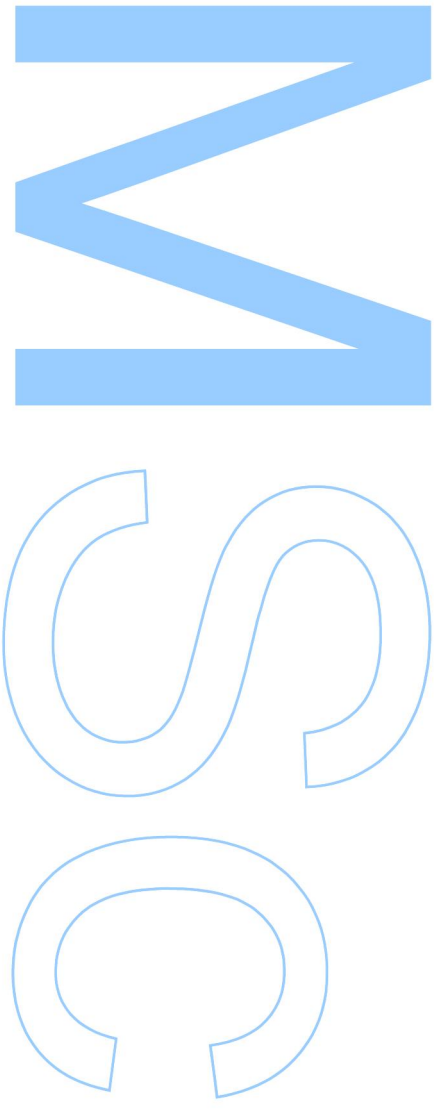
Optimizing call center operations with Reinforcement Learning

Cátia Sofia Coutinho Correia
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciência dos Computadores
2020

Orientador
Nuno Miguel Pereira Moniz
Professor Auxiliar Convidado
Faculdade de Ciências da Universidade do Porto

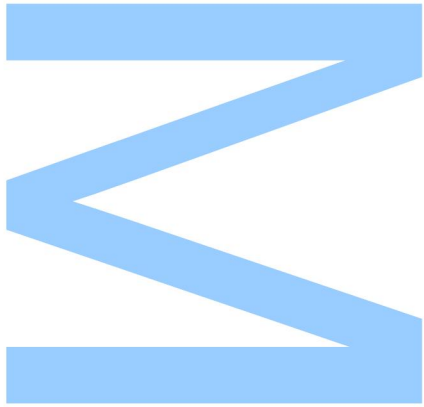
Supervisor Externo
João Pedro Maia Rafael

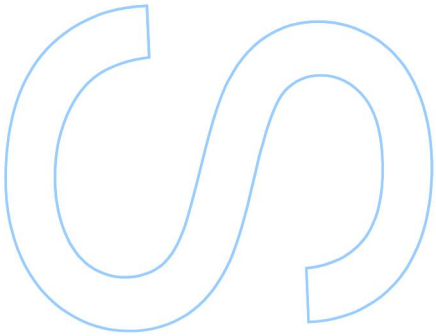


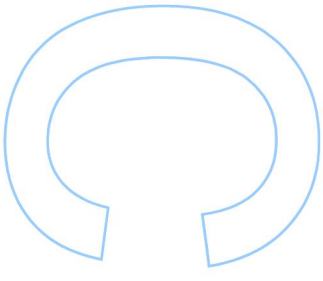
Todas as correções determinadas pelo júri, e só essas, foram efetuadas.

O Presidente do Júri,

Porto, _____/_____/_____







Acknowledgements

I want to express my gratitude to my supervisors, João Rafael and Nuno Moniz, for giving me this opportunity, for all the support throughout this process and all the suggestions that helped improve my work.

I would also like to thank my friends with whom I have spent many arduous work times and shared many priceless moments. They provided me with much-needed laughter, confidence, and support during all the years.

Finally, I would like to thank my dear parents, who worked hard to give me the means to continue my studies and taught me always to seek the best.

Abstract

In a competitive market such as the present, customer satisfaction is a critical factor of difference and progressively appears as a decisive business strategy element. Organizations that have satisfied customers tend to have more clients and, consequently, more profit. Call centers play a fundamental role in customer satisfaction, as they allow direct contact with clients. One of the tasks performed by the call centers is outbound operations, where the company calls clients. Making these calls without any knowledge can impair satisfaction, and it is an inadequate business strategy, so it is essential to know how these operations can be carried out more efficiently. Knowing the best time to contact each customer is the right approach because, with this knowledge, call centers may prioritize their calls to maximize the number of answered calls.

Reinforcement learning is an area that has many successes in optimizing this type of problems, and therefore it is something that must be experimented. For such reason, this thesis aims to solve the problem of estimating the best time to contact a company's customers, using reinforcement learning techniques.

Experiments were carried out to compare reinforcement learning solutions and supervised learning-based solutions. The conclusion reached is that there is no considerable or notable difference between the two approaches. This is an introductory work and, therefore, other reinforcement learning methods should be explored to arrive at a more global comparison concerning a comparison between such approaches.

Keywords: reinforcement learning; optimization; call centers

Resumo

Num mercado competitivo como o atual, a satisafação dos clientes é um fator crítico de diferenciação e surge progressivamente como um elemento decisivo das estratégias empresariais. Organizações que possuem clientes satisfeitos tendem a ter mais clientes e, consequentemente, mais lucro. Os call centers desempenham um papel fundamental na satisação dos clientes, pois permitem o contacto direto com estes. Uma das tarefas realizadas pelos call centers são as operações outbound, onde a empresa liga para os clientes. Realizar estas chamadas sem nenhum conhecimento prévio pode prejudicar a satisfação dos clientes e é uma estratégia de negócio inadequada, por isso é essencial saber como é que estas operações podem ser realizadas de forma mais eficiente. Saber o melhor momento para entrar em contacto com cada cliente é a abordagem certa, pois com este conhecimento, os call centers podem priorizar as chamadas de forma a maximizar o número de chamadas atendidas.

Aprendizagem por reforço é uma área que teve muitos sucessos na otimização deste tipo de problemas, e, portanto é algo que deve ser experimentado. Por esse motivo, esta tese visa solucionar o problema de estimar o melhor momento para contactar os clientes de uma empresa, utilizando técnicas de aprendizagem por reforço.

Experiências foram realizadas para comparar entre soluções de aprendizagem por reforço e soluções baseadas em aprendizagem supervisionada. A conclusão a que chegamos é que não há diferença considerável ou notável entre as duas abordagens. Este é um trabalho introdutório e, portanto, outros métodos de aprendizagem por reforço deviam ser explorados para chegar a uma conclusão mais global sobre a comparação entre as duas abordagens.

Palavras-chave: aprendizagem por reforço; otimização; call centers

Contents

List of Tables

List of Figures

Acronyms

ML	Machine Learning
SL	Supervised Learning
ANN	Artificial Neural Network
RL	Reinforcement Learning
MDP	Markov Decision Process
MDPs	Markov Decision Processes
DQN	Deep Q-Learning
IS	Importance Sampling
WIS	Weighted Importance Sampling

Chapter 1

Introduction

Customer services consist of offering services to clients before, during, and after a purchase. The quality of service provided determines the level of customer satisfaction, which indicates how products and services supplied by a company encounter or overcome customer expectations. In a competitive marketplace, as it is today, businesses have to compete for clients permanently. Therefore, customer satisfaction is a critical difference-maker and progressively appears as a critical element of any business strategy. Organizations that have satisfied customers are bound to increase their customer base – hence profitability. Therefore, it is crucial that industries provide quality service to customers to satisfy them, make them loyal, and retain them.

One of the main tasks in customer services relates to direct contact, be it physically or by telephone and digital channels. When we mention direct contact through telephone, we are mostly referring to call center operations. Call centers are an essential part of companies because they allow obtaining client feedback. Their activity can be divided into two main segments: inbound service, that handles calls received from clients, commonly related to information and billing inquiries; and outbound service, responsible for contacting clients, with the common objective of inquiring them about satisfaction with subscribed services or to promote new ones [Aksin et al., 2007].

Companies face several difficulties concerning outbound services, such as: *i)* which customers have the most appropriate profile for the services the company is trying to sell, or for the questionnaire; *ii)* at which time should one make a call to a particular client; or, how many times should one call unique clients. Calling clients without this knowledge can harm satisfaction, as in situations where clients are contacted several times in a short period; clients are always contacted in an inappropriate time; or, clients are never contacted. Knowing how to answer these questions is essential for the success of outbound operations and, consequently, the company's success.

In this thesis, we focus on the problem of outbound services and how to solve the problem of discovering the best time to contact customers of a particular company. Based on that information, our goal is to determine which customers should be contacted at each hour of the day to maximize the number of clients reached.[1]

[1]This work was developed in collaboration with a Portuguese telecommunications company.

1.1 Motivation

For large companies to be able to contact all their customers, many calls have to be made. This leads to a problem: call centers have limited resources. So, information concerning the best time to contact each client is crucial. When referring to the best time to contact a client, we do not necessarily mean the hour when a client has the most significant probability of answering. What we mean is knowing, for each client, at each hour, how advantageous it would be to call. With such information, call centers may prioritize their calls to maximize the number of answered calls. Therefore, the solution to the success of outbound call center operations can be formulated as an optimization of a sequential decision-making problem.

Reinforcement Learning (RL) [Sutton and Barto, 2018] is an area that has had many successes in optimizing these types of problems. Among recent work in this field, two outstanding success stories arise. The first is the development of an algorithm that learns to play a range of Atari 2600 video games [Mnih et al., 2013] at a superhuman level, directly from image pixels. This work demonstrates that RL agents can be trained on raw, high-dimensional observations, exclusively based on the reward signal it receives. The other success story is AlphaGo [Holcomb et al., 2018], a hybrid Deep RL algorithm [Arulkumaran et al., 2017] that defeated a human world champion in Go, a strategy board game. The innovation in AlphaGo was the utilization of comprised neural networks that were trained using supervised learning and RL, in combination with a traditional heuristic search algorithm [Arulkumaran et al., 2017].

Objectives Related work about the application of Reinforcement Learning in robotics and games is widespread. However, in the context of industry, efforts are less explored. Within the scope of the problem addressed in this thesis - the optimization of outbound operations from a call center, we are only aware of one previous work developed a long time ago, which, therefore, does not leverage the current state-of-the-art in this research field. As such, alongside all the successes that RL systems have demonstrated, and how advantageous it is to companies to have this information, in this work, we formalize our problem as a reinforcement learning task, exploring the contribution of its methods.

In summary, this project aims to explore RL's use to optimize the calling strategy of outbound services operations at a call center, considering simultaneously, client's preferences, operational restrictions, and continued long-term business objectives (e.g., revenue, customer satisfaction).

1.2 Contributions

The development of this work leads to the following main contributions:

- A survey of state of the art on Reinforcement Learning, with particular focus on the context of call center operations;
- A study on the application of Reinforcement Learning approaches to optimize outbound call center operations;
- An analysis and discussion concerning the advantages of Reinforcement Learning methods in this context.

1.3 Thesis Structure

The remaining document is structured as follows. Chapter 2 introduces state of the art in both Reinforcement Learning and call center optimization. Chapter 3 provides the problem definition, the methodology adopted to solve the problem, and a description of the data sets used. Chapter 4 presents an an experimental study based on the data set presented. Finally, chapter 5 concludes and indicates directions where improvement is possible for future work.

Chapter 2

Literature Review

In this chapter, the methods and techniques needed to achieve our goals are reviewed. We start with a brief overview of machine learning, followed by a summary of well-known supervised learning methods. Next, the topic of Reinforcement Learning is presented to provide a better understanding of the work developed in this thesis. The RL area has experienced tremendous growth in recent years. Thus, it raised great interest across several sub-fields. Each of these sub-fields has attempted to solve a different problem resulting in a variety of methods. For this reason, we present an overview covering several concepts, instead of a complete analysis of the current state of the art. Finally, some applications of RL focusing on call center operations are discussed to frame better the magnitude of the contribution made by this dissertation.

2.1 Machine Learning

Machine Learning (ML) is a sub-field of Artificial Intelligence that concerns the task of developing learning capabilities in computer programs [Alpaydin, 2004]. Instead of executing pre-programmed instructions, a machine learning algorithm learns a model from existing data in order to be able to make predictions or infer decisions for a problem related to the data. The aim is to go from data to relevant information that can be used for some propose, such as companies knowing which products they should try to sell for each client.

The problems that Machine Learning has focused on can be divided into four main areas, each characterized by its degree of supervision. In Supervised Learning [Kotsiantis et al., 2006], the algorithm is given data with target values, and the aim is to learn a mapping from the input features to the target output. This is done by training the model in part of the data, called training data, and then in the rest of the data that was not used in training, we hide the target feature and use the trained model to predict what would be the target output of that observations. As we have the real target output, we can compare the results obtained by our model with real ones and know how good our model is. Based on the target value's data type, there are two main tasks in supervised learning: classification and regression. Classification is used when the target output we want to predict is categorical and regression when it is numeric.

Another area is Unsupervised Learning. Here, only input data is used, which means no known target values for each example. So, the goal is to identify similarities and other patterns contained in the data. The methods in this area work similarly to supervised learning, in the sense that a model is also trained

with training data and validated on test data. However, algorithms in such tasks are more challenging to evaluate due to commonly having no ground-truth and, therefore, the lack of a direct comparison – as in supervised learning [Dutton and Conroy, 1997].

The next area is Semi-Supervised Learning. This area represents a middle-of-the-road scenario between supervised and unsupervised learning, as it has both labeled and non-labeled examples. Labeled examples are used to learn the model, while unlabeled examples are commonly used to define the boundaries between classes [Chapelle et al., 2010].

Finally, the area of Reinforcement Learning. For algorithms in reinforcement learning, supervision is provided in the form of rewards and punishments, instead of explicitly desired outputs. The objective in such area is to learn how to map situations to actions and maximize rewards. Given its importance to the work developed in this thesis, we expand on this area in Section 2.3.

2.2 Supervised Learning

As mentioned before, there are two main tasks in supervised learning. In this dissertation we will focus on classification problems due to the connection to our core problem. The following description provides a formal definition of classification tasks.

Given a set of pair observations $D = (xi, f(xi), i = 1, ..., n$, where f represents an unknown function, the objective of a predictive task is to learn an approximation of f. This estimator h, will allow the estimation of values of f in new observations x. Since it is a classification task, $f(xi)$ assumes values in a discrete, unordered set [Gama et al., 2012].

2.2.1 Methods

The range of algorithms in this problem is extensive. As such, we focus our discussion on well-known methods, including Decision Trees, tree-based ensembles, artificial neural networks, and the TabNet approach.

Decision Trees This algorithm is a hierarchical data structure approach that implements a divide-and-conquer strategy. It consists of a root node that links to decision nodes, according to a function applied to the input features. This function continues to be applied to decision nodes until all nodes become terminal nodes (nodes without outgoing edges). Each terminal node is assigned to the class that represents the most appropriate target value for that path. According to the tests' outcome along the path, observations are classified by navigating from the tree's root down to a terminal node [Alpaydin, 2004].

Ensemble of Decision Trees An ensemble method is a combination of multiple learning algorithms, wherein this particular case, decision trees. A problem with decision trees is that they tend to incur in overfitting (i.e., memorizing the data), as they have high variance and low bias [Alpaydin, 2004]. One approach to tackle this difficulty is training different decision trees and combining their predictions. This way, the variance is reduced while maintaining low bias.

Bagging [Breiman, 1996] and Boosting [Schapire, 1990] are the most common ensemble methods utilized. In bagging, each classifier's training set is generated by randomly drawing instances with replacement – allowing some instances to be represented multiple times. Boosting, as opposed to bagging, is an iterative approach where each tree-based model is generated with instances from the training set on the basis of the performance of earlier models. That is, examples incorrectly predicted by previous classifiers are chosen more frequently than correctly predicted examples. Thus, this method aims to obtain classifiers that can better predict the examples that previous classifiers were incorrect [Opitz and Maclin, 1999].

Random Forest Random Forest [Breiman, 2001] is an ensemble machine learning method that adds an extra layer of randomness to bagging. That is, in addition to choosing at random the cases used to train each tree, it also changes the way the trees are created. In typical trees, each node is divided according to the best split among all variables. Whereas in random forest, each node is divided only according to a subset of variables chosen randomly in that node [Liaw et al., 2002]. This algorithm chooses as the predicted class in a classification task, the class most frequently preferred among all trees, i.e., majority voting.

Gradient boosting The gradient boosting algorithm [Friedman, 2001] is a type of boosting that uses gradient descent algorithms to minimize the prediction error in sequential models, making each learner more effective than the previous one [Natekin and Knoll, 2013].

Xgboost Xgboost [Chen and Guestrin, 2016] is a tree-based ensemble algorithm, derived from the gradient boosting algorithm, which focuses on computational speed and model efficiency. To have these improvements, it allows the creation of trees in parallel; uses out-of-core computing to analyze massive data sets; uses distributed computing methods to evaluate complex models; implements cache optimization to make the best use of available hardware and resources.

The next figure illustrates the discussed algorithms and aims to clarify their interpretation.

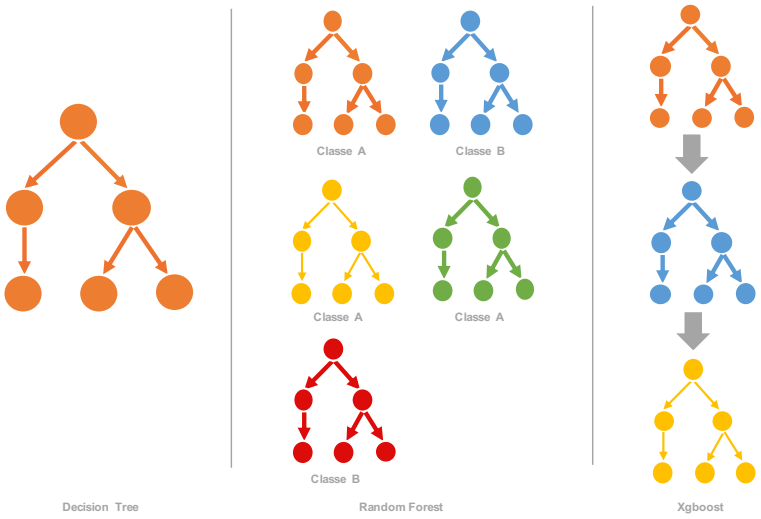


Figure 2.1: Supervised learning algorithms. Figure inspired by [Silipo, 2020].

Artificial Neural Networks An artificial neural network (ANN) [McCulloch and Pitts, 1943; Rosenblatt, 1958] is a distributed system computationally inspired by the neural networks present in the brain. It consists of several simple processing units, called neurons, which are densely connected to form the layers. Each ANN consists of an input layer where data is received, an output layer that provides the result of the computation done within the network, and between these two, an arbitrary number of hidden layers. Learning involves adjusting the parameters of the model - bias and weights - to minimize generalisation error. To be able to make these adjustments, artificial neural networks are trained using the backpropagation algorithm [Rumelhart et al., 1986]. This algorithm is divided into three main steps, as follows:

- in the first step, called feedforward, each training instance is given to the network, layer by layer, until it reaches the output layer. And then, the difference between expected and actual production is calculated;
- next occurs the step known as the reverse/backward step: the process goes through each layer in the reverse direction to measure how much each connection contributes to the error;
- finally, the gradient descent step occurs, where the connection weights are adjusted according to a hyperparameter to reduce the error.

TabNet TabNet [Arik and Pfister, 2019] is an artificial neural network that was designed to think of tabular data. The authors verified that: deep learning for tabular data remains under-explored; and that the vast majority of the algorithms used with tabular data were variants of decision trees, due to the good results they allowed to obtain and their interpretability. Based on this, the authors decided to create the TabNet: a deep neural network that looks at tabular data and tries to have the best parts of both decision trees - interpretability and sparse feature selection - and neural networks - be an utterly differentiable solution and therefore be able to learn end to end. The following is the general idea of TabNet together with figure 2.2, which aims to clarify this explanation.

- Tabular data are comprised of numerical and categorical features. In this algorithm, the raw numerical features are used, and the categorical data are mapping with trainable embeddings [Guo and Berkhahn, 2016];
- Since tabular data can have multiple domains, the next step in the algorithm consists of the resources going through a normalization block;
- After the features are normalized, they pass through a block of stacked layers, named feature transformer [Vaswani et al., 2017], where they are transformed. Part of the resulting features will be used for the final output and the other to calculate the attention of the next layer. Except in the first step where the resulting features are only used for the mask;
- Next, another block pf stocked layers - attentive transformer - emits a mask with a size equal to the number of initial variables and has values between 0 and 1. The goal of this mask is to decide the set of characteristics that must be observed in the next layer;
- The next step is again, a feature transformer followed by an attentive transformer. This process repeats for n steps;
- The decoder is composed of feature transformer blocks, followed by fully connected layers at each decision step. The outputs are summed to obtain the reconstructed features.

Moreover, by observing the masks of each layer we can better interpret the importance of each feature and how they are combined.

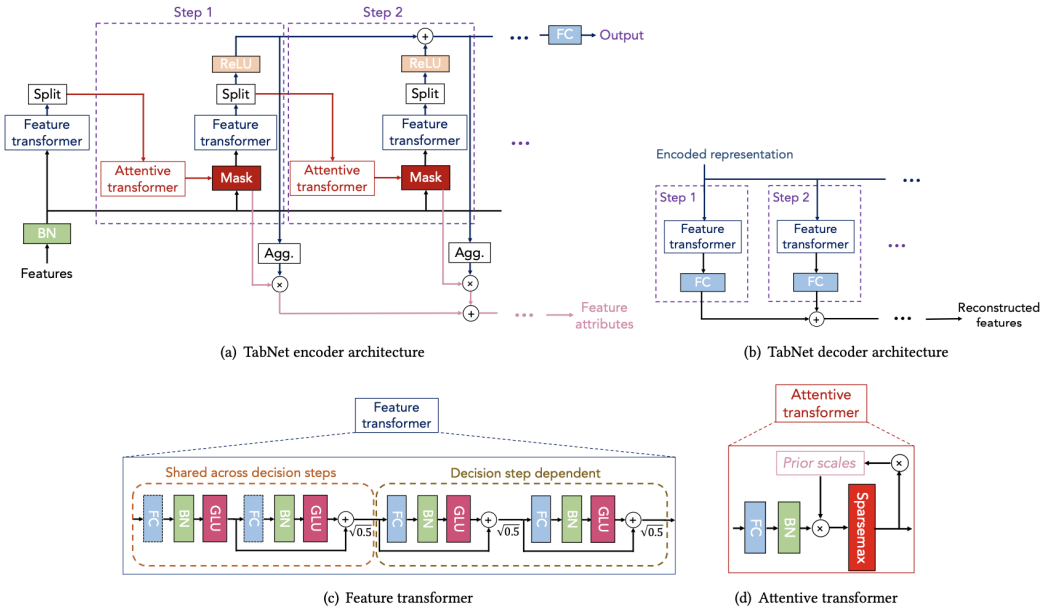


Figure 2.2: TabNet architecture. Extracted from [Arik and Pfister, 2019]. (BN: batch normalization; FC: Fully Conected; GLU: Gated Linear Unit)

2.3 Reinforcement Learning

In Reinforcement Learning, an agent attempts to maximize the expected sum of rewards regarding some objective, such as sequential decision-making processes [Sutton and Barto, 2018]. It is motivated by Thorndike's Law of Effect, which describes that "applying a reward immediately after the occurrence of a response increases its probability of reoccurring while providing punishment after the response will decrease the probability" [Thorndike, 1911].

A RL problem consists of an agent that learns some behavior through trial-and-error interactions with an environment [Kaelbling et al., 1996]. In each iteration, the agent acts on the environment and gets a scalar reinforcement signal, called reward. The reward is a way of informing the agent on how good (or bad) is to apply a given action in a given state. The agent's goal is to learn a policy - a mapping from states to actions - that maximizes the expected sum of rewards, that is, learn the optimal policy.

Based on Sutton and Barto [2018], an RL problem can be described by considering the following items:

Environment Defines the states and actions available to the agent, defines the logic through which actions transform states, and provides the agent's rewards. An environment can be defined in the following forms:

- fully observable environment: the agent knows all the information about the environment;

- partially observable environment: the agent only see part of the information about the environment;
- deterministic environment: the same action in the same state always leads to the same next state;
- stochastic environment: the same action in the same state does not necessarily lead to the same next state;
- discrete environment: when there is only a finite state of actions available for moving from one state to another, like in a chess game, where only a set of moves are available;
- continuous environment: where there is an infinite state of actions available to move from one state to another, as in a maze game where the agent can move a certain number of degrees to anywhere.

Agent Represents the controller of the system, which learns and makes decisions. An environment can have just one agent or multiple agents, which can cooperate or compete.

Policy Defines the behavior of the agent. That is, it maps the states of the environment to actions that the agent can take when in those states. A policy can define which action the agent should take when it is in a given state, called a deterministic policy, or it provides a probability of doing each of the possible actions from that state – in this case, a stochastic policy.

Reward Signal The immediate value that the agent receives after performing a given action in a particular state. So, the reward indicates what is good (or bad) in an immediate sense. However, knowing just information about the immediate reward is not sufficient to know the path of actions that the agent needs to perform to reach the goal.

Value function Indicates for each state the total discounted rewards that the agent can expect to accumulate over the future if it starts in that state and follows a given policy. (s: state; a: action; π: policy; R: reward; γ: discount factor)

$$V_\pi(s) = \mathbb{E}_\pi[(\sum_{k=0}^{\infty}(\gamma^k R_{t+k+1})|s_t = s, a_t = \pi(s))] \tag{2.1}$$

Action-Value function The action-value function takes into consideration the action performed. That is, it corresponds to the cumulative reward that the agent expects to receive if it starts in a particular state, takes a specific action, and follows a given policy.

$$Q(s,a) = \mathbb{E}_\pi[(\sum_{k=0}^{\infty}(\gamma^k R_{t+k+1})|s_t = s, a_t = a)] \tag{2.2}$$

Model Attempts to imitate the environment's behavior to allow inferences to be made about how the environment will behave if an agent is in a given state and performs a particular action. Models are used for planning: to decide which actions should be done by considering possible future situations before they occur. Knowing the model of the environment is not a mandatory element in methods for solving RL problems.

The next section will present some of the different methods for solving a reinforcement learning problem, also based on Sutton and Barto [2018].

2.3.1 Methods

There are several ways of classifying RL methods – Figure 2.3 provides a general overview.

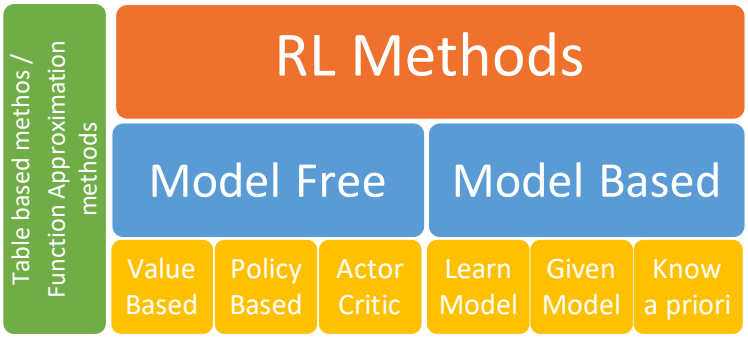


Figure 2.3: Taxonomy of RL algorithms.

A essential difference between RL methods regards whether the method is model-free or model-based. Model-free methods do not need to have a model of the environment. The agent learns how to behave directly from interactions with the environment. These algorithms are generally fast and easy to implement. Model-based methods aim to know how the environment works (know its dynamics) and then plan a solution using that model to find the best policy. The agent can apprehend the model from interactions with the environment, or it can be provided, for example, by another agent, or it can be known a priori such as in chess. Despite the requirement of additional computation power, this type of method is known to be data efficient. However, they tend to fail when the state space is too large. In short, model-free methods rely on learning, while model-based algorithms rely on planning.

The model-free algorithms can be divided into three categories based on the principal method of the learning:

- **Policy-based methods:** such methods try to learn the optimal policy directly from interactions with the environment. This type of algorithm allows better convergence and effectiveness on high dimensional or continuous action spaces. REINFORCE [Williams, 1992] is an example of an algorithm in this category;
- **Value-based methods:** these methods try to learn the optimal action-value function directly from interactions with the environment. Examples of algorithms value-based are Q-learning and SARSA [Rummery and Niranjan, 1994].
- **Actor-Critic methods:** the final category is Actor-Critic, which combines both policy-based and value-based methods. The actor (policy) learns using feedback from the critic (value function), and the goal is to optimize both the police and the value function. An example of an algorithm is Asynchronous Advantage ActorCritic [Mnih et al., 2016].

Another categorization of RL methods lies in how the algorithms learn an estimate of the optimal value function or policy during the learning phase. An off-policy method can learn an optimal policy from data collected by executing a different (non-optimal) policy. Q-learning is an example of an algorithm of this category. An on-policy method chooses actions using the policy derived from the current estimate of the optimal policy, and the updates are also based on the current estimate of the optimal policy. An example of this type of algorithm is SARSA [Rummery and Niranjan, 1994].

The RL methods can be table-based or function approximation methods. Table-based methods are used when the state and action spaces are small enough to represent each state-action pair's approximate value as an array or table. This characteristic allows these methods to find often precisely the optimal value function and the optimal policy. Q-Learning [Watkins, 1989] is an example of this type of method. In function approximation methods, the state and action spaces are arbitrarily large, so it is not expected to find the optimal value function and optimal policy even in the limit of infinite time and data. Instead, our goal is to find a good approximation of both value function and the policy using limited computational resources. This objective was achieved by combining RL methods with existing generalization methods. Many table-based methods can be upgraded into function approximation methods, e.g., Deep Q-Learning(DQN) [Mnih et al., 2015].

Throughout this explanation, different examples of algorithms are described. Now, some of these algorithms are briefly presented.

Q-learning It is an off-policy, model-free algorithm. It creates a table named Q-table, where each line represents a state, each column represents an action, and has as values the Q-values [Watkins, 1989]. The table is initialized to zero, and its values are updated after each action occurs. Whenever an agent is in a state and wants to know what action to do next, it looks at the table and chooses the action according to is ϵ-greedy policy. That is, ϵ times choose a random action, and $1 - \epsilon$ chooses the action which maximizes the accumulated sum of rewards.

Deep Q-Learning As already mentioned, Q-learning is a table-based method, which means that when the number of states and actions is vast (or infinite) it is not computationally feasible to use this algorithm. As a solution, Deep Q-Learning emerged [Mnih et al., 2015]. It is an off-policy, model-free algorithm that uses a neural network to estimate Q-values. When an agent wants to decide the action to do next, it sends the actual state as input to the neural network and receives as output the Q-value of each available action. Similar to the Q-learning, it chooses the action according to an ϵ-greedy policy. Figure 2.4 illustrates the differences between these two methods.

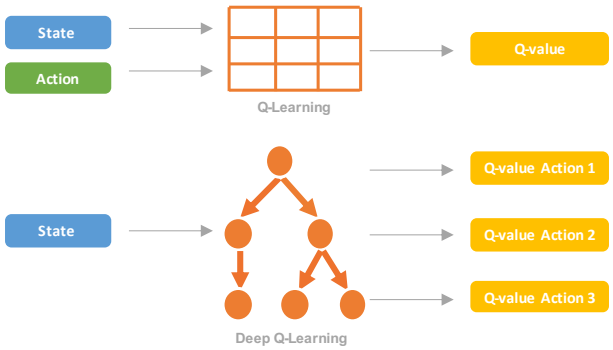


Figure 2.4: Q-learning and Deep Q-learning. Figure motivated by [Choudhary, 2019b].

SARSA This algorithm is similar to Q-learning, except in the way the Q-value is updated [Rummery and Niranjan, 1994]. While the off-policy method Q-learning updates its Q-values using the update rule 2.3, where the *max* operator causes the greedy action - the action that gives the maximum Q-value for the state - to be chosen, and, consequently, the estimation policy to be greedy, which guarantees the Q-values converge to the optimal Q-value, $Q^*(s, a)$. For SARSA, an on-policy method, the behavior policy, and the estimation policy are equal. So, it updates its Q-values based on the current policy's action instead of the greedy action as Q-learning does. Equation 2.4 shows its Q-values update rule. Although this difference, SARSA can converge to optimal Q-values, just like Q-learning.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.3}$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2.4}$$

2.3.2 Batch Reinforcement Learning

Reinforcement Learning tasks involve an iterative collection of experience by interacting with the environment and then using that experience to improve policies [Sutton and Barto, 2018]. However, in some situations where data collection is expensive or risky, this online interaction is impractical. Examples include robotics, health, and autonomous driving. Furthermore, even in circumstances where online interaction is possible, it may be preferable to use previously collected data - for example, if the domain is complex and for effective generalization, large data sets are necessary.

Batch reinforcement learning is the solution to overcome these difficulties. It is a subfield of Reinforcement Learning characterized by learning from a set of collected experiences [Ernst et al., 2005; Lange et al., 2012]. In the most general case of batch RL, the agent can no longer interact with the environment and collect additional transitions. Instead, the learning algorithm is provided with a fixed data set of interactions and must learn the best possible policy using only that data set.

According to [Levine et al., 2020], this type of learning is a difficult problem for multiple reasons:

- Since this algorithm needs to learn exclusively from the static data set, exploration is out of reach. Therefore, if the data set does not contain transitions that illustrate high-reward regions, it may be

impossible to discover them, indicating that it is not always possible to obtain an optimal policy;

- The batch RL goal is to have a learned policy that performs better than the behavior policy (the policy that exists in the data set). Hence, for this objective to be achieved, the sequence of actions performed must be different. However, this leads to distributional shift - when the learned policy was trained under one distribution, and it will be evaluated on a different one. This issue can be addressed in several ways, being the simplest to restrict the learning process so that distributional change is limited.

As a conclusion of this section, Figure 2.5 is presented to clarify the distinct types of learning: online reinforcement learning, off-policy reinforcement learning, and batch reinforcement learning.

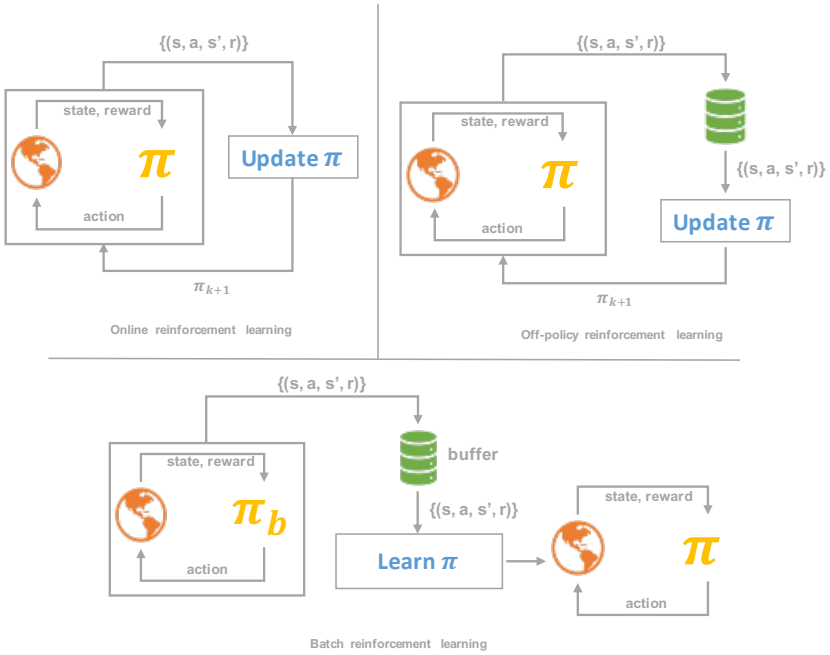


Figure 2.5: Illustration of online reinforcement learning, off-policy reinforcement learning, and offline reinforcement learning. Figure inspired in [Levine et al., 2020].

2.4 Best time for contact

The vast majority of the literature on the optimization of call center operations concentrates on inbound call centers. Avramidis et al. [2010] propose a solution that combines simulation with integer or linear programming to optimize agent scheduling over one day. Chevalier and Van den Schrieck [2008] discuss staffing issues, Dietz [2011] integrates queuing theory, quadratic programming, and a variable-threshold rounding algorithm to derive an optimal schedule, and L'Ecuyer [2006] gives an overview of some central problems in call center management.

Comparatively, there is much less research on outbound call centers. Samuelson [1999] developed an algorithm to deal with the problem of determining when to dial the next call, paying attention to whether a call center operator has already completed the previous call and that the new call is answered without

delay. This solution keeps the workers busy while reducing the number of abandoned calls. Bollapragada and Nair [2010] tries to maximize the right party contact rate - a metric used in banks to know the ratio of calls answered by the account holder. For this purpose, the author divided the work into two sections: estimate the likelihood of contact with each account holder at each hour of the day and schedule the call center calls. The first sub-problem was solved using mathematical programming, while for scheduling the calls, the authors developed a greedy heuristic. This work's idea is very similar to the one purpose in this thesis; however, we propose to estimate the likelihood of contact each client using reinforcement learning techniques.

Being more specific concerning the optimization of call centers with Reinforcement Learning, once again, the amount of previous work found is more significant in inbound operations than in outbound operations. Starting with inbound call centers, Levin et al. [2000] work tries to optimize the dialog system using a combination of supervised learning and reinforcement learning. The supervised learning is used to estimate a model of the user from the available training data. That is, to estimate the Markov Decision Process (MDP) parameters that quantify users' behavior. Then, such information is used by the Monte Carlo method [Metropolis and Ulam, 1949] to estimate the optimal strategy while the system interacts with the simulated user. In this work, it is possible to show that any dialog system can be formally described as an MDP. Lewis and Fabbrizio [2006] applied RL techniques to select the optimal prompts to maximize the success rate in a call routing application. The authors used a simulation of the dialogue outcomes to experiment on different scenarios and demonstrate that RL can make these systems more robust. The Monte Carlo algorithm was used, and the experiments consist of testing how well the algorithm adapts in a new environment, with no assumptions being made concerning the values of each state when the environment is changed.

Lastly, in optimizing outbound call-centers with RL, Greenberg and Stokes [1990] develop an optimal call scheduling strategy for telephone surveys. The authors modeled the problem of contact with households in a telephone survey as a Markov Decision Process, where states include information about the history of calls made to the phone number, the action to be selected is the time of the next call attempt, and the rewards are positive if the call is answered and negative if the survey period ends before the contact is made. The transition probabilities are estimated using a polytomous logistic regression [Engel, 1988] model. The categories predicted by the model are the possible call outcomes. With this work, the authors wanted to minimize the number of calls required to contact a household.

2.5 Summary

This chapter presents an overview of the topics needed to understand the work developed in the context of this thesis. Regarding the RL section, several types of methods have been mentioned to give a overview of the field. However, in the context of this thesis, only a select few methods are suitable. The methods used in this work, and the reason they were chosen, are described below.

Some key differences in reinforcement learning methods were presented. Namely, between table-based and function approximation methods. Here, the difference concerns if it is possible to know precisely the optimal solution or whether it is necessary to use estimators to approximate the optimal solution. As the state space in our problem is infinite, it is impossible to enumerate all possible states; thus, function approximation methods were used.

The next difference was among the model-free and model-based methods. Algorithms that do not

have information about the environment and learn by interacting with it or methods that know the environment's dynamics, respectively. A significant problem with model-based methods is when the environment model is unrealistic. This situation leads to unsatisfying results because the algorithms will focus particularly on areas where the model deviates from reality. As we do not believe that we can have a sufficiently realistic environment model due to the fact that we are dealing with persons, which means that various external factors can influence an observation, we decided to choose model-free agents because we consider that we can have better results this way.

As model-free methods are used, the last difference concerns how the agent learns to behave optimally - if it learns the policy directly, uses the value function, or uses both. That is, whether it is a policy-based method, value-based method, or actor-critic method, respectively. Since most of the information we found about model-free methods was related to value-based methods, we decided to use this type of algorithms in this work. Value-based methods are also similar to the existing solution, facilitating the communication, comparison, and finally, deployment of such an algorithm. We should note the importance of developing future experiments with the remainder types of algorithms.

In addition to the taxonomy of the RL algorithms, this section also mentioned batch reinforcement learning. Subsection 2.3.2 expressed some reasons why this type of learning is used, one of which was "if the domain is complex and for effective generalization, large data sets are necessary". Since our problem's domain fits such description, and the partner company already has an extensive data set with such information, it was decided that batch RL would be used in this work.

Chapter 3

Problem Definition and Data

When a company wants to contact its clients, it has a limited number of calls that can be realized, ergo, it has to select which customers should be called to obtain the best possible outcome. The solution to this problem consists of having a good calling strategy. In other words, estimating which is the best time to contact each client and, based on this information, decide the order by which the calls should be performed at each hour in the call-center.

This work is being developed within the scope of a partner company that has already implemented a process to optimize outbound call center operations. This process consists of estimating for each customer the likelihood of a call being successful at each time of the day using a supervised learning method. Then, based on this information, a single dialing policy is created, indicating which customers should be contacted at each hour of the day. Figure 3.1 illustrates how the existing process works.

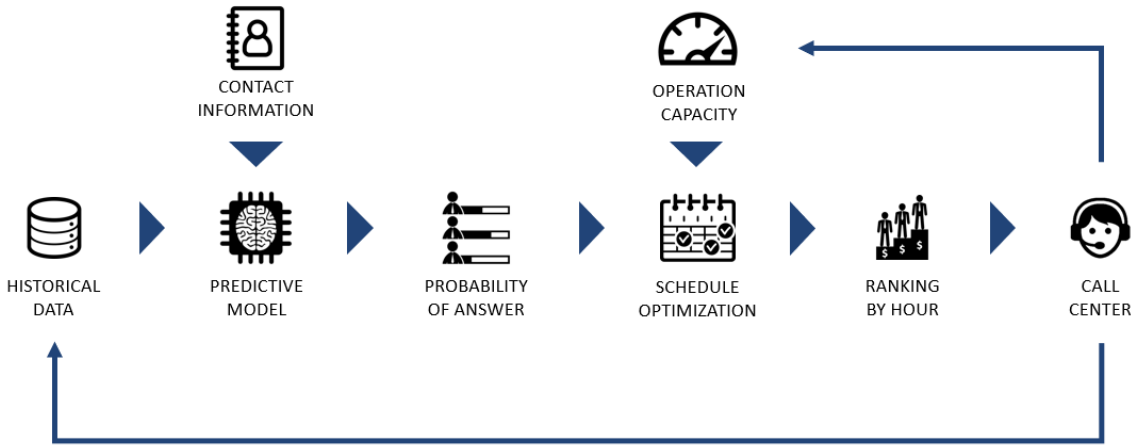


Figure 3.1: Outline of the process currently implemented in the partner company to optimize the outbound operations of the call centers

To be able to create this dialing policy, the company uses a training set. This data set results from cross-referencing call history and customers' information from various resources. When a campaign is ongoing, meaning that the call-center is calling clients, there is a daily exchange of information between the

company and the call-center. The call-center sends to the company three updated files: one indicating the calls made that day and their respective outcome (e.g., handled, no answer, invalid number); another with the current business outcome for each client (e.g., callback, not interested, success); the last one includes clients that will not be contacted the next day because the call center blocked them as a consequence of business rules that are predefined by the call center (e.g., already called to a customer more than a given number of times and the client never answered). This data is consolidated into a relational database that supports the optimization process and other reporting and analytic systems. Based on this information, the company sends a file to the call-center that indicates, in an orderly manner, which clients should be contacted at each hour of the next day.

This thesis proposes to solve the problem of determining the best time to call each client using Reinforcement Learning techniques.

3.1 Formal Definition

According to Sutton and Barto [2018], Markov decision processes (MDPs) are a classic formalization of sequential decision-making, where actions influence not only immediate rewards, but also the following states and, consequently, all future rewards. Thus, MDPs involve trial-and-error search and delayed reward, which are the two most critical distinguishing characteristics of reinforcement learning. As so, Markov decision processes provide the mathematical formulation for RL problems.

A Markov decision process is defined by a tuple (S, A, T, R) with the following composition:

- The set of states, $s \in S$, that the agent can take;
- The set of actions, $a \in A$, that the agent can make;
- A deterministic or stochastic state-transition function $T(s_{t+1}|s_t, a_t)$, which defines the probability of going to the next state s_{t+1} from state s_t by performing the action a_t;
- The reward function $R(s_t, a_t, s_{t+1}) = \mathbf{E}(r_t|s_t, a_t, s_{t+1})$, that indicates a one-step reward for being in state s_t and performing the action a_t.

.

Figure 3.2 represents the agent-environment interaction in an MDP and can be interpreted as follows: at each interaction, the agent in an arbitrary state s_t selects and acts a_t on the environment. This process forces the environment to transform into the next state s_{t+1}, according to the transition function T, and to compute the immediate reward, r_t. The actions that the agent chooses are defined according to a learned policy, denoted by π.

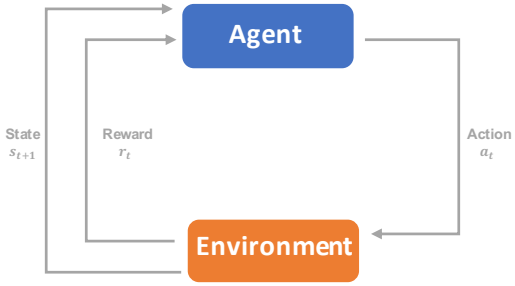


Figure 3.2: Agent-environment interaction in RL problems. Adapted from [Sutton and Barto, 2018].

As in the general reinforcement learning problem defined by Sutton and Barto [2018], the ultimate goal in batch reinforcement learning is to find a policy that maximizes the sum of the expected rewards in the agent-environment loop [Lange et al., 2012]. However, in batch RL, the agent can no longer interact with the environment during learning. That is, instead of observing a state s_t, take action a_a and adapting its policy according to the next state s_{t+1} and reward r_t, the agent only receives a static set of transitions $D = \{(s_t, a_t, r_t, s_{t+1})\}$ sampled from the environment and must learn the best policy it can only from that data. According to Levine et al. [2020], the batch reinforcement learning problem resembles the standard supervised learning problem, and we can consider D as the training set for the policy.

3.2 Challenges

As mentioned before, the partner company has already implemented a process to optimize the call center operations based on supervised learning. This application already produces a significant business impact, however, has identifiable challenges:

1. It does not consider interactions between calls when creating the dialing policy, which causes the algorithm to optimize for the short-term and, thus missing long-term opportunities;
2. As a consequence of the above, the algorithm does not identify calls that have a net loss in expected return and does not prevent such calls from being made;
3. The algorithm does not distinguish different possible outcomes that a call can have: “not answered” or “rejected” are considered the same;
4. The current dialing policy is computed using a simple heuristic: for each hour, perform the calls that minimize the expected regret (minimax-regret/“savage criterion”) [Savage, 1951], where the regret is assumed to be, for each client, the difference between the current probability and the maximum probability until the end of the day. A more complex policy could be used to exploit real-world constraints and would optimize for the long-term.

This work intends to use reinforcement learning techniques to address the challenges listed from one to three. The first challenge can be solved using the value function to know which combination of actions should be performed to obtain the maximum cumulative sum of rewards. Each interaction with the environment will have a reward identifying how good or bad was the performed action. Therefore, it

is possible to avoid actions that always have a terrible outcome (with this knowledge, it is possible to deal with challenges two and three). The final challenge will not be faced at an early stage due to the complexity involved in trying to solve it - it involves optimization with multiple agents competing witch each other - and because of the lack of data regarding operations capacity.

3.3 Methodology

Figure 3.3 illustrates the methodology adopted to solve the target problem. It consists of three distinct phases:

1. **Business and data understanding**: The first step of this methodology is to understand the telecommunications business, specifically the call centers' outbound operations. After having this knowledge, and the partner institution provides the data, the step of data understanding is initialized. The goal is to increase familiarity with the data set: know which information is present on it, what each variable represents, and verify its quality.
2. **Data preparation**: This phase covers all activities, from the partner institution's data set to that used to feed the modeling process. In chapter 4, this step will be presented in more detail, but some of the actions carried out were: data cleaning with some cases and variables being removed, and feature engineering to add relevant information that was missing.
3. **Modeling and evaluation**: In the last step, a suited algorithm for the problem is selected. After applying the model to the data, it is necessary to thoroughly evaluate the results obtained to understand them, know if the objectives have been achieved, and decide the next steps.

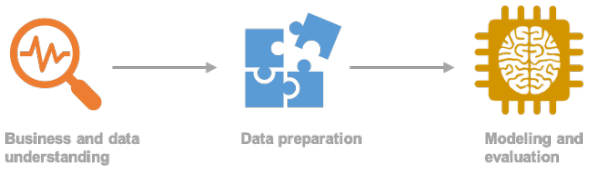


Figure 3.3: The methodology adopted to solve the target problem.

3.4 Data set description

The data set consists of a sample of historical data that describes calls between April 2019 and September 2019. Table 3.1 provides a summary of key indicators of the data set. These calls belong to several campaigns that can be in different phases, that is, as the only restriction to create the sample was the time, it is possible to have, between these dates, campaigns starting or ending. Another relevant information refers to the campaigns' working period: during the week, calls are made between ten and twenty-two hours (inclusive); on weekends, calls are only performed on Saturday between ten and twelve (inclusive).

Number of cases	$>$ 5 million
Number of features	123
Percentage of answered calls	15 %
Percentage of missed calls	85 %
Start date	2019-04-01
End date	2019-09-30

Table 3.1: Data set summary.

There are a total of one hundred twenty-three variables. One of those is the target, which is a binary variable indicating if the call was answered or not. The remaining can be grouped into four sets.

- One group has information about the customers' answered calls, like the number of calls answered during the week;
- The second group provides knowledge about the television box, such as the number of times the customer pressed the TV remote or the number of times the box was restarted;
- The next set brings information about the calls made by the dialer - automatic machine responsible for making calls at the call centers - to the customer. It has variables such as the number of days since the last call answered by the customer and the total number of calls made to the customer;
- Finally, the fourth group describes the client in the company, stating how long he/she has been in the company and the type of service he/she has.

Regarding the target variable, Figure 3.4 shows the distribution of the calls in the data set, where the percentage of calls answered was 15%.

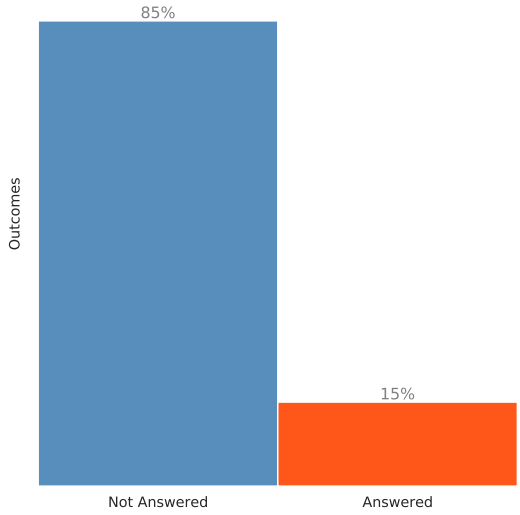


Figure 3.4: Distributions of call outcomes in the data set.

Subsequently, an analysis of the target variable was performed to understand other features' influence on it. Figure 3.5 exhibits the answer rate per hour, which enables understand that exists intervals where the response rate is highest, precisely: among eleven and fourteen; fifteen and sixteen; and finally between twenty-one and twenty-two.

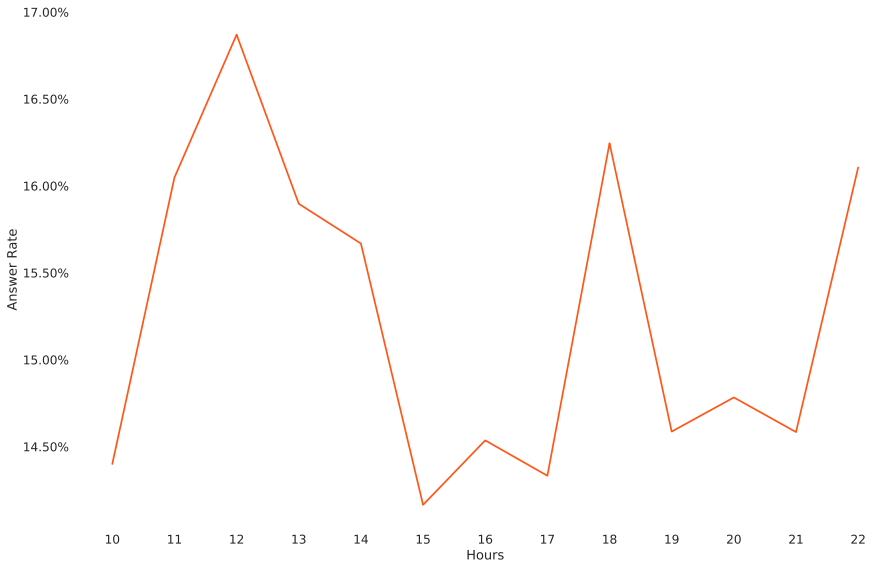


Figure 3.5: Answer rate per hour.

As calls can be made to both landlines and mobiles, it is vital to see the answer rates' distribution by phone type over time. Figure 3.6 depicts this information, which allows drawing the following conclusions: calling mobile phones yields better results, and the best time to contact a mobile phone is not the same as a landline phone.

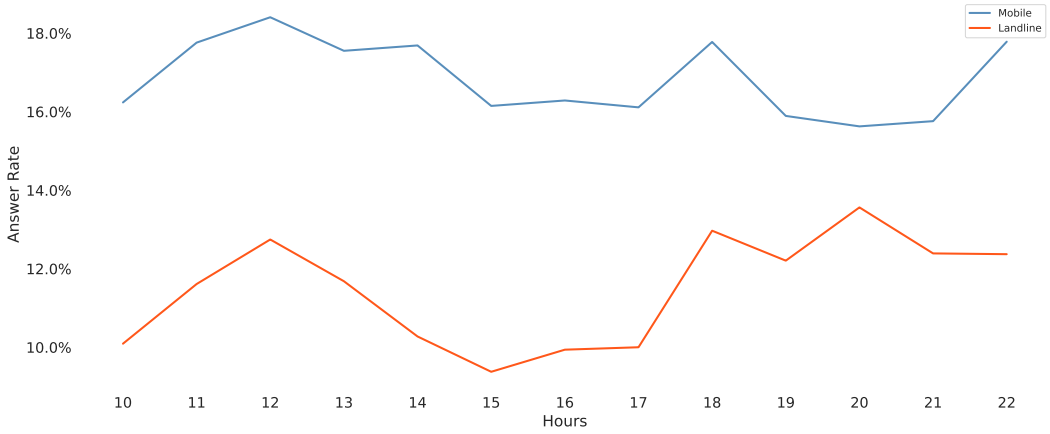


Figure 3.6: Answer rate by phone type over time of day.

Another factor that has a significant correlation with the target variable is the result of the last call made to the customer. Figure 3.7 shows that customers who answered the last call were put on hold and hung up without speaking to an operator, are 40% likely to answer again. Customers who answered the previous call have a 30% chance of answering again.

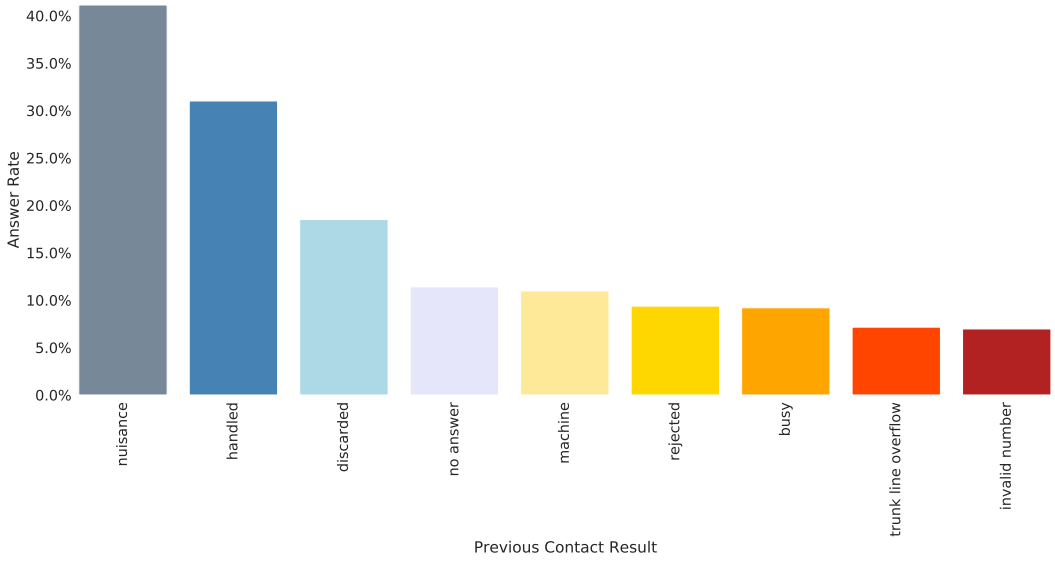


Figure 3.7: Answer rate by previous contact result.

Finally, it was decided to analyze the correlation of the variables *num_tent* and *num_atend* with the target variable. These variables refer to the total number of calls made to each customer and the total number of calls answered by each customer, respectively. These two features have essential information about the clients that can help understand their behavior. Figure 3.8 shows the results of this study. Each line represents the number of calls realized; each column refers to the number of answered calls, and the bar indicates the hit rate of a subsequent call. Thus, each point indicates the percentage of answered calls when the variables *num_tent* and *num_atend* had specific values. In some lines, there are situations where only the first columns are filled, then there is a space and then a point with 100% of answered calls. These points can be explained as situations in which the number of calls made in which the two variables in question had these values were very few, hence the answer rate of 100%. Based on the outcomes obtained, it is possible to see that in most cases, not considering the points mentioned earlier that represent few calls, the more calls previously answered, the greater the probability that the customer will answer again.

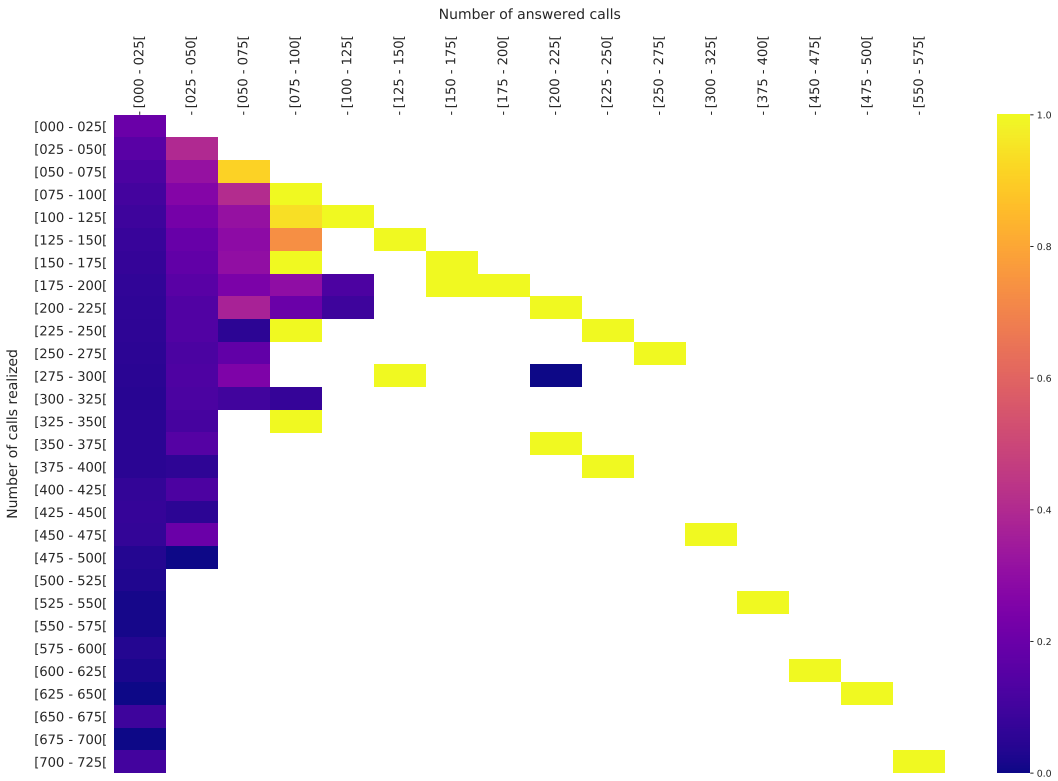


Figure 3.8: Answer rate by both the number of calls made and the number of calls answered.

3.5 Summary

This chapter aimed to provide specific information about the problem addressed in this thesis and the data used. The problem that is intended to be solved consists of estimating each customer's likelihood of answering a call every hour of the day, using reinforcement learning techniques. Regarding the data set, it was found that only 15% of calls were answered and that several factors influence the target variable. The hour of the day when the call is made is an example. A call made between eleven and thirteen hours or

between seventeen and nineteen hours is more likely to be answered than other hours. Also, calls made to mobile phones are always more likely to be answered than to landlines. Finally, it was also found that variables with information regarding previous calls are correlated with the target variable. Namely, the last call's result, the number of calls realized, and the number of calls answered by the customer.

Chapter 4

Experimental Study

This chapter focuses on presenting the results of our experimental study. It begins with details concerning implementation, followed by those regarding data preparation, which includes a description of pre-processing steps. Then, the methods used to solve the learning task are introduced, accompanied by the evaluation metrics used to assess their predictive performance. Finally, experimental results obtained are presented and discussed.

4.1 Implementation

Our experimental work was developed using the Python programming language [Rossum, 1995], which, according to Choudhary [2019a], is one of the most used programming languages in data science. Python provides access to several libraries that simplify data science tasks; it is straightforward to use and integrates seamlessly with RLlib [Liang et al., 2017] - an open-source library for reinforcement learning.

When collecting information concerning RL libraries, a wide range of options was found. In order to decide which one to use, an assessment between the different libraries was carried out. Table 4.1 presents such evaluation, including a multi-aspect comparison of several RL libraries. The color red symbolizes a negative factor, the dark blue a positive factor; other colors express intermediate results. As the RLlib library is the best overall solution according to the selected factors, this was the library chosen to develop our experimental study..

	Dopamine	RLlib	Stable Baselines	Tensor Force	TF - Agents	Coach	Horizon / ReAgent	SLM - Lab	Rlgraph	Keras - RL
Technology implemented	TensorFlow	TensorFlow / PyTorch	TensorFlow	TensorFlow	TensorFlow	TensorFlow	PyTorch	PyTorch	TensorFlow / PyTorch	TensorFlow
Number of algorithms implemented	3	12	12	10	8	24	8	10	10	7
Supports batch RL (0- no; 1- yes)	0	1	0	0	0	1	1	0	0	0
Extensibility: easy to understand and implement new algorithms	4	4	4,5	4	4	4	3,5	3	2,5	3
Number of stars on GitHub	8800	11 100	1900	2700	1300	1700	0	698	239	4600
Number of commits	121	4406	815	1873	1044	501	411	2499	4594	308
Date of the last published commit	19/12/19	16/04/20	15/04/20	12/04/20	15/04/20	14/01/20	19/01/19	15/04/20	05/11/19	11/11/19
Number of unanswered problems on GitHub	42	808	92	1	76	61	0	4	22	9
Number of problems answered on GitHub	60	2571	499	459	177	175	0	33	50	201
Published Papers (0- no; 1- yes)	1	1	0	0	1	0	1	1	1	0

Table 4.1: Comparison between RL libraries. This comparison was performed on 16/04/2020.

4.2 Data preparation

After a process of understanding the data described in Section 3.4, such data was prepared for simulating the RL problem. The first task carried out consisted of data pre-processing, where issues found in the data were corrected. For example, one problem was related to the relevance of the variables. That is, some features contained a single value; in other cases, the information portrayed by them was not relevant or redundant, and, consequently, such features were removed. Such process resulted in the removal of 15 variables. Another difficulty was related to features containing missing values. 57 of the 123 variables presented in the data set had missing values, which corresponds to a missing information percentage of (roughly) 11%. To solve this situation, features with missing data were divided into three groups and treated independently as follows.

1. The first group consists of continuous variables that result from the ratio between two other data set features and have no value when the two features from which they derive has zero as value. Once the possible range of values these variables can get is between zero and one, and the absence of information is not an error, it is reasonable to fill the missing value with -1.
2. The second group is composed of variables with information regarding the last call made (e.g., the result of the last attempt to contact a client; duration of the last call answered by the client at eleven o'clock). To understand whether the missing information was the result of an error, for each observation with this missing information was necessary to verify if were realized calls in the day before and if that calls had all the necessary information. It was possible to conclude that all the missing information was not the result of a mistake and, thus, the missing spaces were replaced by -1 or 'NA' depending on the variable was numeric or categorical, consequently.
3. The third and final group comprises information about the client. After analyzing the data set, it was possible to check that the missing information does not result from an error, and the missing values were replaced by -1 or 'NA' depending on the variable being numeric or categorical, respectively.

After the application of such pre-processing procedures, the step of data preparation is carried out. As

already mentioned, the data set contains all calls made between April 2019 and September 2019, inclusive, and the respective outcome of each call - whether the customer answered or not. However, for the RL task, the data set must have all the timestamps at which a call was possible because, as reinforcement learning aims to maximize the cumulative sum of discounted rewards, it is crucial to achieve such purpose, to have information about the calls that could have been made, but were not.

In order to produce the desired data set, some considerations had to be taken into account:

- For each campaign, it was necessary to know the interval between which each customer was active. That is, knowing the date on which the customer became active and the date on which he/she ceased to be active. The information about the start date exists in a database table, therefore access to the table was sufficient to extract the intended dates. Regarding the end date, there are three possible reasons for closing a customer: if the campaign ends, if a sale was made, or if the customer explicitly states that he/she does not want to be contacted anymore. This information is spread over several tables, and the way to identify the end date is by checking all of them. Here, a problem was found: some customers never appeared in any of the tables, that is, they had no end date. The solution adopted was to define a threshold that limits the maximum number of days that a campaign can be open. Thus, customers who do not enter the checks will have the start date plus the threshold as their end date.
- The threshold used as the maximum number of days that a customer can be open was 60 and was decided according to the results shown in Figure 4.1. The analysis performed consisted of understanding, for all customers, how many days have passed since the activation date and the date they were last contacted. With these results, a threshold that was representative of the population was decided.

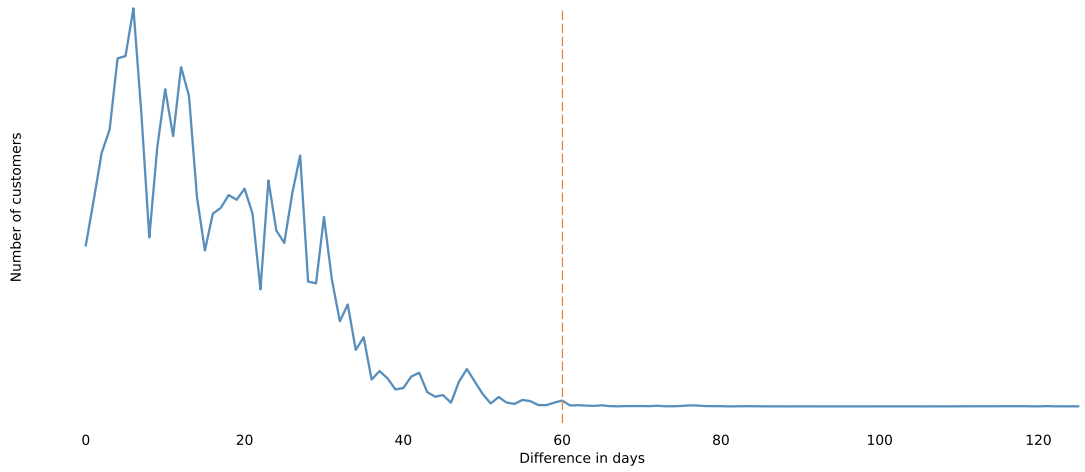


Figure 4.1: Difference between the last contact and the data the customer became active.

As we are trying to solve the problem using RL techniques, defining the components of a reinforcement learning problem in our context is necessary. Different options were considered, but the final decision consisted of defining that:

- An episode, which is a sequence of actions and states, corresponds to all the hours in which a call could have been made for each specific customer, for a given campaign and number;
- a state corresponds to all information about calls made to each customer. It has general customer information and specific information about each number;
- The available actions consist of calling the customer or not;
- The reward can take the values[1]: 100 if the call was answered, -1 if the call was not answered, and 0 if the call was not made. When a call is not made, we cannot extract any information, and therefore a neutral value of 0 is used as a reward. When the call is not answered, we want to show the agent that the action was wrong. However, this value cannot be too low; otherwise, the model would never call because the reward would be terrible if the calls were not answered. Finally, +100 as a reward for answered calls because as we have three possible rewards, 0 when a call is not made if the model learns that the probability of customer answering is low, then the way to maximize the reward is never to call. Therefore, we have to reinforce this value so that the model knows that the ideal action is when a call is answered;
- The agent will be described in sub-section 4.3.

So that the data set is in line with what the RL agent expects to receive, certain information must be added to the data set. Namely, the action, the reward, and a variable that indicates whether the episode ended. The variable action can take the values 1 and 0, if a call was performed or not, respectively. Concerning the reward variable, it is essential to note that we consider the reward per customer, day, and hour. This means that every day, every hour, it is possible to know the result of all the calls that could have been made to each customer's various numbers. It was decided to do it this way because it is enough for the customer to answer a call on any of the phones to be rewarded positively. The maximum reward that a client could have at each hour is 600, and the least is -6. The last variable inserted can have the values: 0 if the episode ends, and 1 otherwise. An episode ends when a terminal state is reached (in our context, when the customer's end date is reached). Table 4.2 exemplifies the resulting data set after all these steps being applied, and Table 4.3 resumes some crucial information about the data set.

Client	**Number**	**Timestamp**	**State variables**	**Action**	**Aux_reward**	**Reward**	**Done**
1	A	2020/04/01 10:00	...	1	-1	99	0
1	A	2020/04/01 11:00	...	1	-1	-1	0
1	B	2020/04/01 10:00	...	1	100	99	0
1	C	2020/04/01 13:00	...	0	0	0	0

Table 4.2: Exemplification of the data set resulting from this subsection.

[1]The reward values were defined by the telecommunications company for the scope of this research. It was not the product of a thorough study concerning an appropriate value of reward for each state. Therefore, it represents an ad-hoc decision that attempts to embody the general magnitude of difference in reward values between the states considered.

Number of episodes	> 20 thousand
Average number of steps per episode	52
Percentage of calls made	1%
Percentage of calls that were not made	99%
Percentage of calls answered	16,5 %
Start date	2019-04-01
End date	2019-09-30

Table 4.3: Reinforcement learning data set summary.

After the data set had all the necessary information, it is necessary to convert it to a JSON file with a specific format, as this is how the RLlib library receives the data that will be used.

4.3 Learning Algorithms

To address the reinforcement learning task at hand, we used the Distributional Q-learning algorithm [Bellemare et al., 2017], which is an extension of the Deep Q-learning algorithm, mentioned (and described) in subsection 2.3.1. The innovation brought by this method focuses on the estimation of Q-value, as shown in Figure 4.2. While in the Deep Q-learning, for each pair (state, action), there is only an estimate of Q-learning, which does not provide information about how good the estimate is or how confident the model is in it. Distributional Q-learning provides a distribution of the Q-value's possible values for that pair (state, action), solving the problems mentioned.

As also mentioned in subsection 2.3.1, the Deep Q-learning algorithm uses a neural network to estimate Q-values, just as distributional Q-learning does. We decided to start by using a multilayer perceptron [Rumelhart et al., 1986] with only three hidden layer because of its simplicity. Furthermore, after verifying that the main concern was resolved, the TabNet architecture was used for the internal model.

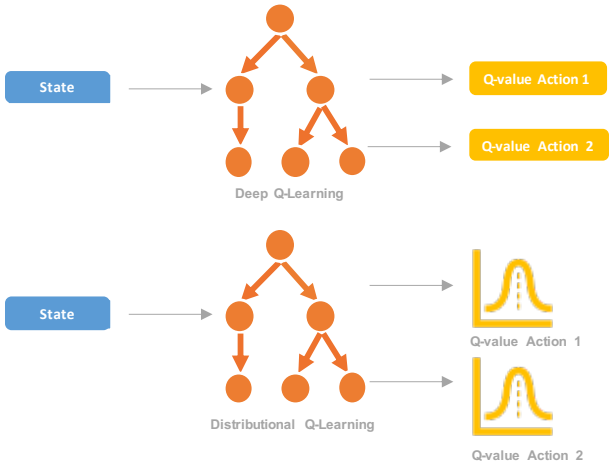


Figure 4.2: Deep Q-learning and Distributional Q-learning. Figure motivated by [Choudhary, 2019b].

4.4 Experimental Methodology

Initially, the intention was to use repeated random sub-sampling as an experimental methodology: 5 samples with 10% of customers each. However, as the RL data set contains all the calls that can be made, and on average, clients are open for a certain amount of days, it was not possible to use this approach due to database and server memory problems. Instead, we used a sample of (roughly) 1% of the clients in each iteration, and because of time constraints, only one sample was used. Summarizing, the methodology used was random sub-sampling with a sample with 1% of the clients. Despite our data's temporal dimension, as RL techniques are being used, the holdout method was used to create the training and testing data set, with 70% and 30% of the data, respectively.

4.5 Evaluation Metrics

As mentioned earlier, this problem is being solved with batch RL, which means that the agent learns from a fixed data set without further interactions with the environment. Therefore, in order to be able to evaluate the policy, the most accurate way is to execute it, according to Jiang and Li [2016]. However, in many real-world applications, like in the one this work describes, this is not viable because doing so is expensive, risky, or unethical/illegal. Off-policy value evaluation is a form of overcoming this difficulty and consists of estimating the value of a new policy based on data collected by a different policy.

According to Jiang and Li [2016], there are two types of approaches to off-policy value evaluation: fit an MDP model from the data, and evaluate the policy against the model, or use an approach based on the idea of Importance Sampling (IS), which corrects the incompatibility between the distributions induced by the target policy and the behavior policy. Since the first approach needs to learn a model of the environment with satisfactory accuracy, and this could be tremendous difficulty in complex real-world problems, it was decided to focus only on the approach based on the idea of IS.

The IS estimator provides an unbiased estimate of the target policy value. It is calculated by estimating

the IS of each episode, according to Equation 4.2, and then the final IS value is simply the average estimate over the episodes, demonstrated by Equation 4.1. $|D|$ is the number of episodes in the data set, and π represent the probability of action for each of the policies.

$$IS = \frac{1}{|D|} \sum_{i=1} V_{IS}^{(i)} \tag{4.1}$$

$$V_{IS} = \rho_{1:H}(\sum_{t=1}^{H} \gamma^{t-1} r_t) \tag{4.2}$$

$$\rho_{1:t} := \prod_{t'=1}^{t} \rho_{t'} \tag{4.3}$$

$$\rho_t := \pi_1(a_t|s_t)/\pi_0(a_t|s_t) \tag{4.4}$$

Typically, IS suffers from very high variance [Jiang and Li, 2016]. Thus, weighted importance sampling (WIS), a variant of IS, which has bias, but is a consistent estimator, was used. It is calculated in the same way as IS; that is, it estimates the WIS for each episode, according to equation 4.5, and then averages all the episodes' estimates.

$$V_{WIS} = \frac{\rho_{1:H}}{\omega_H}(\sum_{t=1}^{H} \gamma^{t-1} r_t) \tag{4.5}$$

$$\omega_t = \frac{\sum_{i=1}^{|D|} \rho_{1:t}^{i}}{|D|} \tag{4.6}$$

4.6 Results and Discussion

A set of agents has been created to assess whether the RL agent can achieve better results than other approaches to this problem. The agents considered were:

- The behavior agent, which is the agent responsible for the behavior of the test data set;
- The baseline agent corresponds to a naive approach, in which the agent acts randomly. This agent is intended to represent the situation where no intelligence would be behind the results;
- A supervised learning agent consists of an XGBoost model trained on the data set described in subsection 3.4, predicting the RL test data set's examples. This agent is used because, since the solution used in the RL task fits into the methods of approximating functions where an SL model is learned, the following question needed an answer: what happens if one of the policies in question instead of being determined by reinforcement learning is determined by supervised learning? Besides, as the partner company's current approach is based on SL, it is necessary to compare the two approaches to understand whether this work allows for obtaining more satisfactory results;
- The last agent to be considered is the RL agent that uses the Tabnet as a neural network.

As the approach used to create each of the agents is different, their information will also be different. Therefore, it was necessary to transform the results to have the same information for each agent. It was necessary to know in each state whether a call should be made or not, that is, obtain the policy of each agent. Regarding the behavior and baseline agents, the information they portray is already the policy, so there was no need to do anything. The SL agent gives the probability of a call being answered at each moment, and the RL agent indicates, at each moment, the Q-value of calling and not calling, that is, the expected reward of a making a call in that state versus the expected reward of not making a call in that state. In order to obtain the policies from these two agents, it was necessary to use a threshold to define which calls should be made. In the supervised learning agent, a call is made when the likelihood of a call being answered is larger than the threshold. Moreover, the RL agent made a call when the difference between the Q-value of call and the Q-value of not call is greater than the limit. This threshold was defined so that the number of calls made by all policies was the same. As a result of this, the IS and WIS for the SL and RL agents are calculated based on a non-deterministic policy in which the probabilities are given according to what is described: the probability of a call being answered and the difference between the Q-values, respectively. Regarding the baseline policy, it is a deterministic policy, and the probability is always one. Besides, as the only policy for which real information is available is the behavior policy, the others have to be estimated. As a result, policies can differ from each other, diminishing our confidence in the estimates produced.

As an introduction to the results and to facilitate the compression of all the information mentioned above, Figure 4.3 shows how each policy behaves throughout an episode. In each graph, it is possible to view the policy's actions, represented by the orange line, and according to the policy under analysis, another line or lines are represented, and these are related to the rewards. Graph 1 shows information related to the behavior policy. By analyzing the results is possible to understand that although several attempts to contact the customer, he/she only answered twice. In graph two, it is possible to see the baseline policy. As this policy acts at random and a call is only made when the result of the random estimate is positive, the estimated reward is always positive and is quite different from the behavior policy. Between states 100 and 140, the estimate of discounted rewards is higher because some calls were answered in other customer numbers. Regarding graph three and the SL policy, it is possible to notice that the model has a high score for an entire day, which is related to the fact that the SL model uses the data from the previous day when the customer answered several calls. However, a policy that says to call every hour of the day is a bad policy. Finally, graph 4 represents the RL policy, where the model says that given the way the rewards are being calculated, it is always positive to make a call. However, in some states, a call was not made. This is because, as explained earlier, to maintain the same number of calls made by each policy, a threshold is used that indicates the value from which calls can be made. So, if the difference between the Q-values is lower than the threshold, the call will not be realized, which is the situation depicted in this episode.

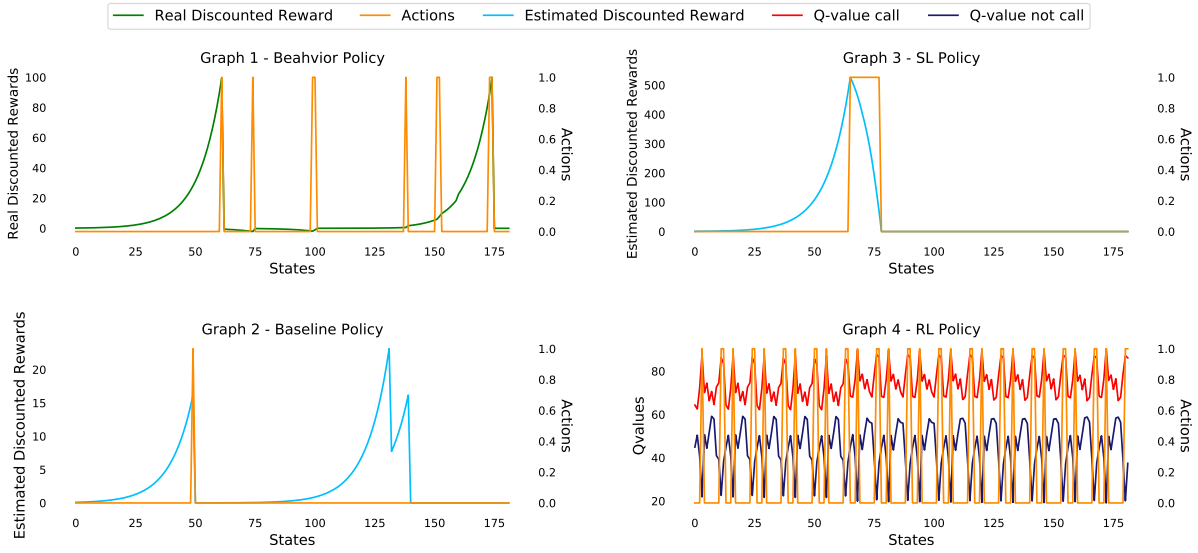


Figure 4.3: Illustration of the policies considered and their comparison concerning a certain episode.

Concerning the results presented in Figure 4.3, policies behave differently. This conclusion raises the question that results cannot answer imperatively: how different are the policies from each other? IS and WIS estimators allow us to answer this question. Table 4.4 shows the importance sampling and weighted importance sampling results for each of the policies.

-	**Baseline Policy**	**SL Policy**	**RL Policy**
IS	51,2686	0,0925	3,4072e-05
WIS	121,9365	100,4331	158,3302

Table 4.4: Results of importance sampling and weighted importance sampling.

It is possible to see that the RL policy is the most different one, while the SL Policy is the most similar to the baseline policy, regarding WIS. Although IS and WIS are approaches that can be used when the goal is to assess the differences between two policies (target and behavior policies), these metrics do not allow the extraction of results in terms of predictive performance because we do not know if the differences are a good or bad signal – this might vary depending on multiple factors. Therefore, it is necessary to have a different understanding of its meaning.

To obtain information about the models' performance and consequently know the impact of the difference between the policies, we analyzed their predictive performance. Figure 4.4 presents the results concerning an AUC-based evaluation.

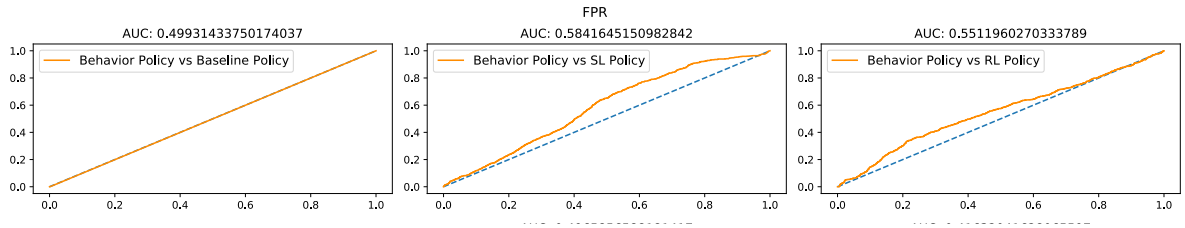


Figure 4.4: Agreement between the policies.

By analyzing the results, we conclude that the baseline policy has an AUC of roughly 50%. Such an outcome is reasonable because, as the model always acts randomly, it does not have the discriminatory ability to distinguish between positive and negative cases. Besides, and concerning the SL and RL policies, we observe that both have a positive outcome ($> 50\%$). However, we should note that results concerning the SL policy demonstrate a slight improvement of the RL policy.

Summary. This subsection presented an analysis and discussion of the obtained results. First, a comparison was presented on how policies behave in a given episode, and it was concluded that they have very different behaviors. Next, the obtained results of IS and WIS were discussed. It is concluded that RL Policy demonstrated the most different behavior w.r.t other policies. Such a conclusion led us to the last analysis, where the goal was to understand the impact of the difference between the policies in terms of predictive performance. Results were very similar between SL e RL policies. However, we acknowledge that the SL policy was slightly better. It is important to note our efforts focused on a single RL method. However, given that RL is a vast area where many different methods can be explored, future work should be carried out.

Chapter 5

Conclusion

This thesis's main goal was defined as an effort to optimize call centers operations by using reinforcement learning techniques to estimate the best time to contact each client. Based on our motivation and previous results concerning RL methods, our expectation was that it would improve the number of calls answered, customer satisfaction, and, consequently, the partner company's profit. However, the results obtained are inconclusive, although pointing towards a negative result concerning our expectations. Such a conclusion is framed because, with this work, it was not possible to prove that RL can be more advantageous than an SL-based solution. However, in any case, we should note that these results were obtained concerning a single iteration of experimental evaluation. Therefore, we are not aware of the magnitude of the variation in the results, which can be one reason for obtaining these results.

5.1 Future Work

The work presented in this dissertation opens various possibilities for several future research paths. The first corresponds to the developed model's operationalization, which allows an evaluation of the real results and compares the estimated results obtained by this work. As an off-policy evaluation was developed, we would like to realize a theoretical study about off-policy estimators and their results in estimated and real environments. This brings additional validation and further insights into the addressed questions and the proposed approaches.

Moreover, optimizing the model used in this work could be beneficial. Other algorithms could be tested, naturally, and an analysis of the hyper-parameters could be presented, or a study about safety policies, like TRPO, could be realized to have policies not as good in estimates but more robust.

Another possible line of future work could be changing the formalization of the problem, e.g., the action instead of being for each client at each hour – call it or not – it could be the indication of the hour of the next day that we should contact each customer. Or the reward is altered to be more in line with the business. To know the costs associated with a particular call and give rewards based on that.

Finally, another approach should be explored. Currently, we are using RL techniques to estimate the likelihood of a customer answer a call at each hour of the day, and based on that information, a heuristic is used to create a single dialing policy that is followed by the dialer. However, another approach could use the RL to create a dialing policy. This could be done using multi-agent policy actors, where multiple

agents are competing or working together in the environment.

Bibliography

Zeynep Aksin, Mor Armony, and Vijay Mehrotra. The modern call center: A multi-disciplinary perspective on operations management research. *Production and Operations Management*, 16(6):665–688, 2007. doi:10.1111/j.1937-5956.2007.tb00288.x.

Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004. ISBN: 0262012111.

Sercan O Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *arXiv preprint arXiv:1908.07442*, 2019.

Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *CoRR*, abs/1708.05866, 2017.

Athanassios Avramidis, Wyean Chan, Michel Gendreau, Pierre L'Ecuyer, and Ornella Pisacane. Optimizing daily agent scheduling in a multiskill call center. *European Journal of Operational Research*, 200:822–832, 02 2010. doi:10.1016/j.ejor.2009.01.042.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

Srinivas Bollapragada and Suresh Nair. Improving right party contact rates at outbound call centers. *Production and Operations Management*, 19:769 – 779, 08 2010. doi:10.1111/j.1937-5956.2010.01156.x.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Olivier Chapelle, Bernhard Schlkopf, and Alexander Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010. ISBN: 0262514125.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

Philippe Chevalier and Jean-Christophe Van den Schrieck. Optimizing the staffing and routing of small-size hierarchical call centers. *Production and Operations Management*, 17(3):306–319, 2008.

Ankit Choudhary. Best languages for machine learning in 2020! https://becominghuman.ai/best-languages-for-machine-learning-in-2020-6034732dd242, October 2019a.

Ankit Choudhary. A hands-on introduction to deep q-learning using openai gym in python. https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/, April 2019b.

Dennis Dietz. Practical scheduling for call center operations. *Omega*, 39:550–557, 10 2011. doi:10.1016/j.omega.2010.12.001.

David Dutton and Gerard Conroy. A review of machine learning. *The Knowledge Engineering Review*, 12: 341 – 367, 12 1997. doi:10.1017/S026988899700101X.

J. Engel. Polytomous logistic regression. *Statistica Neerlandica*, 42(4):233–252, 1988. doi:10.1111/j.1467-9574.1988.tb01238.x.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

João Gama, André Carlos Ponce de Leon Ferreira de Carvalho, Katti Faceli, Ana Carolina Lorena, and Márcia Oliveira. *Extração de conhecimento de dados: data mining*. Edições Sílabo, 2012.

Betsy S Greenberg and S Lynne Stokes. Developing an optimal call scheduling strategy for a telephone survey. *Journal of Official Statistics*, 6(4):421–435, 1990.

Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables. *arXiv preprint arXiv:1604.06737*, 2016.

Sean D. Holcomb, William K. Porter, Shaun V. Ault, Guifen Mao, and Jin Wang. Overview on deepmind and its alphago zero ai. In *Proceedings of the 2018 International Conference on Big Data and Education*, ICBDE '18, page 67–71, New York, NY, USA, 2018. Association for Computing Machinery. ISBN: 9781450363587. doi:10.1145/3206157.3206174.

Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 652–661. PMLR, 2016.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *CoRR*, cs.AI/9605103, 1996.

Sotiris Kotsiantis, I. Zaharakis, and P. Pintelas. Machine learning: A review of classification and combining techniques. *Artificial Intelligence Review*, 26:159–190, 11 2006. doi:10.1007/s10462-007-9052-3.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. *Batch Reinforcement Learning*, pages 45–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN: 978-3-642-27645-3. doi:10.1007/978-3-642-27645-3_2.

Pierre L'Ecuyer. Modeling and optimization problems in contact centers. In *Third International Conference on the Quantitative Evaluation of Systems-(QEST'06)*, pages 145–156. IEEE, 2006.

Esther Levin, Roberto Pieraccini, and Wieland Eckert. A stochastic model of human-machine interaction for learning dialog strategies. *Speech and Audio Processing, IEEE Transactions on*, 8, 02 2000. doi:10.1109/89.817450.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Charles Lewis and GD Fabbrizio. Dialogue strategy optimization with reinforcement learning in an at&t call routing application. In *Proc. AAAI Workshop Statist. Empirical Approaches Spoken Dialogue Syst*, pages 1770–1773, 2006.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Joseph Gonzalez, Ken Goldberg, and Ion Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, page 85, 2017.

Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341, 1949. PMID: 18139350. doi:10.1080/01621459.1949.10483310.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN: 00280836.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, 7:21, 2013.

David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Guido Rossum. Python reference manual, 1995.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

G. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*, 11 1994.

Douglas A. Samuelson. Predictive dialing for outbound telephone call centers. *Interfaces*, 29(5):66–94, May 1999. ISSN: 0092-2102. doi:10.1287/inte.29.5.66.

Leonard J Savage. The theory of statistical decision. *Journal of the American Statistical association*, 46 (253):55–67, 1951.

Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

Rosaria Silipo. Ensemble models: Bagging & boosting. =https://medium.com/analytics-vidhya/ensemble-models-bagging-boosting-c33706db0b0b, Mar 2020.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Edward L. Thorndike. Animal intelligence. *Psych Revmonog*, 8(2):207–208, 1911. doi:10.2307/2176958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.