 (fully interested), with each user u associated with an interest vector $u \in [-1,1]|K|$. User u's interest in document d is given by the dot product $I(u,d) = u^d$ and it evolves over time as they consume different documents. By the consumption of a document d user generates an immediate reward which is a function $f(I(u,d), q^d) = R_t^u$ of user u's interest and document d's quality. In addition, the user's satisfaction is increased at each time step of the reward value obtained ($S_t^u = S_{t-1}^u + r_t^u$) [Ie+19].

The *CP model*, as for the user model, assumes content providers $c \in C$ have various degrees of interest in topics preferences for future creation, with each CP c associated with an interest vector $v_t^c \in [-1,1]^{|K|}$. Provider satisfaction at time t is then defined as the sum of the reward obtained for being exposed, or a penalty $\mu^c < 0$ if it gets no recommendation, plus the user feedback received. Thus, supposing that at time t, provider c receives recommendations of her content $(d_1, ..., d_m)$ and user rewards $(r_{u_1}, ..., r_{u_m})$ and that η_1^c, η_2^c is its sensitivity to content exposure and user feedback, respectively, then its satisfaction will be

$$S_t^c = f(\sum_{s=0}^{t} \mu^c + \eta_1^c m + \eta_2^c \sum_{i=1}^{m} r^{u_m}). \tag{4.4}$$

The creation of new content depends on the reward content providers receive,

which is defined to be incremental provider satisfaction:

$$r_t^c = S_t^c - S_{t-1}^c. \tag{4.5}$$

Each provider starts with a fixed number of content items, but if its current reward r_t^c is positive, the provider will create more content, where the number of new items is proportional to r_t^c . To model content providers adapting their topic preferences based on user feedback, a content provider updates her topic preference using the sum of her recommended content topics, weighted by the corresponding user feedback:

$$v_{t+1}^c \leftarrow v_t^c + \delta^c \sum_{i=1}^{m} r^{u_i} v^{d_i} \tag{4.6}$$

where δ^c represents how sensitive the content provider's topic preferences are to the user feedback. Each content provider also has a viability threshold; if their satisfaction S_t^c is below the threshold, they leave the platform.

4.2.1 Environment Setup

Once the simulation environment has been described, it is possible to use it to train the agent, by collecting data that describe the agent behavior that will be used to update it, and then, once the training is done, to analyze and evaluate its behavior in the environment. On each simulation, EcoAgent interacts with a new environment which initially samples 50 users and 10 content providers uniformly from the state spaces. Each content provider is initialized with 20 content items which are sampled from 15 topics based on the provider's topic preference, and they have the choice of creating more content or leaving the platform as described in Section 4.2.

4.3 Experiments

4.3.1 EcoAgents Training and Evaluation

Once we have defined the environment within which the agent operates, including the interface between agent and environment and how **EcoAgent**

is structured, we can start training EcoAgent. We are going to train several EcoAgents with different λ's (*content provider constant*) varying from 0 to 1, so that we will have:

- **user-only EcoAgent**, where $\lambda = 0$. EcoAgent only optimizes user utility;
- **content provider-user EcoAgent**, where $\lambda = 0.5$. EcoAgent optimizes the utility of both content providers and users;
- **content providers-only EcoAgent**, where $\lambda = 1$. EcoAgent only optimizes the content provider's utility;
- **RandomAgent**, where the agent recommends content randomly from the candidate set of content items.

All EcoAgents and RandomAgent have been trained for 300 epochs. For each training epoch, EcoAgent interacts with 10 new environments as set up above. The environment will be rolled out for 20 steps, suggesting a slate of three items. At each time step of one rollout, all users receive recommendations simultaneously, and the environment updates all users' and content providers' states. We then use collected data to update EcoAgent with Adagrad optimizer. Throughout the experiments, we consider long-term effects on users and content providers by setting discount factors: $\gamma^u = \gamma^c = 0.99$.

As Zhan [Zha+21], we compare the agent models testing them in 50 rollouts of new environments with the same setup they have been trained on, and then in 50 rollouts of new environments for 100 steps (with prespecified user and content provider dynamics but different initial states). We compute the following statistics of each rollout, which summarize how the agent influences users and content providers respectively:

- *user* and content providers' *satisfaction*
- *user* accumulated *reward*: $\sum_{t=1}^{20} r_t^u$;
- *content* provider accumulated *reward*: $\sum_{t=1}^{20} r_t^c$;

- # viable content providers: number of providers at the environment at the current time step.

Both satisfaction and accumulated rewards characterize how satisfied users and content providers are on the platform, while the number of viable providers reflects how agents help less established providers. In Table 4.1 and Table 4.2 we can observe how content providers' satisfaction grows as increases the value of the content provider constant, λ, while decreases the user satisfaction. As regards RandomAgent, we see that the satisfaction of content providers is between that obtained with EcoAgent0 and EcoAgent0.5, while that relative to users is just slightly higher than that obtained with EcoAgent1, which we recall does not take into account the user's objective when choosing the items to recommend. So we can see how RandomAgent, which we consider as our baseline, gains some content providers' satisfaction but provides very little satisfaction to users. The same scenario can be observed in Figures 4.3 and 4.4 where is considered the accumulated reward, defined in Equation 4.5.

	EcoAgent0	EcoAgent0.5	EcoAgent1	RandomAgent
100 steps	31810	36510	37034	34101
20 steps	5161	5698	5785	5643

Table 4.1: Content providers' satisfaction at the end of an episode

	EcoAgent0	EcoAgent0.5	EcoAgent1	RandomAgent
100 steps	2244	2187	2129	2144
20 steps	426	423	410	414

Table 4.2: Users' satisfaction at the end of an episode

Finally, it is important to observe the plots in Figures 4.5a and 4.5b, which represent the mean of viable content providers occurred across the 50 rollouts. We can see how Ecoagent0 and RandomAgent have the highest number of viable content providers, respectively 9 and 10, against 8 available

for EcoAgent0.5 and Ecoagent1. This is confirmed also by Table 4.3, which shows the minimum number of content providers observed across the 50 roll-outs. Here we can see how EcoAgent0.5, and EcoAgent1 have a tendency to leave more content providers behind. These results may seem strange at first glance since we would expect EcoAgent to help content providers to remain on the platform as λ increase, while we have that the number of viable content providers with content provider aware EcoAgents decreases, even though these EcoAgents have larger content provider accumulated reward as compared to a user-only EcoAgent. This is due to the incapability to capture the difference between less established content providers and established content providers, but it is in line with the goal of optimizing the overall content provider utilities [Zha+21] and not to keep the maximum number of content providers.

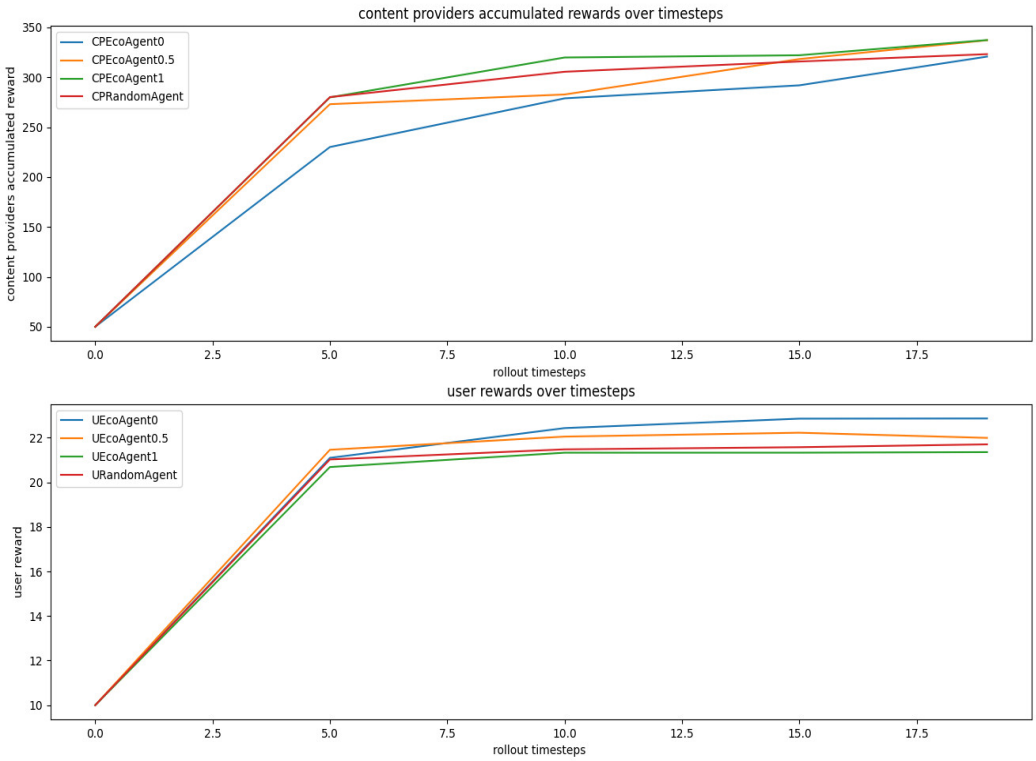


Figure 4.3: EcoAgent evaluation in a 20 steps simulation. EcoAgent (λ close to 1) helps content providers by improving content provider accumulated reward as compared to a user-oriented EcoAgent (λ close to 0).

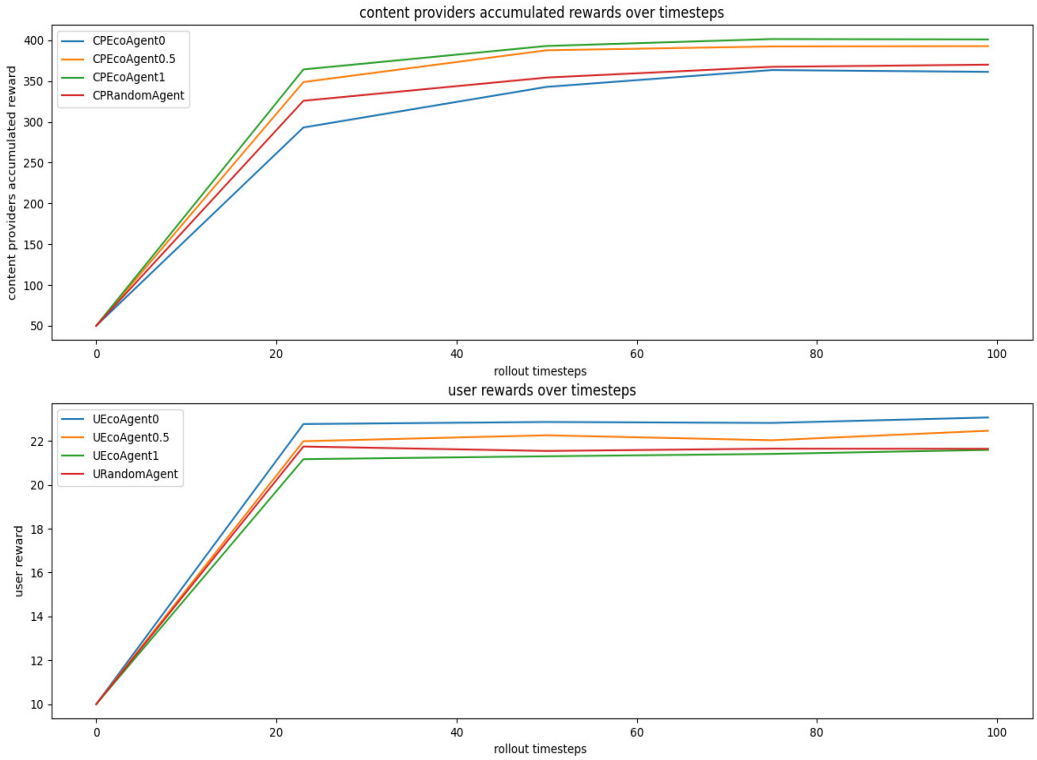
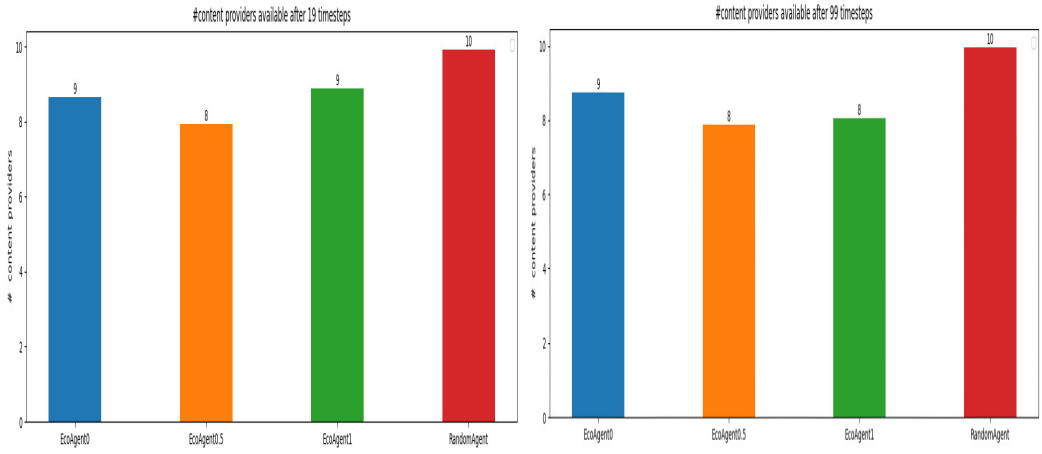


Figure 4.4: EcoAgent evaluation in a 100 steps simulation. EcoAgent (λ close to 1) helps content providers by improving content provider accumulated reward as compared to a user-oriented EcoAgent (λ close to 0).



(a) (b)

Figure 4.5: EcoAgents available content providers at the end of 20 and 100 steps simulation.

	EcoAgent0	EcoAgent0.5	EcoAgent1	RandomAgent
100 steps	7	5	5	9
20 steps	7	6	7	8

Table 4.3: Minimum content providers over timesteps

4.3.2 Simulation Analysis

Once we have defined how the agents have been trained and their validation, we execute the agents on the simulated environment described as in 4.2, which we recall initially samples 50 users and 10 content providers uniformly from the state spaces. Each content provider is initialized with 20 documents which are sampled from 15 topics based on the provider's topics preferences, and they have the choice of creating more content or leaving the platform.

Then, we will study the environment obtained at the end of the simulation to determine how the agent affects the environment's element such as the users, content providers and the document catalog. We chose to test the agent in a 100-step episode to observe the behavior of the system during extended interaction between the parties, and to identify any long-term side effects. Specifically, at the end of the simulations performed with the agent, we have highlighted three main points on the environment that will be justified in the rest of the section, namely:

1. *content provider* abandoned the platform due to unsatisfaction;
2. the EcoAgent's environment presents a *popularity bias* over topics available on the platform;
3. the content providers in the EcoAgent's environment present highly different level of satisfaction, which indicate an EcoAgent's *unfair behavior towards content providers*;

First of all, from the environment analysis we noticed that at the end of the simulation, EcoAgent has 8 content providers available, which means that

two content providers have left the platform due to unsatisfaction. Second, we focused on analyzing the distribution of the catalog topics recorded during the simulation. In Figure 4.6, we trace the probability distribution of every possible argument within the document corpus in every single passage of the episode. At the beginning of the simulation, we note that the topics have different distributions of probability, ranging from 0.04 to 0.9 due to the environment's policy of document creation based on the preferences of content providers sampled by the environment. However, it is interesting to note that the probability distribution of documents about *topic 0* increases faster than the others, reaching a 0.13 probability topic distribution. The same findings can be seen, from a different point of view, in Figure 4.7 where at the end of the simulations *topic0* has 250 documents while other topics have a maximum of 140 documents.

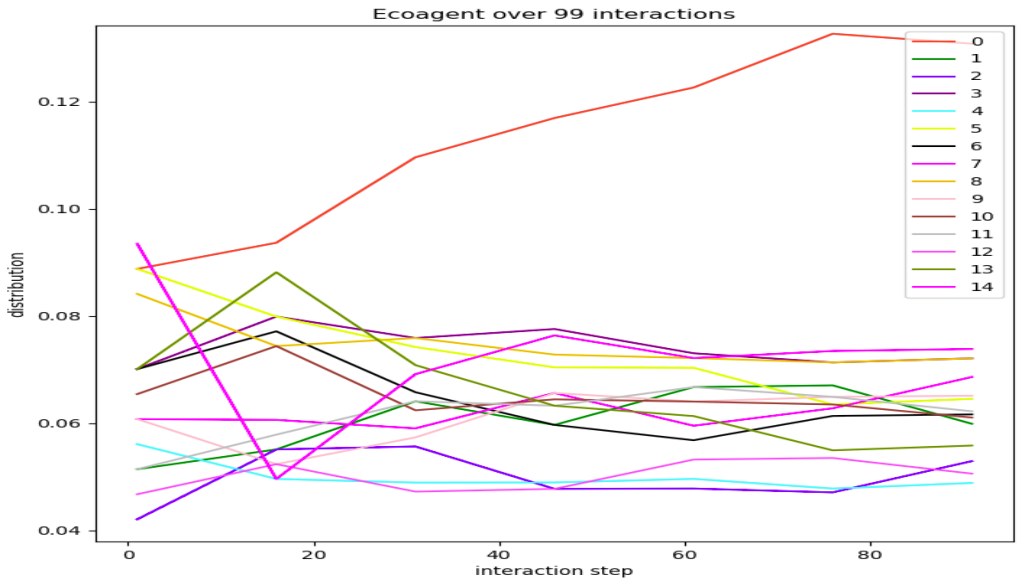


Figure 4.6: Overall topic distribution of documents in the simulation environment over interactions with EcoAgent. In this figure, the plot shows how the overall topics' distribution of the documents changes at each steps of the experiment, where each step represents an interaction between users and content providers with the recommendation system.

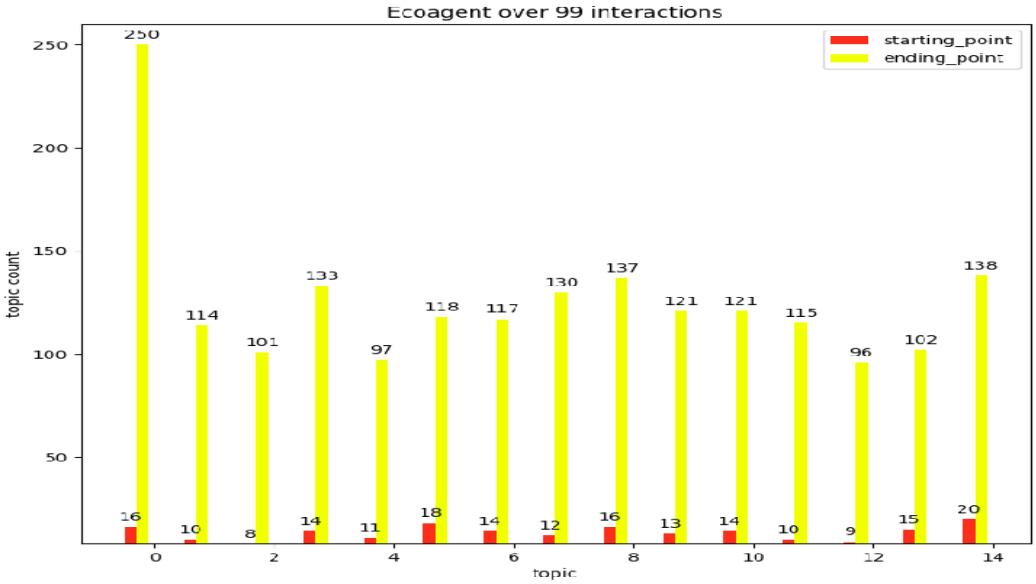


Figure 4.7: In this figure, it's possible to observe the number of documents for each topic at the beginning and at the end of the experiments.

	beginning of episode										end of episode									
Topics	cp0	cp1	cp2	cp3	cp4	cp5	cp6	cp7	cp8	cp9	cp0	cp1	cp2	cp3	cp4	cp5	cp6	cp7	cp8	cp9
topic1	1	0	2	1	3	1	4	2	1	1	4	0	6	2	4	0	222	4	2	6
topic2	0	2	1	2	0	2	0	2	0	1	3	0	11	6	3	0	83	4	1	3
topic3	1	1	1	1	1	2	0	1	0	0	5	0	3	3	2	0	82	2	3	1
topic4	4	1	2	0	2	1	0	0	1	3	6	0	6	2	4	0	108	0	1	6
topic5	1	2	3	1	2	0	1	0	1	0	3	0	5	1	3	0	81	1	1	2
topic6	3	1	0	1	1	1	4	2	2	3	5	0	2	4	2	0	87	4	4	10
topic7	0	2	1	1	2	2	0	1	4	1	0	0	3	3	4	0	93	5	4	5
topic8	3	1	1	1	1	1	1	1	1	1	4	0	8	1	4	0	103	1	4	5
topic9	0	1	1	2	1	1	5	2	1	2	1	0	4	3	1	0	120	2	2	4
topic10	3	1	3	0	2	1	2	0	0	1	7	0	7	2	2	0	98	1	0	4
topic11	1	2	2	2	1	2	1	1	1	1	2	0	6	3	3	0	97	4	3	3
topic12	1	0	0	2	0	1	1	3	0	2	6	0	4	4	2	0	88	4	1	6
topic13	0	1	2	2	0	2	0	1	1	0	2	0	4	3	0	0	78	4	2	3
topic14	0	1	1	2	2	0	0	3	4	2	2	0	4	4	3	0	73	4	5	7
topic15	2	4	0	2	2	3	1	1	3	2	4	0	2	4	2	0	114	3	3	6
Total	20	20	20	20	20	20	20	20	20	20	54	0	75	45	39	0	1527	43	36	71

Table 4.4: Content providers' contribution to the document catalog per topic at the beginning and at the end of an episode for EcoAgent.

Lastly, analyzing how each content provider contributes to the document catalog throughout the simulation, we noticed that using EcoAgent, a single

content provider in this case *content provider 6* owns most of the documents available on the platform, which is 1527 documents against the maximum of 75 documents owned by the other suppliers as shown in Table 4.4. Moreover, he has introduced 222 documents on *topic0*, which is the topic we have identified as suffering from popularity bias in the analysis of the topics' distribution. Since the interaction dynamics defined in the simulation environment anticipate that documents will be generated by content providers that have gained positive rewards, this finding suggests the presence of unfair behavior of the system, where *cp6* is favored among the others. For this reason, we proceed to measure the distributional inequality in the satisfaction of content providers with the Gini coefficient. The Gini coefficient ranges from 0 to 1, where 0 represents perfect equality, which means that all content providers of the platform have the same level of satisfaction, and 1 represents complete inequality, which means that only one content provider of the platform is satisfied, while all others are unsatisfied. It is defined mathematically as the ratio of the area that lies between the line of equality and the Lorenz curve, which represents the proportion of the total content providers' satisfaction (y-axis) generated by the bottom x% of content providers during the experiment. At the beginning of the simulation, we have a Gini coefficient equal to 0, in fact, we can observe in Figure 4.8 that the Lorenz curve at time step 0 coincides with the equality line. Instead, a time step 99 we have a Gini coefficient equal to 0.830637411643133 which is also evident in the plot in Figure 4.8 where we can see that at time step 99, the set of 9 content providers gets less than 20% of the cumulative satisfaction.

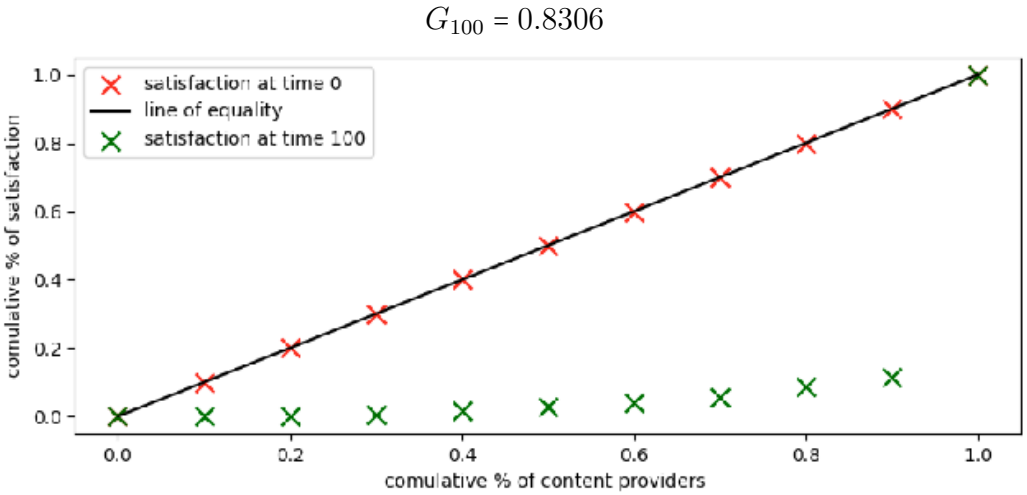


Figure 4.8: Visual representation of the Gini Coefficient at time step 0 and 100 of EcoAgent using the Lorenz curve.

To sum up, from the analysis of the simulation we get that EcoAgent tends to leave content providers behind and that it suffers from unfairness towards content providers. Indeed, we have seen that *content provider 6* has a major impact on the document catalog. Moreover, we have noticed a popularity bias over the creation of document treating *topic0*. However, although unfairness could lead to an unhealthy system, this is not a shocking outcome since EcoAgent, for its definition, is only trying to optimize the overall content creator objective without paying attention to being fair among content providers. Instead, given our interest in the impact of the recommender system on society, we have found more interesting the identification of a topical bias, since it can be hurtful for users interacting with the platform because it means a reduction of topic diversity to which users are subjected, that leads to the creation of echo chamber, where users' interests results narrowed down increasing their risk of being manipulated by content providers who provide them this content. So we wonder whether and to what degree this bias could be created or amplified by both the reduction of content providers available on the platform and the unfairness of the system towards content providers. This point will be analyzed in detail in the next section, where we will define some post-processing approaches that will allow us to study how these two

characteristics of the system affects the topics' popularity.

4.4 Post-Processing Experiments

In the previous section, from the analysis of the environment in which EcoAgent was run, we have highlighted three main points: the tendency of EcoAgent to leave content providers behind, the presence of a popularity bias over topics available on the platform, and the unfairness of EcoAgent towards suppliers. Since, as we have seen, the *reduction of topics* to which users are exposed leads to the creation of an *echo chamber*, we are interested in understanding the role of content providers leaving the platform and of the unfairness of the system in the creation and amplification of such an effect. Therefore, in this section, we are going to answer the following questions:

- does a system that keeps content providers on the platform provide a healthier environment to users?
- does a fair system towards content providers offer a healthier system to users?
- How does unfairness towards content provider reflects in the creation of biases?

To answer these questions, we are going to define some post-processing methods on EcoAgent, one reactive and one proactive, which allows us to manipulate the agent's behavior in order to study whether and how the topic distribution of documents is affected. First of all, assuming that the effect created was caused by the reduction of the content providers available for the platform, we worked to get an agent that guarantees to do everything possible to keep the maximum number of content providers on the platform. Thus, we aim to have a system that given the set of content providers CP, keeps almost 100% of its content provider on the platform after 100 interactions, which means that all content providers' satisfaction at time 100 is above a

certain threshold θ:

$$CP_{100} = \{c \in CP | \forall c \in CP, S_k^c \geq \theta\} \text{ and } |CP_{100}| \geq y\%|CP|$$

We use two different approaches to reach our goal, first a **reactive** approach which does not behave fairly towards content providers, but *greedily* ensures keeping the maximum number of suppliers on the platform. This approach consists of checking content provider satisfaction throughout the simulation and manually intervening in case one of the content providers notifies the system that it's about to leave. The action consists of recommending its documents to the user in order to gain the satisfaction it needs to want to remain on the platform. The second approach is **proactive**, instead. This approach ensures to have the maximum number of suppliers on the platform while trying to act fairly towards content providers throughout the whole simulation by *rebalancing* content creators' satisfaction at every step of the simulation by recommending documents owned by content providers with the lowest satisfaction. Both these approaches will be described in detail in the next sections.

Therefore, the systems obtained will be evaluated over the same simulation environment described in Section 4.2 considering the metrics listed below:

- number of available content providers $\#\{c \in CP | S_k^c \geq \theta\}$;
- *mean* and *median* of *content providers' satisfaction*;
- *Gini coefficient* to measure the distributional inequality in the satisfaction of content providers.

As a result, we expect that both systems obtained from the application of the two different approaches maximize the number of content providers available, while presenting different coefficients of Gini, which reflect their fairness towards suppliers. Next, we will study the environment resulting from the simulation of the agents obtained to assess how the presence of content providers and fairness helps to provide a healthier environment for users.

4.4.1 Reactive Approach

In this section, we will first describe the **reactive post-processing** approach used on EcoAgent and then study how the application of such an approach changes the system behavior during the simulation and how it affects the environment. This approach involves checking on the environment throughout the simulation by verifying the satisfaction of content providers and, eventually, intervening manually in the event that one of the content providers notifies the system it is about to leave the platform.

In other words, assuming that we are at time step t, where we have a precise environment configuration $env_t = (cp_t, u_t, a_t)$. The agent proposes a list of documents (d_1, d_2, d_3) with which users interact leading to an update of their status u_{t+1}, which is followed by an update of the status of content providers cp_{t+1} based on user feedback and the reward for the recommendation. If at the end of the interaction, no content provider is willing to leave the platform, then we update the state of the agent a_{t+1} and initialize the environment of the next time step, $env_{t+1} = (u_{t+1}, c_{t+1}, a_{t+1})$, which will be used in the following interaction. If, on the other hand, a content provider is willing to leave the platform as a result of the interaction, then we will restore the environment of time step environment t, env_t, and modify the slate of documents suggested by the agent (d_1, d_2, d_3). The slate of documents is changed by replacing the document belonging to the content provider with the highest satisfaction with one of the documents of the content provider that has expressed the intention to abandon the platform.

As a result of the application of the post-processing method described above on EcoAgent, we reached the goal of getting a system able to keep the majority of content providers on the platform. In fact, from the evaluation of the agent behavior on the environment with a 100-step long simulation, we get at the end of the simulation the totality of content providers. However, checking the median value of the content providers' satisfaction in Figure 4.9 at the beginning and at the end of the simulation, we can see that a time step 1 we have a symmetrical distribution of the satisfaction among content

providers while at time step 100 we see how the median value is closer to the bottom of the box, and the data are evidently more spread and, therefore, the distribution of satisfaction among content providers is skewed. The same thing is evident in Figure 4.10 where the Gini coefficient value is equal to 0.8615. Thus, we can conclude that by applying the reactive post-processing approach to EcoAgent, we get a system that can keep content providers on the platform, but we have no improvement in terms of its fairness to content providers.

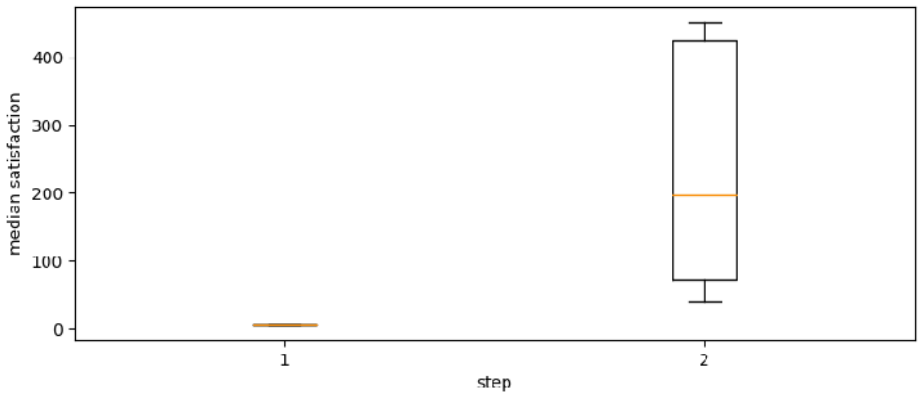


Figure 4.9: Median content providers' satisfaction observed using EcoAgent with **reactive post-processing** at the beginning and at the end of the simulation.

$$G_{100} = 0.8615$$

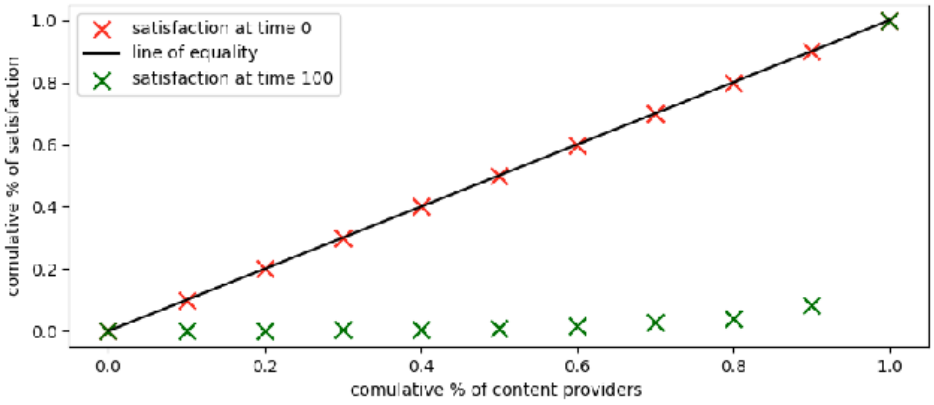


Figure 4.10: Visual representation of the Gini Coefficient at time step 0 and 100 of EcoAgent with **reactive post-processing** using the Lorenz curve.

4.4.2 Proactive Approach

In this section, as in the previous one, we will first describe the **proactive post-processing** approach used on EcoAgent and then study how the application of such an approach changes the system behavior during the simulation and how it affects the environment. As before, we will monitor content provider satisfaction throughout the simulation and act to manipulate the agent's behavior. Contrary to what we did in reactive post-processing, where we only reacted when a content provider expressed its intention to leave the platform, in this case, we will act at every step of the simulation to keep all content providers with a similar level of satisfaction.

In other words, assuming to be at time step t, where we have an environment configuration $env_t = (cp_t, u_t, a_t)$. We start storing the content provider with the maximum satisfaction c_{max}, and its respective satisfaction value $s_t^{c_{max}}$. Following, we store the content providers that register a difference in satisfaction with respect to c_{max} greater than α, which is our unfairness tolerance. The introduction of the unfairness tolerance value is required to preserve the work of the recommendation system, which would otherwise be limited to suggesting documents of the three content providers with the lowest satisfaction. In this case, instead, unless there are evident differences in suppliers' satisfaction, the proposed recommendations are retained. When the agent proposes its slate of suggested documents $slate = (d_1, d_2, d_3)$, we proceed by searching inside the slate for the document belonging to the content provider with the highest satisfaction among the recommended providers. Then, we replace this document with a document of the content provider with the lowest satisfaction. If there are other content providers with a satisfaction difference lower than the tolerated one, the process is repeated again. In the end, we obtain a new slate, $slate'$, with which users interact, leading to an update of their status u_{t+1}, which is followed by an update of the status of content providers cp_{t+1} based on user feedback and the reward for the recommendation. Then, we update the state of the agent a_{t+1} and initialize the environment $env_{t+1} = (u_{t+1}, c_{t+1}, a_{t+1})$ which will be used in the following

interaction.

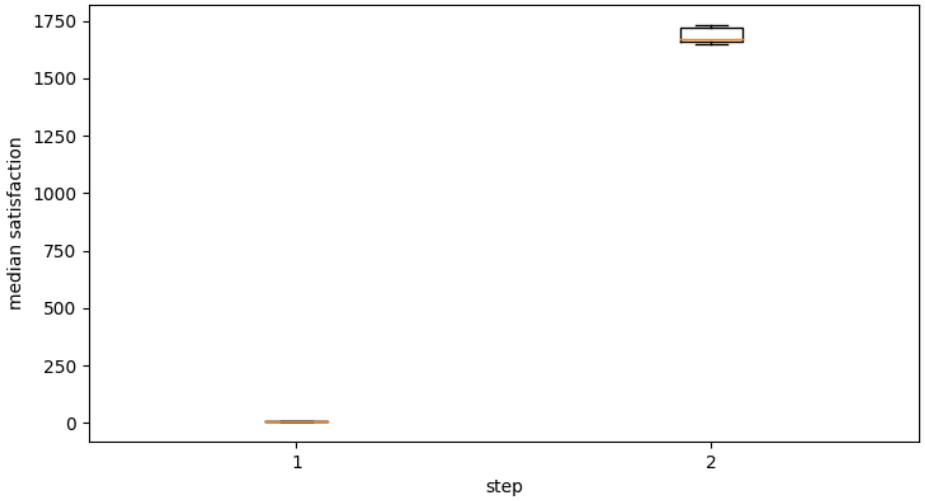


Figure 4.11: Median content providers' satisfaction observed using EcoAgent with a proactive post-processing at the beginning and at the end of the simulation.

$$G_{100} = 0.0596$$

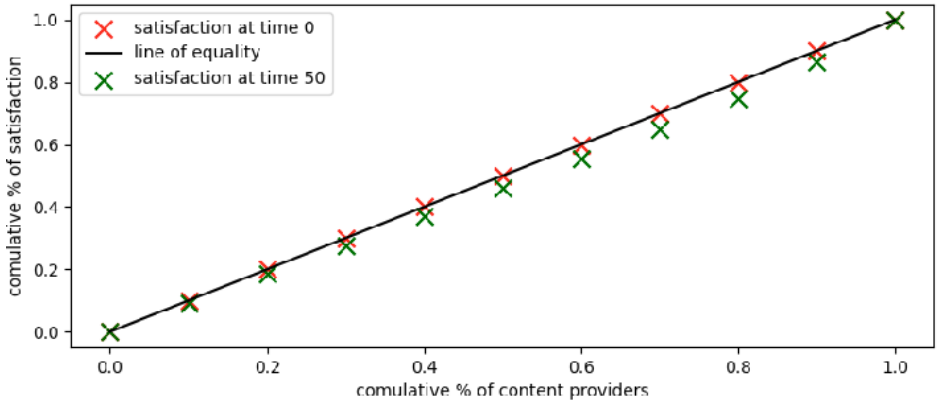


Figure 4.12: Visual representation of the Gini Coefficient at time step 0 and 100 of EcoAgent with **proactive post-processing** using the Lorenz curve.

As a result of the application of the post-processing method described above on EcoAgent, we reached the goal of getting a system able to keep the majority of content providers on the platform. In fact, from the evaluation of the agent behavior on the environment with a 100-step long simulation, we get at the end of the simulation the totality of content providers. However,

checking the median value of the content providers' satisfaction in Figure 4.11 at the beginning and at the end of the simulation, we can see that a time step 100 we still have a median value closer to the bottom of the box, but we can see how the box plot is smaller than the one obtained with the previous approach, so the satisfaction is less dispersed. We do not obtain a perfectly symmetric distribution of the satisfaction because of the unfairness tolerance, which allows some content provider to gain more satisfaction on some time step. However, we can see in Figure 4.12 that the Gini coefficient value is equal to 0.0596, which means that the system is pretty fair towards content providers. Observing the Lorenz curve in Figure 4.12 we can see how the suppliers' satisfaction at the end of the simulation does not fall far from the equality line. To sum up, we can say that by applying the proactive post-processing approach to EcoAgent, we get a system that can keep content providers on the platform and behaves fairly towards them.

4.4.3 Post-Processed Simulation and Results

In the previous section, we have seen that from the application of the **reactive** and **proactive** methods upon EcoAgent, we get two systems:

- **reactive EcoAgent**, able to maximize the number of content providers available, but with a low degree of fairness towards them.
- **proactive EcoAgent**, able to maximize the number of content providers available while acting almost perfectly fairly towards them.

Therefore, we can proceed with the analysis of the simulated environment throughout the simulation in order to study how they affect the environment's elements such as the users, content providers, and the document catalog, with a particular focus on the distribution of topics in the document catalog. Again, we chose to test the agents over a 100-steps episode to observe the behavior of the system during a long-term interaction among parties. This will allow us to understand how the reactive and proactive

agents, respectively, affect the catalog of documents of the environment in order to answer the two questions we asked ourselves initially, namely:

- does a system that keeps content providers on the platform provide a healthier environment to users?

- does a fair system towards content providers provide a healthier system to users?

- How does unfairness towards content provider reflects in the creation of biases?

Does a system that keeps content providers on the platform provide a healthier environment to users, with regard to their exposure to topics? To answer this question, we proceed with the analysis of the simulated environment resulting from running the **reactive EcoAgent**, which guarantees to keep all content providers on the platform, but does not pay attention to be fair. In fact, it presents a Gini coefficient $G_{100} = 0.86$ which is just a little bit above that of EcoAgent. So, the only difference between these two systems is the number of content providers available throughout the simulation. From the analysis of the distribution of topics recorded during the simulation, shown in 4.13, we can see how using the reactive EcoAgent in the simulated environment, we get one topic, specifically *topic10*, prevails over the others. This result is very reminiscent of the result observed from running EcoAgent in the same environment. Thus, the answer to this question is no, guarantying keeping all content providers on the platform does not ensure to provide a healthier system for users, since also in this case they will very likely be exposed to documents on *topic10*, reducing over time the number of topics they are exposed to.

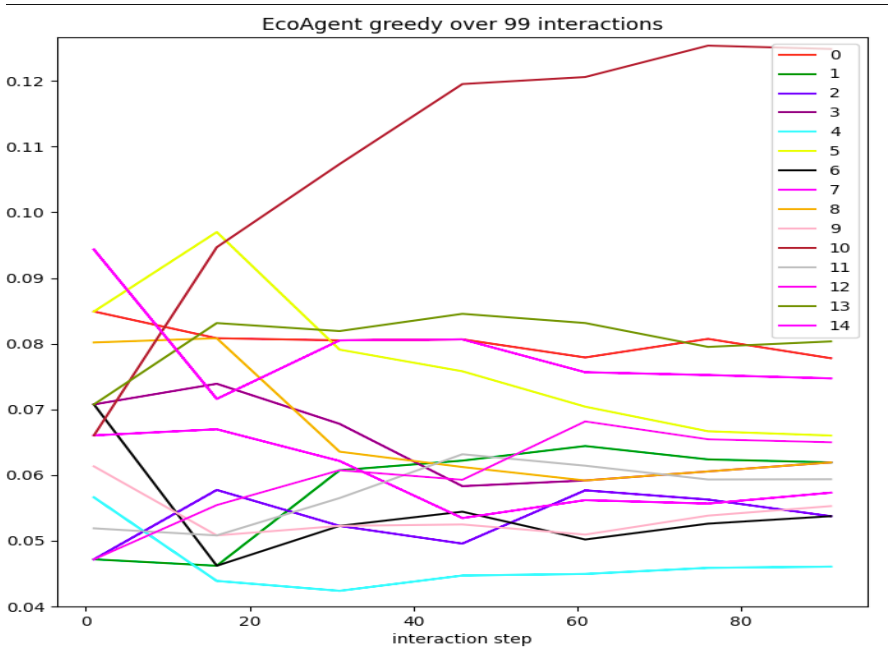


Figure 4.13: Overall topic distribution of documents in the simulation environment over interactions with the **Reactive Agent**

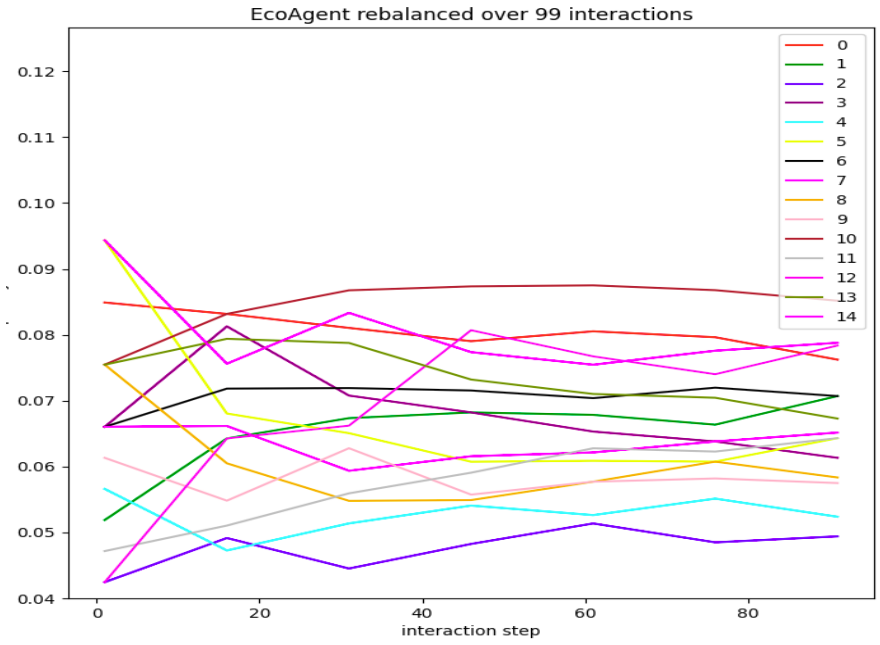
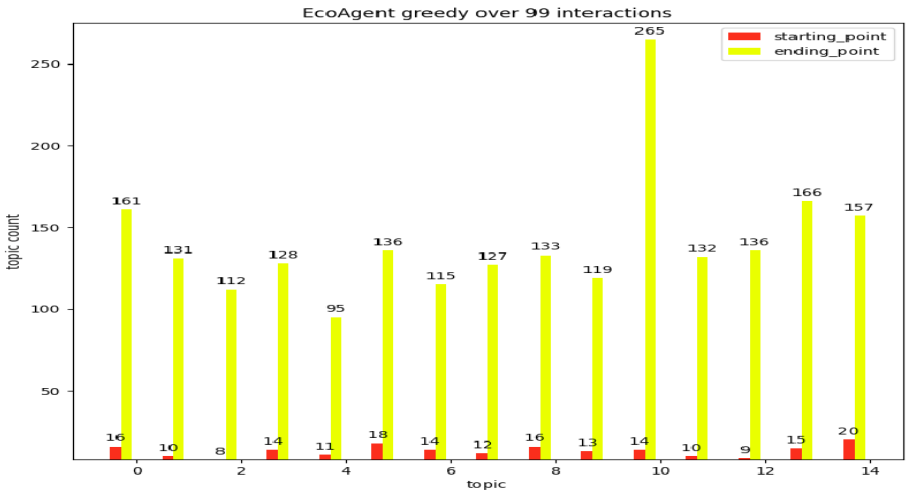
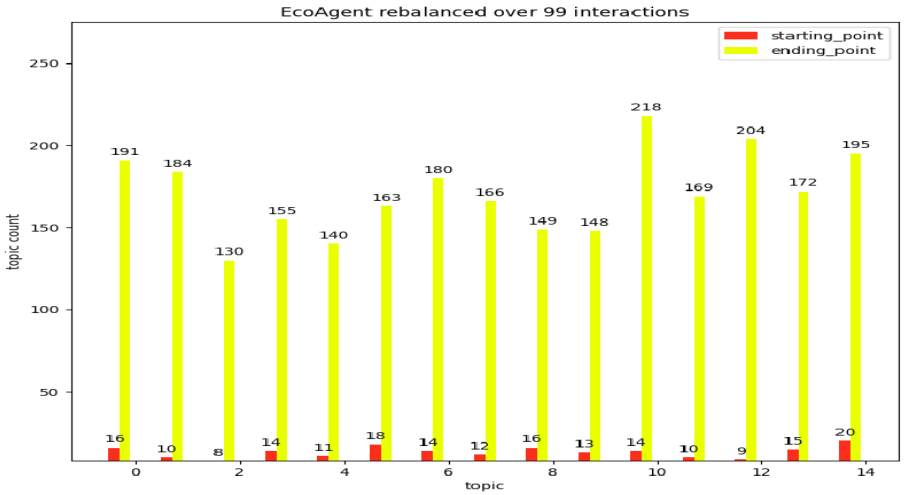


Figure 4.14: Overall topic distribution of documents in the simulation environment over interactions with **Proactive Agent**.



(a) **Reactive Agent**



(b) **Proactive Agent**

Figure 4.15: Number of documents for each topic at the beginning and at the end of the simulation

Does a fair system towards content providers offer a healthier system to users, with regard to their exposure to topics?
To answer this question, we compared the results obtained from the **reactive EcoAgent** and the **proactive EcoAgent**. Both these systems satisfy the requirement of keeping the maximum number of content providers, while they differ in the fairness degree towards content providers. Indeed, while reactive EcoAgent presents a Gini coefficient $G = 0.86$, the proactive Gini coefficient

G = 0.05.So, comparing the results obtained using these two agents on the same simulation environment allows us to define how the *fairness* of the system towards content providers is a discriminating element in creating a healthier environment for users. Contrary to what was observed in Figure 4.13 where the reactive agent was run, in the environment resulted using the proactive EcoAgent, shown in Figure 4.14, we do notice different probability distribution, but none of the topics is prevailing over the others. The same findings can be seen, from a different point of view, in Figure 4.15b, where we can see how in the Reactive agent's environment the sum of documents covering *topic10* is definitely higher than the others. Thus, the answer to this question is yes, providing a healthier environment for content providers helps to provide a healthier environment for users regarding their exposure to topics as in this case we no longer notice a reduction in the number of topics users are exposed to.

How does unfairness towards content provider reflects in the creation of biases? To answer this question, we have further analyzed the contribution of content providers to the corpus of documents. Using the Reactive EcoAgent we can observe that *cp7* in Table 4.5 owns most of the documents available on the platform, which is 1744 documents compared to other content providers that contribute with about 30 documents, which reflects the unfairness of the reactive agent. But more interestingly, we can see how it contributes most to the creation of documents concerning *topic10*. This result is particularly interesting because it allows us to see how favoring a single provider gives him the power to propose more and more frequently the content from which he received the user's approval initially. This not only creates a bias on some topics and reduces the diversity of available content, but also amplifies the risks of users being trapped in the echo chamber, where there is a single source of documents that risk being polarized, preventing the user from knowing all the facets of a given topic.

On the other hand, looking at Table 4.5 we can see that we still have some content provider that is more active in document creation, such as *content*

provider 5 and *content provider 8*, but unlike what happened in the Reactive EcoAgent's environment, in this case, the other suppliers are not cut off from the process and are still satisfied with the platform. This affects the system because to meet the fairness requirement between content providers, the system will also recommend content that deals with more niche topics encouraging suppliers to create new documents on that topic and therefore increasing the likelihood that they will reach users. Doing so increases the diversity of documents to which users are exposed and thus mitigates the popularity bias on topics of which EcoAgent suffers.

	Reactive Agent document catalog at step 100										Proactive Agent document catalog at step 100									
Topic	cp0	cp1	cp2	cp3	cp4	cp5	cp6	cp7	cp8	cp9	cp0	cp1	cp2	cp3	cp4	cp5	cp6	cp7	cp8	cp9
topic1	1	1	3	1	5	7	10	128	2	1	8	9	5	9	23	52	21	9	44	11
topic2	0	3	2	2	1	7	1	114	0	1	6	15	13	10	6	65	6	9	48	6
topic3	1	3	1	1	6	6	2	92	0	0	13	4	6	5	11	44	7	5	31	4
topic4	4	4	3	1	5	6	2	98	1	4	16	11	8	7	16	39	9	6	36	7
topic5	1	6	3	1	2	0	1	79	2	0	4	19	9	5	11	34	4	7	40	7
topic6	4	3	1	1	3	8	8	102	3	3	6	17	8	3	7	44	11	11	40	16
topic7	0	2	1	3	3	4	1	94	5	2	7	6	10	8	13	56	6	14	54	6
topic8	5	3	1	1	2	8	2	100	3	2	13	7	9	9	17	67	7	4	24	9
topic9	0	2	1	3	3	5	6	110	1	2	3	11	4	8	18	39	13	8	37	8
topic10	4	1	3	0	2	6	2	99	0	2	9	7	6	8	10	55	11	7	27	8
topic11	1	4	4	3	2	6	3	238	3	1	9	14	14	4	12	66	9	9	78	3
topic12	2	3	0	2	1	9	1	111	1	2	5	11	6	6	7	70	9	8	43	4
topic13	1	1	2	4	0	8	4	114	2	0	4	9	9	12	10	107	7	5	36	5
topic14	0	1	1	2	4	6	1	143	4	4	2	6	6	6	21	65	7	11	41	7
topic15	2	5	0	2	3	15	3	122	3	2	6	14	7	8	12	76	9	6	46	11
Total	26	42	26	27	42	101	47	1744	30	28	111	160	120	108	194	879	136	119	625	112

Table 4.5: Content providers' contribution to the document catalog at the end of the simulation for the **Reactive Agent** on the left and the **Proactive Agent** on the right.

4.5 Discussion

To summarize, in this chapter, we study a content provider-aware recommendation agent that aims to maximize the combined user and content provider utilities, which we call EcoAgent. We conducted a series of experiments to identify scenarios where a content provider-aware recommender

system can lead to the creation of some biases in the environment that could undermine user behavior in long-term interaction with recommendation systems. In particular, we have identified the presence of a bias on the available topics, specifically, we have noticed that a particular topic, is created with a greater frequency compared to the others in the course of the simulation. Moreover, from the analysis of the experiments, we have noticed two trends of the RS: that is its tendency not to maximize the number of content providers available on the platform and to treat them in an unfair way. So we tried to see if and how changing the behavior of the system towards the content provider could have an effect in the mitigation of the bias detected, and therefore, what was the role of content providers in the generation of this bias.

The results show that providing a platform with numerous content providers does not help to create a healthy environment with reference to the topics proposed by the platform. Instead, it makes a difference to provide a *healthy* environment for content providers, where everyone is treated fairly. In fact, by receiving recommendations, content providers are encouraged to create new documents according to their preferences which include more niche topics, thus increasing the diversity of the proposed topics. In addition, multiple vendors creating documents on the same topics decreased the likelihood that documents on a particular topic are polarized, and thus the user can access different viewpoints. In this way, we reduce the elements that risk trapping users in the echo chamber. Instead, as a result of the mitigation of popularity bias over topics, we have a more diverse catalog from which to choose a recommendation for users. This allows us to satisfy the preferences of the users of which we are aware, but also to verify their position on those not yet mentioned, thus increasing the probability of having novel or serendipitous recommendations.

Conclusion and Future Work

This thesis work aims at studying the impact that recommendation systems have on society. First, we tried to provide a sufficiently solid theoretical basis to allow a full understanding of the tools used in the thesis. This includes an exploration of the principles of Reinforcement-Learning and an examination of possible approaches that can be used to solve the RL problem. Followed by a general overview of the different types of recommender system available in the literature, and an analysis of the undesirable effects that they may have on users as a side effect of long-term interaction.

Next, to study the role played by recommender system in the creation of these effects, a particular RS, *EcoAgent*, is introduced. EcoAgent, in fact, is a multi-stakeholder recommendation system based on RL, able to grasp the dynamics between user-agent-cps, and therefore to represent in the most realistic way possible a real-world application. Moreover, by running simulations in which the agent interacts in an environment composed of users and suppliers, we noticed the tendency of the agent not to maximize the number of content providers available on the platform and to treat them unfairly, plus the presence of a popularity bias on some topics available on the platform. This scenario has been identified as unhealthy, obviously towards content providers, but also towards users. In fact, they are led to interact more and more frequently with documents on a given topic, which have most likely been created by the same supplier, with the consequent amplification of his point of view on the subject; that recalls the definition of echo chamber.

In conclusion, we implemented two post-processing techniques that change the agent's behavior towards content providers, first creating a platform that maximizes the number of providers available and then one that behaves correctly towards them. The results show that providing a *healthy* environment for content providers, where everyone is treated fairly, is reflected in mitigating the biases of popularity found on topics. In fact, by receiving recommendations, content providers are encouraged to create new documents based on their preferences that include more niche topics, thus increasing the diversity of the proposed topics. In addition, since multiple vendors create documents on the same topics, the possibility that documents on a particular topic are biased decreases, and thus the user can access different viewpoints. This reduces the risk of users being trapped in the echo chamber. Instead, as a result of the mitigation of popularity bias over topics, we have a more diverse catalog from which to recommend items for users. This allows us to satisfy users' preferences that we are aware of, but also to verify their position on those not yet mentioned, thus increasing the probability of having a novel or serendipitous recommendations.

However, EcoAgent is a simplistic attempt to capture the complex dynamics between different stakeholders and to study how they affect the environment in which they work. Here, we focused on analyzing the agent's effect on the document catalog, but there are several entities present in the environment that can be studied to ensure a healthy multi stake-holder recommendation system. For example, another possible study path could be to create an RS able to meet some requirements on the diversity of the elements present in a slate of recommendation, for instance ensuring to have a high value of intra-list similarity, which measure the degree of diversity between the elements of a recommendation slate by calculating the average cosine similarity. An RS that satisfy this constraint, would guarantee the user to not be subject to a flattening of his interests, but it would be interesting to study how this would affect the satisfaction of users and suppliers.

Bibliography

[Gol+92] David Goldberg et al. “Using collaborative filtering to weave an information tapestry”. In: *Communications of the ACM* 35.12 (1992), pp. 61–70.

[WD92] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8.3 (1992), pp. 279–292.

[Res+94] Paul Resnick et al. “Grouplens: An open architecture for collaborative filtering of netnews”. In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. 1994, pp. 175–186.

[KLM96] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[RV97] Paul Resnick and Hal R Varian. “Recommender systems”. In: *Communications of the ACM* 40.3 (1997), pp. 56–58.

[Bur00] Robin Burke. “Knowledge-based recommender systems”. In: *Encyclopedia of library and information systems* 69.Supplement 32 (2000), pp. 175–186.

[Gol+01] Ken Goldberg et al. “Eigentaste: A constant time collaborative filtering algorithm”. In: *information retrieval* 4.2 (2001), pp. 133–151.

[Bur02] Robin Burke. "Hybrid recommender systems: Survey and experiments". In: *User modeling and user-adapted interaction* 12.4 (2002), pp. 331–370.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions". In: *IEEE transactions on knowledge and data engineering* 17.6 (2005), pp. 734–749.

[Che+08] Long-Sheng Chen et al. "Developing recommender systems with the consideration of product profitability for sellers". In: *Information Sciences* 178.4 (2008), pp. 1032–1048.

[FB08] Alexander Felfernig and Robin Burke. "Constraint-based recommender systems: technologies and research issues". In: *Proceedings of the 10th international conference on Electronic commerce*. 2008, pp. 1–10.

[DMR09] Aparna Das, Claire Mathieu, and Daniel Ricketts. "Maximizing profit using recommender systems". In: *arXiv preprint arXiv:0908.3633* (2009).

[Par11] Eli Pariser. *The filter bubble: What the Internet is hiding from you*. penguin UK, 2011.

[RRS11] Francesco Ricci, Lior Rokach, and Bracha Shapira. "Introduction to recommender systems handbook". In: *Recommender systems handbook*. Springer, 2011, pp. 1–35.

[Ado+13] Gediminas Adomavicius et al. "Do recommender systems manipulate consumer preferences? A study of anchoring effects". In: *Information Systems Research* 24.4 (2013), pp. 956–975.

[Bob+13] Jesús Bobadilla et al. "Recommender systems survey". In: *Knowledge-based systems* 46 (2013), pp. 109–132.

[VDS13] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. "Deep content-based music recommendation". In: *Advances in neural information processing systems* 26 (2013).

[IFO15] Folasade Olubusola Isinkaye, Yetunde O Folajimi, and Bolande Adefowoke Ojokoh. "Recommendation systems: Principles, methods and evaluation". In: *Egyptian informatics journal* 16.3 (2015), pp. 261–273.

[Sil15] David Silver. *Lectures on Reinforcement Learning*. URL: `https://www.davidsilver.uk/teaching/`. 2015.

[Agg+16] Charu C Aggarwal et al. *Recommender systems*. Vol. 1. Springer, 2016.

[Bro+16] Greg Brockman et al. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[Bur+16] Robin D Burke et al. "Towards multi-stakeholder utility evaluation of recommender systems." In: *UMAP (Extended Proceedings)* 750 (2016).

[DB16] Robin Devooght and Hugues Bersini. "Collaborative filtering with recurrent neural networks". In: *arXiv preprint arXiv:1608.07400* (2016).

[ABM17] Himan Abdollahpouri, Robin Burke, and Bamshad Mobasher. "Recommender systems as multistakeholder environments". In: *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*. 2017, pp. 347–348.

[Aru+17] Kai Arulkumaran et al. "Deep reinforcement learning: A brief survey". In: *IEEE Signal Processing Magazine* 34.6 (2017), pp. 26–38.

[Bur17] Robin Burke. "Multisided fairness for recommendation". In: *arXiv preprint arXiv:1707.00093* (2017).

[NDK17] Phong Nguyen, John Dines, and Jan Krasnodebski. "A multi-objective learning to re-rank approach to optimize online marketplaces for multiple stakeholders". In: *arXiv preprint arXiv:1708.00651* (2017).

[Liu+18] Feng Liu et al. "Deep reinforcement learning based recommendation with explicit user-item interactions modeling". In: *arXiv preprint arXiv:1810.12027* (2018).

[Möl+18] Judith Möller et al. "Do not blame it on the algorithm: an empirical assessment of multiple recommender systems and their impact on content diversity". In: *Information, Communication & Society* 21.7 (2018), pp. 959–977.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Zha+18] Xiangyu Zhao et al. "Recommendations with negative feedback via pairwise deep reinforcement learning". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2018, pp. 1040–1048.

[Abd+19] Himan Abdollahpouri et al. "The unfairness of popularity bias in recommendation". In: *arXiv preprint arXiv:1907.13286* (2019).

[Bou+19] Dimitrios Bountouridis et al. "Siren: A simulation framework for understanding the effects of recommender systems in online news environments". In: *Proceedings of the conference on fairness, accountability, and transparency*. 2019, pp. 150–159.

[Ie+19] Eugene Ie et al. "Recsim: A configurable simulation platform for recommender systems". In: *arXiv preprint arXiv:1909.04847* (2019).

[Zha+19] Shuai Zhang et al. "Deep learning based recommender system: A survey and new perspectives". In: *ACM Computing Surveys (CSUR)* 52.1 (2019), pp. 1–38.

[Abd+20] Himan Abdollahpouri et al. "Multistakeholder recommendation: Survey and research directions". In: *User Modeling and User-Adapted Interaction* 30.1 (2020), pp. 127–158.

[Che+20] Jiawei Chen et al. "Bias and debias in recommender system: A survey and future directions". In: *arXiv preprint arXiv:2010.03240* (2020).

[Mah20] Batta Mahesh. "Machine learning algorithms-a review". In: *International Journal of Science and Research (IJSR).[Internet]* 9 (2020), pp. 381–386.

[Man+20] Masoud Mansoury et al. "Feedback loop and bias amplification in recommender systems". In: *Proceedings of the 29th ACM international conference on information & knowledge management*. 2020, pp. 2145–2148.

[MTF20] Silvia Milano, Mariarosaria Taddeo, and Luciano Floridi. "Recommender systems and their ethical challenges". In: *Ai & Society* 35.4 (2020), pp. 957–967.

[ACF21] M Mehdi Afsar, Trafford Crump, and Behrouz Far. "Reinforcement learning based recommender systems: A survey". In: *arXiv preprint arXiv:2101.06286* (2021).

[Ela+21a] Mehdi Elahi et al. "Investigating the impact of recommender systems on user-based and item-based popularity bias". In: *Information Processing & Management* 58.5 (2021), p. 102655.

[Ela+21b] Mehdi Elahi et al. "Towards responsible media recommendation". In: *AI and Ethics* (2021), pp. 1–12.

[Luc+21] Eli Lucherini et al. "T-RECS: A simulation tool to study the societal impact of recommender systems". In: *arXiv preprint arXiv:2107.08959* (2021).

[Zha+21] Ruohan Zhan et al. “Towards Content Provider Aware Recommender Systems: A Simulation Study on the Interplay between User and Provider Utilities”. In: *Proceedings of the Web Conference 2021*. 2021, pp. 3872–3883.

Acknowledgments

I would like to thank my supervisors, Dr. Asia Biega, for her guidance during the development of this work and her exceptional availability in brainstorming with me any possible ideas, and Prof. Ugo Dal Lago for the experience and insight shared and the time dedicated to the progress of this research.

My biggest thanks to my family for all the support you have shown me throughout my studies and in all my decisions. Finally, I thank my friends: from my lifelong friends to those I met on my journey and who decided to walk by my side.

To you my most sincere love.