

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Inferring User Preferences by Analyzing Their Behavior on Streaming Platforms Using Inverse Reinforcement Learning

António Pedro Dantas



FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Mestrado em Engenharia Informática e Computação

Supervisor: Ana Paula Rocha

Second Supervisor: António Castro

July 27, 2022

Inferring User Preferences by Analyzing Their Behavior on Streaming Platforms Using Inverse Reinforcement Learning

António Pedro Dantas

Mestrado em Engenharia Informática e Computação

July 27, 2022

Abstract

Inverse Reinforcement Learning (IRL) is a rather recent installment in the world of Artificial Intelligence. It is a Machine Learning branch deriving from Reinforcement Learning (RL). The goal of RL is to learn a decision process that maximizes an already established reward function. On the other hand, IRL inverts this problem. Its goal is to extrapolate a reward function by observing an expert agent's behavior on a dynamic environment. This method relies on Markov Decision Processes to learn from demonstrations and explain the experts behavior. Contrary to its counterpart, RL, which is commonly associated being used for robotics and engineering problems, Inverse Reinforcement Learning has the potential to dwell into real-life situations where the reward function is not straightforward and thus very hard to understand.

This leads to the main topic of this dissertation. The plan is to infer user preferences on a social network or streaming platform using these types of algorithms. It is complicated to tell a computer what we do or do not like, human nature is complicated and programming a reward function that tries to address it would be tedious and prone to errors. However, there is a possibility that by observing a user's behavior an IRL model could come up with a reward function that predicts its preferences.

Spotify will be the case study, we will use Spotify's API to specify the group of inputs we want to analyse and then observe the user's behavior over a period of time. The data collected and the shown results can then be used for a plethora of goals. It can improve user recommendations on the platform and improve the overall app's experience, it can enhance advertising algorithms and lead to a better understanding of what the user really wants. Shall the work be successful, it can be extended to further areas of studies. Analyzing users' behavior with this type of algorithm can be used to predict how people interact with many types of web platforms and enhance the overall user-experience.

Keywords: inverse reinforcement learning, user preferences, machine learning, artificial intelligence, social networks, streaming platforms, spotify

Resumo

O *Inverse Reinforcement Learning* é um capítulo bastante recente no mundo da Inteligência Artificial. É um ramo de *Machine Learning* que deriva do *Reinforcement Learning*. O objetivo da RL é aprender um processo de decisão que maximize uma função de recompensa já estabelecida. Por outro lado, o IRL inverte esse problema. O seu objetivo é extrapolar uma função de recompensa observando o comportamento de um agente especialista num ambiente dinâmico. Este método baseia-se em Cadeias de Decisão de Markov para aprender com as demonstrações e explicar o comportamento dos especialistas. Ao contrário da sua contraparte, RL, que é normalmente associada ao uso na robótica e problemas de engenharia, o *Inverse Reinforcement Learning* tem o potencial de ser aplicado em situações da vida real onde a função de recompensa não é direta e, portanto, muito difícil de entender.

Isto leva-nos ao tema principal desta dissertação. O plano é inferir as preferências do utilizador numa rede social ou plataforma de *streaming* usando esse tipo de algoritmos. É difícil dizer a um computador o que gostamos ou não, a natureza humana é complicada e programar uma função de recompensa que tente resolver isso seria tedioso e sujeito a erros. No entanto, existe a possibilidade de que, ao observar o comportamento de um utilizador, um modelo IRL possa sugerir uma função de recompensa que preveja as suas preferências. Para isso, usaremos a API do *Spotify* para especificar o grupo de entradas que queremos analisar e, em seguida, observar o comportamento do utilizador ao longo de um período de tempo.

Os dados e os resultados obtidos podem ser usados para uma infinidade de propósitos. Podem melhorar as recomendações do utilizador na plataforma e a experiência geral na utilização da aplicação, e podem também aprimorar os algoritmos de publicidade levando a um melhor entendimento sobre as preferências do utilizador. Este trabalho pode ainda ser estendido a outras áreas de estudos. A análise do comportamento dos utilizadores com este tipo de algoritmos pode ser utilizada para prever como as pessoas interagem com vários tipos de plataformas na internet e melhorar globalmente a experiência do utilizador.

Keywords: inverse reinforcement learning, user preferences, machine learning, artificial intelligence, social networks, streaming platforms, spotify

Acknowledgements

Quero agradecer aos meus pais e a toda a minha família, por todo o tempo, dinheiro e amor investido em mim para tornar isto possível.

Agradecer à Beatriz por todo o carinho e paciência que tem comigo, proporcionando ajuda nos momentos de frustração.

A todos os meus amigos pelos bons momentos de descontração nos intervalos do trabalho e do estudo. Um agradecimento especial ao Guilherme e ao João por lavarem a loiça e fazerem o jantar enquanto estava ocupado com este projeto.

Mencionar também todos os meus colegas que estiveram do meu lado nesta viagem de 5 anos. Um abraço para o Bernas e o Bernardo, o Cajó, a Kaká, o David, o Macedo, o Matos, o Luís, o Manel, a Matilde, o Miguel, o Tito e o Vítor.

Aos professores Ana Paula Rocha e António Castro, orientadora e co-orientador, por me proporcionarem todas as ferramentas e indicações para tornar este trabalho possível.

Por fim, um agradecimento especial a todas as pessoas que fizeram parte da fase de testes deste projeto e contribuíram para uma análise de resultados mais completa.

António Pedro Dantas

“Não se pode confundir velocidade com andamento.”

Random guy at Café Au Lait

Contents

1	Introduction	1
1.1	Context	1
1.2	Framing and Motivation	1
1.3	Objectives	2
1.4	Document Structure	2
2	State of the Art	4
2.1	Markov Decision Processes	4
2.2	Reinforcement Learning	6
2.3	Inverse Reinforcement Learning	7
2.3.1	Formulation	8
2.3.2	Algorithms	9
2.3.3	IRL v.s. IL	12
2.4	Inferring User Preferences	13
2.4.1	Collaborative Filtering	13
2.4.2	Content-Based Filtering	14
2.4.3	Demographic Filtering	15
2.4.4	Knowledge-Based Filtering	15
2.4.5	Hybrid Filtering	16
2.5	Bridging the Gap	17
2.6	Summary	18
3	Problem and Proposed Solution	19
3.1	Problem	19
3.2	Proposed Solution	20
3.2.1	Information Fetching	22
3.2.2	IRL Implementation	25
3.3	Application	28
3.4	Summary	29
4	Experiments and Results	32
4.1	Experiment	32
4.2	Results & Analysis	34
4.2.1	Results	34
4.2.2	Analysis	38

5	Conclusions and Future Work	40
5.1	Conclusions	40
5.2	Further Work	41
	References	43
A	IRL Code Snippet	46
B	Form	49

List of Figures

2.1	Agent-Environment in Reinforcement Learning Loop [3]	6
2.2	Comparison between IRL and RL methodologies [9]	8
2.3	Non-Optimized Imitation Learning problem [15]	13
3.1	Spotify annual users 2015 to 2022 (mm) [18]	20
3.2	Project architecture	22
3.3	Diagram explaining the information fetching process.	23
3.4	Endpoint returning the user's top tracks.	24
3.5	Endpoint searching for tracks the user might like.	25
3.6	Endpoint getting user spotify recommendations.	26
3.7	Diagram explaining the IRL implementation.	27
3.8	Application's Login page.	29
3.9	Application's feed before recommendation fetching.	30
3.10	Application's feed after recommendation fetching.	31
3.11	Application's Experiments page.	31
4.1	User's Spotify dashboard upon playlist creation.	33
4.2	User's updated Spotify playlists.	34
4.3	Age distribution.	36
4.4	(a) Previously Unknown Songs (b) Previously Unknown Artists	36
4.5	(a) Previously Unknown Songs (b) Previously Unknown Artists	37
B.1	Form's first page.	50
B.2	Form's second page.	51
B.3	Form's third page.	52
B.4	Form's fourth page.	53

List of Tables

2.1	State-Value Table	7
2.2	CF user scores representation [27]	14
2.3	Restaurant representation in a database [19]	15
3.1	Spotify annual subscribers by region 2018 to 2021 (mm) [18]	21
4.1	Form questions	35
4.2	First and Second Questions Results (Number of liked songs)	37
4.3	Fifth Question Results	38

Listings

3.1	Reward system's structure.	26
A.1	Method that solves the Reward function by attributing the best parameter values to the Reward system.	46

Abbreviations

AI	Artificial Intelligence
API	Application Programmimg Interface
CBF	Content-Based Filtering
CF	Collaborative Filtering
DF	Demographic Filtering
HF	Hybrid Filtering
HTTP	Hypertext Transfer Protocol
IL	Imitation Learning
IRL	Inverse Reinforcement Learning
IT	Information Technology
LP	Linear Programming
MDP	Markov Decision Process
ML	Machine Learning
RL	Reinforcement Learning

Chapter 1

Introduction

This dissertation aims to evaluate the capabilities of Inverse Reinforcement Learning when predicting user behavior or infer its preferences in streaming platforms. There are already a lot of algorithms capable of performing this function, but our main objective is to test the potential for this type of artificial intelligence being successful in this area. Inverse Reinforcement Learning, much like its counterpart Reinforcement Learning, has been used almost exclusively on robotics projects like self driving cars for example. It would be interesting to see this working as a way of content recommendation, completing and possibly aiding already existing algorithms.

1.1 Context

Inverse Reinforcement Learning (IRL) is a relatively recent branch of Machine Learning algorithms. It focuses on the problem of predicting hidden behavioral preferences by observing the actions of an agent. Its capability of building autonomous agents that can mimic an expert's conduct using data of previous performances is quite alluring because it doesn't need to intervene in the performance of the task nor use a previously established reward policy of any sort [2].

On a completely different note, the use of recommendation algorithms deriving from the inference of user preferences, not only on streaming platforms but also on e-commerce websites and social networks, seems to be an ever-growing problem. The increasing scale of these services begs for the existence of a way of finding what the user deems more relevant inside the almost infinite stream of content that these platforms hold [30].

1.2 Framing and Motivation

According to Arora and Doshi, IRL's primary use cases have been focused on the world of robotics and vehicle control [2]. This means that it has been widely tested for industrial purposes on assembly-line robots and for easing human life on self-driving cars and other means of transportation. "An early and well-known application that brought significant attention to IRL is helicopter flight control". When it comes to predictions, these algorithms don't stray much further from the

ones described previously and have proved to be successful in forecasting taxi routes and footstep placement for planning legged locomotion.

Besides its high potential for inferring and predicting user behavior, IRL has yet to be widely used on current recommendation systems. These are usually based on other machine learning (ML) techniques, such as clusters, decision trees and artificial neural networks. Instead of relying on the user's behavior, they are based around Bayesian classifiers that identify content that is considered "relevant" or "irrelevant" [1]. There are, already, some studies that show the potential for IRL implementation on user behavior analysis on social networks [12] but this approach seems to lack in popularity in the current state of the art for recommending content on all these types of platforms.

1.3 Objectives

Bearing this lack of research in the field in mind, it would be interesting to have more information about how IRL would behave in this environment. Hence, the proposed objective for this dissertation aims to fill in this gap and bring advancements to the current literature on the topic.

Spotify is the platform chosen to investigate this possibility. Due to the vast and well-documented API provided by the company itself, it will be possible to develop an app that utilizes some user's credentials for authentication and extract its interaction history with this music streaming platform. With this information, an IRL approach should be able to infer its musical tastes and match its profile to music, artists, albums or playlist predictions. The developed app should then, upon login, show the user what its deduced sonic likes are as well as offer a group of content predictions it might be into as well.

Upon this proposed app's development, the time for analyzing the results attained will come. By gathering a considerable population sample that is asked to utilize the resulting platform and rating its overall accuracy, it will be possible to evaluate in detail the success and relevancy of an IRL application of this sort. The results could vary from an impracticality of an Inverse Reinforcement Learning approach to the content recommendation problem to an advancement in the current state of the art or the completion and aid of already existing technologies in the field. Either way, this work has the potential to push the literature on the topic forward by merging two components of the Computer Science area that are rarely seen together.

1.4 Document Structure

This document is composed of three main parts that will hopefully enlighten the reader and provide all information necessary for a clear understanding of the matters at hand.

Chapter 2 focuses on providing all the information relative to the State of the Art of all technologies that are dealt with in this dissertation. This includes a review on both RL and IRL, current content recommendation algorithms and how these two areas have previously been put together. It also holds information about how current content recommendation systems work.

Chapter 3 includes a detailed description of the proposed problem and the approach chosen to hopefully solve it. It also holds an in-depth description of the resulting platform and how it works.

Chapter 4 holds information about how the resulting solution's evaluation was conducted and how the experiments to assess its success were constructed.

Finally, Chapter 5 aims to analyze the results that were obtained and the viability of the achieved solution for later implementation in already existing platforms.

There are also two sections dedicated to showing important parts of the development process. However they aren't crucial to the clear understanding of this dissertation and would slow down the reader, so, they take the form of Appendix A and B.

Chapter 2

State of the Art

This chapter is dedicated to understanding everything related to the matters at hand. It holds information about both the background of the area and the most recent stages on themes of Inverse Reinforcement Learning and Streaming Platforms.

It goes without saying that before mastering all the concepts related to IRL, it will be necessary to look at where it comes from. Understanding these concepts will be vital to developing an effective solution to our problem. Investigating how recommendation algorithms and how user preferences are inferred currently on these platforms will also be fundamental.

2.1 Markov Decision Processes

To understand how Inverse Reinforcement Learning will be computed and applied there needs to be a clear comprehension of **Markov Decision Processes** (MDP). What are they? How do they work? Reinforcement Learning is also dependent and widely influenced by this mathematical framework. MDPs are an instinctual formalism for *decision-theoretic planning*, that is, a way to model an environment so that a computer algorithm can then be able to perform decisions. "The goal is to control the system in such a way that some performance criterium is maximized" [28]. Here, they'll be explained based on their adaptation to both RL and IRL problems. MDP systems are based around the concept of an **environment**. This environment is composed of **states** in which an agent can be at. In order to transition between states it needs to perform an **action**. The new state it transitions to is dependent not only on the state it is currently at, but also in the action it takes. A **reward** is a value the agent receives for entering a certain state or performing an action. It can be either positive or negative, depending on whether the agent should be rewarded or punished for the matter at hand. To easily push these concepts forward, it is commonly assumed that $\mathcal{S}$ is the set of all states, $\mathcal{A}$ represents the set of all possible actions a and the reward function is represented by R .

The reward function is usually defined as $R : S \rightarrow \mathbb{R}$, specifying the rewards attained in each state. However, there are two more distinct definitions specified in Equation 2.1. "The first one gives rewards for performing an action in a state, and the second gives rewards for particular transitions between states" [28].

$$R : S \times A \rightarrow \mathbb{R} \quad \text{and} \quad R : S \times A \times S \rightarrow \mathbb{R} \quad (2.1)$$

Furthermore, it is considered that T is a transition function defined as Equation 2.2. This represents the probabilities of transitioning from one state to another using a certain action [28].

$$T : S \times A \times S \rightarrow [0, 1] \quad (2.2)$$

It may be represented as $T_{ss'}^a$, inferring that this pays respect to the probability of transitioning from state s to state s' using action a .

With this, the tuple $\langle S, A, T, R \rangle$ is created. Finally, Markov decision processes also acknowledge the existence of a policy π , which refers to the probability of taking a specified action on the state the agent is currently in [31]. It is a function that outputs an action for a given state, specified in Equation 2.3.

$$\pi : S \times A \rightarrow [0, 1] \quad (2.3)$$

Sometimes it is also important to introduce a constant γ that serves as a discount factor. Its main goal is to gradually lower the weight of past rewards as the agent makes advancements in the environment. This new variable adds up to the previously established tuple, $\langle S, A, T, \gamma, R \rangle$ [17].

Some other concepts of MDP theory need to be introduced in order to understand how to estimate the performance of a policy. The **Value function**, $V^\pi(s)$, is responsible for outputting the evaluation of the reward values expected to receive when executing the policy π starting from S_1 . But sometimes, it makes more sense to evaluate this as a state-action pair. The **Q-function**, $Q^\pi(s, a)$, comes into play in this situation, by outputting the expected return when taking action a in state s and following policy π afterwards [17][31]. The formulas to these functions are demonstrated in Equations 2.4 and 2.5, considering E as notation for the expectation values and $s' \rightarrow T_{sa}$ being the case where this expectation is with respects to s' according to the transition probability upon taking action a in state s , T_{sa} :

$$V^\pi(s_1) = E[R(s_1) + \gamma R(s_2) + \gamma^2 R(s_3) + \dots + \gamma^{n-1} R(s_n) | \pi] \quad (2.4)$$

$$Q^\pi(s, a) = R(s) + \gamma E_{s' \rightarrow T_{sa}}[V^\pi(s')] \quad (2.5)$$

Then, as expected, the goal of the agent is "to find an optimal policy π^* which gives the optimal state Value function or Q-value function" [31]. They are determined in Equation 2.6 and 2.7:

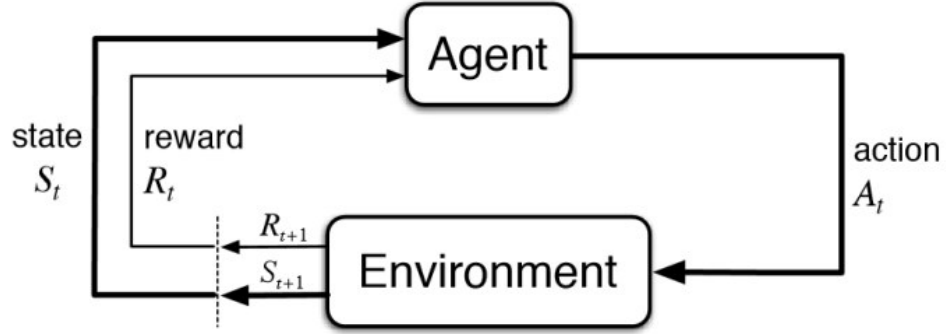


Figure 2.1: Agent-Environment in Reinforcement Learning Loop [3]

$$V^*(s) = \max_a \sum_{s' \in S} T_{ss'}^a (R_{ss'}^a + \gamma V^*(s')) \quad (2.6)$$

$$Q^*(s, a) = \sum_{s' \in S} T_{ss'}^a (R_{ss'}^a + \gamma \max_{a'} Q^*(s', a')) \quad (2.7)$$

2.2 Reinforcement Learning

IRL is directly dependent on Reinforcement Learning (RL) concepts, the part of Machine Learning algorithms it derives from. This field of artificial intelligence (AI), as understood by Sutton and Barto, RL, explores a way of problem-solving based on the interaction between mapped states and actions that tend to maximize an already established reward [23]. For every state, the agent is in, taking an action leads it to another state. Traditionally, an agent is given a reward function that associates numerical values to possible states that are eventually reached during its course. Figure 2.1 illustrates the typical framing of a Reinforcement Learning scenario.

It is also given a set of actions that make it possible for the agent to change states. Its goal is to maximize the rewards that he receives during the course it chooses to take, no actions are given to him. Then, by trial and error he comes up with a solution to the problem, updating the value of each action in a specific state. This is called policy and is the way for the agent to know which action it should take given the current scenario. According to Leonor Baptista Santos, "the RL learning process is all about finding the balance between exploration and exploitation. The system should replicate actions that led to higher rewards (exploitation, greedy actions) and discover new actions that will potentially achieve better rewards (exploration, non-greedy actions)"[8].

To undertake the trial and error phase that makes these algorithms discover the best actions to take, for all the possible states in the environment, $\mathbf{S}$, each one of them holds values for each of its possible actions, $\mathbf{A}$. The values are directly influenced by the reward system, and are constantly updated by whichever reward value, $\mathbf{r}$, it receives when performing an action. This value is dependant on the reward function $\mathbf{R}$. The update method, **update()**, can also vary, depending on

Table 2.1: State-Value Table

States	Values
$S_1(a_1)$	2.4
$S_1(a_2)$	1.5
$S_2(a_1)$	-3.1
$S_2(a_2)$	0.2
...	...

the chosen implementation. For every step it takes, it needs to choose an action. To balance the exploration/exploitation approach, it usually is given an α value between 0 and 1 that holds the probability of exploring or exploiting. After choosing its action, it is ready to update the value of the state according to the reward it received. Equation 2.8 aims to explain this scenario:

$$S_n(a_n) = \text{update}(r_a) \quad (2.8)$$

After the training phase, the agent uses these values to decide the best action to take, depending on the state it is currently in. It now holds results for every possible action in every possible state and uses them to navigate its environment. An example of this is given in Table 2.1. Given its state, it should choose the action with the highest value.

Besides having some good results in applications such as natural language processing or the stock exchange study, usually, these problems take up the form of games. One of its most recognized uses has been on the creation of gaming agents that can beat already experienced players, ranging from some niche market *Atari* games to world of Chess [16]. This difficulty in addressing real world situations is very well explained by Yuxi Li when talking about how reward specification may cause problems [11]. A reward function may not represent the intentions of the designer and for systems of a higher degree of complexity, this task can become utopic. Let's take the example of autonomous driving vehicles. Given the density of the environment that is the real world, it is almost impossible to come up with a finite set of states for the driving agent. Even if there was a way of simplifying this, coming up with a reward system that covers every possible action would be tedious as well. Besides of having the need to understand every road rule, the rewards would have to reflect real human behavior that can be particularly hard to pinpoint, such as defense driving mechanisms, toll and parking payments and fuel management.

2.3 Inverse Reinforcement Learning

Inverse Reinforcement Learning reverses the problem approach to RL in the sense that instead of receiving a reward function and deducing desired behavior by trial and error, it first gets an extensive set of actions from an expert agent and then comes up with its reward function. "The most intuitive representation of an expert's intention is its policy, and it is possible to directly learn a policy from expert's demonstrations" [31]. There are certain situations where thinking this way

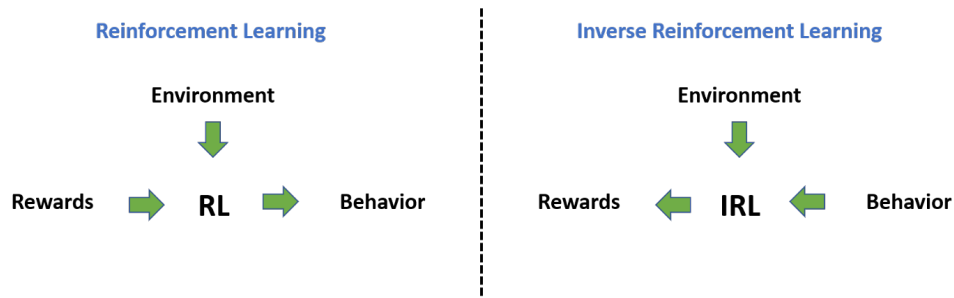


Figure 2.2: Comparison between IRL and RL methodologies [9]

makes sense, specifically when trying to mimic human and animal behavior. In these problems the reward function must be considered unknown in order to attain it. The reasons for this are two-fold. In one hand, specifying a reward function is an empirical hypothesis that may or may not be in tune with the actual environment. On the other, the weights of a multi-attribute reward function cannot be determined *a priori* and so it will be impossible to determine the values of its parameters [22].

Inverse Reinforcement Learning can be seen as a reverse procedure of RL problems. In the case of IRL, the expert's demonstration is considered to be the optimal policy π^* that is influenced by an unknown reward function R^* [31]. This can be easily observed in Figure 2.2.

It is pretty easy to understand the concept behind this AI field but putting it to work can be quite challenging. There needs to be a way to formulate these types of problems to be aware of the best approach to take and algorithms that let us solve them more efficiently.

2.3.1 Formulation

According to Russel, every IRL implementation needs to begin with the following thought experiment [22].

- **Given**

1. measurements of an agent's behavior over time, in various circumstances.
2. measurements of the sensory inputs to that agent.
3. a model of the physical environment (including the agent's body).

- **Determine** the reward function that the agent is optimizing.

The reward function should be pretty straightforward and dependent on the agent's sensory inputs. Thus, it should simplify the expert's behavior in a compact, robust, much simpler way than the direct decision mapping. But what does this mean in practical terms? How will it infer the reward function? How should IRL be applied when computing it directly?

The MDP terminology can come into play when formulating an IRL problem. Taking advantage of the aforementioned notation Russel's thought takes up a new form.

- **Given**

1. a finite space S
2. a set of actions A

3. transition probability T
4. discount factor γ
5. a policy π

- **Determine** a set of possible reward functions R such that π is the optimal policy for the MDP $\langle S, A, T, \gamma, R \rangle$.

The goal is to find a reward function that can explain the observed behavior. Since this is the most specific and characteristic feature of Inverse Reinforcement Learning problems, there will be the need to assume two classic MDP theorems [23].

Theorem 1 (Bellman Equations). *Let a Markov Decision Process $M = \langle S, A, T, \gamma, R \rangle$ and a policy $\pi : S \rightarrow A$ be given. Then, for all $s \in S, a \in A, V^\pi$ and Q^π satisfy:*

$$V^\pi(s) = R(s) + \gamma \sum_{s'} T_{ss'}^{\pi(s)} V^\pi(s')$$

$$Q^\pi(s, a) = R(s) + \gamma \sum_{s'} T_{ss'}^a V^\pi(s')$$

Theorem 2 (Bellman Optimality). *Let a Markov Decision Process $M = \langle S, A, T, \gamma, R \rangle$ and a policy $\pi : S \rightarrow A$ be given. Then, π is an optimal policy for M if and only if, for all $s \in S$:*

$$\pi(s) \in \arg \max_{a \in A} Q^\pi(s, a)$$

2.3.2 Algorithms

To come up with a practical IRL algorithm, the matters at hand can be looked at as a Linear Programming (LP) problem. The constraints correspond to the optimal policy. The current literature identifies three prominent cases on the IRL problem formulation [31]:

- Finite-state MDP with known optimal policy.
- Infinite-state MDP with known optimal policy.
- Infinite-state MDP with unknown optimal policy, but demonstrations are given.

Finite-State Spaces

In the case of a finite-state space, a known model and a complete observed policy it is possible to deduce a set of reward functions that satisfy the given policy. The intention is to find a way of characterizing a solution set. Assuming that $\pi(s) \equiv a^*$, the best way to achieve this is using Equation 2.9. T_{a^*} is the transition probability of taking the optimal action a^* , and T_a is the transition probability of other policies with $a \in A/a^*$.

$$(T_{a^*} - T_a)(I - \gamma T_{a^*})^{-1} R \geq 0 \quad (2.9)$$

This formula is helpful to understand how it is possible to get to the unknown reward function. However, it has some issues. The main one being that it accepts multiple optimal policies under the same reward function and multiple reward functions to the same policy [14]. One of these

solutions is the one that states that $\mathbf{R} = 0$, which means that "there are no distinctions among actions and makes $\pi \equiv a^*$ one of the optimal solutions"[31].

To counteract this, two additional criteria are introduced to differentiate the vast number of solutions to a good choice for $\mathbf{R}$. The idea is to *maximize the difference between the best and second-best solutions* and *give preference to simplified solutions*. Equation 2.10 bears these ideas in mind, where $\min_{a \in A/a^*} \{(T_{a^*}(i) - T_a(i))(I - \gamma T_{a^*}(i))^{-1} R\}$ is the return distance between the best and second best policy at time step i for the number of states N , and $\lambda \|\mathbf{R}\|_1$ is a penalty term added to balance simplicity of the reward function. Here, λ is an adjustable coefficient that balances between the goals of having weak reinforcements and of maximizing the sum of all the value differences of not taking the optimal action. The Equation can then be solved using Linear Programming (LP) [17].

$$\max \left\{ \sum_{i=1}^N \min_{a \in A/a^*} \{(T_{a^*}(i) - T_a(i))(I - \gamma T_{a^*}(i))^{-1} R\} - \lambda \|\mathbf{R}\|_1 \right\} \quad (2.10)$$

Large State Spaces

Considering the case of an Infinite-State Space, it would be unfeasible to search through all reward functions. In this setting, the set of possible states would be $S \rightarrow \mathbb{R}^n$ meaning that $\mathbf{R}$, the set of all reward functions, would be $\mathbb{R}^n \rightarrow \mathbb{R}$ [17]. Instead, the reward function is commonly reached using a linear approximation expressed in Equation 2.11. It assumes that $\phi_1, \dots, \phi_d$ are fixed, known, bounded features (basis functions) mapping S into $\mathbb{R}$, and $\alpha_i(s)$ are the unknown parameters that need to be fine-tuned.

$$R(s) = \alpha_1 \phi_1(s) + \alpha_2 \phi_2(s) + \dots + \alpha_d \phi_d(s) \quad (2.11)$$

Given that R is linear in the variables it will optimize, a linear programming formulation will apply here as well. $V(\pi)$ can then be represented according to Equation 2.12 and 2.13, where α is a weight vector and $\mu(\pi)$ is a feature vector. Furthermore, since $\pi(s) \equiv a^*$ is the optimal policy, Equation 2.14 also holds, meaning that the expert, given its policy π^E , will always outperform or match any other policy $\pi^{(i)}$ [31].

$$V(\pi) = \sum_{i=1}^d \alpha_i \mu_i(\pi) = \alpha \mu(\pi) \quad (2.12)$$

$$\mu_i(\pi) = E \left[\sum_{t=0}^{\infty} \gamma^t \phi_i(s_t) | \pi \right] \quad (2.13)$$

$$\alpha \mu(\pi^E) \geq \alpha \mu(\pi^{(i)}) \quad (2.14)$$

"There are however two problems with the current formulation" [17]. Firstly, there are infinitely many features that need to be looked at due to the infinity of the space state. To circumvent

this problem, usually, it is only sampled a large but finite subset S_0 of the states and extracting features from those states only. The second problem has to do with the linear approximation used in Equation 2.11, meaning that there may not exist an R other than $R = 0$ for which π is optimal. With this in mind, the optimization problem is relaxed with a penalty when the constraint of Equation 2.14 is violated. Here, $q(x)$ is a penalty function and c is a positive constraint [31].

$$\max \left\{ \sum_{s \in S_0} \min_{a \in A/a^*} \{q(\alpha\mu(\pi^E) \geq \alpha\mu(\pi^{(i)}))\} \right\} \quad \text{and} \quad s.t. |\alpha_i| \leq 1, i = 1, 2, \dots, d \quad (2.15)$$

$$q(x) = \begin{cases} x & \text{if } x \geq 0 \\ cx & \text{if } x < 0 \end{cases} \quad (2.16)$$

Evaluation

A Bayesian approach could be used to evaluate the uncertainty of the obtained reward function. A high uncertainty would mean that the R reached is not a good solution, whereas a small uncertainty would suggest the opposite. Since uncertainty exists in the obtained reward function, it is only natural to use a probability distribution to model this. "Bayesian IRL approaches the IRL problem from this perspective, and treats the demonstration sequences as the evidence and calculates a prior on the reward function" [31]. To make this work, the previously mentioned MDP $\langle S, A, T, \gamma, R \rangle$ needs to be considered along with an expert agent, χ , operating in the MDP. There also exists an R that χ follows chosen from a (known) prior distribution P_R . The IRL agent receives a set of expert behavior observations, $O_\chi = \{(s_1, a_1), (s_2, a_2), \dots, (s_n, a_n)\}$, meaning that χ was in state s_i and took action a_i at time step i . By assuming that χ tries to maximize the total accumulated reward according to R and that it is executing a stationary policy, meaning that new actions are not dependent on previous ones, the following independence assumption is reached taking the form of Equation 2.17.

$$Pr_\chi(O_\chi|R) = Pr_\chi((s_1, a_1)|R)Pr_\chi((s_2, a_2)|R)\dots Pr_\chi((s_n, a_n)|R) \quad (2.17)$$

"The expert's goal of maximizing accumulated reward is equivalent to finding the action for which the Q^* value at each state is maximum. Therefore the larger $Q^*(s, a)$ is, the more likely it is that χ would choose action a at state s " [21]. This is modeled in Equation 2.18 by an exponential distribution for the likelihood of (s_i, a_i) , with Q^* as a potential function, where α_χ is a parameter representing the degree of confidence we have in χ to choose the best action to take and Z is the appropriate normalizing constant:

$$Pr_\chi(O_\chi|R) = \frac{1}{Z_i} e^{\alpha_\chi Q^*(s_i, a_i, R)} \quad (2.18)$$

Now, by applying the Bayes theorem in Equation 2.19, the posterior probability of the reward function R can be computed:

$$Pr_{\chi}(O_{\chi}|R) = \frac{Pr_{\chi}(O_{\chi}|R)Pr(R)}{Pr(O_{\chi})} = \frac{1}{Z_i} e^{\alpha_{\chi} Q^*(s_i, a_i, R)} Pr(R) \quad (2.19)$$

2.3.3 IRL v.s. IL

Imitation Learning (IL) is another form of artificial intelligence paradigm that relies on the observation of expert agents' behavior. Since the premise of these two approaches is so similar and they both share Markov properties to achieve the respective solutions, their concepts often get mixed up or even misunderstood. In order to correctly develop an Inverse Reinforcement Learning approach to the problem of user preferences, it is crucial to be conscious of where these types of algorithms differ.

"Imitation learning works by extracting information about the behavior of the teacher and the surrounding environment including any manipulated objects, and learning a mapping between the situation and demonstrated behavior" [10]. This means that the concept of States and Actions are interchangeable between these two approaches. However, IL assumes that the expert agent's policy is the only one that needs to be reproduced and does not usually take into account a reward function. Due to this fact, IL implementations are not usually chosen when dealing with a dynamic environment. In order for them to work, optimization through trial and error needs to be introduced, which means coming up with reward systems that are designed specifically for each task. Defining rewards for such problems is complicated and in many cases unknown as well. IL algorithms that do not use concepts of RL to some degree in order to optimize their results are commonly referred to as Behavioral Cloning. It uses Supervised Learning to understand the expert's policy. "In some applications it can work excellently. Although, for the majority of the cases, behavioural cloning can be quite problematic. The main reason for this is that in MDP an action in a given state induces the next state, which breaks the Supervised Learning assumption that state action pairs are distributed. This also means, that errors made in different states add up, therefore a mistake made by the agent can easily put it into a state that the expert has never visited and the agent has never trained on. In such states, the behaviour is undefined and this can lead to catastrophic failures" [15]. Figure 2.3 tries to make it easier to understand how this can become a problem.

It is now safe to assume that the differences between IL and IRL are clear. IL tries to mimic a behavior while IRL tries to understand what foundation is responsible for the expert agent to behave that way. "On the one hand, IRL is a paradigm relying on Markov Decision Processes (MDPs), where the goal of the apprentice agent is to find a reward function from the expert demonstrations that could explain the expert behavior. On the other hand, IL consists in directly generalizing the expert strategy, observed in the demonstrations, to unvisited states (and it is therefore close to classification, when there is a finite set of possible decisions)" [20]. That is not to say that one is better than the other. They are just different, meaning that they can adapt differently to different kinds of proposed problems.

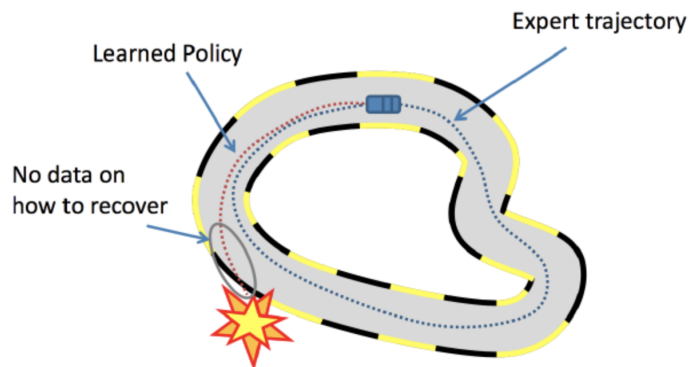


Figure 2.3: Non-Optimized Imitation Learning problem [15]

2.4 Inferring User Preferences

The success of the solution to this dissertation's problem is also dependent on how well current recommendation algorithms are understood. "Most streaming platforms have search services and some of them have some recommendation mechanism. The methods used are diverse, from those based simply on the popularity or genre of the songs, to those based on measures of similarity between songs or users. Some companies such as Spotify, Apple and Pandora have developed playlist generation algorithms based on content analysis performed by human experts or collaborative filtering. However, the details of these algorithms are unknown and there are not available data about their reliability" [25]. With this in mind, it will be virtually impossible to know with absolute certainty how Spotify's recommendation algorithms work. However, by having a broad understanding of the most common techniques of this field, an IRL implementation of this sort will be much more straightforward and will have higher chances of success.

2.4.1 Collaborative Filtering

Collaborative Filtering (CF) is one of the most popular ways of recommending content not only on streaming platforms but also on social networks and e-commerce websites. CF is based around rating systems. Its prime objective is to predict a target user's rating of an item considering other users with similar preferences regarding previously rated items [25].

Memory-based, also commonly referred to as user-based or user-user, CF predictions for a given user are based around a nearest neighbor approach. The user's nearest neighbors are other users that are expected to have similar preferences to the active user since they have rated items in common with a similar score [24]. Table 2.2 shows how a user's rating would be predicted according to other users. In this case, the score Carol would give to *Harry Potter* would be similar to Joe's, since they rated previous movies similarly.

However, this approach tends to create what is referred to in the literature as the **Sparsity Problem**. It arises when "the number of ratings needed for prediction is greater than the number

Table 2.2: CF user scores representation [27]

	Joe	Bob	Carol
Titanic	5	1	5
Schindler's List	1	5	2
Harry Potter	4	2	?

of the ratings obtained from the users" [24]. This happens because CF requires the user's explicit ratings for specific items in order to infer its preferences. This process can be very resource and time consuming. If the number of ratings needed for prediction is greater than the ratings obtained from the user the algorithm will fail.

On the other hand, Model-based, also commonly referred to as item-based or item-item, CF predictions are made by pre-computing the similarity between items. This is to avoid scalability problems usually associated with memory-based methods. It usually does not interfere with later ratings because new ones probably won't change the similarity between the items. However, they don't seem to be as good at recommending new content to users as Memory-based CF [4].

Currently, hybrid techniques are the most extensively implemented in content recommendation systems. These methods combine a plethora of CF methodologies or Collaborative Filtering with other recommendation techniques such as **Content-Based Schemes** [25].

2.4.2 Content-Based Filtering

Another way to infer user preferences on the web to later recommend relevant content to them is through **Content-Based Filtering** (CBF). Content-Based recommendation systems analyze item tags or descriptions to identify the ones that may be of particular interest to the end-user. In most web applications, items that can be shown or recommended to the user are often stored in a database of some sort. Table 2.3 shows an example of a database describing three restaurants. "This is an example of structured data in which there is a small number of attributes, each item is described by the same set of attributes, and there is a known set of values that the attributes may have" [4].

With this information, machine learning algorithms are then capable of extrapolate the preferences of a user's profile based around a combination of all sets of attributes he is most fond of. The preferences of the user's profile are mainly consisted of two different types of information [19]:

1. A model of the user's preferences, i.e., a description of the types of items that interest the user.
2. A history of the user's interactions with the recommendation system.

This information can not only be used to predict new content the user might interact with but also display old content it might want to revisit. These recommendation systems also provide the

Table 2.3: Restaurant representation in a database [19]

ID	Name	Cuisine	Service	Cost
10001	Mike's Pizza	Italian	Counter	Low
10002	Chris's Cafe	French	Table	Medium
10003	Jacques Bistro	French	Table	High

user independence through the use of its exclusive ratings and allow the suggestion of very recent items, not yet rated by anyone.

However, just like a Collaborative Filtering approach, Content-Based schemes have their downsides. One of them is the difficulty of evaluating whether or not a good recommendation was made. Since the user doesn't have to rate the items it has interacted with, there is no way to see if it matches its expectations and desires. It is also difficult to generate the attributes for items in certain areas and, due to the fact that Content-Based algorithms advocate the same type of items, it usually suffers from an overspecialization problem (the impossibility of assessing the relationship between different types of items) [27].

2.4.3 Demographic Filtering

Although not as popular as other recommendation systems, a **Demographic Filtering**, (DF), introduces a new, possibly essential, approach and solution to these types of problems. In DF, recommendations are established on the demographic profile of the user. Data such as age, gender, type of education and geographical location can be used to create a demographic profile. Different types of demographic profile might want completely different things. "For instance teenagers prefer different products than the elderly and the rich may want different products than middle and lower classes and are willing to pay more. Therefore demographic recommendation categorizes the user based on personal attributes and makes recommendations based on demographic classes" [26].

One of the advantages of this technique is that it is domain-independent, meaning that user-item ratings are not used and features about the items are not needed. Therefore, new users can get recommendations as soon as they arrive at the platform. However, it is still unrefined for recommendations with a high level of penalization. It is also important to mention that gathering user's information recklessly can lead to privacy issues as well [26].

2.4.4 Knowledge-Based Filtering

This type of content recommendation system uses knowledge about products and users to pursue a knowledge-based approach [6]. It suggests to users items of their interest based on the link of both items' characteristics and users' profile data.

"In order to properly work, this kind of recommender systems need a thorough modeling of items and users; the usual barrier to their development is, therefore, the availability of the necessary

knowledge and its maintenance over time" [26], meaning that its main drawback is the need for an engineered knowledge of constantly up-to-date database to make beneficial recommendations. On the other hand, this system doesn't depend on any database of user ratings, hence, "it does not have to gather information about a particular user because its judgements are independent of individual tastes" [6]. Moreover, they don't need a massive amount of data to compute recommendations, but a minimal amount of users in the system is required.

2.4.5 Hybrid Filtering

Keeping in mind that the previously discussed techniques all have their rewards and weaknesses, it's easy to argue that a **Hybrid Recommendation System** approach could be more effective in a variety of cases. The main motive of a hybrid approach is to aggregate Collaborative and Content-Based filtering to improve recommendations' accuracy. This can be implemented in various ways [27].

1. Implement collaborative and content-based methods individually and aggregate their predictions.
2. Integrate some content-based characteristics into a collaborative approach.
3. Comprise some collaborative characteristics into a content-based approach
4. Construct a general consolatory model that integrates content-based and collaborative characteristics.

Netflix is an excellent example of the Hybrid Filtering (HF) approach's success. They propose recommendations relying on CF, comparing the exploration habits of similar exploiters, and on CBF, by providing movies that share similar features of previously high-rated items by the exploiter. A data set containing approximately 100 million anonymous movie ratings showed how this combination of techniques can have a higher degree of success than independent implementation [5].

Depending on the way that this methodology is implemented, there are several ways that they can be classified. Burke separated them into the following seven classes [7]:

- **Weighted:** Different recommendation components scores are combined statistically. This class aggregates scores from each factor using the additive formula.
- **Switching:** From available recommendation components system chooses a particular component and applies the picked out one.
- **Mixed:** Different techniques provide their recommendation that will be introduced together. This class is based on merging and presenting multiple rated lists into single rated lists.
- **Feature Combination:** Contributing and actual recommending are two different recommendation components that exist for this class. The working of a real recommender depends on the data modified by the contributing one. The contributing one throws features of one source onto the other components' source.

- **Feature Augmentation:** This class is similar to the feature combination hybrids but the only difference is that the contributor gives novel characteristics. It is more elastic than the feature combination method.
- **Cascade:** This class plays the role of a tiebreaker. Here, for every recommender, assign some priority and then, according to that given priority, lower priority recommenders play the role of tiebreakers over higher priority.
- **Meta-level:** There exists contributing and actual recommenders, but the early one completely substitutes the data for the latter one.

2.5 Bridging the Gap

Now that the **Inverse Reinforcement Learning** and **User Preferences** literature has been revised, it is time to start thinking about how these two fields can be combined. The idea is to create a platform that can extract the users' preferences and recommend them new content accordingly using an IRL approach. In this case, the goal is to use a Spotify's account information to predict new artists, songs or records the user might like. This chapter focuses on indicating how content recommendation works in the music streaming industry and how IRL can be applied to a problem of this sort.

The HP laboratories at Cambridge conducted a research that focuses on the music recommendation issue. The paper approaches the problem from a song set perspective. It gathers an ensemble of songs the user likes to recommend a new one, with similar properties, it has never listened to. "The hope is that if several songs are chosen as representative of the 'sound' the user is seeking, we will have more information on which to base our automatic recommendation" [13]. They specify the term "song set" because, unlike a playlist, the order in which the songs appear is not relevant to the technique. The audio files for each track are then analysed to extract acoustic information and compare it with other files. Unfortunately, this cannot be done to examine songs from Spotify, since their files are not available to the public. However, we can take a similar approach using tags that have been attributed to each tune in its wide catalog. A distance measure can then be computed to compare the similarity between songs. After testing out different approaches in distance comparison between the song set and test songs, the results show that the best technique is to simply favor the track that has the minimum distance to any song in the set.

This means that the reward system for an IRL approach to the same problem could take in parameters such as a song's genre, duration and popularity and compare them to the tracks the expert is already into, thus, formulating the reward function accordingly.

This method is also used to infer user preferences in a social network using IRL in joint research conducted by the universities of Milan and Bologna. The chosen platform to understand why users perform specific actions like posting a message or clicking on an ad was **Twitter**. In this case, the expert is given a series of *tweets* (generated by other users) and has to decide whether to *retweet* it or not. The algorithm is dependent on three variables: the popularity of the tweet, the number of retweets recently performed by the agent, and the retweet time. The results of this

research indicate that different types of accounts tend to favor different post attributes and perform different actions. Standard Twitter users are more interested in retweeting posts with an average popularity score and following other users, whilst most commercial accounts (bots and companies) usually prefer retweeting posts with a high popularity score and do not worry on following other accounts [12].

Bridging this information to the case study of the dissertation, an IRL approach on a streaming platform like Spotify should be able to identify users that do not share the same fondness for different musical styles and, therefore recommend them new content in accordance with their personal tastes.

2.6 Summary

After studying the current state of the art in all the fields this proposal delves into, it is safe to assume that an IRL approach to the inference of user preferences is not only possible but has also been done. Despite being already tested on a social network, there isn't any evidence of it being applied to the music streaming business. The information gathered in this chapter is a good starting point for understanding which inputs from the Spotify's API will be useful for the IRL implementation and which of its variants should be used to solve the proposed problem.

At first glance, it can already be assumed that the **Markov Decision Processes** studied will be used to implement a **Large State Space** IRL algorithm that takes advantage of useful **inputs** from a **song set** the user already likes to extrapolate a **reward function**. The results attained from this reward function will then be used to recommend the user new content using a **Content-Based Filtering** or possibly a **Hybrid Filtering** approach.

Chapter 3

Problem and Proposed Solution

3.1 Problem

Despite its potential benefits, scaling IRL to real-world applications has historically demonstrated to be more challenging than RL. The most widespread applications are limited to the domains where the environment can be accessed or based on simulation, such as robotics. Hence, the main goal of this dissertation is to test its success in inferring user preferences. It is widely connected to the Information Technology (IT) research field because it will evaluate the accuracy of an IRL algorithm in something that has not yet been, or has barely been tested.

The first step is to trim down the problem at hand. The goal is to use IRL to infer user preferences, and there are many ways this could be approached. This field of artificial intelligence could be applied to various services like e-commerce websites or social networks, where the urge to solve this problem is everlasting. However, the chosen applications will be streaming services, which usually focus only on one content format like music or video. That will make it easier to apply this previously untested type of algorithm and make the job of recommending new content much more manageable. With that in mind, the kind of content this dissertation will focus on is music, more specifically, the Spotify music streaming service.

There are a lot of reasons for this decision. On the one hand, audio media is a much lighter file format than video, which will make the task of analyzing these files and their features, shall there ever need to be done so, much simpler. On the other, Spotify's API is a well-documented, ever-changing, and updating collection of endpoints, making the task of fetching information pretty straightforward. It is also one of the most subscribed music streaming platforms in the world and has a collection of over 70 million songs.

Short history of Spotify

According to Business of Apps [18], Spotify started as a Swedish small start-up, founded on 2006. The goal was to have an audio streaming and media services provider as a response to the growing

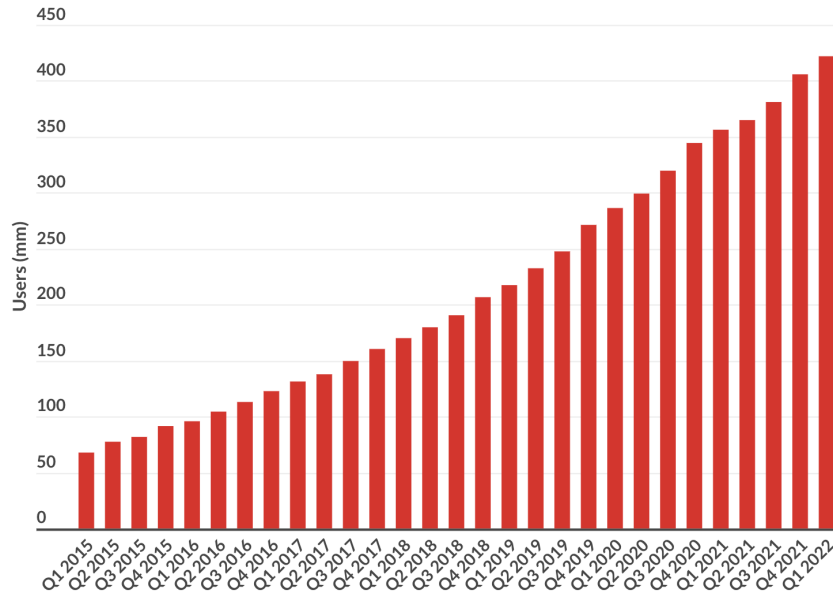


Figure 3.1: Spotify annual users 2015 to 2022 (mm) [18]

piracy problem the music industry was facing.

Spotify was launched two years later, in 2008, and currently is the world's most popular audio streaming subscription service with 422 millions of users across 183 markets.

In Spotify's apps, music can be browsed or searched for via various parameters, such as artist, album, genre, playlist, or record label. Users can create, edit and share playlists, share tracks on social media, and make playlists with other users.

Spotify had 422 million users in the first quarter of 2022. The evolution of the number of users from 2005 to 202 is represented in figure 3.1.

Spotify is heard all over the world, with the largest number of subscribers in Europe and North America. Table 3.1 shows the number of annual subscribers by region from 2018 to 2021.

Users of Spotify can register for free to have access to one of the largest collections of music in history. This wide range of users and content makes the job of testing an IRL approach and analysing its results much more accurate.

Now, the problem has been cut down to testing the efficiency of an Inverse Reinforcement Learning approach in inferring user preferences on Spotify.

3.2 Proposed Solution

The goal is to develop an app that should be able to predict what type of music the user is fondest of at the moment and the probability of liking a song or following an artist. This platform should gather information from Spotify and use an Inverse Reinforcement Learning algorithm to predict the user's preferences. The Spotify's API is very well documented and offers a plethora of possible

Table 3.1: Spotify annual subscribers by region 2018 to 2021 (mm) [18]

Year	Europe	North America	South America	Rest of the World
2018	33	26	17	7
2019	43	32	22	11
2020	54	40	29	15
2021	66	48	33	18

inputs for our algorithm to base its decisions. It could be possible to infer the user's preferences by analysing its history regarding the followed artists, followed playlists, liked songs, the playback state, recently played tracks and even whether or not he presses skip on a song. To implement this both a client side interface and a back-end server will be needed.

The client side will be handled by the **React** library. This allows us to easily create a user friendly environment that can communicate with the back-end entity where most information will be handled. The server side will use the **Flask** framework, since it can receive requests via Hypertext Transfer Protocol (HTTP) and handle them using **Python**, which is one of the best programming languages for dealing with AI.

Figure 3.2 presents the system architecture, showing how these technologies work and interact with each other. The overall procedure can be described by the following steps:

- [1] *User Logs In* to the platform and gives permission to access its profile information
- [2] The client side application *Requests User Info* from the Spotify API (top artists, top songs, ...).
- [3] The Spotify API *Returns User Info* so that it can be arranged by the client side application.
- [4] The client side application *Sends User Info* in the correct format to the server side application.
- [5] The server side *Retrieves User Preferences using IRL* to the client side application.
- [6] The client side application *Displays User Preferences* to the user.

The platform will be able to take requests from various users. They will only be allowed communication with the client-side application, which is in charge of receiving and displaying information in a user-friendly manner. This part of the application is also in charge of fetching data from the Spotify API after the user's authentication is approved. The client-side will choose which information should be extracted and arrange it accordingly to later send it to the server-side application. In this part of the platform, the information collected from the client-side application will be used to extrapolate the user's preferences using Inverse Reinforcement Learning. The reward function resulting from the IRL implementation will calculate the probability of a user liking a song, which in turn can be used to recommend new content to the user. This information is again sent to the client-side application in order to be shown to the user. This consists in

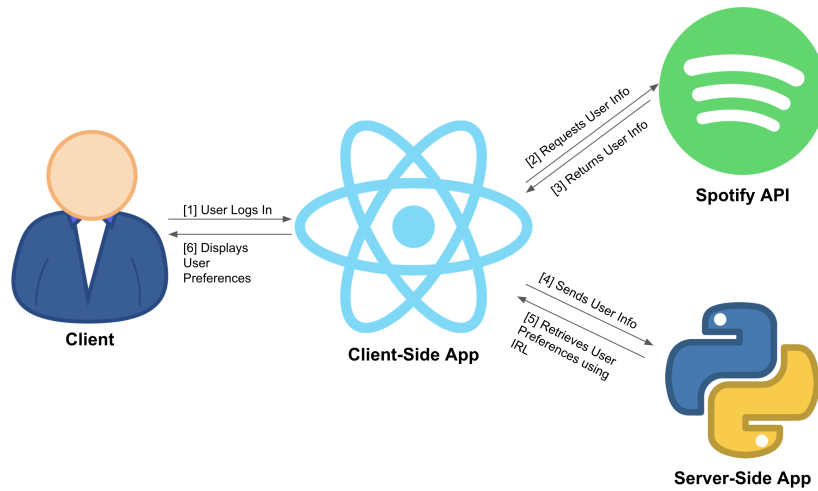


Figure 3.2: Project's architecture with its technologies and how they interact.

displaying a collection of recommended songs ordered by the user's enjoyment probability as well as giving him the option of creating a playlist in his own personal profile containing those recommendations. This means that the client-side application exchanges information with three different entities, while the server-side only communicates with one.

The next subsections show how the problem was tackled in both the client and server side applications, since the first one is concerned on what information should be extracted to make the algorithm work and the second one focuses on the implementation of the IRL algorithm.

3.2.1 Information Fetching

The real challenge of developing the **Client-Side Application**, besides developing a user friendly environment, is to extract relevant information from the Spotify's API. This information needs to be carefully chosen to distinguish what needs or does not need to be used and hence, make an IRL implementation not only possible but also accurate. With this in mind, this subchapter will focus on this information fetching process.

An Application Programming Interface (API) is a "connection between computer programs" [29], it is a type of software interface that offers a service to other pieces of software. In other words, it is a collection of endpoints that an application provides for outside users to fetch information. This eases the data fetching process because the Client-Side Application only needs to make HTTP requests to these endpoints to get what it needs. It is a simple and fast process. However, it is at the mercy of what the Spotify group gives access to. The application can only access data that has been meticulously chosen to be publicly available. This makes the decision of what information should be fetched even more difficult.

The Inverse Reinforcement Learning algorithm, described in the next subsection, will need data about: i) how the user proceeds in the environment, in this case, which songs the user likes, and ii) new untested environment states to evaluate what to do with them, in this case, new songs

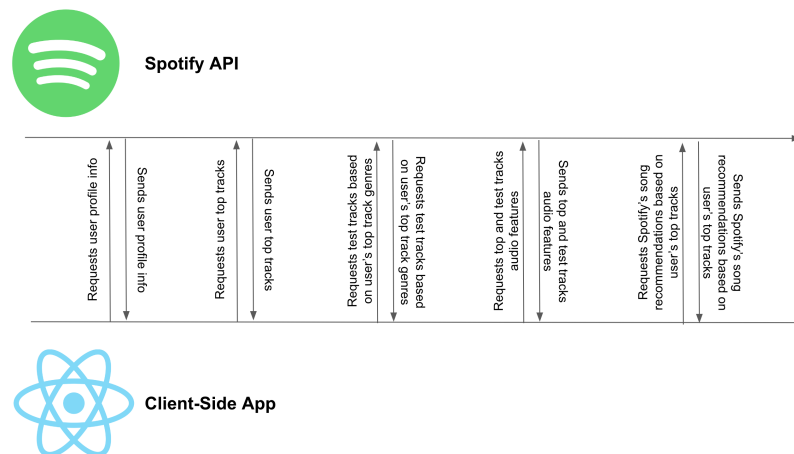


Figure 3.3: Diagram explaining the information fetching process.

to eventually recommend to the user. With this in mind, the approach chosen was similar to the one provided by the HP Laboratories [13]. So, it is necessary to: i) gather a set of tracks that Spotify knows the user is fond of, and ii) gather another set holding songs that will be evaluated after the user's intentions and preferences have been inferred. After completing the process of implementing the Spotify's API authentication system, it was possible to fetch this data from anyone that logs in to the Client-Side Application, as can be seen in Figure 3.3.

The following paragraphs discuss the information gathering process for obtaining data i) and ii) as well as other data relevant to the IRL algorithm described in section 3.2.2.

Information Fetching: getting user's top tracks

The process of getting the collection of liked tracks of the user is pretty straightforward. Spotify provides an endpoint that returns the logged user's top tracks. It is a set of songs that the user has repeatedly listened to on a specified time frame. Figure 3.4 shows that endpoint. This means that it gathers dynamic information about the user's taste and will let our algorithm change its outputs depending on what the user is most fond of at the time. In other words, since the data about the user's preferences is constantly changing depending on what he has been listening to at the time, the reward function will be different every time he logs into the application and in turn its taste predictions and further recommendations can follow its current preferences.

Information Fetching: getting test tracks

The process of getting the collection of test tracks was a little bit more complicated. There exists an endpoint that lets developers search for tracks based on a specified music genre, so that would be the best option for fetching our test data. It even gives the option of specifying tags that influence the returned results. For this, the *hipster* tag was used to retrieve songs with a low popularity and

```
const USER_TRACKS_ENDPOINT =  
  "https://api.spotify.com/v1/me/top/tracks?limit=50";
```

Figure 3.4: Endpoint returning the user's top tracks.

hence, improve the chances of the user not having previously heard those tracks. Figure 3.5 shows that endpoint.

The problem arises when deciding which music genres should be used in the search. At first it was planned to extract the genre of each track in the top tracks song set. Although this seems like a good idea, the Spotify's API doesn't return the tracks genre on a song's GET request. It needs to see what the track's artist is and then perform a GET request to that artist and retrieve its genre. This is completely possible and doable on the application but poses serious synchronization problems and risks to a smooth user experience and in some cases it exceeds the number of allowed requests done to the API (rate limit). The work around for this is to simply get the user's top artists and their respective genres. In this way, the application searches for genres that are dependent on the user's preferred artists instead of their preferred songs.

Information Fetching: getting tracks' features

Now, with both the top tracks and test song sets available, there needs to be a way of comparing them. There needs to be a way of extracting the tracks' information to be able to calculate the distance between each other. The HP Laboratories [13] use acoustic measures to achieve this. They take the raw audio of each song and have a method of extracting relevant song information such as the instrumentation or timbre. However, it is impossible to get the raw audio data for songs on a streaming platform. Fortunately, the Spotify's API has an audio features endpoint that retrieves values for different parameters of a track. These parameters include values for:

- **danceability**, describes how suitable a track is for dancing based on a combination of musical elements, including tempo, rhythm stability, beat strength, and overall regularity
- **energy**, represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy.
- **instrumentalness**, predicts whether a track contains no vocals. For instance, rap or spoken word tracks are clearly "vocal".
- **liveness**, detects the presence of an audience in the recording.
- **loudness**, the overall loudness of a track in decibels.
- **speechiness**, detects the presence of spoken words in a track.
- **tempo**, is the speed or pace of a given piece and derives directly from the average beat duration.
- **time signature**, is a notational convention to specify how many beats are in each bar (or measure).

```
const USER_SEARCH_ENDPOINT =  
  `https://api.spotify.com/v1/search?q=genre:${genre}+tag:hipster&type=track&limit=50`;
```

Figure 3.5: Endpoint searching for tracks the user might like.

- **valence**, describes the musical positiveness of the track. Tracks with high valence sound more positive (happy, euphoric), while songs with low valence sound more negative (sad, depressed)

The way the streaming platform assigns these values is unclear, however, besides not being as accurate as extracting information from the raw audio files, it should give a basic foundation for a good distance measure to be used in the Inverse Reinforcement Learning algorithm.

All this data is then sent to the **Server-Side Application** which will find a way of applying an IRL approach to eliminate tracks from the test song set that the user will not like and order the ones it does from the best song match to the worst.

Information Fetching: getting recommendations from Spotify

Last but certainly not least, there needs to be a way of evaluating the success of the IRL retrieved recommendations. Comparing them with the current Spotify recommendation algorithm could be a way of doing that. By putting these two approaches besides each other, test users of the application implemented in this dissertation could have a say in how the IRL performs in comparison to this large company's efforts in the matter and tell if this AI approach has anything to add to the current state of the art. The Spotify's API has an endpoint dedicated to its recommendation methodology. The request for recommendations, based on a collection input tracks, is met with a response with similar tracks to the ones provided. In this case, the request to the endpoint is made with the user's top tracks as input, hoping to get a series of song recommendations that match the user's taste. It can be viewed on Figure 3.6. This way, test users will be able to compare the two sets of recommendations and give important feedback on the behavior of the IRL implementation. These results will be further discussed on Chapter 4.

3.2.2 IRL Implementation

The **Server-Side Application** is in charge of receiving the data from the **Client-Side Application** and outputting a recommendation, that is, a preference ordered song set to later be displayed to the user using an IRL approach, in which this chapter is mainly focused on. It uses the **Flask** framework in order to take advantage of web development concepts and the simplicity and vast library list of the **Python** programming language.

The IRL concept, in this work, will be used to infer the users' preferences, related to songs in the Spotify platform. The main idea of the IRL methodology, as mentioned in chapter 2.3, is to derive a reward function that explains the observed behavior of the user. This reward function can then be used to infer the user preferences. The core of the IRL is then the right formulation

```
const SPOTIFY_RECOMMENDATIONS_ENDPOINT =  
  `https://api.spotify.com/v1/recommendations?seed_tracks=${track.id}`;
```

Figure 3.6: Endpoint getting user spotify recommendations.

of the reward function, that should be filled in and updated according to the intentions of the user's policy. After that, the set of test songs should be ordered according to their reward function score, which will result in the user's recommendations. Figure 3.7 makes it easier to visualize this process, being the steps illustrated there described in the remaining of this section.

IRL Implementation: instantiating the reward function

To do this, a first implementation of the reward function arises. The **Reward** system takes the form of a class that holds values for every instance of the audio features parameters. In this case, it has values for **danceability**, **energy**, **instrumentalness**, **liveness**, **speechiness** and **valence**. The **time signature**, **tempo** and **loudness** parameters were excluded from the reward system because they do not tell a lot about a track's sonic features and focus rather on technical aspects, that are independent of what a user might appreciate in a song. The code snippet below shows its initialization. The algorithm that updates the reward function will then analyse the parameters for every song in the top tracks song set and output the "perfect" value for every parameter in the reward function. We are referring to this as the "perfect" values because they are the parameter values that will be held in the reward system to later compare them to the test tracks. In theory, if a song has the same values as the ones on the reward system it has a 100% chance of liking the track.

```
1 class Reward:  
2     danceability = 0  
3     energy = 0  
4     instrumentalness = 0  
5     liveness = 0  
6     speechiness = 0  
7     valence = 0
```

Listing 3.1: Reward system's structure.

This implementation works just fine but can be improved. Since each music genre tends to share similar parameter values between songs, this can be used to modify the implementation of the reward function and improve its accuracy. In a first attempt, the reward system held a weighted average of the parameter values of all songs, which can be prone to error. The new reward system implementation holds different parameter values for each music genre that appears on the top and test songs sets.

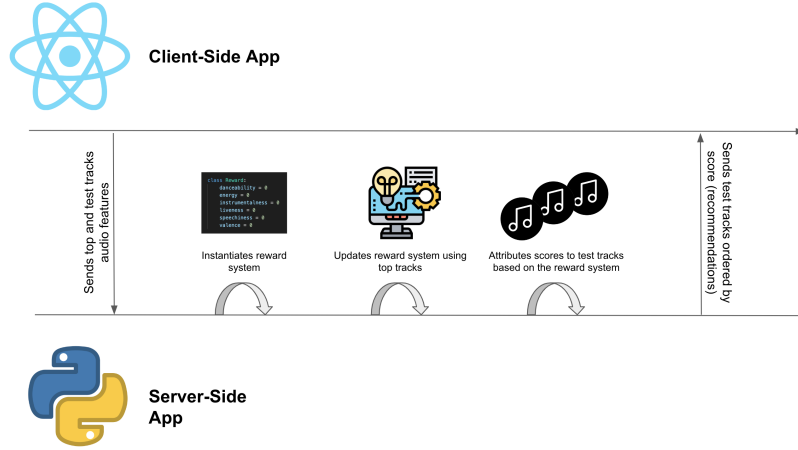


Figure 3.7: Diagram explaining the IRL implementation.

IRL Implementation: filling and updating the reward function

Linear programming must be used to update the reward function and create the perfect song parameter values. Since the value function implies that smaller distances mean that a song will be more successful, this becomes a minimization LP problem. It needs to find out which reward parameter values have a smaller distance between every song in the top tracks set. This means minimizing the sum of the differences between the reward parameter values and all the top songs' parameter values, as can be viewed on Equation 3.1. The equation takes every song in the top tracks set S , and minimizes the sum of all differences between the parameters P present in the reward system R and the parameters p of each song s . The result means that the reward system will have the perfect song parameters for every genre.

$$\min \left\{ \sum_{s=0}^S \left\{ \sum_{p=0}^P |R_p - s_p| \right\} \right\} \quad (3.1)$$

To make this not only possible but also straightforward, the Pulp library was used to achieve the solution for the Reward function. A code snippet from the implementation is included in Appendix A. Now that the reward function is complete, we can measure the distance between each track in the test-song set and the reward parameters.

IRL Implementation: getting the reward values for each test song

That distance is calculated by the sum of the absolute differences between each song parameter values and the reward system parameter values. Equation 3.2 shows how it works, taking $V(s)$ as the value function (distance) to given a song s , P as the set of parameter values p of the reward function and R as the reward function. This methodology approach means that the slimmer the distance a song has from the reward system, the best it is and the probability of the user liking

it will be higher. This makes it possible to formulate the user's **Policy** as well. In this case, the policy is simply the decision liking the song or not liking the song. With this in mind, the decision can be taken by taking the track from the top song set that has the highest distance to the reward system, meaning that every track from the test song set that has a difference greater than one track is not going to be liked by the user. Equation 3.3 shows how to get to the policy limit, taking π^* as the policy limit and $V(S)$ is the value function (distance) to the top tracks song set.

$$V(s) = \sum_{p=0}^P |R_p - s_p| \quad (3.2)$$

$$\pi^* = \max\{V(S)\} \quad (3.3)$$

This means that, contrary to most RL and IRL value functions, the higher the distance a song has to the "perfect" reward values, i.e., the higher the reward value a track gets from the value function, the most likely a user is going to dislike it. If a song has the same parameter values as the reward system, theoretically, the user has a 100% chance of liking it. Hence, the test tracks that will be sent to the Client-Side Application are ordered by their score, in descending order, providing they have a smaller score than the policy limit.

Summing up, and taking a look at the bigger picture, hopefully it is now clear to understand how this implementation relates to IRL. Each song that is analysed by the algorithm corresponds to a state s in the collection of states S that takes part of the environment. The reward system R is simply a collection of values that are present in every song divided by music genre, that can be discovered after minimizing the sum of the differences between the values of the reward function and the top tracks set. The possible actions A are simply liking or disliking a track. This decision is directly dependent on the policy π limit that is calculated after the reward system R has been discovered using Linear Programming. After the reward system is complete, it is also possible to attribute a score to each song on the top test tracks that is used to order them by preference of the user. Transition probabilities T were not taken into account because to do so there would be the need to access the users' listening history, which is not possible. These state transition probabilities would depend on what type of music the user is listening at a specific time frame and what his actions are beforehand and after.

3.3 Application

The resulting platform consists in the implementation of both the **Client-Side** and **Server-Side** Applications. In the case of it ever being deployed, users will be able to access it on the internet through a web browser. It is a simple Web Application consisting of three routes: the **Login** page, the **Feed** and the **Experiments** page.

The **Login** page simply holds a button that sends the user to the Spotify's API authentication system (see Figure 3.8). Shall this process be successful, an access token is created, to further make more complex requests, and the user is redirected to the **Feed**.

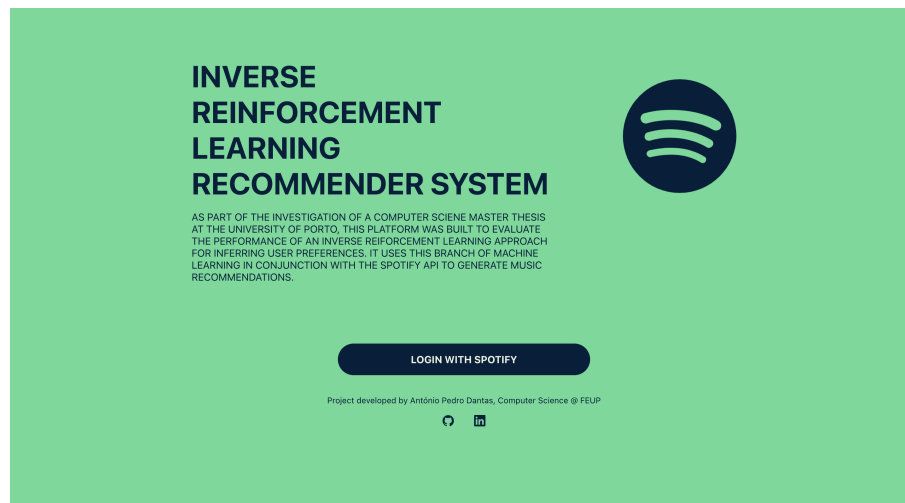


Figure 3.8: Application's Login page.

On this page, illustrated in Figure 3.9, the **Feed**, the application fetches all user information it needs while displaying it on the screen. It asks if it wants to retrieve its recommendations. If so, recommended songs are then displayed in a user friendly manner and the user is given the option of creating a Spotify playlist containing them. This is illustrated in Figure 3.10. There is also a component requesting help in the development of this platform. It serves as an entry to the **Experiments** page.

After clicking on that section, a playlist containing the top 25 IRL recommendations and another containing the top 25 Spotify generated song recommendations based on the user top tracks are created on the user's profile. It then redirects to the **Experiments** page, where information on how to proceed is given, as can be seen in Figure 3.11. The user should listen to both playlists and then answer a small survey that is linked on that web page as well. This will be very important to analyse the success of the IRL implementation. These results can be seen on Chapter 4.

Finally, a *logout* button that terminates the session and redirects to the **Login** page is displayed on both the **Feed** and **Experiments** page. Figures 3.8, 3.9, 3.10, 3.11 serve as a good reference on what is going on on the website and how the application works as a whole.

3.4 Summary

The resulting platform is able to fetch user data from the Spotify's API and use an Inverse Reinforcement Learning approach to infer its preferences and then recommend new content. This proves that an implementation of this AI branch is possible in solving content recommendation problems.

Since in this case there are no transition probabilities of any sort, the algorithm becomes much simpler than the approaches looked at in Chapter 2. However, the probability of the user transitioning into a specific track depending on the kind of music it is listening to at the moment

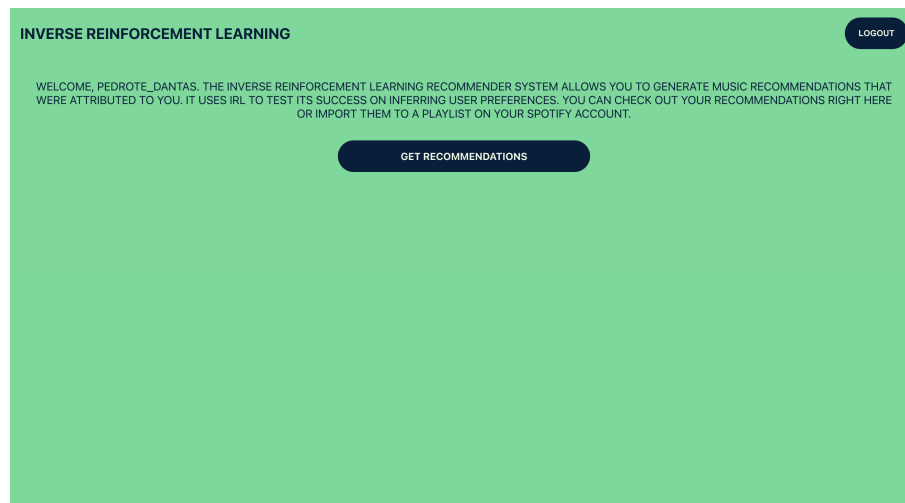


Figure 3.9: Application's feed before recommendation fetching.

could also be taken into consideration. To do that, there would be the necessity of accessing a user's listening history of some sort which, unfortunately, is not provided by the API in question. It would also have to be strictly tested in an environment where the user knew exactly what it wanted to hear at the moment to attain accurate results. A lot of users use the *Shuffle* listening option present in most audio players that plays the next song in the queue in no apparent order. If this would be the case it would mislead the developed algorithm.

Therefore, besides the implementation being a success, there are a lot of other details that could be added to this approach. This is one of the problems of not being able to access all the desired information, being at the mercy of what the API has to offer.

The next step is to test the current implementation and evaluate its results in a real-world scenario. Chapter 4 is mainly concerned with this problem.

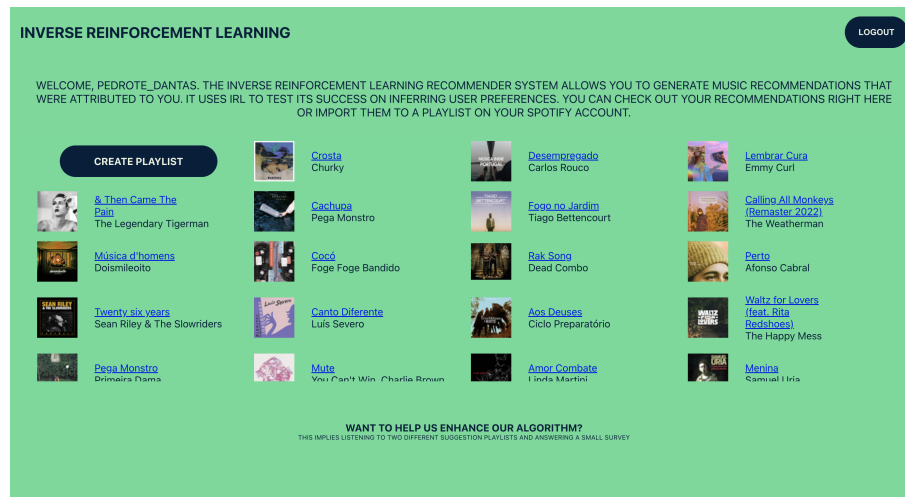


Figure 3.10: Application's feed after recommendation fetching.

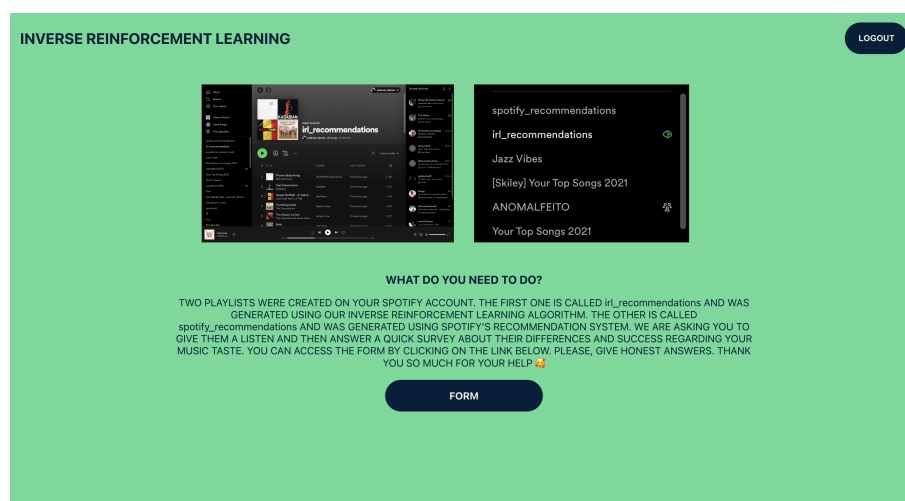


Figure 3.11: Application's Experiments page.

Chapter 4

Experiments and Results

After coming up with a solution to the problem discussed in Chapter 3 it is time to test its success and accuracy. This is crucial to understand whether or not the proposed methodology could be relevant to the current state of the art of content recommendation and deduction of user preferences. Shall these experiments be successfully applied and their results proven positive, this could mean that Inverse Reinforcement Learning methods should be able to improve the field. This chapter focuses on explaining how these experiments were conducted and on evaluating the attained results.

4.1 Experiment

To test the success of the proposed IRL approach to the problem of inferring user preferences and content recommendation as opposed to the current academic/industry standard on the field, the main idea is to compare the recommendation results attained from this AI field with the way Spotify currently recommends new content to users.

The developed application allows test users to create two playlists (both with 25 tracks) on their Spotify accounts. One of them, the playlist recommended by the proposed IRL approach, is called *irl_recommendations*, whilst the other, the proposed by Spotify, is called *spotify_recommendations*. Figure 4.1 and 4.2 show the created playlists on the user's Spotify account. Users are then asked to inspect and listen to both of them in order to be able to extract relevant information regarding the differences between an Inverse Reinforcement Learning recommendation algorithm and what the streaming platform currently offers.

After listening to both playlists, test users will have to answer some questions regarding how their recommendations behaved. The goal is to understand to which degree the IRL implementation differs from the already existing recommendation mechanism. It could generate a better playlist than the one provided by Spotify, both in terms of new content and its quality. However, it could fail in comparison to an already refined content recommendation algorithm developed by

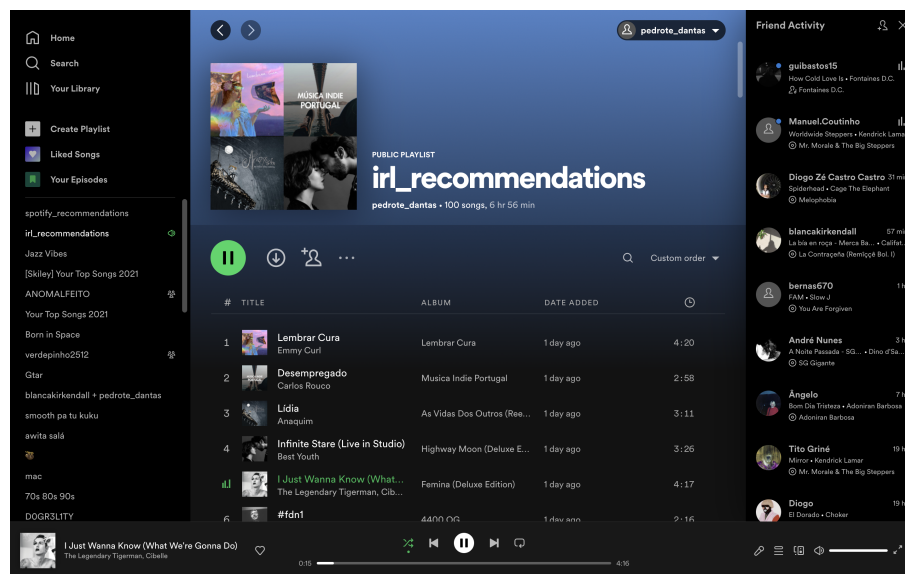


Figure 4.1: User's Spotify dashboard upon playlist creation.

a giant company. That's why developing an intricate and useful questionnaire will be of extreme importance to understand this phenomena. For this, **Google Forms** was used to allow users to easily answer the proposed questions and to gather information in an orderly fashion.

The form is divided into two sections, one for the *irl_recommendations* playlist and the other for the *spotify_recommendations* playlist. The questions in each section are the same to better verify the differences between the two. Table 4.1 holds the questions at hand.

The first and second questions aim to evaluate the percentage of songs that the user likes in each playlist. This is pretty straightforward because if a playlist has a higher number of enjoyed tracks, it points to it being a success and draws it closer to being a good recommendation algorithm. The fact that these are divided into the first 10 and then the last 15 tracks serve to evaluate whether or not the order in which the songs are recommended plays a role. Knowing this is especially useful for the Inverse Reinforcement algorithm since the songs are ordered by their score on the value function. So, in theory, the *irl_recommendations* playlist should be ordered from the track the user is most likely to enjoy to the one last likely.

The third question aims to pinpoint the percentage of recommended tracks the user has previously never heard of, while the fourth question does the same relating to previously unknown artists. This is important to understand the degree to which the playlist recommends new content. A good recommendation algorithm that attributes new tracks to a user that already knows the songs or artists featured in them is not very helpful.

To wrap these sections up, the fifth question asks the users to rate both these playlists from 1 to 10 in order to get a global sense of their satisfaction level. This will come in handy; shall the percentages of liked tracks in each playlist differ by a small margin. If the user rates both playlists similarly, it could mean that both approaches have similar results independently of that difference.

The form ends with a third section, where users are asked their age and to leave a small

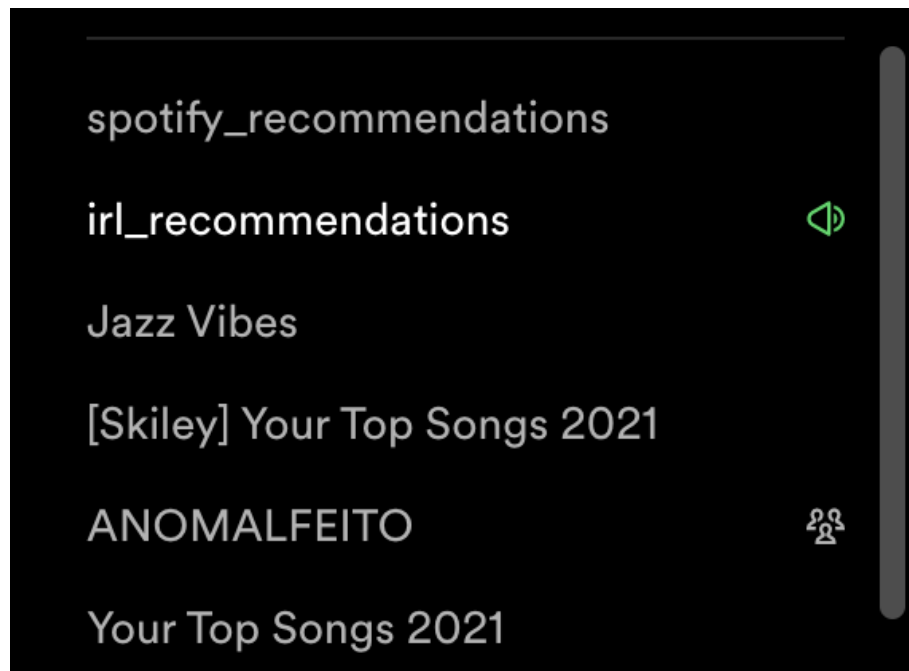


Figure 4.2: User's updated Spotify playlists.

comment if they found anything that they feel is worth mentioning for a better analysis of the gathered results. However, these questions are not mandatory. The form sent to test users is presented in Appendix B.

4.2 Results & Analysis

The results of the aforementioned inquiry are present in this chapter. At the end of this experimentation process, a total of 21 responses were collected from people ranging from the age of 15 to 57. The age distribution can be viewed in Figure 4.3. The responses will be used to infer the success of the IRL implementation against Spotify's already established algorithm.

4.2.1 Results

The first two questions of each section of the form (one related to *irl_recommendations* and the other to *spotify_recommendations*):

- *Take the first 10 tracks from the playlist. How many of them did you enjoy?*
- *Take the last 15 tracks from the playlist. How many of them did you enjoy?*

are useful to compare the raw results of how the users feel about each song in the playlists individually.

A percentage of the first 10, the last 15, and the whole 25 songs can be extrapolated to quickly understand how the two playlists behaved. These percentages can be obtained by calculating

Table 4.1: Form questions

Form Questions	
1	Take the first 10 tracks from the playlist. How many of them did you enjoy?
2	Take the last 15 tracks from the playlist. How many of them did you enjoy?
3	From the tracks you enjoyed, how many songs in the playlist have you NEVER heard?
4	From the tracks you enjoyed, how many artists featured in the playlist have you NEVER heard of?
5	Rate your level of satisfaction on this playlist from 1 to 10

each question's acquired results mean and can be viewed in table 4.2 along with the mode and the standard deviation. For the *irl_recommendations* playlist, users enjoyed 45,2% of the first 10 tracks, and 41,6% of the last 15, giving an all-around 43% of well-recommended tracks that make up the 25-song list. These results were attained by calculating the mean of the results and dividing that by 10 in the first question, calculating the mean of the results and dividing by 15 on the second question and adding up each individual result of each question, calculating the mean and dividing by 25 to get the total value of enjoyment. The most frequent answers (mode) were 6 and 10, respectively, and the standard deviation was 2,7 and 3,4. On the other hand, the *spotify_recommendations* playlist gathered a 57,6% of positive track results on the first 10 songs, 56,2% for the last 15, and an all-around 56,8% of good recommendations. The mode of these questions were 5 and 12, and the standard deviation for this playlist playlist was 2,0 and 3,3, respectively.

The third and fourth questions provide a sense of the degree to which each algorithm performs when recommending new content to the user.

- *From the tracks you enjoyed, how many songs in the playlist have you NEVER heard?*
- *From the tracks you enjoyed, how many artists featured in the playlist have you NEVER heard of?*

This is not a feature that was intentional or directly implemented, but these results can be interesting and important to evaluate its correct performance. Suppose the algorithm has a good percentage of new content recommendations. In that case, it will mean that it is not constructing its user preference score based on any kind of song or artist's popularity. It should infer the user's preferences based solely on each track's features and never on outside factors. These values are attained by comparing each individual answer and dividing the number of tracks and artists the user has never heard of by the total number of songs it enjoyed. Each of these percentages are then used to calculate the average results of new content recommendation. Figure 4.4 holds this information for the IRL playlist and Figure 4.5 does the same to the one generated from Spotify. As it can be inspected, the *irl_recommendations* playlist has an 63% chance of recommending new songs and an 42% chance of recommending new artists to the user, whilst the *spotify_recommendations* playlist has a 43% likelihood of recommending new songs and a 33% chance of recommending new artists. The mode values for the *irl_recommendations* playlist is 100% on both the third and fourth questions and the standard deviation is 0,37 and 0,36 respectively. On the other hand, the

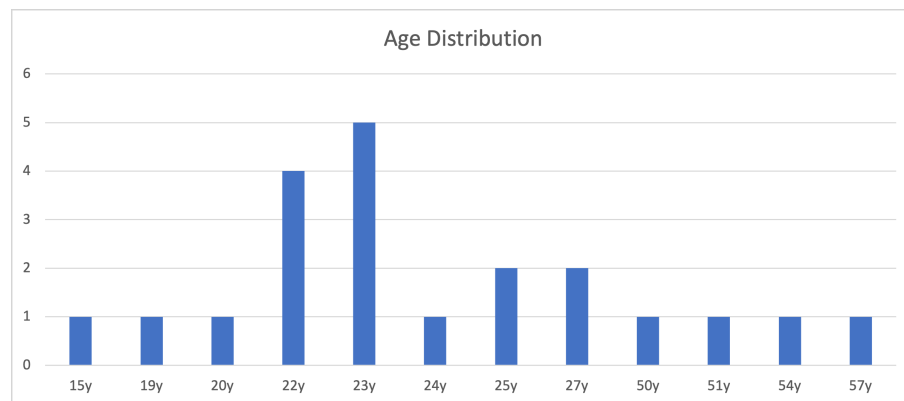


Figure 4.3: Age distribution.

mode and standard deviation for the *spotify_recommendations* playlist are, respectively, 50% and 0,44 for the third question and 0% and 0,39 for the fourth question.

The fifth question marks the end of each of these sections and evaluates the overall user enjoyment of the playlists they got.

- *Rate your level of satisfaction on this playlist from 1 to 10*

It serves not only as a confirmation of the results that were previously attained but also as a way of understanding a little bit better how the small nuances of the previous questions affect the success of each playlist. Table 4.3 holds this information. The IRL playlist got an overall score of 5,9 and the one from Spotify got a 7,4. The results were calculated using the mean of each question's answers. The mode and standard deviation for the *irl_recommendations* were 9 and 2,4 while the respective values for the playlist generated by Spotify were 8 and 1,8.

Test users also got asked if they wanted to leave a comment on shall they had found anything interesting about their playlists and wanted to share it. These answers can be used to explain phenomena that have happened in previous questions and confirm the acquired results. Here are some of them.

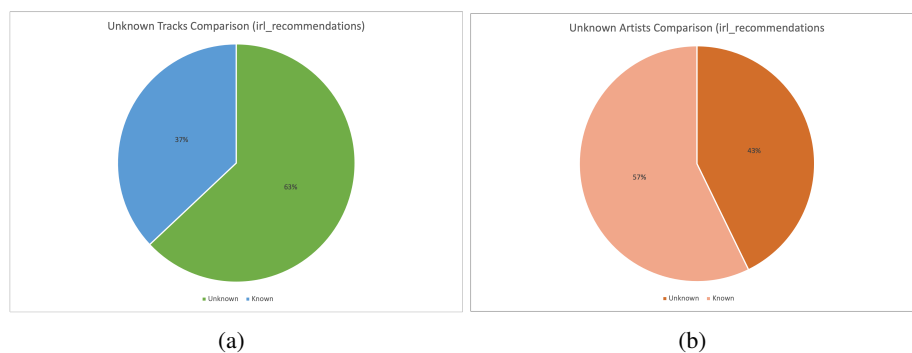


Figure 4.4: (a) Previously Unknown Songs (b) Previously Unknown Artists

Table 4.2: First and Second Questions Results (Number of liked songs)

Playlist	First 10 Questions			Last 15 Questions			Total
	Likes	Mode	Std. Deviation	Likes	Mode	Std. Deviation	Likes
irl_recommendations	4,5	6	2,7	6,2	10	3,4	10,7
spotify_recommendations	5,8	5	2,0	8,4	12	3,3	14,2

1. I enjoyed the playlist and the suggestions given made sense. Despite the songs that I didn't like, I understand some suggestions given, make sense according to some artists I've heard.
2. In general I enjoyed it, and would used it for sure.
3. I really liked both playlists, however irl_recommendations was the one that had more different songs from the ones I know and showed me more variety. In the spotify_recommendations playlist most of the songs I already knew however I still liked that one.
4. The spotify playlist had more "popular artists" than in IRL playlist. But IRL was more original/likeable at least for me.
5. While the spotify_recommendations playlist seemed to pick up multiple genres and "moods" from what I have been listening to recently, the irl_recommendations playlist pickup one broad genre and got fixated on only recommending songs from that genre.
6. It was really helpful in introducing me to new musicians. When compared to spotify recommendations, which presents songs that I most relate with, most artists and many of the songs were already familiar to me. Irl recommendation wins in this category because it introduces new discoveries.
7. Some songs in the irl playlist where super obscure and lo-fi, also 4 songs in a row came from a compilation of love songs from the 80s. Spotify probably recommends songs also based on their success or that more people listen to.
8. The irl playlist was much superior in terms of recommending new artists similar to my taste, however there were some weird picks for recommendations that didn't really fit my taste or

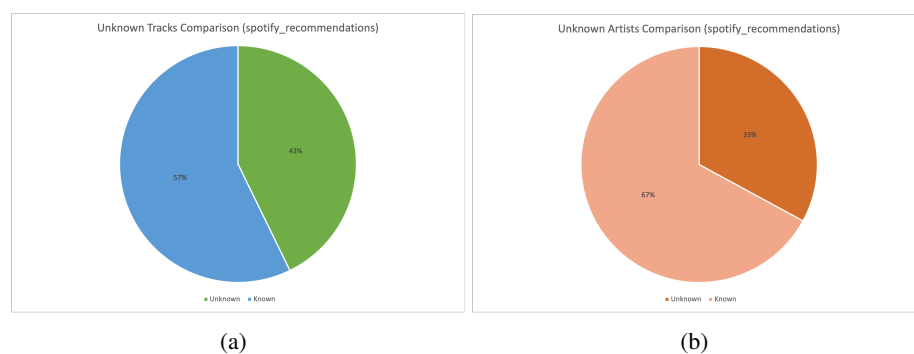


Figure 4.5: (a) Previously Unknown Songs (b) Previously Unknown Artists

Table 4.3: Fifth Question Results

Playlist	Overall Score [1-10]	Mode	Std. Deviation
irl_recommendations	5,9	9	2,4
spotify_recommendations	7,4	8	1,8

what i listen to. Specially the last 4 songs in the playlist. All the other recommendations were mostly around the same few genres and what I've been listening to more, lately.

4.2.2 Analysis

Analyzing the results attained is crucial because it enables us to understand what these values mean in the project's scope. Drawing these conclusions is essential to understand what the IRL implementation can or cannot do and deduce new ways of improving it.

Let's start by addressing the results acquired in Question 5. This is the one regarding the overall score of each playlist. It is a good starting point to dive into before drawing other conclusions because it allows us to look at the bigger picture of the global user enjoyment score of these playlists. As we can see, the *irl_recommendations* playlist got an overall positive score since it got almost 2 points above half of the score limit range. It is safe to say that in a system ranking the playlists from 1 to 10, bringing a 5,9 average score is not a negative result. However, it loses by quite a significant margin when comparing it to the average score of the *spotify_recommendations* playlist. The 7,4 average score of the song recommendation system of this streaming platform is undoubtedly superior. The results acquired from the first two questions also seem to support these findings since the average test user enjoyed 43% of tracks from the playlist using an IRL implementation against an almost 57% of the one generated by Spotify. The mode and standard deviation for Questions 1 and 2 also seem to reinforce these findings. The only result that contradicts this is the mode values for Question 5, where the IRL playlist obtains a score of 9, and the one from Spotify gets an 8. Nonetheless, these results should be taken with a grain of salt because the standard deviation for the *irl_recommendations* playlist is much higher, meaning that many people also gave it a negative score.

Despite that, the results from Questions 3 and 4 allow us to conclude that this IRL implementation brings something new to the table. These answers tend to indicate that the *irl_recommendations* playlist did something that the *spotify_recommendations* playlist couldn't: recommend new content. The percentage of both previously unheard of tracks and artists is higher in the playlist generated by the IRL implementation than on the one from Spotify. Not only higher, but higher but quite a large margin, having a 20% difference when it comes to recommending new songs and a 9% difference when it comes to recommending new artists. This becomes more apparent after looking at the most common answer to these questions. The IRL playlist has a mode value of 100% for both the third and fourth questions, meaning that all of the songs and artists they enjoyed were previously unknown to them. This is much higher than the mode value of these questions regarding the Spotify playlist while maintaining a lower standard deviation for both. The answers

to the final question on the form provided to the test users also seem to support this finding. There, people were asked to leave comments that they found would help analyze the questionnaire results. They weren't required to do so, but still, some of them have. From a total of 12 small comments left on this question, half of them mentioned the impact they felt this playlist had when recommending new content compared to *spotify_recommendations*.

Now, can a reason be found to explain these phenomena? Why does the playlist generated by Spotify get a higher overall score, but fails in comparison to an IRL implementation when it comes to recommending new content? This was not a feature that was thought of when coding up the platform, and surely, the IRL algorithm did not consider this. It must be something to do with how Spotify organizes and attributes scores to its recommendations. Although how the streaming platform's algorithm works is unknown, it is safe to assume that it must gather a much higher and diverse set of track and artist features and information that come into play when considering which content it will recommend to the user. Data regarding the users' nationalities, their listening history, saved playlists and albums, and content release dates are only a few parameters that could be part of their refined recommendation algorithm. They have access to all of this information, after all. On the other hand, the IRL implementation only focuses on the users' top tracks audio features and genres, which can result in a disregard for content popularity or any other outside factor. Focusing on content's sonic characteristics and ignoring outside factors explains the lower grading of the overall enjoyment of the Inverse Reinforcement Learning playlist while maintaining a higher rate of new content recommendation. Hence, the users' preferences are only inferred in relation to the content's features.

Chapter 5

Conclusions and Future Work

This project kicks off with a pretty strong premise in terms of problem analysis and solution examination. The goal is to evaluate the success and accuracy of an Inverse Reinforcement Learning implementation in a real-world scenario with already some found solutions. Therefore, this dissertation is all about research and discovery of something that can push the state of the art forward in the areas of user preference prediction and content recommendation.

For this, a lot of research needed to be done in terms of IRL problem formulation and its direct counterpart, RL. However, everything seems to have ran smoothly and there aren't any big problems encountered on the development of this project. This chapter focuses on understanding whether or not the expectations for this project were successfully achieved. The possibility of further investigation and improvement on this project will also be tackled here.

5.1 Conclusions

Taking a look back at what was first proposed in this dissertation allows us to understand a little bit better how far this project has gone a bit more in-depth.

The idea was to test the reliability and accuracy of an Inverse Reinforcement Learning approach to the problem of inferring user preferences and, furthermore, bring advancements to the current literature on this topic. It sounds like a simple premise but the lack of information on the field and the highly theoretical knowledge that is still being developed for this type of A.I. made it almost impossible to tackle this problem in a straightforward manner. This is due to the fact of this Machine Learning branch being a relatively new installment on the computer science world and it still being mainly used for solving robotics problems.

Despite that, this dissertation proves that an application of this sort is not only possible but also adds new nuances when it comes to inferring user preferences and recommending new content. In this particular case, a reward system and value function were created based on the listening habits of users of a music streaming platform. This implementation focuses only on the parameters that

specify a track's audio features and its genres, which means that the IRL algorithm can order songs according to the users' preferences and, therefore, recommend content that he is likely going to enjoy. This has the potential to be extended to another types of streaming platforms or even completely different applications that are dependent on human behavior analysis.

This is not to say that this type of implementation will completely change and revolutionize the current State of the Art. It failed in comparison to the already established Spotify's recommendation system when it got tested in the experiments phase. This approach is still very deficient since it does not consider a lot of specifications that set apart an IRL implementation. For once, it does not have state change probabilities, since the streaming platform's API does not give access to users' listening history. The set of data used to infer the users' preferences in the first place is also very small due to rate limitations of the application. Also, the design of the different parameters that make up the reward system is very incomplete. It uses a small set of song's sonic features to develop parameter values that make up the the type of content the user will most likely enjoy. Increasing the size of that set and adding different weight to the parameters could perfect the algorithm by a large margin. This, together with the ability to get more information regarding users' behavior on the platform could contribute to a much more accurate and trustworthy implementation.

One thing is for certain: even with all these limitations and an adapted approach to a theoretically pure IRL implementation, the resulting algorithm brought some promising results that did not need abundant information to quickly understand what the user is into. The future work that could be done in this field, together with a combination of other recommendation algorithms and approaches (Hybrid Filtering comes to mind) could truly change the way experts in the field tackle the problem of predicting users' behavior and inferring their preferences.

5.2 Further Work

Future work on this dissertation and its project are essential to improve the success of the IRL implementation and, consequently, its evaluation.

The first thing that could be done to complete this project would be to make the resulting application open to the public. To do that, many requirements, such as the terms of service, design guidelines, and privacy policies, need to be addressed. This is very important because for the app to go live, it requires Spotify's approval, which can take up to six weeks. This was considered during the development process, but due to time limitations was not implemented. Shall this issue be addressed, it would result in a much bigger test sample for the experiments conducted in Section 4.1, which would turn the results attained much more trustworthy.

Shall Spotify approve this platform and it becoming public in the future, a lot of improvements could be made in the Information Fetching process present in Section 3.2.1. One of the implementation problems faced in this process was the rate limits that Spotify's API had. The rate limit is calculated based on the number of calls the app makes to Spotify in a rolling 30-second window. If the app exceeds the rate limit, we'll see 429 error responses from the API. The limit

varies depending on whether the app is in development or extended quota mode. If the platform had this quota extension (became public), it could increase the top and test tracks collection, and, in turn, make the IRL algorithm have a bigger learning set and improve the accuracy of the reward system.

Last but not least, there are also some issues regarding the IRL implementation that could be improved in the future. The possibility of adding weights to the song parameters present in the reward system would be one of them. Currently, all parameters weigh the same, meaning they all have the same importance. Maybe adding different weights to each one of these parameters could result in a better extrapolation of the user's preferences. In turn, the Server-Side Application could return a better set of recommendations. This, however, could only be done by testing the different weight applications on test users and waiting for their feedback to know which combination would be best. This process would be long and tiresome, resulting in delayed dissertation delivery. Another way of improving the Inverse Reinforcement Learning implementation would be to add state transition probabilities to the agent's behavior. The implementation has been adapted not to include these because the Spotify's API does not give access to a user's listening history. If we had access to this information, the probability of it changing states (tracks) based on the content he is listening to at the moment could also be inferred and in turn make the algorithm much more complete.

References

- [1] Gediminas Adomavicius and Alexander Tuzhilin. Recommendation Technologies: Survey of Current Methods and Possible Extensions. *NYU Working Paper No. 2451/14115*, 2004.
- [2] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *THINC Lab and Institute for AI, Dept. of Computer Science, University of Georgia*, 2021.
- [3] Mohammad Ashraf. Reinforcement learning demystified: A gentle introduction. Available at <https://m-elsersy96.medium.com/reinforcement-learning-demystified-36c39c11ec14>, April 2018.
- [4] Sarwar Badrul, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web*, pp. 285-295., 2001.
- [5] James Bennett and Stan Lanning. The netflix prize. *Proceedings of KDD cup and workshop*, vol. 2007, p. 35, 2007.
- [6] Robin Burke. Knowledge-based recommender systems. *Encyclopedia of library and information systems* 69, no. Supplement 32: 175-186, 2000.
- [7] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction* 12, no. 4: 331-370, 2002.
- [8] Leonor Baptista da Costa Silva Santos. Learning from Demonstration using Hierarchical Inverse Reinforcement Learning. Master’s thesis, Faculdade de Engenharia da Universidade do Porto, 2021.
- [9] Alexandre Gonfalonieri. Inverse reinforcement learning: Introduction and main issues. Available at <https://towardsdatascience.com/inverse-reinforcement-learning-6453b7cdc90d>, December 2018.
- [10] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)* 50, no. 2: 1-35, 2017.
- [11] Yuxi Li. Reinforcement Learning Applications. *arXiv preprint*, 2019.
- [12] Amarildo Likmeta, Alberto Metelli, Giorgia Ramponi, Andrea Tirinzoni, Matteo Giuliani, and Marcello Restelli. Dealing with multiple experts and non-stationarity in inverse reinforcement learning: an application to real-life problems. *Mach Learn* 110, 2541–2576, 2021.
- [13] Beth Logan. Music recommendation from song sets. *ISMIR*, pp. 425-428, 2004.

- [14] Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. *Machine Learning and Knowledge Discovery in Databases*, pp. 31–46, 2009, 2009.
- [15] Medium. A brief overview of imitation learning. Available at <https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c>, September 2019.
- [16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning. *DeepMind Technologies*, 2013.
- [17] Andrew Ng and Stuart Russel. Algorithms for Inverse Reinforcement Learning. *Icml. Vol. 1*, 2000.
- [18] Business of Apps. Spotify revenue and usage statistics. Available at <https://www.businessofapps.com/data/spotify-statistics/>, June 2022.
- [19] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. *The adaptive web*, pp. 325–341. Springer, Berlin, Heidelberg, 2007.
- [20] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Bridging the gap between imitation learning and inverse reinforcement learning. *IEEE transactions on neural networks and learning systems* 28, no. 8: 1814–1826, 2016.
- [21] Deepak Ramachandran and Eyal Amir. Bayesian inverse reinforcement learning. *JCAI (Vol. 7, pp. 2586–2591)*, 2007.
- [22] Stuart Russel. Learning agents for uncertain environments. *Proceedings of the eleventh annual conference on Computational learning theory*, 1998.
- [23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 1998.
- [24] Diego Sánchez-Moreno, Ana Gil González, Dolores Vicente, Vivian Batista, and María Moreno-García. Recommendation of songs in music streaming services. *International Conference on Practical Applications of Agents and Multi-Agent Systems*, pp. 206–213. Springer, Cham, 2017.
- [25] Diego Sánchez-Moreno, María Moreno-García, Nasim Sonboli, Bamshad Mobasher, and Robin Burke. Inferring user expertise from social tagging in music recommender systems for streaming services. *International Conference on Hybrid Artificial Intelligence Systems* pp. 39–49. Springer, Cham, 2018.
- [26] Ruchita V. Tatiya and Archana S. Vaidya. A survey of recommendation algorithms. *IOSR J. Comput. Eng* 16, no. 6: 16–19, 2014.
- [27] Poonam B. Thorat, R. M. Goudar, and Sunita Barve. Survey on collaborative filtering, content-based filtering and hybrid recommendation system. *International Journal of Computer Applications* 110, no. 4: 31–36, 2015.
- [28] Martijn van Otterlo and Marco Wiering. *Reinforcement Learning and Markov Decision Processes*. Springer, Berlin, Heidelberg, 2012.

- [29] Wikipedia. Api. Available at <https://en.wikipedia.org/wiki/API>, May 2022.
- [30] Seok Yu. The dynamic competitive recommendation algorithm in social network services. *Dept. of Computer Science, Sookmyung Women's University, Seoul*, 2011.
- [31] Shao Zhifei and Er Joo. A Review of Inverse Reinforcement Learning Theory and Recent Advances. *WCCI 2012 IEEE World Congress on Computational Intelligence*, 2012.

Appendix A

IRL Code Snippet

This code fragment shows how the reward solution values are attained. By splitting the reward system into genres, it gets all parameter values from the top tracks and uses linear programming to find out which reward values minimize the sum of the differences between the two.

```
1 from pulp import *
2
3 # method creates values for reward function on every parameter
4 def update_reward_function(reward_function, tracks_data):
5
6     # ===== using linear programming to minimize the
7     # difference between the perfect parameter values with track genres
8     # =====
9
10    # creates all genres for the reward function
11    for track in tracks_data:
12        if track["genre"] not in reward_function:
13            reward_function[track["genre"]] = Reward()
14
15    # for every genre in the reward function iterates through top tracks and finds
16    # out the best values for them
17    for genre in reward_function:
18        # array that holds all the tracks belonging to a genre
19        # it uses this array to create the parameter values for that genre
20        genre_tracks = []
21        # flag that tells if a genre is not present in the top tracks
22        genre_undefined = 0
23        for track in tracks_data:
24            if track["genre"] == genre:
25                # if it finds a song belonging to that genre adds it to the array
26                # updates flag and says that it has at least one genre
27                genre_tracks.append(track)
28                genre_undefined = 1
29
30        # if it did not find any song for that genre, adds all the songs to the "
31        # undefined" genre
```

```

27     # this will create parameter values undifined genre that are common to
        every song
28     if genre_undefined == 0:
29         genre_tracks = tracks_data
30
31
32     # array that holds all genre_tracks parameter values
33     danceability_array = []
34     energy_array = []
35     instrumentalness_array = []
36     liveness_array = []
37     speechiness_array = []
38     valence_array = []
39
40     for track in genre_tracks:
41         danceability_array.append(track["danceability"])
42         energy_array.append(track["energy"])
43         instrumentalness_array.append(track["instrumentalness"])
44         liveness_array.append(track["liveness"])
45         speechiness_array.append(track["speechiness"])
46         valence_array.append(track["valence"])
47
48     # elementary features
49     lp = LpProblem("Reward_Problem", LpMinimize)
50
51     # defined variables
52     danceability = LpVariable(name="danceability", lowBound = 0, cat="Float")
53     energy = LpVariable(name="energy", lowBound = 0, cat="Float")
54     instrumentalness = LpVariable(name="instrumentalness", lowBound = 0, cat="
        Float")
55     liveness = LpVariable(name="liveness", lowBound = 0, cat="Float")
56     speechiness = LpVariable(name="speechiness", lowBound = 0, cat="Float")
57     valence = LpVariable(name="valence", lowBound = 0, cat="Float")
58
59     t_danceability = LpVariable(name="abs_helper_d", lowBound = 0, cat="Integer
        ")
60     t_energy = LpVariable(name="abs_helper_e", lowBound = 0, cat="Integer")
61     t_instrumentalness = LpVariable(name="abs_helper_i", lowBound = 0, cat="
        Integer")
62     t_liveness = LpVariable(name="abs_helper_l", lowBound = 0, cat="Integer")
63     t_speechiness = LpVariable(name="abs_helper_s", lowBound = 0, cat="Integer"
        )
64     t_valence = LpVariable(name="abs_helper_v", lowBound = 0, cat="Integer")
65
66     # objective function
67     lp += t_danceability + t_energy + t_instrumentalness + t_liveness +
        t_speechiness + t_valence
68
69     # constraints

```

```

70     lp += t_danceability >= get_sum_difference_value(danceability,
71                                                       danceability_array)
72
73     lp += t_energy >= get_sum_difference_value(energy, energy_array)
74     lp += -t_energy <= get_sum_difference_value(energy, energy_array)
75
76     lp += t_instrumentalness >= get_sum_difference_value(instrumentalness,
77                                                           instrumentalness_array)
78
79     lp += t_liveness >= get_sum_difference_value(liveness, liveness_array)
80     lp += -t_liveness <= get_sum_difference_value(liveness, liveness_array)
81
82     lp += t_speechiness >= get_sum_difference_value(speechiness,
83                                                       speechiness_array)
84
85     lp += -t_speechiness <= get_sum_difference_value(speechiness,
86                                                       speechiness_array)
87
88     # solving the lp
89     status = lp.solve()
90     print("Status: ", status)
91
92     for var in lp.variables():
93         print(var, " = ", value(var))
94     print("OPT = ", value(lp.objective))
95
96     # gives lp solution variables to the reward function
97     reward_function[genre].danceability = value(danceability)
98     reward_function[genre].energy = value(energy)
99     reward_function[genre].instrumentalness = value(instrumentalness)
100    reward_function[genre].liveness = value(liveness)
101    reward_function[genre].speechiness = value(speechiness)
102    reward_function[genre].valence = value(valence)

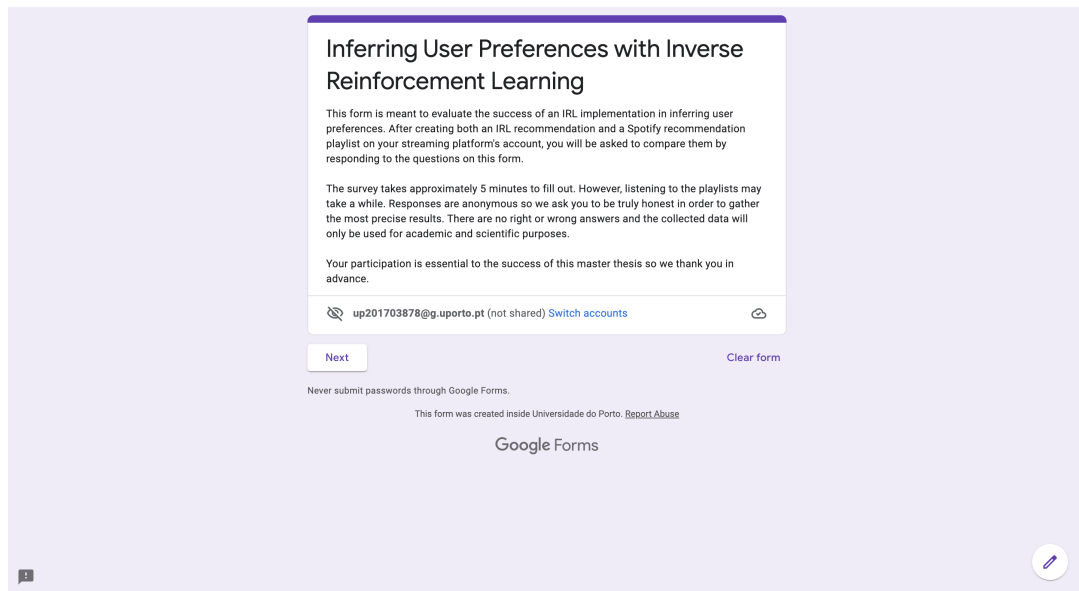
```

Listing A.1: Method that solves the Reward function by attributing the best parameter values to the Reward system.

Appendix B

Form

This chapter shows the Google Form file that was used to test the IRL implementation. It is composed to two sections with similar questions aimed at comparing the `irl_recommendations` playlist with the `spotify_recommendations` playlist and an extra section for final comments.



Inferring User Preferences with Inverse Reinforcement Learning

This form is meant to evaluate the success of an IRL implementation in inferring user preferences. After creating both an IRL recommendation and a Spotify recommendation playlist on your streaming platform's account, you will be asked to compare them by responding to the questions on this form.

The survey takes approximately 5 minutes to fill out. However, listening to the playlists may take a while. Responses are anonymous so we ask you to be truly honest in order to gather the most precise results. There are no right or wrong answers and the collected data will only be used for academic and scientific purposes.

Your participation is essential to the success of this master thesis so we thank you in advance.

 up201703878@g.uporto.pt (not shared) [Switch accounts](#) 

[Next](#) [Clear form](#)

Never submit passwords through Google Forms.

This form was created inside Universidade do Porto. [Report Abuse](#)

Google Forms

Figure B.1: Form's first page.

Inferring User Preferences with Inverse Reinforcement Learning

 up201703878@g.uporto.pt (not shared) [Switch accounts](#) 

*Required

Inverse Reinforcement Learning Recommendations

To answer this part of the form, it is advised that you first listen to the irl_recommendations playlist that was created on your Spotify account.

Take the first 10 tracks from the irl_recommendations playlist. How many of them did you enjoy? *

1

Take the last 15 tracks from the irl_recommendations playlist. How many of them did you enjoy? *

1

From the tracks you enjoyed, how many songs in the irl_recommendations playlist have you NEVER heard? *

1

From the tracks you enjoyed, how many artists featured in the irl_recommendations playlist have you NEVER heard of? *

1

Rate your level of satisfaction on this playlist *

0 1 2 3 4 5 6 7 8 9 10

Not good ☐ ☐ ☐ ☐ ☒ ☐ ☐ ☐ ☐ ☐ ☐ Excellent

[Back](#) [Next](#) [Clear form](#)

Never submit passwords through Google Forms.

This form was created inside Universidade do Porto. [Report Abuse](#)

Google Forms

Figure B.2: Form's second page.

Inferring User Preferences with Inverse Reinforcement Learning

 up201703878@g.uporto.pt (not shared) [Switch accounts](#) 

*Required

Spotify's Recommendations

To answer this part of the form, it is advised that you first listen to the spotify_recommendations playlist that was created on your Spotify account.

Take the first 10 tracks from the spotify_recommendations playlist. How many of them did you enjoy? *

1

Take the last 15 tracks from the spotify_recommendations playlist. How many of them did you enjoy? *

2

From the tracks you enjoyed, how many songs in the spotify_recommendations playlist have you NEVER heard? *

1

From the tracks you enjoyed, how many artists featured in the spotify_recommendations playlist have you NEVER heard of? *

1

Rate your level of satisfaction on this playlist *

0 1 2 3 4 5 6 7 8 9 10

Not good ☐ ☐ ☐ ☐ ☐ ☒ ☐ ☐ ☐ ☐ ☐ Excellent

[Back](#) [Next](#) [Clear form](#)

Never submit passwords through Google Forms.

This form was created inside Universidade do Porto. [Report Abuse](#)

Google Forms

Figure B.3: Form's third page.

The screenshot shows a Google Form titled "Inferring User Preferences with Inverse Reinforcement Learning". The form is created by the user "up201703878@g.uporto.pt (not shared)" and includes a "Switch accounts" link. Below the title bar, there is a "Final comments" section with the text: "If you feel like being extra helpful, you can answer these questions, besides them not being mandatory." The form contains two text input fields. The first field is labeled "Please tell us your age." and has a placeholder "Your answer". The second field is labeled "Is there anything you want to comment on? Did you enjoy the irl_recommendations playlist? Did you hate it? Is there anything you noticed that could help us evaluate our results?" and also has a placeholder "Your answer". At the bottom of the form, there are three buttons: "Back", "Submit", and "Clear form". Below the buttons, there is a footer that reads: "Never submit passwords through Google Forms. This form was created inside Universidade do Porto. [Report Abuse](#)".

Figure B.4: Form's fourth page.