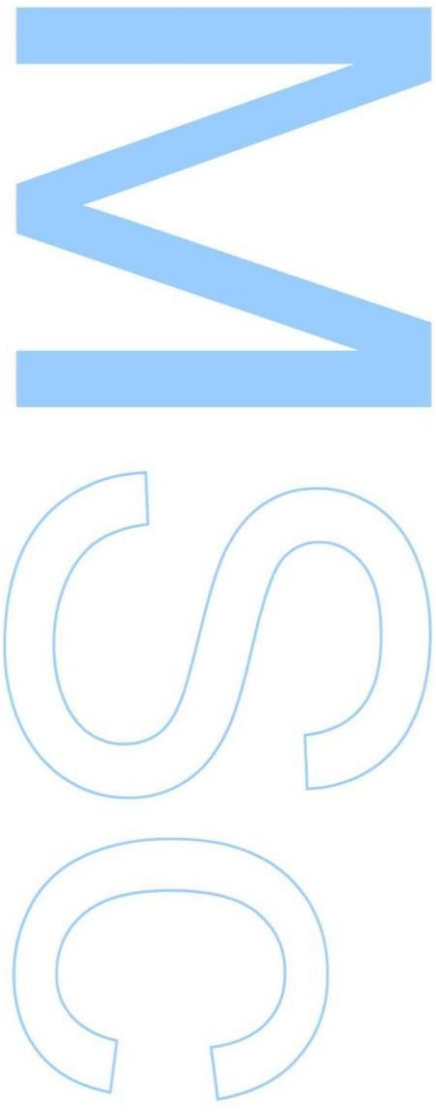
Next Best Action Recommendation

Rui Jorge Eduardo Ramos
Mestrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência de Computadores
2022

Orientador
João Nuno Vinagre Marques da Silva, Professor Auxiliar Convidado,
Faculdade de Ciências da Universidade do Porto



Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto,_____/_____/_____

Acknowledgments

And so, as if in the blink of an eye, five years have passed. I remember the beginning of this journey as if it were yesterday. Somewhere in September 2017, I walked through the door of the Biology Department to make my enrolment as a official FCUP student. Since then it was all a carousel of emotions.

First of all, I have to thank my family and my girlfriend Inês, who has been with me even before I joined FCUP, as they were my great emotional pillars during this journey. I would not be here if they did not provide me with all they could, regardless of the cost. I also want to thank my closest friends, who have always been with me since this all started.

Next, I would like to thank my tutor João Vinagre, as I could not have made a better choice as the person who would guide me in this final year. He proved to be always present, responsible and dedicated to answer all my questions and help me with any difficulty throughout the academic year.

During this journey, I also joined INESC TEC, first with an internship, followed by several research opportunities until today. I would like to give special thanks to Lino Oliveira, my research supervisor during my period at INESC TEC as a research scholarship holder, and also to thank him for the opportunity to continue in the institution after the end of my master's degree.

Finally, but most especially, I want to thank my grandfather José Eduardo. If today I am the man I am, it is much thanks to him. I carry with me all the lessons he taught me to this day. He will always be my biggest hero. My eternal gratitude to him.

Thank you all and may the future come.

To my father Avelino Ramos, my mother Maria José Ramos, my brother Luís Ramos, my girlfriend Inês Dias, my grandmother Lucinda Eduardo, my godmother Carla Eduardo, my close friends Rui Lopes, André Tse and Rúben Lôpo, my supervisor João Vinagre, my co-workers Lino Oliveira, Mafalda Castro and Alexandre Costa, who always supported me during this long journey, and especially to my grandfather José Eduardo, who will always be my biggest reference in everything I do ...

Abstract

In a sequential decision-making paradigm, a user needs to choose the most appropriate actions during the various stages of a flow and it is important that these actions are chosen in the best possible way to obtain good results. We, as human beings, spend a lot of time listening to music, and choosing the next song to listen to is not always easy. This choice is influenced by a number of factors, one of the most important being the user's individual preferences.

This dissertation presents a solution, named SkipAwareRec, which aims at solving the problem of choosing the next best action in sequential decision-making paradigms. By collecting negative implicit feedback, in the form of skips, in an iterative way, our algorithm is able to recommend the next best musical category to listen to and specific songs taking that category into account. The presented results show that our solution adds value to the sequential recommendation landscape and can be embedded in a music streaming platform.

Our contributions are based on a sequential recommendation algorithm of the next best action, based on Reinforcement Learning, whose learning is done by interpreting quantified implicit negative feedback in the form of skips. SkipAwareRec also has a second phase, where the recommendation of specific items is generated taking into account the recommended action, through a hybrid incremental recommendation algorithm.

Keywords: Sequential Recommendation, Next Best Action Recommendation, Implicit Negative Feedback, Reinforcement Learning.

Resumo

Num paradigma de tomada de decisão sequencial, um utilizador necessita escolher as ações mais apropriadas durante as várias etapas de um fluxo e é importante que essas ações sejam escolhidas da melhor forma possível para obter bons resultados. Nós, enquanto seres humanos, passamos muito tempo a ouvir música, e escolher a próxima canção a ouvir nem sempre é fácil. Esta escolha é influenciada por uma série de fatores, sendo um dos mais importantes as preferências individuais do utilizador.

Esta dissertação apresenta uma solução, de nome SkipAwareRec, que visa em resolver o problema de escolher a próxima melhor ação em paradigmas de tomada de decisão sequenciais. Através da recolha de feedback implícito negativo, na forma de *skips* (a ação de saltar um determinado item), de forma iterativa, o nosso algoritmo é capaz de recomendar a próxima melhor categoria musical a ouvir e canções específicas tendo essa categoria em conta. Os resultados apresentados mostram que a nossa solução acrescenta valor ao panorama da recomendação sequencial e pode ser inserida numa plataforma de streaming musical.

As nossas contribuições baseiam-se num algoritmo de recomendação sequencial da próxima melhor ação, baseado em Aprendizagem por Reforço, cuja aprendizagem é feita através da interpretação de feedback implícito negativo quantificado na forma de *skips*. O SkipAwareRec conta ainda com uma segunda fase, onde são geradas recomendações de itens específicos tendo em conta a ação recomendada, através de um algoritmo de recomendação incremental híbrido.

Palavras-Chave: Recomendação Sequencial, Recomendação da Próxima Melhor Ação, Feedback Implícito Negativo, Aprendizagem por Reforço.

Contents

List of Tables

List of Figures

List of Algorithms

Acronyms

AI Artificial Intelligence

APG Automatic Playlist Generation

CF Collaborative Filtering

CNN Convolutional Neural Network

FMDP Finite Markov Decision Process

GRU Gated Recurrent Unit

HR Hit Ratio

ISGD Incremental Stochastic Gradient Descent

MAB Multi-Armed Bandits

MDP Markov Decision Process

ML Machine Learning

NBA Next Best Action

PCA Principal Component Analysis

RL Reinforcement Learning

TD Temporal Difference

WMF Weighted Matrix Factorisation

Chapter 1

Introduction

The decision-making process accompanies us on a day-to-day basis, many times without us realising it. We have all faced various types of decisions, such as choosing what time to take the train, what clothes to wear, or even what movie to watch.

Decision making is a cognitive process that results in the selection of one option among several alternatives. Whenever we decide on something, we try to choose the best possible option in order to get the best return for the decision we make. Nevertheless, it is sometimes difficult to make a decision that is right and meets our expectations.

This dissertation aims to help solving the problem of choosing the Next Best Action (NBA) recommendation in sequential decision music stream paradigms, using quantified implicit negative feedback in the form of skips, allowing humans to easily identify the best songs to listen to at particular moments of a stream.

1.1 Motivation

Music helps in our intellectual development as well as in stimulating creativity and also in the possibility of expressing our various feelings through sounds [4]. It is recognised by many researchers as a kind of modality that develops the human mind, promotes balance, providing a pleasant state of well-being, facilitating concentration and the development of reasoning, especially in reflective issues aimed at philosophical thought. Due to of all the benefits of music, we as humans spend an average of a lot of time listening to it [5], and choosing the next song to listen to is not always easy. The music that we choose to listen to is influenced by a number of things, one of the most important of which is individual user preferences.

In sequential decision-making paradigms, people need to choose the most appropriate actions in the various steps of a flow and it is important that those actions are taken in the best way possible in order to obtain good results. However, choosing the finest solutions during this process is not always easy, considering that decision-making is filled with issues that are not

always easy to overcome. For example, in many applications, the quantity of alternative options might be overwhelming, analogously to the exponential growth in the number of songs available on digital music platforms, making us more likely to choose the wrong or less appropriate action.

Recommender system techniques use various types of information to generate the best possible recommendations. In a real scenario, a recommender system is in constant interaction with a user and the recommendation task can be seen as an iterative process: the user receives a recommended item and provides feedback to the system, and in the next iteration, the recommendation model incorporates the feedback received by the user and makes a new recommendation. The key aspect for this is data: while the software of recommender systems is identical for all users, their behaviour must constantly be adapted based on the experiences of different individual users.

Despite the notorious importance of user iterative and sequential feedback for these systems, many of those that currently exist do not take it into account, as they completely ignore the interaction with the user. Consequently, they become unable to update the recommendation models in real time with new information, thus generating less satisfactory results. In addition to this, the few contributions that use sequential and iterative recommendation techniques use only explicit feedback, inputs given by users according to their preferences for a item, or implicit positive feedback, users' positive interests acquired through indirectly monitoring its behaviour. There is a lack of iterative sequential recommendation contributions using negative implicit feedback, especially when it is quantified and can take various values with different meanings.

1.2 Main Goals

The ability of recommender systems to adapt their behaviour to individual preferences is becoming more and more valuable as their relevance to humans increases [9]. The proposed solution aims to study and adapt the sequential decision-making strategies to create a next-action recommendation system that interacts in real time with the user. More specifically, our solution is named SkipAwareRec, a Reinforcement Learning (RL) based Recommender System whose objective is to recommend the best possible next music categories, at certain moments of a music stream, by collecting quantified negative implicit feedback during the interaction of the recommender agent with the user. It gets easier to choose one song, or multiple songs, to recommend after having a musical category to recommend and the initial set of alternatives decreased to a much smaller one.

Therefore, the goal of our research is to address the following research questions:

- **Q1**: How does the implicit negative feedback – i.e. the skipping of songs – reflect the user's opinion on them and how is it related to other context information associated with music listening sessions?
- **Q2**: How to create a recommendation system that learns user preferences in real time through collected quantified implicit negative feedback?

- **Q3**: Is the online category selection able to improve recommendations in an incremental setting?

Having that in mind, for this work the main goals are:

- Study and perform the best way to execute a song clustering, exploring the attributes present in data and understanding which is the best way to create relationships between songs.
- Collect, clean, standardise and interpret the information that will be used to feed an Artificial Intelligence (AI) algorithm that support the basis of the recommender system, in order to conclude how the best actions are chosen by humans, by learning user preferences using RL.
- Explore the best way to create a MDP and a reward function that internalises quantified implicit negative feedback, and study how to link the RL paradigm to a next best action recommender system.
- Generate recommendations through the proposed tool in conjunction with an incremental recommendation algorithm.

1.3 Contributions

This dissertation will have the following contributions:

- We developed an iterative sequential next best action recommender system whose learning is based on the interpretation of quantified implicit negative feedback. In a specific music recommendation landscape, our work is distinguished by the use of negative implicit feedback to generate recommendations. In a general overview of recommendation, we distinguish ourselves through the use of quantified implicit negative feedback to generate recommendations.
- We present a hybrid recommendation solution, SkipAwareRec, with two components (Figure 1.1): the recommendation of the best possible musical category at a given moment of a stream, and the specific recommendation of items having in account the recommended category, preserving the incrementality of the solution.

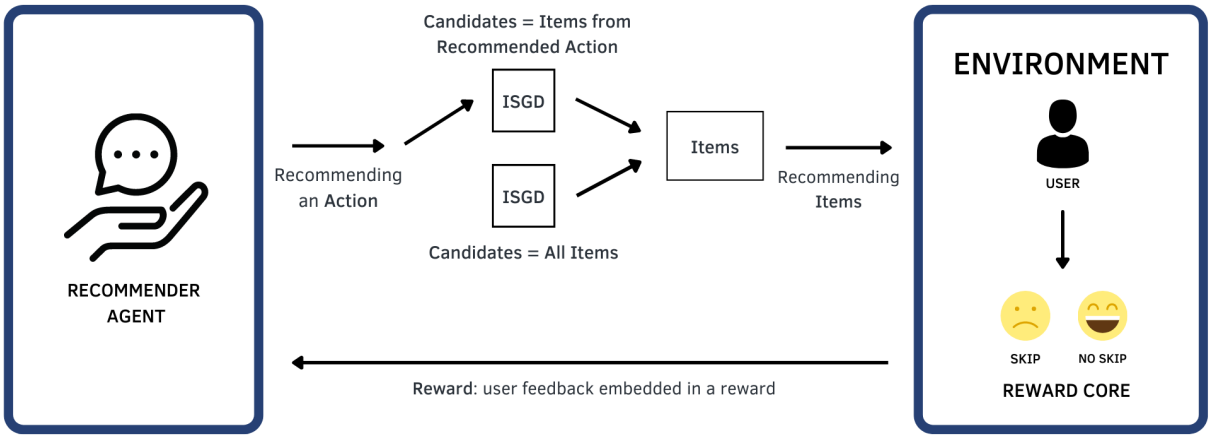


Figure 1.1: SkipAwareRec overview

1.4 Dissertation Layout

The remainder of the dissertation is organised as follows. Background information essential for a full understanding of this work is provided in Chapter 2. Chapter 3 presents a literature review of sequential recommendation techniques that currently exist, specifically applied to a musical landscape. In addition to this, there is also a literature review of recommendation techniques that use implicit negative feedback and how this type of feedback has been explored in the music recommendation landscape. Chapter 4 explains the strategy used to generate the set of musical actions/categories. Chapter 5 demonstrates the composition of the RL task, from the formulation of the MDP and all its components, to the learning itself and the recommendations generated - the first part of SkipAwareRec. Chapter 6 discusses the incremental algorithm that was adapted to generate recommendations of specific items - the second part of SkipAwareRec. Chapter 7 presents a case study applied to this problem and its results. Finally, in chapter 8 the conclusions and future work are presented.

Chapter 2

Fundamental Concepts

Intelligence is the ability to solve new situations efficiently and successfully, with adaptation to them through acquired knowledge, i.e., it is the art of understanding, thinking, reasoning, and interpreting [2]. The word Artificial refers to something produced by art or industry of the human being, and not by natural causes. Artificial Intelligence is the ability of software devices to work similarly to human thought, becoming agents capable of simulating human actions and thinking logically, making it possible to create solutions for many different aspects of our lives.

With the modernisation of society and the evolution of technology, human beings have begun to think of new ways to automate their routine and repetitive jobs in order to save resources and increase productivity, and thus the need for the use of AI has emerged. Besides this, this area emerges in several fields of study, being crucial in many of them, such as finance, security, health, justice, transportation, environment, among many others [1].

The basis of a Recommender System relies on several AI-related paradigms. In this chapter, an overview of what Machine Learning is and what types of tasks it consists of and is divided into will be presented. By exploring the key aspects of this topic, we will be able to recognise and understand how ML impacts the results generated by recommender systems. Moreover, this chapter also discusses the recommendation systems as a whole.

2.1 Machine Learning

Machine Learning (ML) is a sub-area of AI that can be seen as an automatic process of learning from data: the machine analyses all known observations through data entered for this purpose and extracts knowledge from them. With the knowledge formulated, ML algorithms become capable of solving various real problems, and these can be divided into three major sub-themes:

- **Supervised Learning**: Given a set of observations (described by variables) and a target variable, the goal of this learning is to find a function that best approximates the target

variable. In other words, it consists of scenarios where the system already knows which inputs are associated with which outputs, and the challenge is to understand how this association is made in new cases not yet known.

- **Unsupervised Learning**: Given a set of observations (described by variables), the goal of this learning is to find common patterns, or more specifically, implicit relationships in a data set of which the system has no information about the association between the input and output data.
- **Reinforcement Learning**: This learning is based on the maxim of "learning what to do", that is, how to map situations to actions in order to maximise a reward signal. In this paradigm, the machine is stimulated to learn based on trial-and-error search, and the process is optimised through practice, where the system learns to prioritise certain habits.

2.1.1 Supervised Learning

In supervised learning, the system has access to a dataset that has an input and the corresponding output and the challenge is to learn how this association is made.

The machine starts by receiving a set of N training examples, where each example i in the data set contains a feature vector ai and a corresponding label bi:

$$N = [(a1, b1), ..., (aN, bN)]$$

The goal is to learn a function *f:* $X \rightarrow Y$ that represents the relationship between input X and output Y as accurate as possible. In this way, through an input corresponding to a feature vector it is possible to discover with which label is associated.

The most common tasks associated with supervised learning are Classification and Regression. These two types of problems are quite similar, with the main difference being the type of label to be determined by the function *f*.

In Classification, the variable to be determined is categorical, that is, Y represents a nominal variable and the output is a single value, as we can see in Figure 2.1. This type of problem is usually divided into two types of classification:

- **Binary Classification**: The target variable takes on only two possible values (classes), usually referred to as positive class and negative class, for example "yes or no".
- **Multi-Class Classification**: The target variable takes on more than two possible classes, for example "small, large or medium". There are algorithms that cannot handle multi-classes and the alternatives are to combine several binary classifiers.

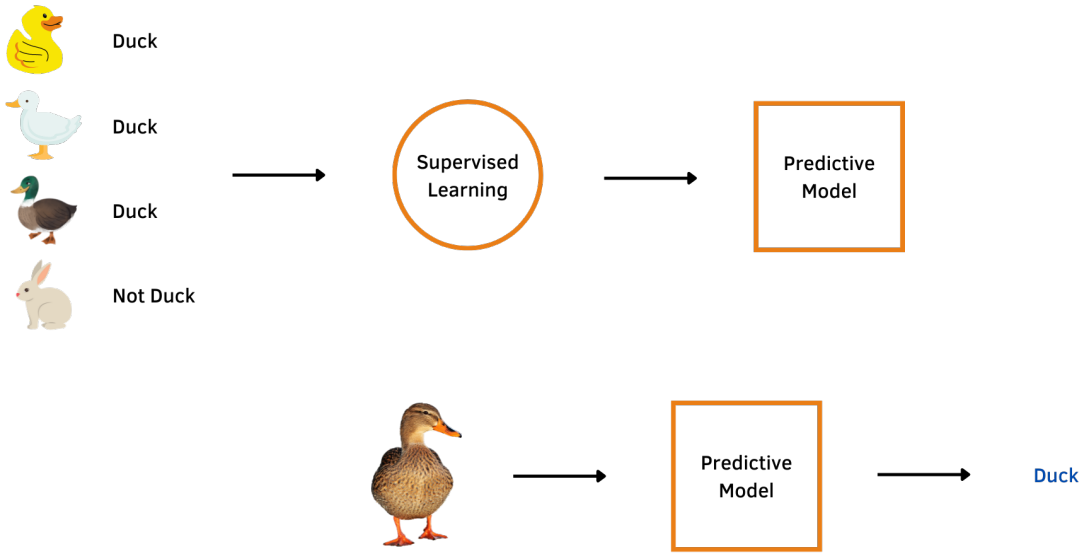


Figure 2.1: Supervised Learning - Classification Problem Example

As stated earlier, a regression task is similar to a classification task, the main difference being that the variable Y in this case is numeric, in other words, regression maps the input data object to continuous real values. Given that the target is numeric, an order used to differentiate the values is usually created - many regression algorithms operate under a form where the output is not a single value, but an ordered ranking of the best-fitting values.

2.1.2 Unsupervised Learning

In unsupervised learning, the system has no information about the association between the input and output data, in other words, the learning agent is only provided with unlabelled training examples. The machine starts by receiving a set of N training examples, where each example i in the data set contains a feature vector ai:

$$N = [(a1), ..., (aN)]$$

The goal is to uncover a structure present in the training set.

One of the best known methodologies of unsupervised learning is called Clustering. This technique aims to create clusters of observations that satisfy a specific condition, that is, that are similar in some way, as we can see in Figure 2.2. In this way, points with similar characteristics will be close together in space, within the same cluster.

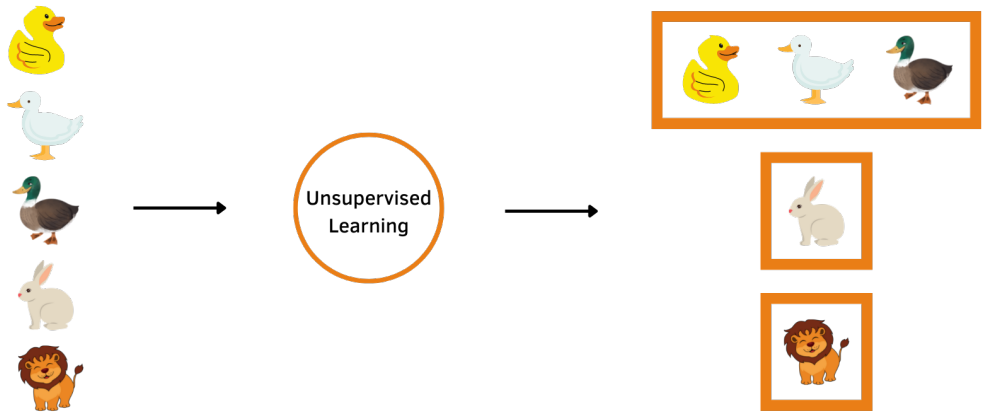


Figure 2.2: Unsupervised Learning - Clustering Problem Example

There are several ways to calculate the distance between points. Some of the best known formulas for calculating the distance between two points in space are the Manhattan Distance and the Euclidean Distance, which result in the Minkowski Distance, which is a metric that serves as a generalisation for the other two previous ones. This last metric is defined as such:

$$minDistance(x, y) = (\sum_{i=1}^{n} |xi - yi|^p)^{1/p}$$

It is important to note that when p=1, we are dealing with the Manhattan Distance, and when p=2, we are dealing with the Euclidean Distance.

Just as there are several ways to calculate the distance between points, there are also some ways to perform clustering, such as:

- **Partitioning clustering**: Techniques that subdivide data sets into a set of k groups (Figure 2.3), where k is a number predefined by the user. The goal is to minimise the distance of points within the same cluster and maximise the distances of points from different clusters.
- **Hierarchical clustering**: An alternative technique to partitioning that does not require pre-specification of the number of clusters to be generated. The goal is to obtain a hierarchy of groups (Figure 2.4), where each level represents a possible solution with x groups. This process can be agglomerative or divisive, depending on how the hierarchy is generated.

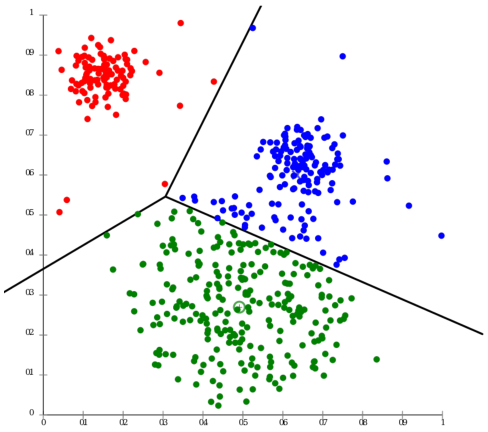


Figure 2.3: Partitioning clustering example

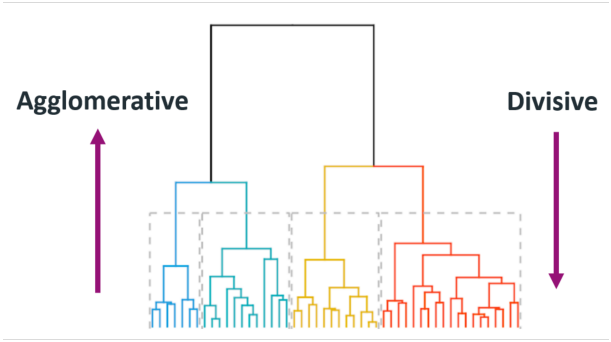


Figure 2.4: Hierarchical clustering example

2.1.2.1 K-means

The K-means algorithm is a partition-based clustering method that obtains k clusters from a dataset in such a way so that data points within the same cluster are as similar as possible (intra-class similarity), while data points from different clusters are as dissimilar as possible (inter-class similarity). In this technique, each cluster is represented by its center, called a "centroid", which corresponds to the arithmetic mean of data points assigned to the cluster. The algorithm is defined by the following steps:

1. Initialise the centres of the k groups to a set of randomly chosen observations.
2. Repeat:
   (a) Allocate each observation to the group whose center is nearest.
   (b) Re-calculate the center of each group.
3. Until the groups are stable, i.e. there is no significant decrease or there is an increase on the minimise criterion *h(C)* - this technique uses Euclidean distance as criterion.

On one hand, its advantage is that it is a fast algorithm that scales well. On the other hand, it does not always ensure an optimal clustering, because we need to previously find the optimal value of k.

2.1.3 Reinforcement Learning

Reinforcement learning is the most dynamic and different from the three types of learning discussed. It is based on the premise of "learning what to do", that is, learning how to map situations to actions in order to maximise a numerical reward signal. It is a learning where an agent interacts with an environment in a trial and error manner in order to broaden it knowledge horizon.

This approach is based on the trade-off between *Exploration* and *Exploitation* - the learner is not told what actions to take, but tries several and gets a reward. Exploration is the act of experiencing actions never performed before and Exploitation is the act of performing actions already performed in order to obtain rewards - with both of these techniques combined, the learner is able to obtain enough information to carry out it search for knowledge.

Rewards can be of various types and are commonly defined by the user. For example, in a helicopter flight, a positive reward would be to follow the desired trajectory and a negative reward would be to crash. In a game, a positive reward would be to win and a negative reward would be to lose.

RL is based on the concept of state, a signal that conveys to the agent some sense of "what the environment is like" at a given time. At each step, the agent performs an action and receives an observation and a reward from the environment, as we can see in Figure 2.5 - in short, it implements a Sequential Decision Problem where the agent's utility depends on a sequence of decisions and the goal is to maximise the future total reward.

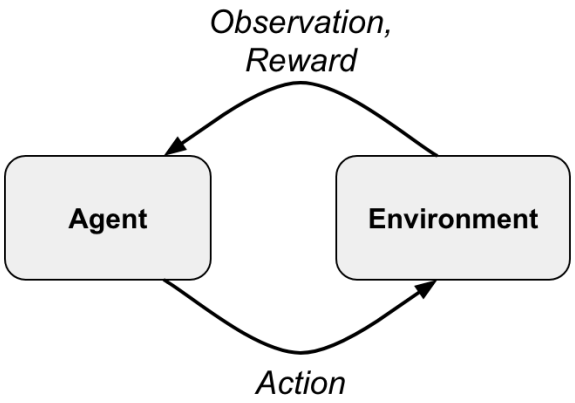


Figure 2.5: Reinforcement Learning Interaction Overview

2.1.3.1 Markov Decision Process

In an RL problem, the agent learns through interactions with the environment in order to maximise the total reward by sequential decision making. Markovian decision processes (MDP) formally describe an environment for RL, where this environment is completely observable, i.e., the agent directly observes the state of the environment.

Thus, it is possible to define an MDP as a reward process with decisions in a environment in which all states are Markov. Like a RL problem, an MDP has a policy, a value-function, and a model. Concretely, an MDP is a tuple <S, A, P, R, Y>, where:

- S is a finite set of states.
- A is a finite set of actions.
- P is a matrix of probability transactions between states.
- R is a reward function.
- Y is a discount factor - 0 or 1 (this parameter is used because it is mathematically convenient to discount rewards - discounting avoids infinite returns in cyclic Markov processes).

In an MDP there are transitions between states (that represent actions) and probabilities associated with each transition, representing the probability that an agent will jump from one state to another, and each state has a reward. For representing the probabilities of jumping from one state to another, there are State Transition Matrices, which as the name suggests, store these probabilities. These matrices define the probabilities of all states S for all their successors S', as we can see in Figure 2.6.

$$\mathcal{P} = \begin{matrix} \\ from \end{matrix} \begin{matrix} \begin{matrix} to \end{matrix} \\ \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix} \end{matrix}$$

Figure 2.6: State Transition Matrix

One of the most relevant conditions of this paradigm is that the future is independent of the past, given the present. In other words, a state St is considered Markov if and only if:

$$P[St+1|St] = P[St+1|S1, ..., St]$$

This condition is called the Markov Property and says that a state captures all the relevant information in the history, i.e. once a state is known, the whole history can be thrown away.

2.1.3.2 Policy and Value-Function

An agent is expected to gradually accumulate greater rewards during each interaction, therefore actively learning by reinforcement. Each state is followed by an action, which in turn leads to the next stage and reward.

A policy is a strategy that an agent employs to achieve its objectives - it dictates a map determining what action to be taken at each state. In the simplest case, the policy for each state refers to an action that the agent should perform in that state - this type of strategy is called deterministic policy and can be displayed in a table, where an action can be selected in different states. In a more general way, a policy assigns probabilities to every action in every state and, as a result, it represents a probability distribution over all feasible actions for each state - such a strategy is called a stochastic policy. Several actions can be chosen in a stochastic policy, each with a probability of non-zero and the sum of all being 1. As a result, it is possible to say that an agent's behaviour can be characterised by a policy that assigns states to a probability distribution across actions.

The value-function denotes how valuable it is for the agent to be in a particular state - it describes the expected return from a given state. It is vital to note that a state value-function is defined in relation to a certain policy because the expected return is determined by it. RL relies heavily on value-functions, because it allows an agent to query the quality of his current situation rather than waiting for the long-term result. This has two great advantages: first, the return is not immediately available, and second, the return may be random due to the policy's stochasticity and the environment's dynamics. By averaging the returns, the value-function summarises all future possibilities and, as a result, it can be used to evaluate the quality of various policies.

2.1.3.3 Exploration versus Exploitation Trade-Off

As mentioned earlier, RL is trial-error learning where positive or negative rewards are received for actions taken. At the beginning of the problem, there is a lot of uncertainty and lack of knowledge about the environment and therefore the agent will have to explore the various actions available.

Here comes the trade-off between exploration and exploitation. Recalling what was mentioned earlier, exploration corresponds to the act of exploring actions never selected before, while exploitation consists in trying what has been tested before. It is important to have a healthy balance between these two techniques in order to avoid problems: if we only exploit, we may miss some unknown optimal actions (long-term sacrifices) and if we only explore, we may miss known rewards (short-term sacrifices).

One of the best known solutions concerning this trade-off problem is Epsilon Greedy. This paradigm consists of randomly selecting, with a 50% probability, whether to exploit or explore: 1-ϵ of the time the algorithm exploits and ϵ of the time the algorithm explores. Still, it is difficult to find an ϵ value that satisfies our needs regarding the desired balance: it is good to have early exploration, in order to quickly arrive at an optimal solution, but in the long run we want to exploit more knowledge. In some cases, we are limited to a fixed amount of time to perform the actions, which often means that there is not enough time to explore and find the optimal solution.

There are some alternatives to solve this problem, one of them being to use optimistic initial values that represent a way to encourage early exploration. The intuition is simple: we assume that the actions are good until we find out that they are not, i.e., this solution fares worse in the early stages, but finds the optimal action more often. Still, this approach brings another problem to the table, as we want to prioritise exploitation as our knowledge improves (as we do more exploration), and not just encourage early exploration.

In order to solve the problem of non-prioritisation of exploitation, there is an enhancement of the Eplison Greedy called the Decaying Epsilon Greedy. In this technique, a decay of the value of ϵ over time is defined, i.e. the more steps we perform, the more knowledge we gain and the less exploration we need. Since 1-ϵ defines the time in which we do exploitation, the smaller the epsilon is, the higher the exploitation will be.

In short, the problem of trade-off between exploration and exploitation is one of the important issues when it comes to RL, and should always be addressed. There are several ways to find a proper balance, and it is up to the user to choose the one that suits him or her best.

2.1.3.4 Temporal Difference Learning

One of the issues with the environment is that rewards are rarely visible immediately. In games like chess or others, we only know the rewards when we make the final move (terminal state), and all other actions will yield no immediate rewards.

Temporal Difference (TD) learning is an unsupervised technique to predict a variable's expected value in a sequence of states - is an agent learning from an environment through episodes with no prior knowledge of the environment. It is a type of model-free reinforcement learning method that learns by bootstrapping from the current value-function estimate.

TD Learning interpolates ideas from Dynamic Programming and Monte Carlo methods. These strategies take samples from the environment, like Monte Carlo methods, and update them depending on current estimates, similar to dynamic programming methods.

2.1.3.5 On-Policy versus Off-Policy Methods

On-policy methods attempt to evaluate or improve the same policy that is used to make decisions, that means it will try to evaluate and improve the same policy that the agent is already using for action selection. Some examples of on-policy algorithms are: Policy Iteration, Value Iteration and Sarsa.

Off-Policy learning algorithms evaluate and improve a policy that is different from policy that is used for action selection - Q-Learning is an example of this type of learning strategy. Using off-policy algorithms can have some benefits, such as:

- **Continuous exploration**, where an agent is learning other policy then it can be used for continuing exploration while learning optimal policy, whereas on-policy learns sub-optimal policy.
- **Learning from demonstration**: the agent can learn from the demonstration.
- **Parallel Learning**, enabling the acceleration of learning convergence.

2.1.3.6 Policy-Based, Value-Based and Actor-Critic RL

Policy-Based Reinforcement Learning: In policy-based reinforcement learning, the logic is to start with a random policy and, through this policy, find the first value-function. After this, we find a new policy through the value-function from the previous step, thus becoming a repetitive process that only stops when the optimal policy is found. In this type of RL, the policy is updated directly.

Value-Based Reinforcement Learning: In value-based approaches, the value-function is initially random and then a new value-function is found. Similar to policy-based RL, this is a repetitive process, however now until the optimal value-function is found. The intuition is that the policy that follows the optimal value-function will always be the optimal policy, and consequently the policy is implicitly updated from the value-function - the function that measures the goodness of the state (state-value) or how good is to perform an action from the given state (action-value).

- **Q-Learning**

Q-Learning is a value-based off-policy temporal-difference (TD) reinforcement learning algorithm whose goal is to learn the value of an action in a given state, by learning the action-value function - "Q" refers to the function that the algorithm calculates: the expected reward for an action taken in a given state. For any Finite Markov Decision Process (FMDP), this algorithm finds an optimal policy in the sense of maximising the expected value of the total reward over any and all successive steps from the current state.

The Bellman Equation (Figure 2.7) is used to determine the value of a particular state by deducing how rewarding it is to be in that state/to move to that state. This equation uses the current state, and the reward associated with that state, along with the maximum expected reward and a discount, which determines its importance to the current state, to find the next state for the agent. The learning rate determines how fast or slow, the model will be learning.

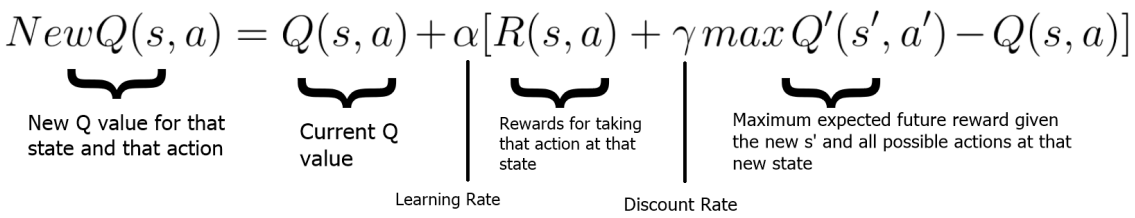


Figure 2.7: Bellman Equation of Q-Learning

Actor-Critic: Finally, there are Actor-Critic algorithms, which incorporate the advantages of the value-based and policy-based approaches. These algorithms try to estimate a value-function, while they adopt the policy gradient to be searched in the policy space. By combining the policy-based method, Actor, with the value-based approach, Critic, it is possible to learn the policy and the value-function in parallel. In other words, the Actor trains the policy according to the value function from the Critic's feedback, while the Critic trains the value-function and uses the TD method to update it in one step.

2.1.3.7 Multi-Armed Bandits

Multi-Armed Bandits (MAB) can be seen as a simple but powerful instantiation of RL. The term MAB comes from a hypothetical experiment in which a person must choose between multiple actions, each with an unknown payoff, and the goal is to determine the best or most profitable outcome through a series of consecutive choices.

In a hypothetical way, let us imagine that we have a player in front of several gaming machines. This player will have to decide which machine to play on and how many times to play on each, in order to decide whether to continue playing on the same machine or to switch. The arms (machines) are stochastic and the rewards produced by them come from a fixed distribution, that is, the distribution of rewards does not change over time. A MAB would perform as follows:

- At each step, it chooses an arm to play and observes the reward.
- After each move it decides whether to continue playing the same arm or to switch in the hope of receiving a greater reward.

The objective of playing is to maximise the total sum of rewards after a fixed number of tries. Thus, a MAB is able to determine the best or most profitable outcome through a series of consecutive choices.

2.2 Recommender Systems

We deal with recommendations every day, even if we do not notice them. These suggestions are intended to help us in various decision-making processes, such as what music to listen to, what book to read, or game to play.

Recommender systems are software techniques that provide suggestions of items to be recommended, being mechanisms capable of analysing and understanding the behaviour of users of a platform and serve two important functions: helping companies make more profits and helping users deal with information overload by giving them personalised recommendations.

2.2.1 The Recommendation Problem

The recommendation problem can be defined by the following factors:

- Set of users U.
- Set of items S to be recommended to the users.
- Each user in U is defined with a user profile that includes user characteristics, tastes, preferences, etc.

The task is to learn the utility function that measures the usefulness of item Si to user Ui and predict the utility value of each item to each user.

2.2.2 Predictions and Approaches

There are two types of prediction tasks:

- **Item Prediction**: prediction of an ordered list of items that a user is likely to buy or use.
- **Rating Prediction**: prediction of a score that a user is likely to give to a previously unseen item.

Just like different types of tasks, there are also different approaches possible to use when implementing a recommender system:

- **Content-based filtering**: the user will be recommended items similar to the ones he preferred in the past.
- **Collaborative filtering (CF)**: the user will get as recommendations items that people with similar tastes to his own have liked in the past. In this type of recommendation we

$$sim(i,j) = cos(\vec{i}, \vec{j}) = \frac{\sum_{u \in U} i_u \cdot j_u}{\sqrt{\sum_{u \in U} i_u^2} \cdot \sqrt{\sum_{u \in U} j_u^2}}$$

Figure 2.8: Item-based Cosine similarity

$$sim(u,v) = cos(\vec{u}, \vec{v}) = \frac{\sum_{i \in I} u_i \cdot v_i}{\sqrt{\sum_{i \in I} u_i^2} \cdot \sqrt{\sum_{i \in I} v_i^2}}$$

Figure 2.9: User-based Cosine similarity

don't need to know anything about an item except who else has liked, viewed or ignored it - two items are not consider similar because of their content, but because they were liked, viewed or ignored by a similar set of users.

It is possible to use user similarity (**User-Based CF**), i.e., similar users have similar ratings on the same item, or item similarity (**Item-Based CF**), i.e, similar items are rated similarly by the same user.

- **Hybrid approaches**: combination of the two previous types of recommendation, incorporating characteristics from both.

To measure the similarity between users or items, several measures can be used, such as Cosine similarity (Figure 2.8 and Figure 2.9), Pearson correlation coefficient, Jaccard index.

2.2.3 Evaluation

The evaluation of a recommender system should be of quality and meet several criteria, such as having predictive power, being novel, diverse, robust, and being able to preserve the privacy of the users. To measure the success of a recommendation model there are several types of evaluation:

- **Offline Evaluation**: cheap and repeatable, but without user feedback.
- **Online Evaluation**: with user interaction, but more expensive and non-repeatable.
- **User Studies**: with quality feedback and captures user behaviour, but it is expensive and has limited samples.

In addition to the types of evaluation, there are metrics that can help us understand the quality of the models. These metrics are quantifiable measures used to analyse the outcome of a specific process, action, or strategy. In general, they are indispensable for measuring the success of the recommendations generated. The best-known metrics for Top-K Item Recommendation are:

- **Recall** (or Sensitivity): represents the fraction of relevant instances that were retrieved.

$$recall = \frac{\#(Hidden \cap Recommended)}{\#Hidden} = \frac{\#Hits}{\#Hidden}$$

- **Precision** (or Positive Predictive Value): represents the fraction of relevant instances among the retrieved instances.

$$precision = \frac{\#(Hidden \cap Recommended)}{\#Recommended} = \frac{\#Hits}{\#Recommended}$$

- **F1**: harmonic mean between Recall and Precision.

$$F1 = \frac{2 * recall * precision}{recall + precision}$$

Just as there are metrics for Top-K Item Recommendation, there are also metrics for Recommendations with Ratings. The most widely used are Root Mean Squared Error, the standard deviation of the residuals (prediction errors) and Mean Absolute Error, the average of all absolute errors.

2.3 Summary

In this chapter, some general notions of the main topics that this dissertation addresses were presented: Machine Learning, with specific focus on Reinforcement Learning, and Recommendation Systems. These major topics are the basis of all the work developed.

Chapter 3

Literature review

The ability of recommender systems to adapt their behaviour to individual preferences is becoming more and more valuable as time goes on and their relevance to humans increases [9]. Recently, the iterative recommendation approach has been successfully applied to different real-world recommendation tasks in several specific scenarios, such as e-learning, recommendation of music, movies, news, professional skills [16].

With respect to recommendations, there are several types of user feedback that can be used to update the models responsible for generating suggestions. Most existing contributions exploit explicit feedback, which effectively demonstrates the user's feelings towards the recommendations. In addition to this, positive implicit feedback is also widely used, however, when it comes to negative implicit feedback there are few works that cover this type of user response. Furthermore, the few contributions that cover this type of feedback use it in boolean form, that is, without quantification, just existence or not, and are not iterative and sequential, being unable to increment the models in real time.

In this chapter, we present some state of the art techniques for sequential recommendation from RL and, more specifically, generation of music recommendations through the same approach. In addition to this, we also cover contributions that focus on recommendation with negative implicit feedback.

3.1 Sequential Recommendation with Reinforcement Learning

One of the focus areas of RL-based recommender systems is sequential recommendation, one of the main topics of the present dissertation. These systems aim at predicting future user preferences in order to recommend items successively through interaction with the user. Some existing work [16] has revealed that RL algorithms deal well with sequential recommendation problems, since such problems can be naturally formulated as an MDP.

3.1.1 Sequential Recommendation via Value-Based RL

In sequential recommendation, it is of utmost importance to be able to fuse long-term user involvement and user-item iterations during the formation of recommendation models. Recommendation policy learning from implicit feedback recorded by RLs is a promising direction, however, it faces challenges in implementation due to the pure off-policy setting.

Xin et al. [22] present a novel self-supervised RL method. The next item recommendation problem is modelled as an MDP, in which an agent interacts with the environment to maximise cumulative rewards. The data used in the reward function, in order to optimise the recommendation model, contains purchase interactions, long-term user engagement and item novelty. This method resulted in two models developed by the authors: a self supervised Q-learning model jointly trains two layers with the logged implicit feedback, and a self-supervised Actor-Critic model treats the self-supervised layer as an actor, to perform ranking, and the RL layer as a critic, to estimate the state-action value. In this way, this contribution is able to generate sequential recommendations through an incremental recommendation model.

3.1.2 Sequential Recommendation via Policy-Based RL

It is not an easy task to capture long-term user preferences in sequential recommender systems. Since the interaction between the user and items may be limited and sparse, learning users' interests through random exploration strategies is unreliable for RL algorithms.

Zhang et al. [23] analyse user's sequential learning. The authors note that recent attention-based recommendation models can distinguish the effects of different historical courses when recommending different target courses. However, the attention-based recommender systems will perform poorly when the users enroll in diverse historical courses, since the effects of the contributing items are diluted by many different items. In order to solve the challenge and remove the noisy items without direct supervision and recommend the most relevant ones at the next time, the authors designed a profile reviser by two-level MDPs, in which a high-level task decides whether to revise the user profile, and a low-level task decides which item should be removed. The agent in the Hierarchical Reinforcement Learning framework completes hierarchical tasks under the revising policy, which is updated by the REINFORCE algorithm, and then receives a delayed reward from the environment after completing the last low-level action. Finally, to improve recommendation results, the profile reviser is trained with an attention network-based recommendation model.

3.1.3 Sequential Music Recommendation through Reinforcement Learning

Sequential music recommendation is a topic that has been addressed for some years by researchers and, with the advancement and development of RL, there has been an increase in the appearance of several works that use this technique to create recommendations. Although there

are many contributions in the field of music recommendation, this state of the art focuses on those that fall under iterative/sequential recommendations. In Table 3.1, we have an overview of the MDPs formulated for each state-of-the-art technique we address below. Table 3.2 presents the evaluation techniques and data used by those same techniques.

Paper	**Algorithm**	**States**	**Actions**	**Reward**
Playlist Recommendation Based on Reinforcement Learning [13]	Q-Learning	Last K songs listened clusters	Recommending a song	A function of user listening, collecting, and downloading
A Reinforcement Learning Approach to Emotion-based Automatic Playlist Generation [8]	Sarsa and Q-Learning	Last k songs listened emotion classes	Choosing an emotion class	Listening time, number of replays and user rating
A Hybrid Recommendation for Music Based on Reinforcement Learning [21]	Hierarchical searching heuristic	Last k songs listened	Listening to a song	A function of user's preferences for songs and song transitions
DJ-MC: A Reinforcement Learning Agent for Music Playlist Recommendation [15]	Tree-search heuristic	An ordered list of all songs in the playlist	Selecting the next song	A function of user's preferences over individual songs and song transitions
Automatic, Personalized and Flexible Playlist Generation using Reinforcement Learning [19]	Policy-gradient	All songs recommended	Recommending a song	Song diversity, novelty, freshness and coherence

Table 3.1: Synthesis of the MDPs formulated by the State of the Art techniques

Paper	**Dataset**	**Evaluation Method**
Playlist Recommendation Based on Reinforcement Learning [13]	Music dataset with 349949 users, 10842 songs and 5652232 feedbacks	Offline
A Reinforcement Learning Approach to Emotion-based Automatic Playlist Generation [8]	Music dataset with 526 pop songs by 76 artists - each song associated with emotional ratings that were manually annotated by pre-trained users	Offline (simulation) and user studies
A Hybrid Recommendation for Music Based on Reinforcement Learning [21]	Million Song Dataset, Taste Profile Subset Dataset, Historical Song Playlist Dataset	Offline
DJ-MC: A Reinforcement Learning Agent for Music Playlist Recommendation [15]	Million Song Dataset, and data from Yes.com and Last.fm with music transitions	Offline (simulation) and user studies
Automatic, Personalized and Flexible Playlist Generation using Reinforcement Learning [19]	*KKBOX Inc.* private data: 10000 playlists, each of which composed of 30 songs, and a total of 45243 unique songs	Offline

Table 3.2: Synthesis of the Evaluation Methods and Data sets used by the State of the Art techniques

In [13] Hu et al. modelled music recommendation as a MDP, and considered the problem as a playlist recommendation task. In this work, each state represents a sequence of songs listened by the user, and listening to a song is considered an action. In order to solve problems regarding spatial dimensionality of states, researchers presented a state compression algorithm to transform individual songs into clusters of songs, through user feedback and characteristics of the songs themselves, for further model learning. Also for efficiency reasons, it was designed a novel recommendation strategy based on tree structure. The authors proposed a model based on Q-learning, an efficient value-based off-policy temporal-difference (TD) reinforcement learning algorithm, with e-greedy strategy, in order to make a efficient trade-off between exploration and exploitation, named RLWRec. Q-learning can evaluate the rewards of performing each action in each state, knowing as Q-values. Hu et al. took advantage of this feature to design a recommendation strategy to recommend song cluster according to the probability of Q-value and recommend song according to the probability of preference score.

As is normal in RL problems, there is an interaction between the agent and the environment,

and it is during this interaction that user feedback is collected by the recommender system, with the objective of better understanding the user's preferences. The feedback used in the reward function comprises information regarding each song: listen, collect and download. Chi et al. [8] also used implicit feedback in their work. The authors argued that the Automatic Playlist Generation (APG) problem is better modelled as a continuous optimisation problem, and proposed an adaptive preference model for personalised APG based on emotions, also modelled as a MDP. For this, Chi et al. defined two types of reward functions: implicit feedback and explicit feedback. Implicit feedbacks include the listening time and the number of repetitions of each song. On the other hand, users can also rate each recommended song, and this rating is used as explicit feedback. Both types of feedback can reflect whether the user likes the recommended song or not.

In their work, researchers categorised all the songs in the experimental dataset into four classes manually pre-annotated by users with correspondence to their emotion class. To deal with problems of spatial dimensionality, the authors adopted the N-Grams model which is commonly used in natural language processing to predict the next item in a sequence - their representation of a state contains only the last k songs emotion classes, rather than the songs themselves. Similar to Hu et al., authors also used the properties of Temporal Difference, thus resulting in two learning methods: SARSA and Q-Learning - both methods are primarily concerned with estimating the value of performing any action in each state (Q-value), their main difference is that, while Q-Learning is an off-policy control method, SARSA is an on-policy control method.

In [21], Wang proposed a personalised hybrid recommendation algorithm for music based on reinforcement learning to recommend song sequences that match listeners preferences better. The researcher first used weighted matrix factorisation (WMF) and convolutional neural network (CNN) to learn and extract the song feature vectors. Secondly, in order to capture the changing preferences of listener users, Wang made use of RL by applying an iteration process that allowed him to update the model continuously - the states are made up of sequences of songs listened to, while the actions boil down to listening to songs. The feedback used as reward was the user's preference not only for individual songs, but also for transitions between songs.

In order to recommend the next song, the author used a hierarchical searching heuristic method. Given the huge size of the search space, it was not practical to select songs from the entire song dataset and, in order to solve this, the author performed a cluster of songs by song type to reduce the complexity of searching. With each new song recommendation, the recommender agent evaluates user feedback to determine whether the model is recommending songs in the right direction or should be updated.

Similar to this contribution, also E. Liebman et al. [15] considered not only user preferences for individual songs, but also for song transitions to be important. Researchers proposed DJ-MC, a novel reinforcement-learning framework for music recommendation that does not recommend songs individually but rather song sequences, or playlists, based on a model of preferences for both songs and song transitions. Analogous to other works, their MDP is not very different: states represent sequences of songs listened to, and actions represent the choice of the next song

to be listened to. The reward function, as noted above, encompasses user preferences for songs and for transitions between songs.

The presented solution contains two main components: learning user parameters and planning a song sequence. The learning part is also divided into two parts: initialisation and real-time learning. Note that initialisation, in this type of system, is very important to keep the listeners attention while the agent converges on a good enough model - in this contribution, the authors initialised the parameters by directly asking the users favourite songs and transitions. Real-time learning allows the system to continuously improve until it converges on a reliable model, and is done through user-agent interaction. Finally, to plan the sequence of recommended songs, the authors employed a tree-search heuristic.

Unlike the mentioned works, Shih and Chi [19] considered music playlist generation as a language modelling problem and solved it by a proposed attention language model with policy gradient, having developed a systematic and interactive approach so that the resulting playlists can be flexibly tuned according to user preferences. In this contribution, states represent all the songs already recommended, while actions represent the recommendation of a song. Regarding the reward function, it is composed by the combination (sum) of several reward sub-functions involving parameters of songs: Diversity, Novelty, Freshness and Coherence.

First, the authors generated base playlists based on users preferences. In detail, given the relationship between users and songs, they constructed a corresponding bipartite graph at the beginning. With the graph, they were able to calculate the embedding characteristics of songs and thus obtain the base playlist for each songs by finding its *k-nearest* neighbours. Second, by formulating base lists as sequences of words, they were able to pre-train an RNN language model to obtain better initial parameters for the following optimisation using policy gradient reinforcement learning. Finally, given user profile preferences and pre-trained parameters, they were able to generate personalised playlists by exploiting policy gradient reinforcement learning techniques with corresponding reward functions.

Lastly, there is one contribution that, although dealing with different reward mechanisms, should be mentioned in this state of the art. Hong et al. [12] proposed a novel policy for music recommendation non intrusive-sensing and RL based Recommender Systems. The proposed framework enables detection, learning and adaptation to the user's current preference based on wireless detection and reinforcement learning in real time during a listening session. Unlike any other work on this topic, the authors used wireless sensing signals (breathing and heartbeat) as feedback used to build the reward function - with this type of feedback, and an MDP set for RL, they used tree-search heuristic to plan the sequence of recommended songs.

3.2 Implicit Negative Feedback

Although there are many contributions that use implicit feedback as a source of user response, few papers use this feedback in negative terms. Furthermore, the few works that focus on negative

implicit feedback for ML Recommendation algorithms use it as boolean and not quantified, and only one is iterative.

3.2.1 Recommendations based on Implicit Negative Feedback

The few contributions that use negative implicit feedback in recommendation algorithms are mainly targeted to e-commerce, where the meaning of this feedback differs from the one used in the context of music streams. In e-commerce websites, the negative implicit feedback is not quantified, i.e., the feedback is present or not (boolean) and has no different interpretations depending on the significance or the way it is transmitted. Furthermore, existing recommendation techniques that use this type of feedback are usually non-iterative, except for the first one mentioned in this subsection, and thus are unable to increment their models in real time.

Zhao et al. [24] state that the recommended items that users skipped can influence recommendation performance and are necessary to obtain better results, although most systems ignore this type of feedback. With this in mind, the authors proposed a pairwise DRL for the recommendation framework with the ability to continuously improve its strategies during interactions with users by collecting both positive and negative feedback. In their work, the authors model the sequential interactions between users and a recommender system as an MDP and leverage Reinforcement Learning to automatically learn the optimal strategies by recommending items on a trial-error basis and receive reinforcement of these items from user feedback.

In order to integrate the two types of feedback into the framework, the states were defined as follows: $s = \{s+, s-\}$, where positive state $s+ = \{i1, ..., it\}$ is the previous t items that a given user clicked or purchased, and negative state $s- = \{j1, ..., jt\}$ denotes the previous t items that the user skipped. Thus, when an item x is recommended to the user, if the user skips the item, $s+$ must keep untouchable and update $s- = \{j1, ..., jt, x\}$. In case the user does not skip the item x, the process is the other way around.

The authors constructed a novel DQN for the proposed framework, in which a Gated Recurrent Unit (GRU) captures user's sequential behaviours as positive state s+, and negative state s-. The positive and negative signals are fed into the input layer separately, which assists the new DQN to distinguish contributions of the positive and negative feedback in recommendations. In the first phase, user model building takes place - through iteration, the recommender agent builds a user model to learn user preferences based on user information or feedback. In the second phase recommendations are generated - the recommender agent generates a list of items that best match user preferences.

Analogous to this contribution, also Peska and Vojtáš [18] studied the implication of using implicit negative feedback in e-commerce oriented recommendation systems. The authors presented several methods to identify some of the feedbacks implied as negative user preference, how to aggregate several types of feedbacks together and how to recommend based on that.

In their work, the researchers used data representing user iterations in a travel e-commerce website. As the basis of their recommendation model, they used implicit feedback such as page visits, mouse movements, scrolling, among others. To study the implication of negative implicit feedback, they created a generalised metric where they combine both types of feedback: positive and negative (e.g. user opened an object, but leaves it quickly). The main goal in the authors experiments was to corroborate that negative implicit feedback can improve the quality of recommendations and for that they performed recommendation tests based on aggregated feedback. In all test cases, the combined negative preferences were superior to other methods for larger train sets, suggesting that negative preferences may become important while trying to sort already good objects. The experiment results showed that negative implicit feedback can be a valuable addition for the recommender system, improving recommendations quality.

Lastly, in [14] Lee et al. introduce a simple mechanism to infer negative implicit feedback and test the feasibility of this type of user response as a way to represent what users want in the context of a job recommendation system. As context for this application, there is a jobs website, with summarised descriptions of various offers. The user can perform various actions on the platform, such as opening a job, saving it and rating it. As negative implicit feedback, the researchers used the action of opening a job offer and closing it, rather than saving it.

A study was conducted to understand the influence of different types of feedback towards the recommendation model. The collected information was compared with a list of recommended jobs that were rated by users. The quality of the feedback was measured in several settings: positive preference only, negative preference only, and composite preference. As explained, when participants click on the title of a job offer in a shortlist, they can see the details of the job in a separate window. If they find the job interesting, they can rate it and even save it to generate recommendations. Otherwise, they just close the window without saving - which is considered negative implicit feedback.

After conducting the experiments with the different types of feedback, this study showed how implicit negative feedback affects the quality of recommendations. Compared to cases using only positive preferences, the distinction between good and bad jobs was significantly clear when using negative preferences. This study shows that negative preferences, as inferred, can increase the quality of recommendations.

3.2.2 Implicit Negative Feedback in Music Listening Sessions

Understanding the meaning and distribution of negative implicit feedback, skips, in music sessions turns out to be quite important in deciding to use this feedback to generate recommendations through an iterative sequential process. Meggetto et al. [17] argue that understanding skip activity during music listening sessions has acquired considerable importance. The authors proposed a data transformation and clustering-based approach to identify and categorise skipping types. For this, they used real-world data from *Spotify*, the same data used in the experiments of this dissertation, which can be read in Chapter 7.

In their work, it was presented a subsequent analysis of short, medium, and long listening sessions that demonstrates that these types of skips are consistent across sessions of varying length, meaning that dominant behaviours have no strong relation with the absolute length of a session and are thus generalisable. In their results, we can also note that two pairs of complimentary behaviour can be observed: listener (listen to the songs until the end) with skipper (always skips songs) and listen-then-skip with skip-then-listen. Finally, the researchers also admit that the distribution of skip types varies under different listening context information. This type of signal can be considered as a measure of users' satisfaction (dissatisfaction or lack of interest) in music listening sessions, affecting their engagement with the platforms.

In [11], Casper et al. argue that modelling user preferences at the beginning of a music session is a practical and effective way to address challenges arising from this type of problem. In their work, researchers observed that recent past consumption and session-level contextual variables (such as time of day or type of device used) are indeed predictive of the music a user will listen to. Driven by these findings, authors proposed CoSeRNN, a recurrent neural network embedding model that learns the sequential listening behaviour of users, and adapts it to the current context. CoSeRNN predicts, at the start of a session, a vector of preferences, based on past consumption history and current context.

This contribution was focused on two research questions: the question of whether music consumption depends on the context and the question if sequential and context-dependent user embeddings better anticipate a user's music consumption. For this, authors defined the context as time of the day (morning, afternoon, etc.) and the device used to access the service (mobile, desktop, etc.). They found clear evidence that, for a given user, sessions that share the same context (e.g., sessions that happen in the morning) are more similar to each other than sessions from a different context. They also found that the more the songs a user listens to during a session deviate from their average preferences, the more likely they are to hit the skip button - a negative clue sign of satisfaction.

In addition to this, authors used boolean skip information to separate played and skipped tracks and, through their experiments, argued that separating played tracks from skipped tracks is beneficial for model performance. There was a performance benefit - although small, this highlights the dissimilarities between skipped and listened tracks, and helps improve the model's abilities to understand the user's musical preferences.

3.3 Summary

In this chapter, the current state of art of sequential recommendations using different types of RL was presented, with a special focus on sequential recommendations in music landscape. We realised that most sequential recommendation techniques are based on MDPs, that can be differentiated from several possible factors, such as actions, states, reward type, and others. In addition to this, a review of some works that address implicit negative feedback and others that

mention it in music streaming sessions was also presented.

Chapter 4

Actions Set Generation

In a recommendation paradigm, one of the most important tasks is to learn what the users preferences are. The tastes of each user can be represented by topics, or categories, that represent sets of similar specific items in a given context. For example, in a TV content recommendation context, it is important to understand what the users favourite topics are, such as comedy, sports, action, among others.

In this dissertation, one of the major goals is to learn users preferences regarding a given set of topics in order to produce recommendations in line with their interests. For this, one of the most important steps is to define the set of possible actions that represents given topics, or categories, in a given recommendation context. This actions represents groupings of similar specific items, as exemplified in Figure 4.1.

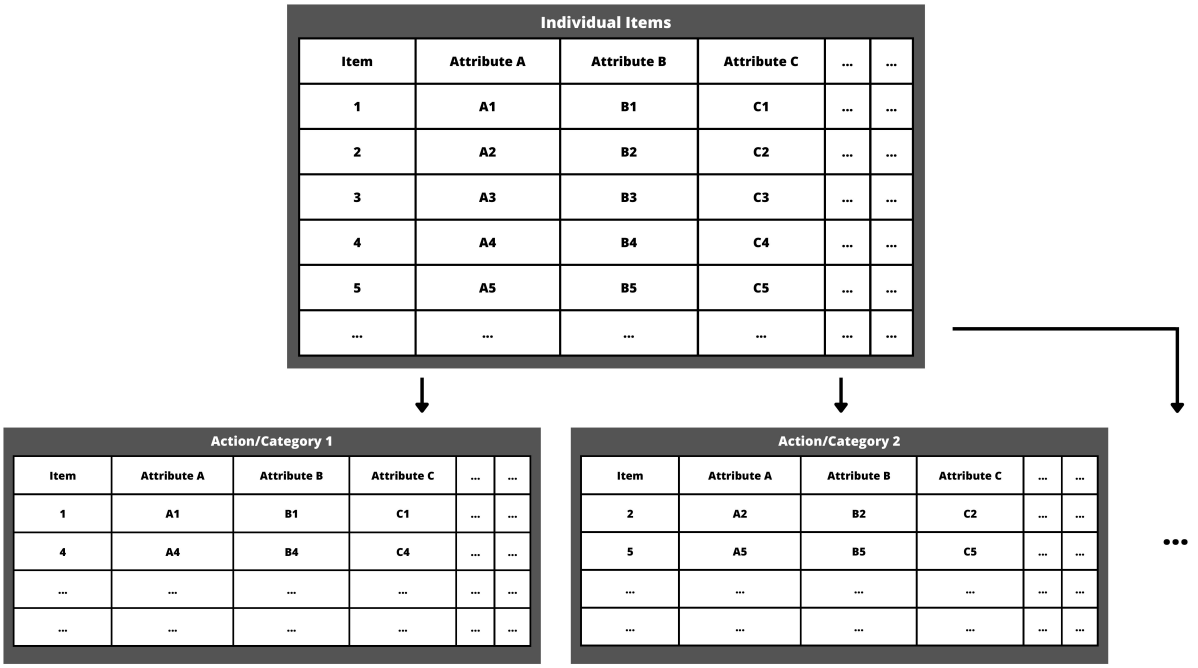


Figure 4.1: Exemplification of grouping individual items into actions/categories

It is also important to note that an RL approach with all available items represents a giant action space and is very difficult to model. The task of grouping similar items into categories also allows reducing the search space, improving the performance of the solution.

In this chapter, the strategy used to group sets of similar specific items into more general actions/categories that represent them is presented, and there is also a presentation and description of the data set that is used for the development of this dissertation's work.

4.1 Anonymized Items Grouping

4.1.1 The Data

The major contribution we make is the use of quantified implicit negative feedback data (further specified and presented in Chapter 5.1.3.1) to generate sequential recommendations during an iterative process. Despite the great diversity of this type of data, few public sources currently provide it. Consequently, due to the lack of data containing the mentioned type of feedback, this work focuses on a specific dataset from the Spotify Sequential Skip Prediction Challenge [3].

This data is divided into two components: one component containing information about anonymized songs and its respective characteristics, and another component containing data regarding music listening sessions containing user feedback and other associated variables. Since the goal of this step of the work is to define the set of possible actions, it was necessary to develop a strategy to group the anonymized individual songs into musical categories where their content contained similar songs.

As mentioned, the songs are anonymized, which makes it more difficult to evaluate the proposed recommendation method, but more intuitive to define the different musical categories (actions), considering that the groupings are not influenced by information such as music genre, artist name, among others, but by audio characteristics, as well as two others not audio related variables. With this in mind, we argue that grouping similar anonymized items can always be more rewarding and less biased.

Therefore, the musical features present in the songs data set are track ID (unique identifier for the track played), duration, release year, United States popularity estimate (estimate of the United States popularity percentile of the track as of October 12, 2018), and many features related to the audio of each song: acousticness, beat strength, bounciness, danceability, dynamic range mean, energy, flatness, instrumentalness, key, liveness, loudness, mechanism, mode, organism, speechiness, tempo, time signature, valence and eight acoustic vectors.

4.1.2 Clustering

Clustering is the task of dividing a population of data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters. In order to perform the grouping of individual anonymized songs into musical categories/actions, a clustering algorithm was applied.

4.1.2.1 K-Means

The clustering method chosen to solve the task of grouping similar anonymized items was K-Means. Recalling what was mentioned in Chapter 2.1.2.1, k-Means is a partition-based clustering method that obtains k clusters from a dataset in such a way so that data points within the same cluster are as similar as possible, while data points from different clusters are as dissimilar as possible. In this technique, each cluster is represented by its center, which corresponds to the arithmetic mean of data points assigned to the cluster. As demonstrated previously, the algorithm is defined by the following steps:

1. Initialise the centres of the k groups to a set of randomly chosen observations.
2. Repeat:
   (a) Allocate each observation to the group whose center is nearest.
   (b) Re-calculate the center of each group.
3. Until the groups are stable, i.e. there is no significant decrease or there is an increase on the minimise criterion *h(C)* - this technique uses Euclidean distance as criterion.

This algorithm was chosen because of its advantages in terms of fast scalability for large data sets, convergence guarantees, and generalisation to clusters of different shapes and sizes. Furthermore, through experimental tests performed, it was easy to see that clusters produced were of enough quality, by analysing the number of clusters present on average in the sessions. However, this algorithm also has some drawbacks, which we had to work around.

4.1.2.2 Choosing the Number of Clusters

K-Means requires the number of clusters be chosen manually. We performed tests, with the Elbow Method, to visualise and choose which number of centres is most appropriate for each different data sample, in our problem in specific.

The Elbow Method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. In statistics,

explained variation measures the proportion to which a mathematical model accounts for the variation (dispersion) of a given data set. Given that we are clustering anonymized songs, and that we want a number of clusters that is not too small, we had to adapt this method to meet a trade-off between the minimum error variation between clusters and a minimum number of clusters chosen.

4.2 Summary

In this chapter, it was explained the importance of obtaining a set of possible actions in each recommendation context, and also the method used to generate this set. With the set of available actions defined, it was possible to move on to the next step, the sequential and iterative learning and NBA recommendation through the quantified implicit negative feedback, discussed in Chapter 5.

Chapter 5

Next Best Action Recommendation

In a real scenario, a recommender system is in constant interaction with a user. The recommendation task can be seen as an iterative process in which the user receives a recommended item and provides feedback to the system, allowing that in the next iteration the recommendation model incorporates the feedback received and makes a new recommendation, closer to the user's preferences. Despite the notorious importance of user feedback for these systems, many of those that currently exist do not take it into account, since such systems completely ignore the interaction with the user. Consequently, become unable to update the recommendation models in real time with new information, generating less satisfactory results.

When listening to music, the choice of the next song to listen to is conditioned by several factors, with user preferences being the most important one to consider. With this in mind, the goal of the work presented in this chapter is to use quantified negative implicit feedback, more specifically song skips, to train an AI algorithm in order to learn user preferences during music streaming sessions, and generate good actions/categories recommendations based on this learning, as we can see in Figure 5.1.

To solve the problem of sequentiality and iterativity, it was defined the use of a RL approach to learn user preferences during music streaming sessions, through the mentioned feedback, in real time. Therefore, learning can be done through reinforcement, improving the knowledge and intelligence of the recommender agent through sequential iterations with users.

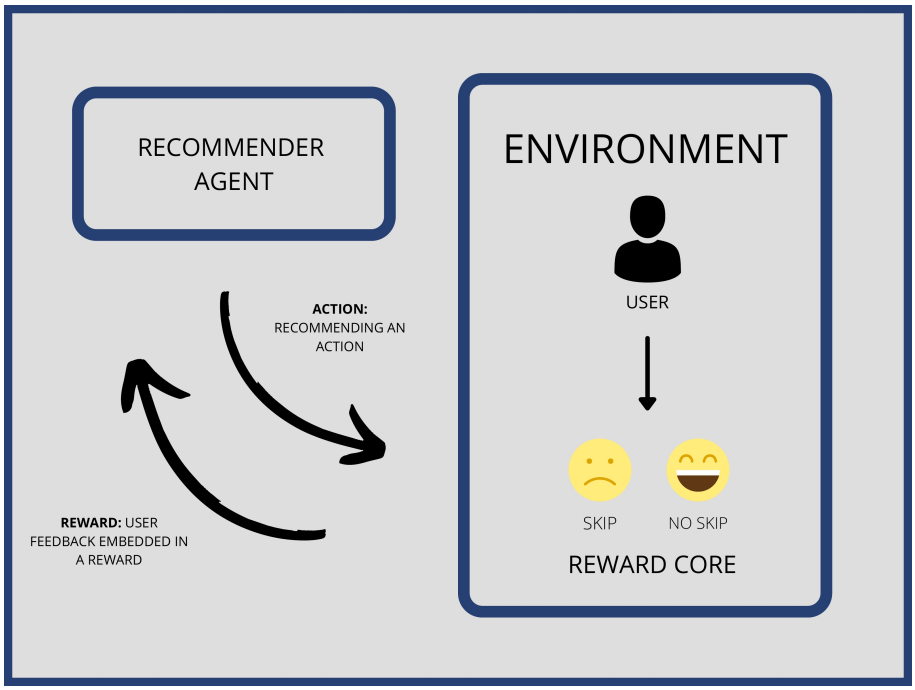


Figure 5.1: Iterative Learning via Implicit Negative Feedback

In this chapter, the RL formulation used to solve the problem of iteratively learning user preferences is presented and described. The section dedicated to the MDP reward function presents several exploratory data analyses in an attempt to find relationships between different variables, as well as a statistical study conducted for the purpose of understanding people's skip behaviours on music playlists of their own making. In the end, it is demonstrated how action recommendations are generated through the developed technique.

5.1 The Markov Decision Process Formulation

In a RL problem, there is a learner and decision maker called an agent and the surrounding with which it interacts is called the environment. The environment, in return, provides rewards and a new state based on the agent's actions. Thus, in this type of learning, we do not teach an agent how it should behave, but present it with rewards either positive or negative based on its actions.

As explained in Chapter 2.1.3.1, MDPs formally describe an environment for RL, so we take advantage of it. The following subsections describe the formalisation of the MDP used to work as the basis of the RL technique developed.

5.1.1 Agent and Environment

In the MDP's paradigm, the environment is completely observable, i.e., the agent directly observes the state of the environment, which allows us to make the analogy to a recommender system that directly observes the state of a user using it - it is one of the best strategies to represent and simulate the interaction of a recommender system with its users.

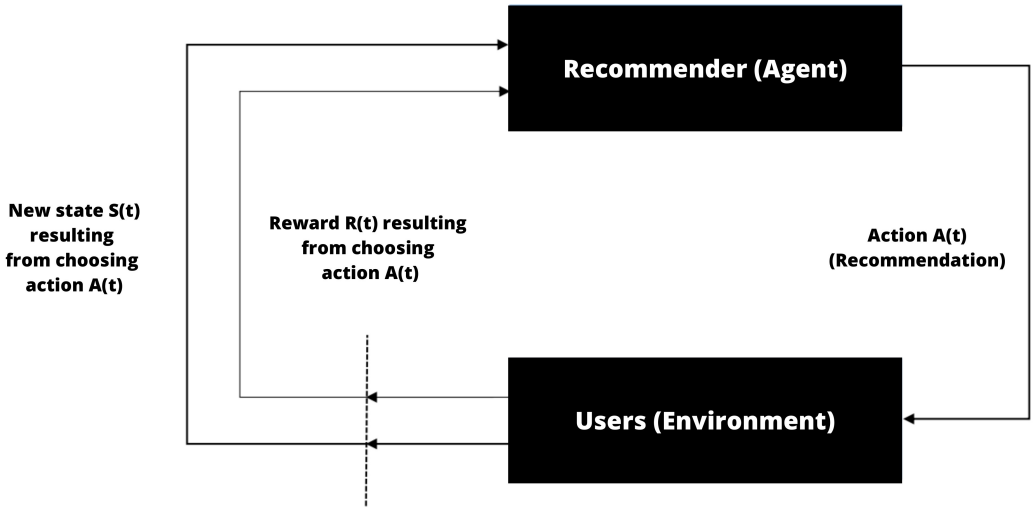


Figure 5.2: Recommender-Users interactions in MDP

As shown in Figure 5.2, the agent is defined by the recommender system which, as the name suggests, provides recommendations (actions) to the user. The user, which in this context represents the environment, interprets this action and produces feedback, which is used as input to a function that produces a reward. After an action has been performed, a new state is produced and the process repeats.

5.1.2 States and Actions

The states set S_n, in an MDP, define the context of the environment, i.e., it is a set of tokens that represent every state S_i that the agent can be in. An action A, on the other hand, is what allows the environment to change its state:

$$S_{t+1} = Transition(S_t, A)$$

To define states and actions, we get inspiration from the strategies presented at Chapter 3.1.3. In the specific case of our problem, an action represents the listening of a song from a certain cluster, or in other words from a certain musical category. With this in mind, it was decided to define the states by the set of the last K actions performed by the user.

The main reason for each state being defined by the set of the last K music categories listened

to by the user was the fact that, in this way, we could preserve historical information regarding the transition between musical categories by the user, and at the same time be able to handle recent information. While Chi et al. [8] grouped songs with manually pre-annotated classes information by users, and Hu et al. [13] through user feedback relative to the songs and some not musical characteristics (singer, release time, popularity, language, gender and so on), we clustered songs just based on their audio characteristics, as well as two others variables not audio related, without user interference, in order to obtain a set of possible actions, as explained in Chapter 4.

Figure 5.3: Examples of K=3 state size transitions through actions

It is important to note that when the user changes the context in which they are listening to music, the state is reset, as shown in Figure 5.3, because in this case there is no actual transition between songs. The context defines the stream source of the songs a user is currently listening to, like radio, personal playlists, among others.

5.1.3 Reward

In RL paradigms, a reward is the output of a real-valued function which measures the expected reward for all the possible states that can be reached from a certain state, i.e., it indicates "how good it is to choose/be in a certain state". A Reward R_{S_t} is received for being at the state S_t. The objective in RL is to maximise long-term future reward., that is, to choose action A so as to maximise $R_{t+1}, R_{t+2}, R_{t+3}, ..$ in order to get the best long-term accumulation of rewards.

5.1.3.1 Quantified Implicit Negative Feedback

The central element and focus of the reward function defined for this dissertation is the implicit negative feedback. This type of response from the user, despite not always representing a negative opinion, presents an interesting value regarding how the user feels about a certain

Hour	Period
00:00 - 05:59	Dawn
06:00 - 12:59	Morning
13:00 - 17:59	Afternoon
18:00 - 20:59	Evening
21:00 - 23:59	Night

Table 5.1: Map from hours of day to periods

recommendation. Our challenge is to create a system to recommend the next best action, or in this specific work the next best musical category, by interpreting this kind of feedback.

The few works that exist that use implicit negative feedback to generate recommendations use it in a boolean form, i.e., this feedback is simply present or not (it is given or not, by the user). One of our major contributions is the use of quantified implicit negative feedback - quantified in the aspect that it is presented with different levels of relevance and timing of transmission. In our work, feedback can take different values depending on the moment it is given. This feedback is represented by song skipping, and these skips have different interpretation values depending on the moment of the song at which it is given, but there are other scenarios where negative implicit information can be quantified, such as the time it takes to skip a short-video on a social network, or even in a more opposite and extreme scenario, in a non-technological and food recommendation-driven way, the amount of food left on a plate.

5.1.3.2 Inter-variable Relationships

In order to understand if there were relationships between users skip activity and some other variables associated with listening to songs, a data analysis was performed. For this analysis, we stopped considering the skip as quantified feedback, and looked at it in an absolute way - the skip was performed or not.

First, we defined a map of the hours of the day, when the songs were listened by users, to day periods, in order to understand if the distribution of skip activity depended on when it was performed. For this, we used a common mapping that is widely used in various contexts and works, which is demonstrated in Table 5.1. We used this mapping to create a chart that expresses the relationship between this variable and skip activity in a data set with approximately 8 million entries regarding music auditions, that can be analysed in Figure 5.4.

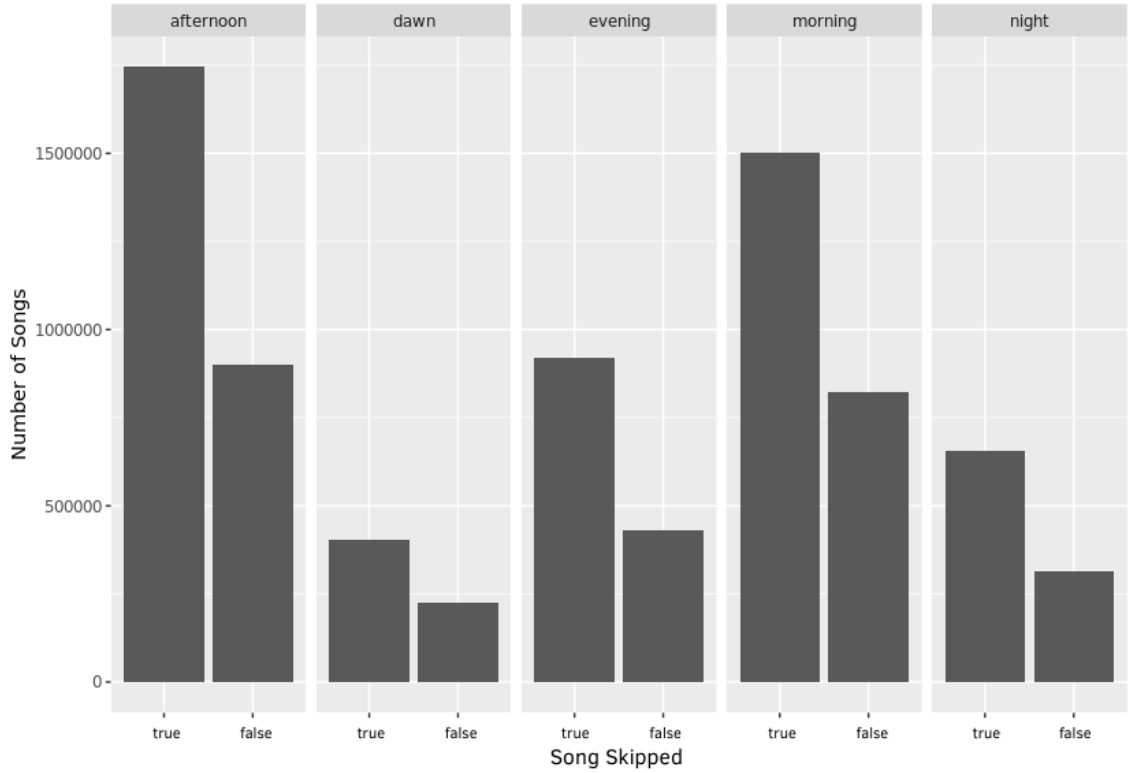


Figure 5.4: Relationship between day period and skip activity

Contrary to what we expected, due to the fact that we thought that behaviour during the normal working period of the world population - starting in the morning and ending in the evening [7] - would be different to nocturnal behaviour, the distribution remains approximately equivalent regardless of the period of day.

In second place, our goal was to understand if there was any deviation from the distribution of skip activity when compared to the context in which it was performed. The user can listen to music in various contexts, which may be important for the skip activity to happen. In the data we used for this analysis, the same as the previous one, there are 6 types of contexts: catalog, charts, editorial playlist, personalised playlist, radio and user collection. The chart referring to this analysis can be seen at Figure 5.5.

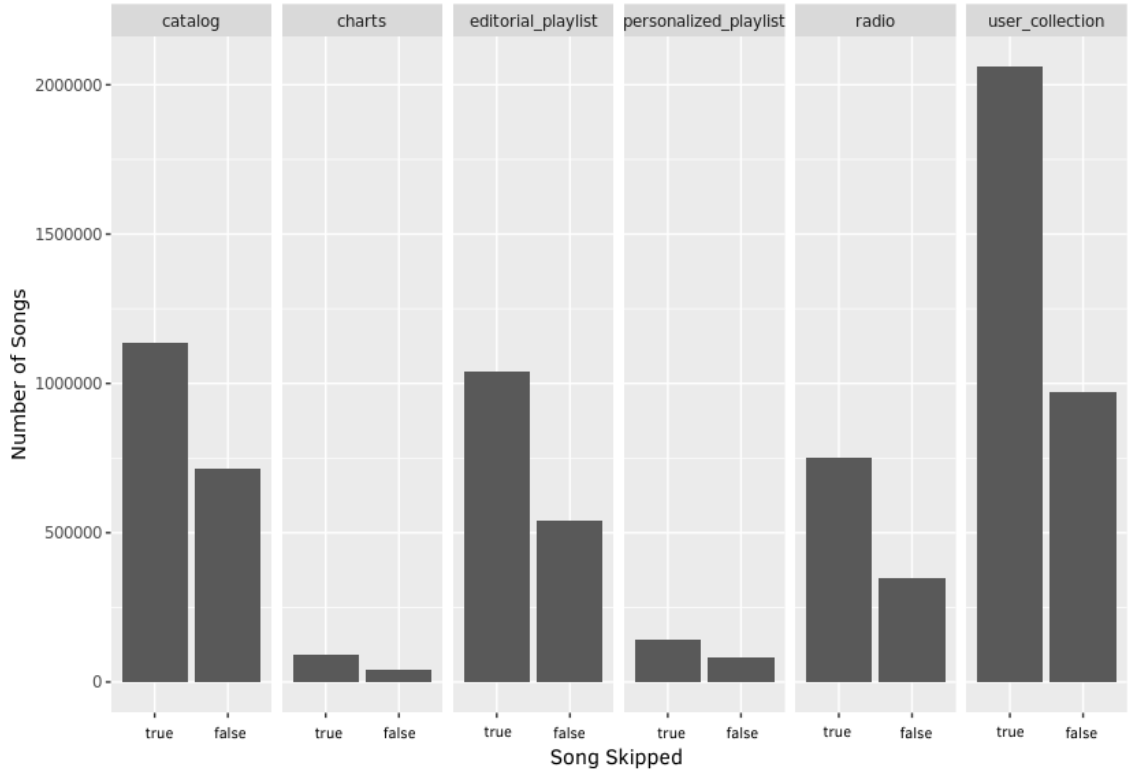


Figure 5.5: Relationship between context and skip activity

Analogously to the analysis of the day period, again we found unexpected results: the distribution remains quite similar in all context types. We expected different results mainly in the context of users private collections, since everything indicates that if a user have added a song to a private collection, he likes it. This is one of the examples why this analysis is so important, as skip information is not always purely negative, as it can happen due to numerous factors. For example, a user may be with a group of friends and skip a song simply because it is not the right song for the moment, or a taxi driver may skip a song because a client requested another one. It is important to realise and be aware that these cases can happen, which is why our reward function should be as finely adjusted as possible.

5.1.3.3 A Study on the Behaviour of Music Streaming Platform Users

Regardless of the statistical conclusions drawn on the previous subsection, we maintained our point that a skip effectively has a different value when it is given in a user's private collection. To support the hypothesis that a skip on a song in a user's collection has a value that is rarely negative, a study, whose results can be read in Appendix A, was conducted on users of music streaming platforms to understand the value of their behaviour in this type of context. Through conducting this research, we were also able to obtain other information that supported other insights we used in the reward function, in relation to song skips.

The study consisted on a anonymous survey to people of different age groups. There were 87

Age Group	Percentage of Answers
12-17	2.2%
18-24	85.1%
25-49	12.6%

Table 5.2: Distribution of survey A answers by age group

	"I do not like the song"	"I do not want to listen to that specific song at the moment"	"I am trying to find a specific song through the jumps between songs"
"Imagine a scenario in which you are listening to the radio or to someone else's public playlist on a music streaming platform. What is/what are the main reasons for skipping a song?"	80%	67.1%	18.8%
"Now imagine a different scenario in which you are listening to a playlist of your own (playlist created by you, with songs chosen by you) on a music streaming platform. What is/what are the main reasons for skipping a song?"	2.4%	95.3%	47.1%

Table 5.3: Synthesis of two questions from survey A

responses, and the distributions by age group can be visualised in Table 5.2. Although the sample is not of large variability in therms of ages, the most frequent users of Spotify, in the United States, were revealed to be adults aged between 18 and 29 years old [10], which is in line with our sample. 97.7% of the people who answered the survey stated they use music streaming platforms, and more specific questions were asked to them about their behaviours on these platforms.

Regarding specific questions about behaviour on music streaming platforms, 90.6% of respondents said that they perform a skip when they do not like a particular song. When faced with a question asking why they would skip a song while listening to radio or a public playlist created by another user on streaming music platforms, 80% of the respondents have stated that one of the reasons is that they do not like the song, and 67.1% stated that one of the reasons is that they do not want to listen to that specific song at the moment.

On the opposite, when confronted by the same question but in a different context, that is, while listening to songs from their own collection, only 2.4% stated that one of the reasons for skipping is not liking the song, supporting our thoughts regarding this topic. Table 5.3 summarises the results of this last two questions, the ones of most importance for our interpretation of

Skip Moment	Description
1	The track was only played very briefly
2	The track was only played briefly
3	Most of the track was played
4	The track was almost completely played
Not Skipped	The track was played in its entirety

Table 5.4: Split of quantified implicit negative feedback into moments of transmission

behaviours. From this, we were able to confirm that a skip on a song from private collections has no negative value, while a skip on radio songs or public playlists has most of the time a negative value.

Thus, we used this insight to shape our reward function by considering a less valuable skip when it is performed in the context of users collections.

5.1.3.4 Reward Function

As stated before, a reward is the output of a real-valued function which measures the expected reward for all the possible states that can be reached from a certain state, i.e., a reward indicates "how good it is to choose/be in a certain state". The definition of this function has an huge weight on RL paradigms, given the fact that it defines how the choice of a given action is evaluated, through the feedback coming from the user.

Considering that we are working with quantified feedback, it is first important to explain how we define the range that exists in this user response. For this, we have divided the transmission of feedback, by the user, into five different moments, as can be seen in Table 5.4. This way, we get a defined range that allows us to assign different weights to the feedback depending on its moment of transmission.

Once the transmission intervals were defined, the next step was to choose how to frame the listening context with the reward function. As stated in Chapter 5.1.3.3, we consider skips given to private user collections to be less valuable, as they mostly represent feelings different from bad ratings. With this in mind, we limit skips to private collections as always positive, only with more or less value depending on the time of transmission.

This way, we were able to define a reward function, which takes into account the moment of feedback transmission and also the context in which it is transmitted. The values resulting from this function are shown in Table 5.5.

Skip Moment	Reward - User Collection	Reward - Other Context
1	0.7	0.1
2	0.7	0.3
3	0.7	0.7
4	0.9	0.9
Not Skipped	1	1

Table 5.5: Reward function output

5.1.4 Reinforcement Learning Algorithm

As mentioned in Chapter 2.1.3.6, in value-based approaches, the value-function is initially random and then a new value-function is found. This is a repetitive process, until the optimal value-function is found or until a certain condition defined by the user is reached. The intuition is that the policy that follows the optimal value-function will always be the optimal policy, and consequently the policy is implicitly updated from the value-function.

To solve the problem of sequentially choosing the next music category, from which the recommendations will arise, a value-based RL approach was used.

5.1.4.1 Q-Learning

Q-Learning, already mentioned and explained in Chapter 2.1.3.6, is a value-based off-policy temporal-difference (TD) reinforcement learning algorithm whose goal is to learn the value of an action in a given state, by learning the action-value function. This algorithm finds an optimal policy in the sense of maximising the expected value of the total reward over any and all successive steps from the current state.

This algorithm was chosen because of its great advantage of being able to compare the expected utility of available actions without requiring a model of the environment. With this type of model-free approaches, we can update the whole value function and policy with every new sample, allowing learning to become more data efficient.

5.2 The RL-based Recommendation Strategy

Now that we have explained the MDP formulation that acts as the basis of our iterative learning scheme, it is time to move on to the recommendation strategy. This section will discuss the organisation of the algorithm, as well as how iterative learning and recommendation itself are performed.

5.2.1 Learning

As stated numerous times before, in an iterative and sequential application it is mandatory that the model is constantly updated with each new iteration. To do this, we use the properties of RL to update our model each time the user transmits feedback on a given item.

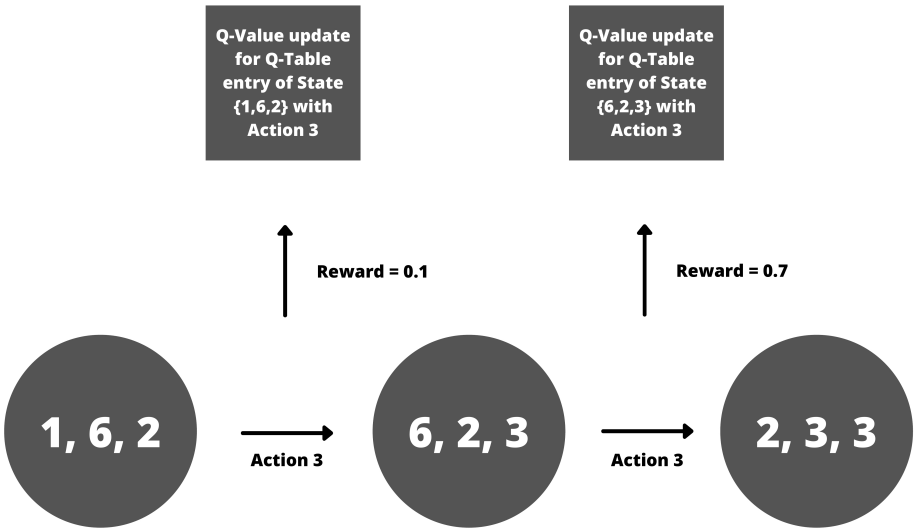


Figure 5.6: Q-Learning update moment

As shown in Figure 5.6, whenever an information is read or received that a user has chosen a action, in addition to the state transition performed in the MDP, the Q-Table in Q-Learning is also updated from the Bellman equation (Chapter 2.1.3.6), that is given by:

$$Q_{new}(S_i, A) = Q_{current}(S_i, A) + \alpha[R(S_i, A) + \gamma Q'_{max}(S_{i'}, A') - Q_{current}(S_i, A)]$$

This equation is used to determine the value of a particular state by deducing how rewarding it is to be in/to move to that state. It uses the current state $Q_{current}(S_i, A)$, the reward $R(S_i, A)$ associated with that state, along with the maximum expected future reward $Q'_{max}(S_{i'}, A')$ given the new state and all possible actions at that new state, and a discount γ, which determines its importance to the current state, to find the next state for the agent.

5.2.2 Next Best Action Recommendation

The constructed Q-Table by learning users interactions is used to solve the problem of sequentially choosing the next best action from which the recommendations will arise, and is constantly updated. The problem of recommending the next best action is unblocked by

choosing the action with the highest Q-value, which in other words can be expressed as the most advantageous action for the user when in a certain state.

Figure 5.7: Q-Learning - Choose next action

At each moment of a user's music stream, by checking the state the user is in, the action with the highest Q-Value for that state is chosen in the Q-Table, as we can see in Figure 5.7.

5.2.2.1 Exploration vs Exploitation

It is very important sometimes to explore new actions to recommend to the user, because user preferences may change over time and it is important that the algorithm retains its knowledge about the user. Although it is good to have early exploration, in order to quickly arrive at an optimal solution, in the long run we want to exploit more knowledge than we have.

In order to solve the Exploration vs Exploitation problem, mentioned in Chapter 2.1.3.3, an E-Greedy strategy is used to balance the two techniques during recommendations. This paradigm consists of randomly selecting, with a defined probability, whether to exploit or explore: 1-ϵ of the time the algorithm exploits and ϵ of the time the algorithm explores, as we can see in Figure 5.8.

The user will receive recommendations aligned with their already demonstrated preferences, as well as new and different recommendations that allow the agent to discover new information about his/her preferences.

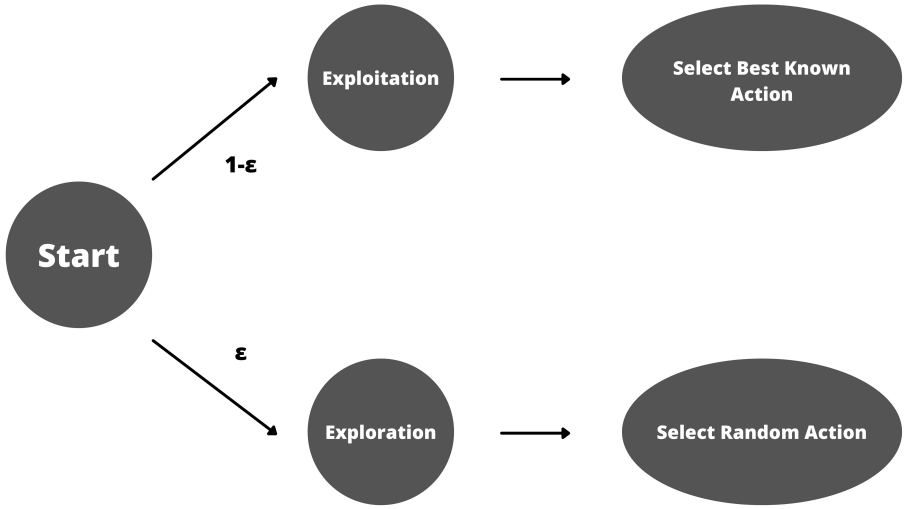


Figure 5.8: E-Greedy approach

5.2.3 Stages

There are two stages involved in our algorithm for recommending the next best action, as can be seen in Figure 5.9.

In an initial phase, the algorithm is only fed with data in order to build its Q-Table. Considering that there is enough real data to perform a good learning, it is valuable not to start recommendations right away after the start of learning, because, as the name of the approach suggests, the algorithm learns by reinforcement, and with little interaction, its performance will not produce good results. It is preferable to have an initial training phase with real historical data, rather than using the approach right away to generate recommendations with few interactions performed.

In a second phase of the RL approach, the connection of the model is now only and exclusively with a specific user. In this way, the algorithm when it is framed with a single user, already has a good basis of how to recommend, and starts to mould itself to that user in the long term.

By doing so, our approach is able to mitigate the effects of a cold start, allowing the recommendation system to have general information on how users behave on music streaming platforms, regarding iterations between different music categories.

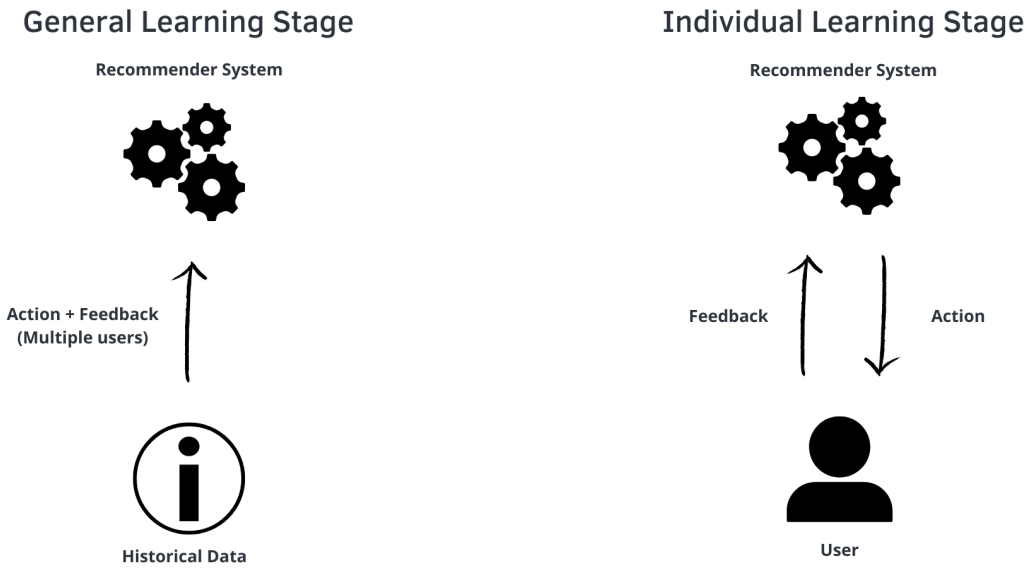


Figure 5.9: Stages of NBA recommendation algorithm

5.3 Summary

In this chapter, we presented the RL formulation used to solve the problem of iteratively learning user preferences. We also demonstrated the entire structure and strategy of the algorithm, as well as how it works. Furthermore, statistical studies were presented to support the decisions made in the construction of the reward function, through the interpretation of the implicit negative feedback.

Chapter 6

Next Best Items Recommendation

In the previous chapter, we demonstrated our strategy for recommending NBAs through an iterative and sequential process of reinforcement learning which uses negative implicit feedback as a reward. Nevertheless, it is also important to have a good strategy to provide specific item recommendations. For this purpose, we adopted the use of the Incremental Stochastic Gradient Descent (ISGD) [20], an fast incremental Matrix Factorisation algorithm, to item prediction - having already known the recommended action, we use this algorithm to recommend a specific item.

In this chapter, we present a introduction to the algorithm used as the basis of the recommendation of specific items and demonstrate how we adapted it to transform it into a hybrid recommendation system that takes into account the NBA.

6.1 The ISGD Algorithm

As mentioned earlier, in real world systems user feedback is continuously being generated at unpredictable rates and we have no control over the data rate or the ordering of the arrival of new ratings. One way to deal with this data stream is to perform online model updates as new data points become available. This requires algorithms able to process data at least as fast as it is generated and ISGD solves this problem, as it approaches the data as a stream. Starting with a simple iterative SGD algorithm, the authors adapted its mechanics to work incrementally using only the currently available observation.

6.1.1 SGD

Supposing we have a user-item matrix R, the algorithm decompose R in two factor matrices U and V, where U spans the user space and V spans the item space. As authors explains in [20], given this formulation, a predicted rating by user u to item i is given by the dot product $\hat{R}_{ui} = U_u V_i^T$. Training is performed by minimising the L2-regularised squared error for known

values in R and the corresponding predicted ratings:

$$\min_{U.,V.} \sum_{(u,i)\in D} (R_{ui} - U_u V_i^T)^2 + \lambda(||U_u||^2 + ||V_i||^2)$$

In the above equation, D is the set of user-item pairs for which ratings are known and λ is a parameter that controls the amount of regularisation - the regularisation term $\lambda(||U_u||^2 + ||V_i||^2)$ is used to avoid overfitting. This term penalises parameters with high magnitudes, that typically lead to overly complex models with low generalisation power. Given a training dataset consisting of tuples in the form <user,item,rating>, SGD performs several passes through the dataset – iterations – until some stopping criteria is met – typically a convergence bound and/or a maximum number of iterations. At each iteration, SGD sweeps over all known ratings R_{ui} and updates the corresponding rows U_u and V_i^T, correcting them in the inverse direction of the gradient of the error, by a factor of $\eta \leq 1$ – known as step size or learn rate. For each known rating, the corresponding error is calculated as $err_{ui} = R_{ui} - \hat{R}_{ui}$, and the following update operations are performed:

$$U_u \leftarrow U_u + \eta(err_{ui} V_i - \lambda U_u)$$
$$V_i \leftarrow V_i + \eta(err_{ui} U_u - \lambda V_i)$$

6.1.2 ISGD

SGD's optimisation procedure is a batch procedure that requires numerous iterations over the dataset in order to train a model. While this may be permissible in a stationary environment, it is not appropriate when dealing with streaming data. Repeatedly revisiting all available data becomes too expensive to be performed online as the number of observations grows and becomes possibly unlimited.

Researchers suggest ISGD (Algorithm 1), which updates factor matrices U and V based purely on the current observation and is designed to work as an incremental process. Despite its formal similarity to SGD, this method has two major distinctions: the learning process requires a single pass over the available data and no data shuffling – or any other pre-processing – is performed.

As previously explained, we use negative implicit feedback to learn users preferences for actions/categories. In this stage of the recommendation, we stop considering that feedback, and work only with users iterations with items. For this dissertation, in ISGD, the user-item matrix can be seen as a boolean value matrix, where true values indicate a user-item interaction, and false values indicate the absence of such interaction.

Given that we are dealing with positive-only feedback, we approach the boolean matrix R by assuming the numerical value 1 for true values. This way, the error is measured as $err_{ui} = 1 - \hat{R}_{ui}$,

and the rows in U and V^T are updated using the update operations in Algorithm 1.

Algorithm 1: ISGD

Data: stream $D = \{(u,i)_1, (u,i)_2, \ldots\}$
input : latent features z, iterations $iter$, regularization λ, learn rate η
output : factor matrices U and V

for $(u,i) \in D$ **do**
  if $u \notin \texttt{Rows}(U)$ **then**
    $U_u \leftarrow \texttt{Vector}(\mathsf{size} : z)$
    $U_u \sim \mathcal{N}(0, 0.1)$
  if $i \notin \texttt{Rows}(V)$ **then**
    $V_i \leftarrow \texttt{Vector}(\mathsf{size} : z)$
    $V_i \sim \mathcal{N}(0, 0.1)$
  for $n \leftarrow 1$ **to** $iter$ **do**
    $err_{ui} \leftarrow 1 - U_u V_i^T$
    $U_u \leftarrow U_u + \eta(err_{ui} V_i - \lambda U_u)$
    $V_i \leftarrow V_i + \eta(err_{ui} U_u - \lambda V_i)$

6.2 Hybrid ISGD

Despite the fact that we have developed a strategy that recommends the NBA in a sequential recommendation paradigm, there are always some possible recommendations that are seen with great value and should not be completely discarded during the recommendation process. For example, let us imagine that actions/categories, in a hypothetical scenario, represent music genres and our algorithm has recommended Jazz as NBA to a user. There may be a song from Blues genre that is a good recommendation for the user and by solely using NBA during the specific recommendation, it will be dropped and ignored. Besides this, if the NBA fails, the recommendation is guaranteed to fail. Thus, it is preferable to always have unfiltered (NBA-independent) items in the recommendation list.

Keeping this in mind, we developed a hybrid adapted strategy for specific recommendation that does not only take into account the NBA recommended in the previous step. We adapt the use of the ISGD algorithm to generate a Top K recommendations, whose K/2 recommendations come from the recommended NBA and K/2 recommendations from all existing items in the data space, as can be seen in the example of Figure 6.1. It is important to bear in mind that if the recommendations of the two ISGDs differ, we have a total of K recommendations. Otherwise, no recommendations are added, and we recommend only the unique ones in the list resulting from the junction of the two sets, resulting in a total of recommendations between K/2 and K-1.

This way, we present the user with less limited and concentrated recommendations. Thus, there is no compromise of the ISGD algorithm with the NBA generated by the previous SkipAwareRec

step.

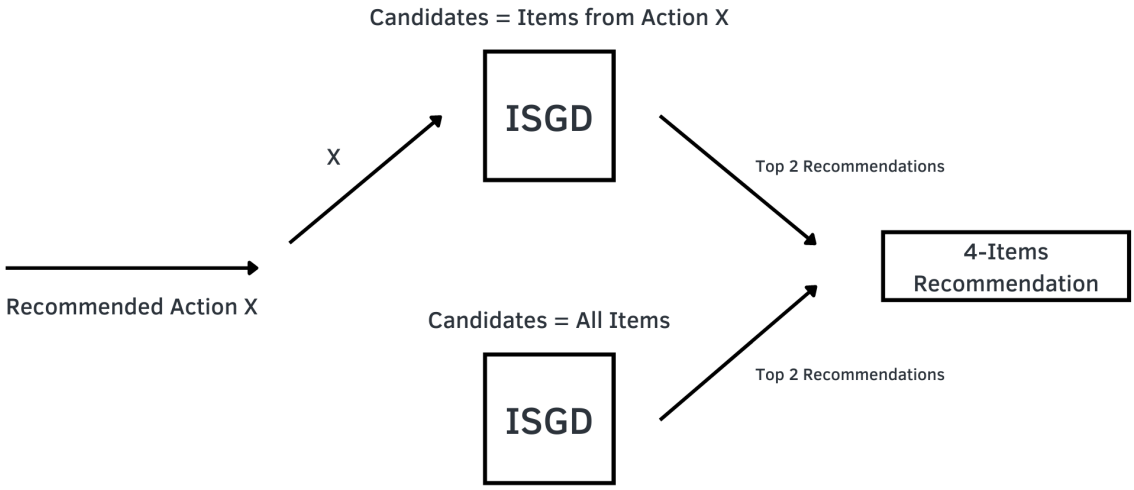


Figure 6.1: Example of Hybrid ISGD with K=4

6.3 Stages

Similarly to the algorithm proposed to recommend the NBA, in Chapter 5, hybrid ISGD also has 2 stages, as shown in Figure 6.2. In the first stage, the recommendation model is fed only with historical data from several users that contain information between iterations of these users with different specific items. In this way the recommendation model starts building its base to, when it reaches the user, start recommending items. This first phase is paralleled with the first phase of the NBAs recommendation algorithm in order to reduce the training time, thus improving the performance of the solution.

The second stage starts when the algorithm is delivered to online users. In this way, ISGD continues to recommend and collect information about user iterations with items, in order to update its model in real time, thus preserving the incrementality of the solution overall.

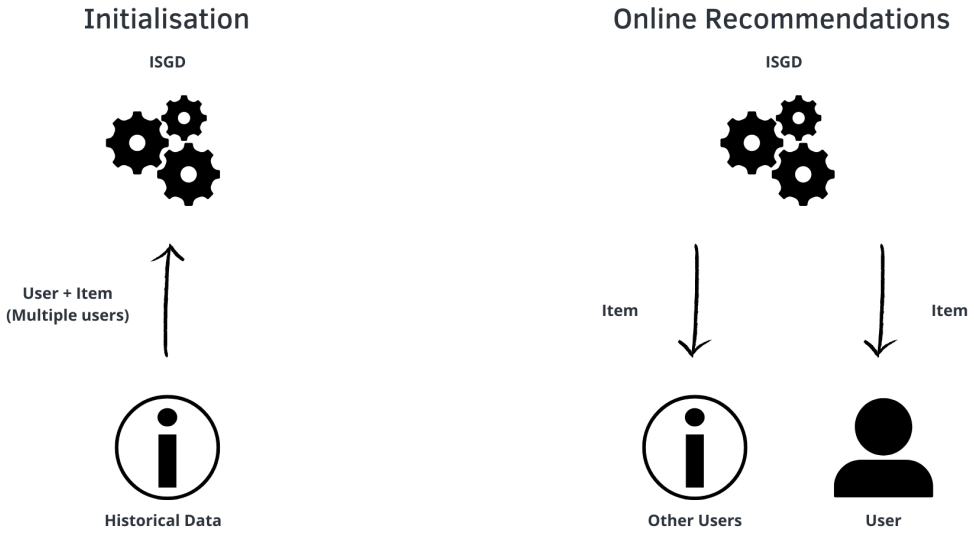


Figure 6.2: Hybrid ISGD Stages

6.4 Summary

In this chapter, we provided an overview of the algorithm that underpins the recommendation of specific items, as well as how we modified it to create a hybrid recommendation system that incorporates the NBA generated on the previous step of SkipAwareRec. In addition, the way the algorithm steps were stratified was presented, as well as the relational form of these steps with those of the NBAs recommendation algorithm.

Chapter 7

Results and analysis

In this chapter a practical experiment conducted with real music listening data from a streaming platform is presented, as well as the entire pipeline of work to that same data, including its preprocessing, the generation of the set of actions and the AI task that underlies the recommendation system. Also in this chapter, the evaluation strategy developed is presented, as well as the major limitations and difficulties encountered in carrying out this evaluation. At the end, the results are discussed.

7.1 The Data

In order to test SkipAwareRec, we used data from the Spotify Sequential Skip Prediction Challenge [3], a challenge where the objective was to predict whether individual tracks encountered in a listening session will be skipped by a particular user. This data is divided into two components: one component containing information about anonymized songs and its respective characteristics, and another component containing data regarding music listening sessions containing user feedback and other associated variables.

The data made available for this competition is of a gigantic size, and given that for the specific task of this dissertation such a large set is not necessary, we collected a smaller portion of this data. Thus, our data, referring to music sessions, contains 1,000,003 entries, where each entry represents the listening of a song in a given session. We have 59,062 sessions, of a length between 10 and 20 songs each, and a total of 174,768 individual songs associated with all these sessions.

As explained previously, in Chapter 4.1.1, the data of the songs contains attributes with values referring to audio features, and two other non-audio related features. The data for the music sessions contains various information, such as: session id, song id, session size, 4 attributes referring to skip moments, an attribute referring to whether there was a context switch, some attributes referring to pause actions, date, hour of day, an attribute referring to the account type used, and the context type.

7.1.1 Data Preprocessing

Prior to starting the experiments with the proposed methodology, we did some data preprocessing. First we filtered the data to only use data from premium sessions. Spotify, the company that provided the data for the aforementioned competition, currently has two types of account plans: a basic, free mode, where there are ads between songs and, on mobile devices, the user cannot choose specific songs to listen to, and another mode, premium, where the user has access to the entire platform, on any device, without any ads. As our problem is not focused on this specific music streaming platform, we only use data from premium sessions, where the user has increased freedom of actions.

In the second place, in order to make our universe of songs more compact, we filtered the existing songs to work only with those that exist in the previously selected sessions dataset. Thus, clustering is done only with songs actually listened to by the users/sessions contained in our sample.

7.2 Actions Set Generation Task

The first main task is to generate the set of possible actions. As explained in Chapter 4, this set is generated from a clustering task performed with the feature data of the individual songs.

7.2.1 Principal Component Analysis

Principal component analysis, or PCA, is a statistical procedure that allows to summarise the information content in large data tables by means of a smaller set of "summary indices" that can be more easily visualised and analysed. PCA forms the basis of multivariate data analysis based on projection methods. The most important use of PCA is to represent a multivariate data table as smaller set of variables in order to observe trends, jumps, clusters and outliers. Principal component analysis today is one of the most popular multivariate statistical techniques, it has been widely used in the areas of pattern recognition and signal processing and is a statistical method under the broad title of factor analysis [6].

PCA can be combined with clustering to obtain better visualisation of the clustering result, or simply to understand the pattern in the dataset. With this in mind, we performed a simple PCA analysis to see how important each variable is in the dimensions of our clustering algorithm.

In Figure 7.1, we can visualise the importance of the different variables for the two dimensions in which we will see the clustered data further ahead. Some variables have more impact on a given dimension, while others are significantly present in both. Finally, in Figure 7.2 we can observe the distribution of observations in the space defined by the PCA performed.

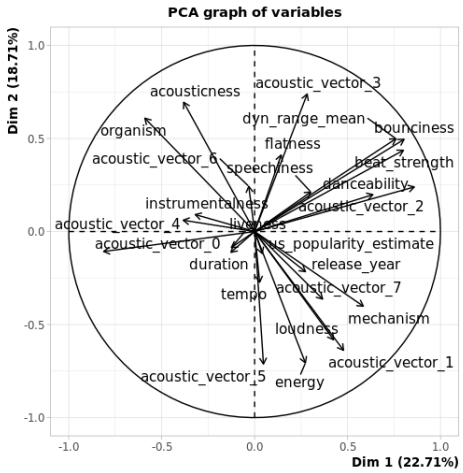


Figure 7.1: PCA - Variables importance

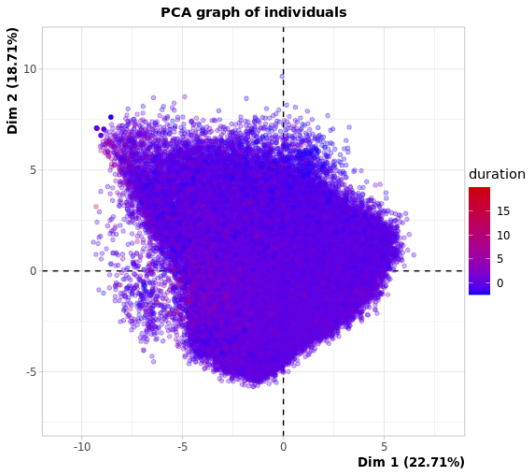


Figure 7.2: PCA - Individuals distribution

7.2.2 Choosing Number of Clusters

The basic idea behind K-means consists of defining k clusters such that total within-cluster variation (or error) is minimum, taking into account the minimum number of clusters we want to define. In practice, it is a trade-off that we have to define between the measure and our goal.

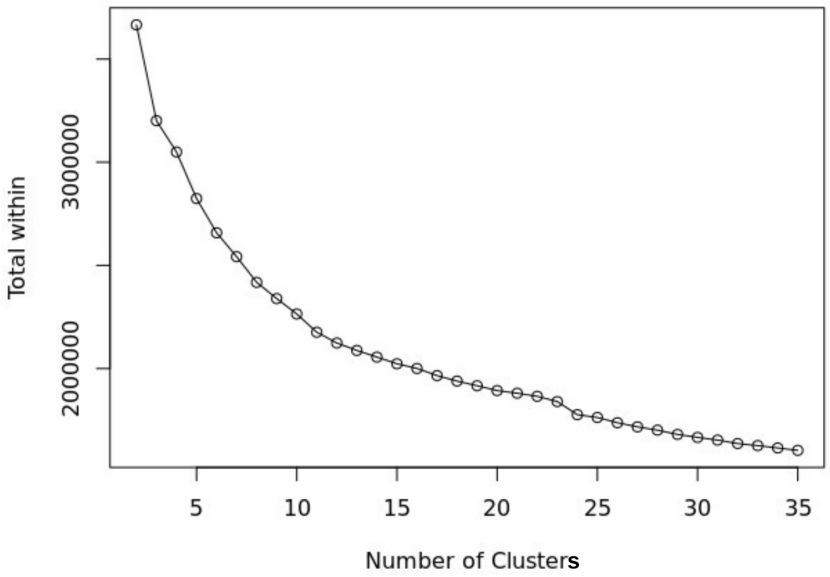


Figure 7.3: Elbow Method applied to data

In order to solve the problem of choosing the right number of centres, we used the Elbow Method, explained in Chapter 4.1.2.2. After the application of this method, where the results can be viewed in Figure 7.3, we end up choosing 23 as the number of centres to use in the clustering task, considering that this number is minimally large and the drop in the chart, compared to the next number, does not have such a steep slope in the curve.

7.2.3 Clustering

In Figure 7.4, we can visualise the distribution of the different clusters in the space generated by the first two dimensions of the PCA performed earlier. The function *fviz_cluster*, from the *factoextra* package on *R*, transforms the initial set of variables into a new set through PCA, in order to show a more elegant visualisation of partitioning methods results. In Figure 7.5, we can see the distribution of the number of songs for each cluster. By observing the graph, we can conclude that the clusters with the most observations are 14, 21 and 8.

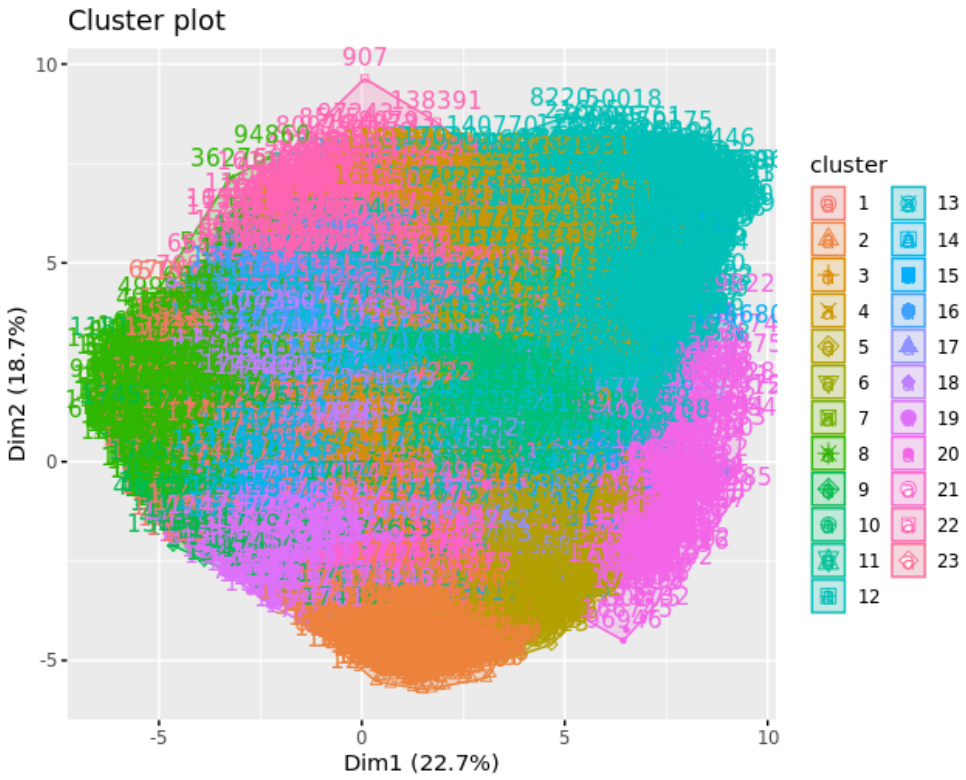


Figure 7.4: Clusters visualisation

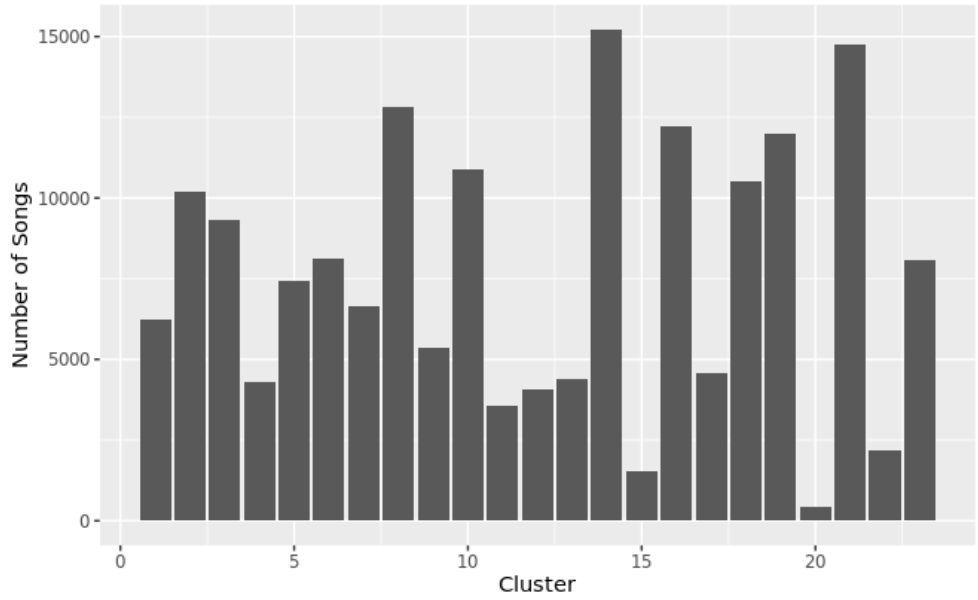


Figure 7.5: Clusters distribution

Our goal with this clustering is to obtain good representations of music categories, and therefore our validation method aims to evaluate the mode amount of clusters per session. Given that sessions can be anywhere from 10 to 20 songs in size, and that most sessions have 20 songs, we assume that an mode amount of 4/5 clusters per session would be reasonable for our analysis. As we can see in Figure 7.6, the mode of clusters per session is 4, and the second major value is 5, which represents a successfully accomplished clustering task for our specific work.

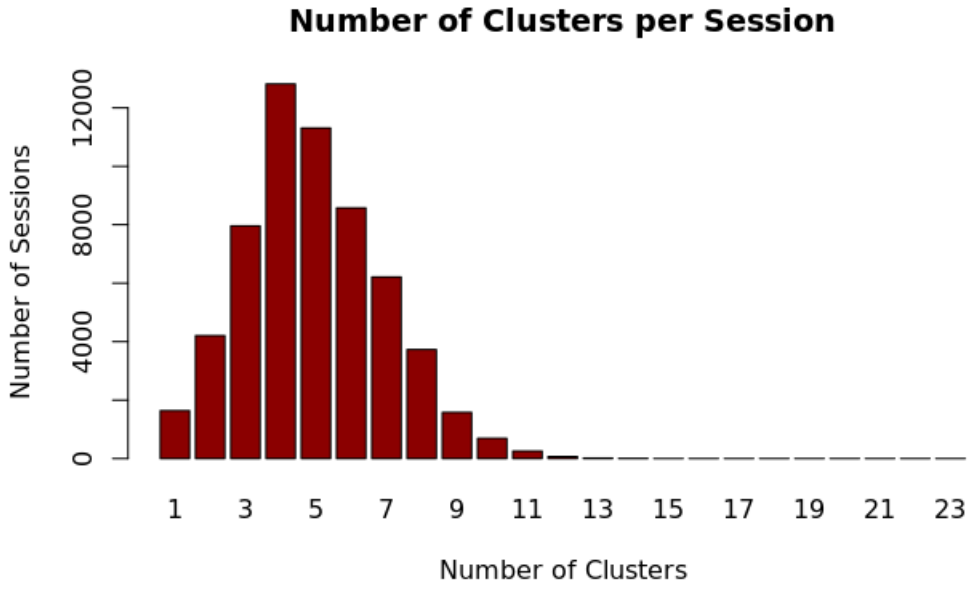


Figure 7.6: Number of clusters per session

Description	Skip 1	Skip 2	Skip 3	Not Skipped
The track was only played very briefly	true	*	*	*
The track was only played briefly	false	true	*	*
Most of the track was played	false	false	true	*
The track was almost completely played	false	false	false	false
The track was played in its entirety	false	false	false	true

Table 7.1: Skip information in data

7.3 General Learning Stage and ISGD Initialisation

In order to train the suggested NBA and ISGD models, we broke the sessions data into two sets, a training set with 85% of the data and a test set with the other 15%. From the training set, we feed the proposed RL-base NBA recommendation algorithm by reading the data, in the general learning stage, considering, in each line referring to music listening, the quantified implicit negative feedback transmitted by the user and the action listened. With this, we built sequential states, for each session, and updated our AI model as explained in the respective section. In parallel, this same training dataset was used to build the ISGD model that will also be incrementally updated in a next stage. The goal of parallelising the creation of these two models is to minimise the time they take to train.

The data used to develop this solution, contains 5 different skips information, relating to different moments of the songs, as showed in Table 7.1, which were crucial for the definition of the MDP reward function, detailed in Chapter 5.1.3.4. We also defined a state sequence maximum size of 4 clusters.

After this first stage of learning done, the RL-based algorithm finds itself with a base that is able to start recommending actions by reading an initial action by a given user. As explained, after connecting to a single user, the algorithm starts to learn his preferences and begins to mould itself to him. Similarly, the ISGD model is ready to start receiving pairs of {session, song} from online users and generate specific song recommendations.

7.4 Evaluation

Once the working pipeline has been executed until the General Learning and ISGD initialisation stages on the presented dataset, we defined a form of evaluation to test the technique presented in this dissertation project. In this section, we present the evaluation strategies used, as well as the difficulties and limitations faced. Finally, we comment on the results generated.

7.4.1 Difficulties and Limitations

Despite the data used being good enough for the development of the technique presented, the evaluation was more difficult to define and perform. Some difficulties and limitations were encountered, and various strategies were created to try to work around these setbacks.

The main difficulties faced in carrying out the evaluation were:

- **Data size**: The sessions respective to each user are too short, generating one problem: The algorithm does not have enough data to evaluate a constant and prolonged iteration with the user, not being able to mould itself to them.
- **Impossibility of online evaluation**: Since the songs are anonymized from scratch, it is not possible to perform an online evaluation with real users. Such a task would take too many resources and time, considering that we would have to generate a new big data set with previously collected feedback.

7.4.2 Strategies and Results

Considering the limitations and difficulties presented, it was necessary to think of customised strategies to carry out an evaluation of the proposed tool.

7.4.2.1 Presence Percentage of Action Recommendations

The sessions provided by the data are too short to perform a quality evaluation for when the algorithm is iterating with a user for a large period of time. In this first evaluation strategy, we created a way to evaluate how good our algorithm can be, with just learning the first music categories listened by a user.

For the evaluation, we used the previously mentioned test set. Given that the sessions have a size of between 10 and 20 songs listened, with the majority having a size of 20, we defined as learning size of the algorithm for a specific user/session, the first 4 observations, which represents 20% of the size of most sessions, using the rest of each session to evaluate the performance. In other words, the algorithm looks at the first 4 music categories listened by the user, as well as the associated feedbacks, and generates an action recommendation that is evaluated through the amount of times its listened on the rest of the session.

After evaluating the recommendations individually for each individual session, we pool all the results into a single presence percentage that considers the absolute hits of the recommendations across the entire test set. We define the presence percentage by the following equation:

$$PP = \frac{\sum_{n=1}^{Numberof Sessions} Number\ of\ times\ recommendation\ appears\ in\ rest\ of\ session}{\sum_{n=1}^{Numberof Sessions} Size\ of\ rest\ of\ session}$$

It is important to note that we do not consider sessions where there are context changes, for this evaluation, because of the fact that we are evaluating the algorithm only with the knowledge of few user preferences, due to the lack of quality data.

We applied our algorithm on the test set and used a Most Popular recommendation algorithm, and a Random recommendation algorithm as a comparison for this evaluation, obtaining the results presented in Table 7.2. SkipAwareRec stands out from the other two used for comparison, as can be seen on the table. A percentage of 26.56%, although at first glance it seems a low number, is an excellent recommendation indicator, taking into account that, as seen and analysed previously, the mode of the number of clusters per session is 4. Consequently 1 cluster represents, on average, 25% of each session. The Most Popular recommender also shows a relatively reasonable percentage, much due to the fact that the most popular cluster appears many times in the data: 189,830 presences in 1,000,003 observations (18.98%).

Recommendation Algorithm	Presence Percentage
SkipAwareRec	26.56%
Most Popular Recommender	19.36%
Random Recommender	4.47%

Table 7.2: Presence percentage of action recommendations results

The results demonstrate that SkipAwareRec can produce correct recommendations even with little knowledge of user preferences, taking into account that due to the lack of data, it has only learned the first 4 observations of each session for specific individual users. The main idea, as explained earlier, is for the algorithm, after the general training stage, to fit in with a single user and match that user's preferences over time. Despite the limitations presented for the evaluation scenario, we believe that the algorithm improves more and more through constant and prolonged iteration with the user, becoming more and more framed with their preferences.

7.4.2.2 Iterative Action Recommendations

Despite the limitations presented above, our second evaluation strategy is based on the evaluation of each recommendation generated sequentially to each observation read. At each user observation, the algorithm generates an action recommendation by interpreting the current state. If this recommendation is equal to the action listened by the user in this observation, the strategy returns a positive value, otherwise it returns a negative value. The final value of this strategy is represented by the division between the total number of recommendations with a positive value (recommendations that are consistent with the user's data) and the total number of recommendations made, and is given by the following formula:

$$Hit\ Ratio\ (HR) = \frac{\#\ Guessed\ Recommendations}{\#\ Recommendations} * 100$$

The algorithm correctly hit 31,398 actions out of a total of 107,864 recommendations generated - thus showing a HR of 29.11%. As shown in Table 7.3, the algorithm again outperformed the two other recommendation models proposed for comparison in the evaluations.

Recommendation Algorithm	Guessed Recommendations	Number of Recommendations	HR
SkipAwareRec	31398	107864	29.11%
Most Popular Recommender	20529	107864	19.03%
Random Recommender	4675	107864	4.33%

Table 7.3: Iterative action recommendations results

It is important to remember that in this evaluation we are using different sessions and not a single one, as would be the intention of the algorithm, due to the poor quality of the available data. In case the algorithm was linked to a single user, the learning would be much more consistent.

7.4.2.3 Iterative Items Recommendations

Finally, we have the evaluation of the recommendation of specific items. Contrary to the two strategies mentioned above, this was an evaluation that led to the adaptation of the recommendation algorithm presented. In other words, these results gave rise to the idea of adapting the ISGD into a hybrid recommendation method.

Analogous to the last evaluation method, we evaluated each recommendation in each iteration read in the different sessions. Whenever new information <user, item> is received, a recommendation is generated. If this recommendation is consistent with the item read, it is considered a right recommendation, otherwise it is a wrong recommendation. This evaluation equation can also be described by the HR formula.

Firstly, we look at the results as a whole. Table 7.4 demonstrates the results of two ISGDs applied to the aforementioned recommendation environment. The first ISGD recommends 2 items only from the action generated by the NBA recommendation algorithm, while the second ISGD recommends 2 items from the entire universe of songs. As can be seen in the table, the ISGD algorithm that recommends any item from the universe of songs performs better, by 1.32%. This value is easily explained by the results of the evaluation strategy in Chapter 7.4.2.2, where it was possible to see that, in the data used to evaluate the NBA recommendation algorithm, the HR did not exceed 29.11%. Thus, in sensibly 7 out of 10 observations, the algorithm produces an action recommendation that does not correspond to the observed item, always failing the specific recommendation in view of this recommendation evaluation model.

In the second place, we wanted to check the results of the ISGD algorithm when SkipAwareRec

Recommendation Algorithm	Guessed Recommendations	Number of Recommendations	HR
ISGD with SkipAwareRec items	6626	107864	6.14%
ISGD with all items	8043	107864	7.46%

Table 7.4: Iterative items recommendations results in all recommendations moments

hits the cluster - Table 7.5 demonstrates these results. As can be seen, in this scenario, the ISGD algorithm that recommends only items from the generated NBA is much higher (by 9.98%, almost double) than the ISGD that recommends items from the entire universe of songs.

Recommendation Algorithm	Guessed Recommendations	Number of Recommendations	HR
ISGD with SkipAwareRec items	6626	31398	21.10%
ISGD with all items	3493	31398	11.12%

Table 7.5: Iterative items recommendations results when SkipAwareRec guesses the next action

This insight allowed us to arrive at what is currently SkipAwareRec's item-specific recommendation strategy: the hybrid ISGD. Given that when getting the category right, the ISGD that recommends only items from that same category is much better, and that overall, the ISGD that recommends items from the music universe is slightly better, we decided to create the hybrid strategy. We put together the recommendations from both ISGDs presented and arrive at the results in Table 7.6. As can be seen, the Hybrid ISGD had a performance of 10.36%, higher than the values visualised in Table 7.4. Chart 7.7 presents a performance comparison of the different approaches.

Recommendation Algorithm	Guessed Recommendations	Number of Recommendations	HR
Hybrid ISGD	11176	107864	10.36%

Table 7.6: Iterative items recommendations results with Hybrid ISGD

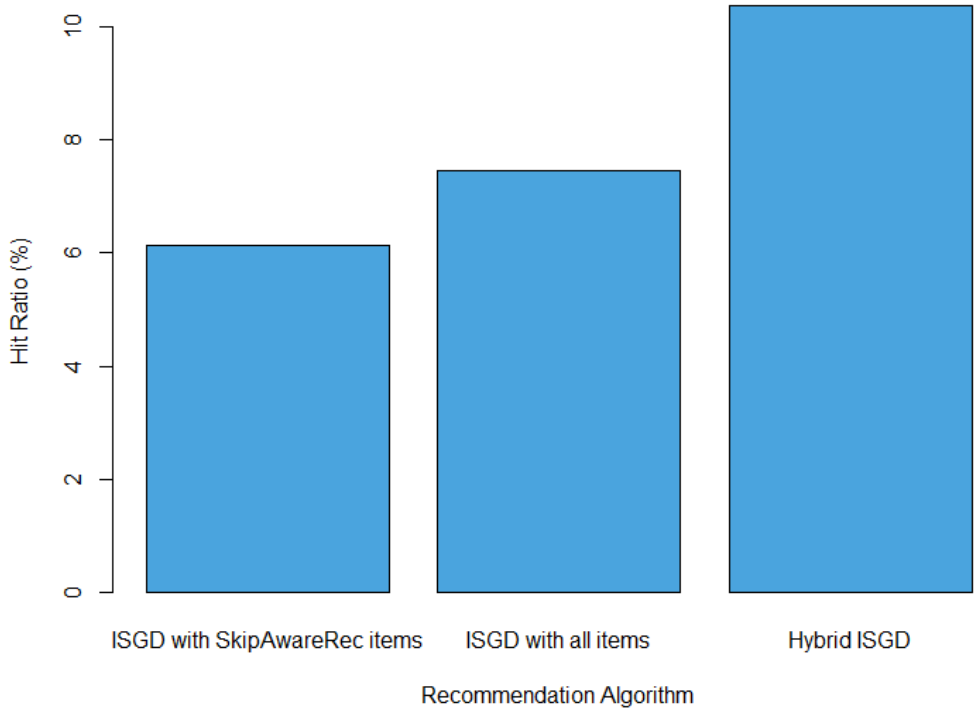


Figure 7.7: Performance comparison of the different approaches

Finally, it is very important to remember that we are recommending only a maximum of 4 specific songs, out of a universe of a total of 174,768 songs. A random recommender would have a hit probability of 0.0023%, while our Hybrid ISGD has a HR of 10.36%.

7.5 Summary

In this chapter we presented a practical experiment conducted with real music listening data from a streaming platform, as well as the whole work pipeline for this same data, including its pre-processing, the generation of the set of actions and the AI task that underlies the recommendation system. Also in this chapter, we presented the evaluation strategies developed, as well as the main limitations and difficulties encountered. Finally, we showed and commented on the results obtained.

We defined evaluation strategies that allowed not only to evaluate each component of SkipAwareRec, but also to evaluate the whole process as one. We showed that our proposed solution for Next-Best Items recommendation paradigms presents quite optimistic results and can be implemented in a real sequential music recommendation scenario.

Chapter 8

Conclusions

We presented in this dissertation a proposed solution to sequential recommendation paradigms where user feedback is quantified implicitly negative. Our solution, SkipAwareRec, is divided in two parts: (1) a RL-based sequential NBA recommendation algorithm, and a (2) hybrid algorithm for recommending specific items taking into account the recommended actions. We also presented a solution that allows to perform the task of generating a set of actions/categories through a set of specific items.

The results, despite the impossibility of testing SkipAwareRec in a situation where there is a prolonged interaction with a user, showed that our solution presents optimistic values even when the algorithm does not have much knowledge about a user's preferences. Furthermore, it has been shown that SkipAwareRec, in a sequential decision paradigm, improves the ISGD algorithm through its presented hybrid strategy.

8.1 Future Work

Finally, some limitations should be noted, which also suggest future research directions. Firstly, as already mentioned in the results section, the data that currently exists online is not of great quality to perform a consistent evaluation, given that the sessions are very small and do not allow to evaluate the connection of the algorithm to a user in the long term. One suggestion for future work is to test the evaluation of SkipAwareRec online, in contact with a real user, or using much larger session data that allows the algorithm to learn more about the user. In this way, validation of the algorithm's long-term properties would be carried out.

Second, this work focused on the music sequential recommendation paradigm, but there are other topics where SkipAwareRec can be adapted and used. For example, our proposed solution can be applied to a scenario of recommendation of publications in social networks, such as TikTok, Instagram Reels and YouTube Shorts. The time a user takes to skip a video can be seen as quantified implicit negative feedback and used to feed SkipAwareRec. This way, publications, after being aggregated into different sets of actions, can be recommended to users through our

solution. Another possible application example is in a news recommendation paradigm. It is easy to estimate the reading time from the text and record what percentage of that time was spent on a given news item. This way, the resulting time can also be used to feed SkipAwareRec.

Appendix A

Study on the Behaviour of Music Streaming Platform Users

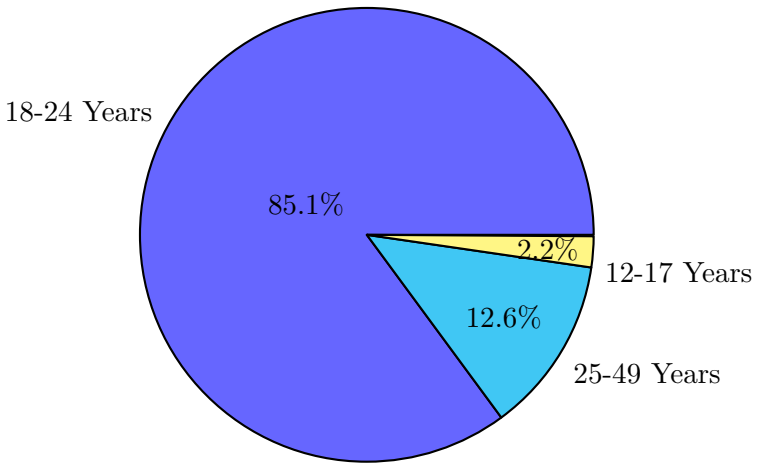
A.1 Results

A.1.1 Number of Answers

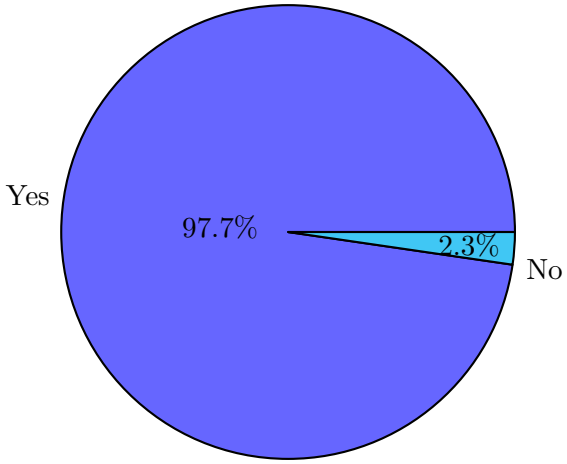
The present survey was online between 4 April 2022 and 11 April 2022, with a total of 87 responses subsequently analysed in the following sections.

A.1.2 General Questions

A.1.2.1 What age group do you belong to?



A.1.2.2 Do you use music streaming platforms? (Examples: Spotify, YouTube Music, Apple Music)



A.1.3 Music Streaming Platforms Behaviour

The answers in this sub-section, were collected only from those respondents who answered "Yes" to the question A.1.2.2.

A.1.3.1 Which music streaming platforms have you used?

Music Streaming Platform	Number of Answers	Percentage of Answers
Spotify	84	98.8
YouTube Music	27	31.8
Apple Music	10	11.8
Tidal	7	8.2
Soundcloud	2	2.4
Deezer	2	2.4
YouTube	2	2.4

A.1.3.2 Which music streaming platforms do you currently use?

Music Streaming Platform	Number of Answers	Percentage of Answers
Spotify	79	92.9
YouTube Music	8	9.4
Apple Music	1	1.2

A.1.3.3 Imagine a scenario in which you are listening to the radio or to someone else's public playlist on a music streaming platform. When you do not like a particular song, what do you usually do?

Answer	Number of Answers	Percentage of Answers
I skip the song	77	90.6
Switch playlist/radio	17	20
I look for a specific new song	14	16.5
I listen to the song till the end	4	4.7

A.1.4 Behaviour in Different Types of Playlists on Music Streaming Platforms

The answers in this sub-section, were collected only from those respondents who answered "Yes" to the question A.1.2.2.

A.1.4.1 Imagine a scenario in which you are listening to the radio or to someone else's public playlist on a music streaming platform. What is/what are the main reasons for skipping a song?

Answer	Number of Answers	Percentage of Answers
I do not like the song	68	80
I do not want to listen to that specific song at the moment	57	67.1
I am trying to find a specific song through the jumps between songs	16	18.8

A.1.4.2 Now imagine a different scenario in which you are listening to a playlist of your own (playlist created by you, with songs chosen by you) on a music streaming platform. What is/what are the main reasons for skipping a song?

Answer	Number of Answers	Percentage of Answers
I do not like the song	2	2.4
I do not want to listen to that specific song at the moment	81	95.3
I am trying to find a specific song through the jumps between songs	40	47.1

Bibliography

[1] Ai applications: Top 14 artificial intelligence applications in 2022. https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/artificial-intelligence-applications. Accessed: 2022-06-05.

[2] Human intelligence. https://www.britannica.com/science/human-intelligence-psychology. Accessed: 2022-06-05.

[3] Spotify sequential skip prediction challenge. https://www.aicrowd.com/challenges/spotify-sequential-skip-prediction-challenge. Accessed: 2022-06-06.

[4] The surprising ways music benefits your brain and body. https://www.betterup.com/blog/benefits-of-music. Accessed: 2022-06-05.

[5] Weekly time spent listening to music in the united states from 2015 to 2019. https://www.statista.com/statistics/828195/time-spent-music/. Accessed: 2022-06-05.

[6] What is principal component analysis (pca) and how it is used? https://www.sartorius.com/en/knowledge/science-snippets/what-is-principal-component-analysis-pca-and-how-it-is-used-507186. Accessed: 2022-04-12.

[7] Working time. https://en.wikipedia.org/wiki/Working_time. Accessed: 2022-06-07.

[8] C.-Y. Chi, R. T.-H. Tsai, J.-Y. Lai, and J. Y. jen Hsu. A reinforcement learning approach to emotion-based automatic playlist generation. In *Proceedings of the 2010 Conference on Technologies and Applications of Artificial Intelligence TAAI2010*. National Taiwan University, Yuan Ze University, 2010.

[9] F. den Hengst, E. M. Grua, A. el Hassouni, and M. Hoogendoorn. Reinforcement learning for personalization: A systematic literature review. *Data Sci.*, 3(2):107–147, 2020.

[10] M. C. Götting. Spotify usage frequency in the u.s. 2017, by age. In *statista*, 2021.

[11] C. Hansen, C. Hansen, L. Maystre, R. Mehrotra, B. Brost, F. Tomasi, and M. Lalmas. Contextual and sequential user embeddings for large-scale music recommendation. In R. L. T. Santos, L. B. Marinho, E. M. Daly, L. Chen, K. Falk, N. Koenigstein, and E. S. de Moura, editors, *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, pages 53–62. ACM, 2020.

[12] D. Hong, Y. Li, and Q. Dong. Nonintrusive-sensing and reinforcement-learning based adaptive personalized music recommendation. In J. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, and Y. Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 1721–1724. ACM, 2020.

[13] B. Hu, C. Shi, and J. Liu. Playlist recommendation based on reinforcement learning. In Z. Shi, B. Goertzel, and J. Feng, editors, *Intelligence Science I - Second IFIP TC 12 International Conference, ICIS 2017, Shanghai, China, October 25-28, 2017, Proceedings*, volume 510 of *IFIP Advances in Information and Communication Technology*, pages 172–182. Springer, 2017.

[14] D. H. Lee and P. Brusilovsky. Reinforcing recommendation using implicit negative feedback. In G. Houben, G. I. McCalla, F. Pianesi, and M. Zancanaro, editors, *User Modeling, Adaptation, and Personalization, 17th International Conference, UMAP 2009, formerly UM and AH, Trento, Italy, June 22-26, 2009. Proceedings*, volume 5535 of *Lecture Notes in Computer Science*, pages 422–427. Springer, 2009.

[15] E. Liebman, M. Saar-Tsechansky, and P. Stone. DJ-MC: A reinforcement-learning agent for music playlist recommendation. In G. Weiss, P. Yolum, R. H. Bordini, and E. Elkind, editors, *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2015, Istanbul, Turkey, May 4-8, 2015*, pages 591–599. ACM, 2015.

[16] Y. Lin, Y. Liu, F. Lin, P. Wu, W. Zeng, and C. Miao. A survey on reinforcement learning for recommender systems. *CoRR*, abs/2109.10665, 2021.

[17] F. Meggetto, C. Revie, J. Levine, and Y. Moshfeghi. On skipping behaviour types in music streaming sessions. In G. Demartini, G. Zuccon, J. S. Culpepper, Z. Huang, and H. Tong, editors, *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, pages 3333–3337. ACM, 2021.

[18] L. Peska and P. Vojtás. Negative implicit feedback in e-commerce recommender systems. In D. Camacho, R. Akerkar, and M. D. Rodríguez-Moreno, editors, *3rd International Conference on Web Intelligence, Mining and Semantics, WIMS '13, Madrid, Spain, June 12-14, 2013*, page 45. ACM, 2013.

[19] S. Shih and H. Chi. Automatic, personalized, and flexible playlist generation using reinforcement learning. In E. Gómez, X. Hu, E. Humphrey, and E. Benetos, editors, *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018, Paris, France, September 23-27, 2018*, pages 168–174, 2018.

[20] J. Vinagre, A. M. Jorge, and J. Gama. Fast incremental matrix factorization for recommendation with positive-only feedback. In V. Dimitrova, T. Kuflik, D. Chin, F. Ricci, P. Dolog, and G. Houben, editors, *User Modeling, Adaptation, and Personalization - 22nd*

International Conference, UMAP 2014, Aalborg, Denmark, July 7-11, 2014. Proceedings, volume 8538 of *Lecture Notes in Computer Science*, pages 459–470. Springer, 2014.

[21] Y. Wang. A hybrid recommendation for music based on reinforcement learning. In H. W. Lauw, R. C. Wong, A. Ntoulas, E. Lim, S. Ng, and S. J. Pan, editors, *Advances in Knowledge Discovery and Data Mining - 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11-14, 2020, Proceedings, Part I*, volume 12084 of *Lecture Notes in Computer Science*, pages 91–103. Springer, 2020.

[22] X. Xin, A. Karatzoglou, I. Arapakis, and J. M. Jose. Self-supervised reinforcement learning for recommender systems. In J. Huang, Y. Chang, X. Cheng, J. Kamps, V. Murdock, J. Wen, and Y. Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 931–940. ACM, 2020.

[23] J. Zhang, B. Hao, B. Chen, C. Li, H. Chen, and J. Sun. Hierarchical reinforcement learning for course recommendation in moocs. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 435–442. AAAI Press, 2019.

[24] X. Zhao, L. Zhang, Z. Ding, L. Xia, J. Tang, and D. Yin. Recommendations with negative feedback via pairwise deep reinforcement learning. In Y. Guo and F. Farooq, editors, *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, pages 1040–1048. ACM, 2018.