UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Deep learning and semi-supervised methods for addressing learning with unlabeled data in telco package recommendation models

Luís Pedro Silva Santos

Mestrado em Ciência de Dados

Trabalho de Projeto orientada por:
Professora Doutora Márcia Afonso Barros

2023

Abstract

Recommender systems have seen an increasing adoption across many industries that aim in achieving a better relationship with the customer and increase their profits. Hence, the incorporation of recommender systems by telecommunication companies is being explored in to recommend offers that aligned with the customer's needs and, consequently, increase profits.

The following work project was developed for a telecommunication company [1]. Currently, the recommender system implemented in the organization is composed of a heuristic model and an adhesion's propensity model trained with data derived from outbound campaigns. These are campaigns where the operator presents the recommended offers to the client via phone-call.

Due to this scenario, tracking the offers that are presented to the customer is not possible, creating an uncertainty factor in labelling a refusal offer. Therefore, in this thesis, we propose two different strategies that aim to tackle the stated problem, a methodology based on Positive Unlabeled Learning and an approach established on recommender systems using the Two-Tower Model architecture. Concerning the Positive Unlabeled Learning, we test different methods, being the method PU-Bagging the one that achieves better results on the evaluation metrics and the A/B testing done on campaigns. Regarding the recommender system approach, we implemented a model following the Two-Tower Model architecture, in which each tower is composed og an autoencoder. Each tower aims to represent the users' features and items' features in a learned latent space. In this space, we compute the dot product between the clients and offers vectors and use this score to produce the ranking of offers to be recommended. In terms of results, we observe that the Two Tower Model attained a satisfactory performance on the recall@k metric and the A/B testing, which on the online tests recorded the best hit rate among the three groups being tested.

The main contributions of this thesis are testing and validating Positive Unlabeled Learning methods and the Two-Tower Model approach on real data.

Keywords: recommender systems, deep learning, semi-supervised learning, positive unlabeled learning,two-tower model

[1]For anonymity reasons, the company's name will not be revealed.

Resumo

Os sistemas de recomendação são cada vez mais usados em diferentes indústrias com foco em melhorar a sua relação com o cliente e aumentar os seus lucros. Assim, a inclusão de sistemas de recomendação por parte de empresas telco tem vindo a ser explorada com o intuito de produzir recomendações de ofertas telco alinhadas com as preferências e necessidades dos clientes, bem como aumentar o lucro da empresa. O presente projeto foi desenvolvido numa empresa de telecomunicações [2]. No que diz respeito ao funcionamento do sistema de recomendação, este é direcionado para clientes empresariais e tem como objetivo recomendar a melhor oferta aos clientes. Este é composto por uma fase de geração de possíveis ofertas a serem recomendadas ao cliente. Deste universo de ofertas candidatas é empregue um modelo para realizar uma seleção e propor a melhor oferta ao cliente. Atualmente, o sistema de recomendação implementado na organização é composto por um modelo heurístico e analítico. Havendo a ambição de que gradualmente, o modelo analítico substitua o modelo heurístico. Relativamente ao modelo heurístico, este funciona com base em regras de negócio. Contrariamente, a abordagem analítica corresponde a um modelo de propensão à adesão treinado com dados vindos de campanhas outbound em que o operador apresenta as propostas ao cliente por via telefónica. Devido a este cenário, o rastreamento das ofertas apresentadas ao cliente não é possível de ser feito, resultando numa incerteza quanto ao mapeamento de uma recusa.

O principal objetivo deste projeto é melhorar o sistema de recomendação da empresa. Como tal são propostas duas estratégias que visam resolver a incerteza subjacente ao mapeamento de uma recusa. Portanto apresentamos uma abordagem focada em *Positive Unlabeled Learning* e uma metodologia baseada em sistemas de recomendação alicerçada no paradigma modelo de duas torres. No que diz respeito à abordagem de *Positive Unlabeled Learning*, a aprendizagem foi feita com dados provenientes de campanhas de *outcomes*, constituídos por variáveis que caracterizam o cliente e a oferta mapeada para o cliente. Esta abordagem consiste em treinar um classificador binário com um conjunto de dados positivos e desconhecidos. Estas observações desconhecidas podem ser tanto positivas como negativas. Nesta abordagem foram testados diferentes métodos de *Positive Unlabeled Learning*, nomeadamente o *Spy Method* e *PU-Bagging*. O primeiro método consiste em retirar uma amostra de observações positivas e mudar a etiqueta das mesmas para desconhecidos, a estas observações denominados de espiões, é treinado um classificador binário e posteriormente são selecionados das observações desconhecidas como negativos fiáveis aqueles que contêm uma probabilidade de serem positivos inferior ao primeiro decil da probabilidade atribuída aos espiões. Quanto ao PU-Bagging, este consiste em treinar n classificadores com as observações positivas e n amostras de dados identificados como desconhecidos. Foram realizados diferentes testes sendo que o PU-Bagging registou os melhores resultados nas métricas *offline Area under the ROC*

[2]Por razões de anonimidade, o nome da empresa não será revelado.

Curve e *PUF-Score*. De igual modo, foram feitos testes A/B em que se selecionou a taxa de adesão como o *Key Performance Indicator* definido para avaliar o desempenho do modelo, este registou a melhor taxa de adesão de 3,41%, face à 2,63% obtido com o modelo analítico em produção e 2,58% com o modelo heurístico. Por outro lado, foi também adotada uma abordagem de sistemas de recomendação. Nesta metodologia, a aprendizagem é feita com base nas alterações de portfólio do cliente. Ademais, a recomendação de ofertas telco diferencia-se de outro tipo de recomendações de produtos, dado que estas agregam um conjunto de serviços. Como tal, a definição adotada para identificar um produto consistiu na utilização dos upgrades propostos por cada oferta, conjuntamente com um intervalo de diferença monetária percentual entre o pacote subscrito pelo cliente e o pacote proposto. Assim, o modelo aprende com qualquer cliente que efetuou uma nova subscrição a um pacote telco proveniente de qualquer canal de venda na qual desta nova subscrição não resultou uma perda de valor ou desativação de serviços. Assim, o modelo desenvolvido consistiu um modelo de duas torres. Cada torre diz respeito a uma *feedforward neural network*, especificamente um *encoder* em que cada torre tem como objetivo aprender um espaço latente do cliente e das ofertas telco. Com base nestes espaços latentes aprendidos, é calculada a similaridade, recorrendo ao produto interno entre pares cliente-oferta. Posteriormente os *scores* obtidos com o produto interno são usados para o ranking das ofertas a recomendar. Para efeitos de comparação, também foi desenvolvida uma *baseline* baseada numa abordagem de *Content-based*. Em termos de resultados verifica-se que o modelo de duas torres tanto na métrica *recall@k* como nos testes A/B obteve uma performance satisfatória. Pelo que ao nível do *recall@1* obteve um valor de 7,4% face a 3,4% obtidos com o *Content-based*. Em termos de taxa de adesão o modelo de duas torres obteve 10,57% face a 9,00% atingidos pelo modelo de propensão à adesão e 8,91% obtidos com o modelo heurístico.

Para trabalho futuro destaca-se a necessidade de experimentar técnicas de *transfer learning*, visto que o *dataset* de treino utilizado é de dimensão reduzida. De igual modo, incorporar abordagens de *Sequential-Aware Recomender systems*, poderia trazer mais ganhos de *performance* ao modelo. Futuramente, é proposto testar abordagens de *multi-recommender systems*, dado que objetivo da empresa não é somente recomendar a oferta mais alinhada com as preferências e necessidades do cliente, mas é também conseguir maximizar o lucro, tendo em conta diferentes níveis de período de fidelização.

Como principais contribuições desta tese elencamos a validação e a experimentação das abordagens anteriormente referidas num cenário de dados reais aplicado à indústria de telecomunicações. Bem como, o objetivo traçado e a melhoria do sistema de recomendação atual.

Palavras chave: sistemas de recomendação, aprendizagem profunda, aprendizagem semi-supervesionada,aprendizagem positivo-desconhecido , modelo duas torres

Acknowledgments

I would like to express my gratitude to all the people who helped me develop and conclude this project. Thus, I would like to thank Doutora Márcia Barros for the supervision provided and shared advice through this year. Then, I would like to thank the organization where I developed the work. Specifically, I want to express my gratitude to Catarina Freitas, who guided me through the entire process, such as teaching me best practices and sharing valuable advice that greatly contributed to the outcome of this project. Also, I want to thank all the B2B Advanced Analytics and NBO teams that welcomed me, made me feel integrated, and were always available to clarify any doubts I had.

Furthermore, I want to thank my family for all the support provided and for helping me to pursue this path. Also, I would like to thank all my friends who always had kind and uplifiting words, which made me do my best.

To all of them thank you very much.

Contents

List of Figures

List of Tables

List of Abbreviations

AUROC Area Under the Receiver Operating Characteristic Curve

B2B Business-to-Business

CAE Código Atividade Económica

CV Cross-Validation

DBN Deep Belief Network

DCN Deep Cross Network

MD Mobile Data

DNN Deep Neural Network

EDA Exploratory Data Analysis

EV Expected Value

GB Gigabyte

i.i.d independent and identically distributed

FI Fixed Internet

KNN K-Nearest-Neighbors

MBPS Megabits per second

MLP Multilayer Perceptron

MSE Mean Squared Error

NBO Next Best Offer

PCA Principal Component Analysis

PDF Probability Density Function

PU Positive Unlabeled

ReLU Rectified Linear Unit

RGU Revenue Generating Unit

RMSE Root Mean Squared Error

SHAP SHapley Additive exPlanations

SSL Semi-Supervised Learning

TV Television

B2C Business-to-Consumer

MV Mobile Voice

GAN Generative Adversarial Networks

SAR Selected at Random

SCAR Selected Completely at Random

Chapter 1

Introduction

This chapter aims to present the motivation of the project and its objectives, the methods used, the main contributions of this thesis, and the organization of the document.

1.0.1 Motivation

The project developed took place at a telecommunication company, in the department of Advanced Analytics in the Business-to-Business (B2B) Segment, on the team Next Best Offer (NBO). The NBO B2B squad developed a recommender system of telecommunication bundles for their corporate clients. This system is composed of a heuristic model that produces recommendations based on scripted rules, and an analytical model. From a pool of possible offers assigned to a client, the goal is to be able to select and recommend proposals that are more aligned with the preferences and needs of the customers. Consequently, this increase adhesion and revenue for the company. Concerning the analytical model, the data used to train the model derives from commercial outbound campaigns, which consists of having the operators reaching the customers via phone calls and propose a set of possible offers. This group of offers are based on the recommendations produced by the model. However, the operator is, also, allowed to present other offers that do not belong to the set of recommendations. To date, it is not possible to track what offers were actually made during the call. However, by default, we assume that the operators make the model recommendations. Then, we check changes in the portfolio to find our adhesions and refusals. The current analytical model is a binary classification that learns that a non-adhesion of the offer is considered a refusal. However, due to the problem stated above, this strategy induces bias in labeling a refusal. Therefore, we have a binary classification model learning with noise regarding with the target.

1.0.2 Objectives

The main objective of this work is to improve the current recommender analytical model. Hence, the purpose of this project is to:

1. Test and experiment with semi-supervised learning approaches, specifically, Positive Unlabeled (PU) Learning methods to deal with the uncertainty dimension presented in the data;

2. Test and experiment with a recommender system approach based on the Two-Tower Model methodology;

3. Evaluate both methods aiming to improve the adherence rate, also known as hit rate.

1.0.3 Methods

This work presents two different methodologies which aim to tackle the previously mentioned problem. The PU Learning approach focuses on learning with positive and unlabeled observations, being considered a type of semi-supervised learning technique. Hence, this method aims to test a different learning way while keeping the logic of the propensity model used on the NBO.

As for the recommender system approach, the recommender system will learn from all the portfolio upgrades that occur across all the selling channels, such as stores and inbound of the company. This allows us to overcome the problem stated before, as we are no longer dependent on a label expressing if a customer has accepted an assigned offer. In this method we have a multi-class classification task, where each label corresponds to each purchased item by the client. Being a recommender system, we focus on learning the similarities between users and items to produce the best recommendations. The selected method is based on a deep learning approach, which applies a Two-Tower Model architecture. Each tower is an encoder, to learn the latent representations of the user's vector and the item's vector. These latent vectors will be used to compute the dot- product between user-item vectors. This computed score will be used as a recommendation score to rank the top items to be recommended to the customer.

1.0.4 Main Contributions

The expected contributions of this project are:

1. Test and validate the previously referred methods on real-world data;
2. Improve the analytic model, which will translate into an increase in the hit rate;
3. Test and present the potential of a recommender system approach in a telecommunication setting.

1.0.5 Document Organization

The present thesis is organized in the following way:

1. Chapter 2 - introduces the theoretical concepts that support the developed work. It presents a general overview of the concepts of machine learning and model ensembles. Then, it provides a more in-depth explanation of semi-supervised learning and PU Learning. Followed by an exposition of the concepts of deep learning, autoencoders, and recommender systems;
2. Chapter 3 - presents the state-of-the-art the academic fields that this thesis addresses;
3. Chapter 4 - explains in detail the methodologies adopted on this project alongside an exploratory data analysis of both datasets used in each approach. Also, it presents how the NBO project works, including its models, with special detail on the analytical model;
4. Chapter 5 - shows the results obtained with both methodologies and discusses them;

5. Chapter 6 - concludes the project and presents ideas for future work.

Finally, a Gantt Chart of the developed activities of this project is presented in Figure 1.1.

Chapter 2

Background

In this chapter, we present the theoretical background that supports the developed work.

2.1 Recommender Systems

The digital world has become a hub of user interactions and behaviors, with highlighted interactions between consumers and products. This data can be used to infer the interest of a customer, the fundamental idea of a recommender system. Thus, the goals of a recommender system can be defined by four dimensions [1]:

- A recommender system must recommend items that are relevant to the user. Hence, a product must be of the interest of the user. This is fundamentally the main operational goal of a recommender system. However, alone is not sufficient for a recommender system to thrive;
- A recommender system must be capable of recommending new items to the users, never seen before. The novelty element is key to the success of the business;
- A notion related to novelty is the concept of serendipity, wherein the products recommended are unexpected. This concept distinguishes from novelty, because it provides more than a discovery, it adds a surprise factor to the recommendations;
- Finally, the recommender system must be capable of generating a pool of recommendations that are diverse, increasing the likelihood that the user likes or is interested in at least at one item.

Furthermore, in a recommender system scenario we can collect data regarding the interactions that the user had with it. Consequently, this type of data can be classified as explicit or implicit feedback. One the one hand, explicit feedback regards the direct and unequivocally interests of the user, such as ratings, likes, or comments. On the other hand, implicit feedback refers to the type of feedback that is inferred about the users' preferences, such as viewing a video or clicking on an ad [15]. Therefore, explicit feedback assumes to be a much more reliable and better source of information. Nonetheless, in most applications and scenarios the only information regarding the user's preferences and tastes is the implicit feedback since there exist fewer explicit datasets due to requiring a direct action or input from the user. Yet, implicit feedback datasets have some challenges, such as the nonexistence of negative feedback. With this type of feedback, we can only collect information

about the users' behavior and infer which items that they will likely enjoy. But we cannot do the same reasoning for the items they will not like [12]. In consequence, implicit feedback is inherently noisy, because by observing the users' behavior, and we can only guess the true intentions of their actions, we cannot infer with certainty the users' preferences [12].

Regarding the main approaches to recommender systems we have collaborative filtering, content-based, and hybrid approaches.

2.1.1 Collaborative Filtering Approach

In Collaborative Filtering the main idea is to leverage the data regarding the user-item interactions. Typically, in collaborative filtering, we are dealing with a rating matrix, that represents the recorded user-item interactions. This matrix can either incorporate user-item interactions collected through explicit or implicit feedback. Generally, this matrix is sparse, hence the objective of collaborative filtering is to output the unspecified ratings. This is done assuming that similar users tend to like similar items, leading to high correlations between the observed user-item interactions [1].

Moreover, there are two approaches commonly applied in collaborative filtering, which are the memory-based methods and model-based methods. Memory-based methods, also known as neighborhood-based, were among the earliest collaborative filtering algorithms, in which the ratings of user-item combinations are predicted based on their neighborhoods. The selection of neighborhoods is based on a similarity approach, such as the cosine similarity measure. When compared with the model-based methods, the memory-based approach has the advantages of being simple to implement and the recommendations are often easy to explain.

Model-based methods use machine learning and data mining methods in the context of predictive models. They showed to be adequate for collaborative filtering problems, as these problems can be viewed as a special case of classification or regression problems. Therefore, the use of machine learning algorithms can be exploited offering some advantages over the more simple and early memory-based methods:

- They are space-efficiency, as the learned model is much smaller than the original ratings matrix;
- They are faster at the training step and the prediction phase;
- The summarizing approach of model-based methods can help to avoid overfitting.

One specific type of model-based method is the Latent Factor/Matrix Factorization model. These methods focus on using dimensionality reduction strategies to estimate the rating matrix. The idea is based on the fact that in the utility matrix, there exist highly correlated rows, meaning similar users. Hence, the data matrix can be well approximated by a low-rank matrix or a latent matrix. By finding a coordinate system to represent a reduced data matrix, the pairwise correlations between dimensions are removed and the completely specified representation can be robustly estimated from an incomplete data matrix [1].

Thus, the rating matrix $m \times n$ of *rank* k, with $k \ll min\{m, n\}$ can be approximated by the product of rank-k factors:

$$R \approx UV^T. \tag{2.1}$$

Being R the rating matrix, U a matrix $m \times k$ and V a matrix $n \times k$, where each column of U or V is denominated as latent vector or component and the ith row of U, u_i, or V, v_i is referred to as latent factor. Consequently, each latent factor of the matrix U can be viewed as a user factor related to the its affinity towards k concepts in the rating matrix [1]. To illustrate this, let us consider a utility matrix 4 × 3, where the rows represent users, and the columns represent books. This matrix is filled with values of 0 and 1, being a scenario of implicit feedback.

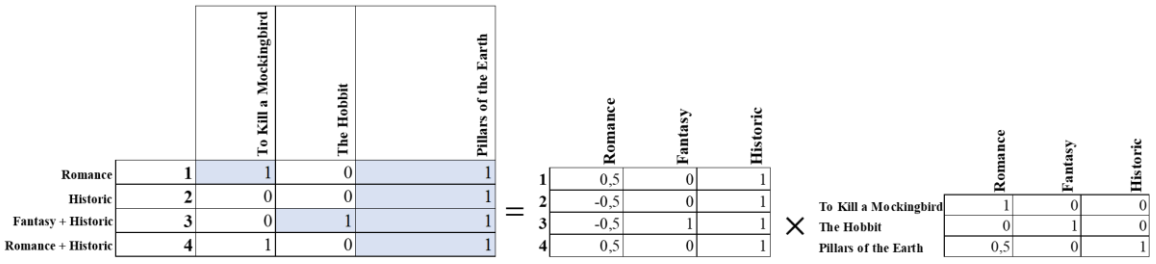


Figure 2.1: Example of rank-3 matrix factorization.

In Figure 2.1, we have three concepts related to the book genres the users have read. As stated before, we can interpret the latent factors as affinities for the captured concepts. Therefore, each rating could be viewed as [1]:

$$r_{ij} \approx \sum_{s=1}^{k} u_{is} v_{js}. \tag{2.2}$$

As, u_{is} represents the affinity of the user *i* to the concept *s*, and v_{js} represents the affinity of the item *j* to the concept *s*.

2.1.2 Content-based Approach

In the content-based setting, the items' description is used to produce the recommendations. Such features can be the title of a movie, the cast members and the movie's synopsis. Hence, the ratings and the user's buying behavior are combined with the content information available in the items. Therefore, the item descriptions labeled with ratings are used as training data to create user-specific classification or regression modeling problems. Moreover, content-based methods have the advantage of making recommendations for new items, when sufficient rating data are not available for that item. In other words, the content-based approach tries to find items that are similar to the ones that the user liked in the past. This similarity is based on the features of the items and not on the users' ratings[1]. To find the similarity between items, we can employ methods based on distance such as the K-Nearest-Neighbors (KNN) or more complex machine learning algorithms. Nonetheless, content-based methods have the disadvantage of providing obvious recommendations, reducing the diversity of recommended items. Moreover, these methods are not effective at providing recommendations for new users.

Content-based methods find most of their applicability in settings with text-rich information and unstructured domains, such as the recommendation of web pages, products and news.

2.1.3 Hybrid Approach

Hybrid models aim to tackle the problems that arise when using a collaborative filtering or content-based approach. The idea is that by mixing and combining different recommendation approaches, the gains will be greater than when using only one approach [1].

According to [4], the hybrid systems can be classified according to the combination methods that is employed:

1. Weighted: Refers to the type of hybrid models that the recommendation score is produced by an ensemble of recommendation models. For instance, the final score combines the scores of a collaborative and content-based models. We can assign weights to each model as a form of attributing a degree of importance;

2. Switching: Consists in changing between recommendation models according to a given criterion, one might employ a content-based approach in the first stage and then switch to a collaborative filtering approach to produce the final recommendation;

3. Mixed: The recommendations are produced by several models. This differs from the Weighted approach, because we are not combining in a final score, but rather outputting different recommendations produced by different models. This might work well for cases where we expect to produce composite recommendations;

4. Feature Combination: With this technique, we are combining different features from different data sources, one could use the collaborative data and add that to the data applied to the content-based model;

5. Cascade: In this we have a technique similar to a boosting approach, where a recommendation model refines the recommendations from a previous model;

6. Feature Augmentation: In this method, the output of a recommender system will be used the input feature of another recommender system.

7. Meta-level: This technique consists of using a model applied by a first recommender system as input to another system. This differs from Feature Augmentation, as we are using a learned model to produce features from our next model. Here, we are using the whole model as input. This is similar to a transfer learning approach used on neural networks.

Thus, the hybrid recommender systems are used either to leverage the power of multiple data sources or/and to improve the performance of the overall recommender system. Consequently, hybrid recommender systems have the goal of leveraging the strengths of different recommendation approaches to minimize the weaknesses of each recommendation model.

2.1.4 Evaluation of Recommender Systems Setting

Evaluating a recommender system can be performed via offline with historical datasets or online. Both approaches offer advantages and disadvantages and often a combination of both proves to be a good strategy to measuring the performance of our recommender system [1].

The offline evaluation methods using historical data are the most popular methods to test recommender systems. They consist of using evaluation metrics that measure the accuracy of the recommender systems over selected datasets like the Movielens or the Netflix Prize. Examples of such metrics can be the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and recall@k, among others. Nevertheless, these methods do not allow us to perceive the actual behavior of the user reacting to the recommender system in future situations. Moreover, this kind of measures do not capture the serendipity and novelty dimensions that should compose any recommender system.

Regarding the accuracy metrics, they focus on evaluating the prediction accuracy of the estimated ratings or the accuracy of top-k ranking predicted items by a recommender system. Considering a matrix $\boldsymbol{R}$, where r_{uj} refers to the rating of a given user $\boldsymbol{u}$ to an item $\boldsymbol{j}$ and $\hat{r}_{uj}$ refers to the corresponding rating predicted, and $|E|$ corresponds to the cardinality of the rating matrix. We have $e_{uj} = \hat{r}_{uj} - r_{uj}$ as the equivalent error for the user $\boldsymbol{u}$ for the item $\boldsymbol{j}$. So, we can leverage this for all ratings and compute the MSE, which is given by the formula:

$$MSE = \frac{\sum(u,j)\epsilon E^{e_{uj}^2}}{|E|}. \tag{2.3}$$

For the case the RMSE we take the root of the MSE:

$$RMSE = \sqrt{\frac{\sum(u,j)\epsilon E^{e_{uj}^2}}{|E|}}. \tag{2.4}$$

The recall@k consists of the proportion of items the user interacted with, found on a list of size $\boldsymbol{k}$ composed by the top-$\boldsymbol{k}$ recommendations of the recommender system. So, we can define the recall@k as:

$$recall@k = \frac{number\ of\ recommended\ items\ @k\ that\ are\ interacted}{number\ of\ interacted\ items} \tag{2.5}$$

Intuitively, this metric allows us to evaluate the quality of our top-$\boldsymbol{k}$ recommendations made for a given user and how well are they aligned with the user's preferences.

Concerning the online evaluation of the recommender system, one common approach is using the use of A/B testing. This technique measures the direct impact of the recommender system on the end user, by using the conversion rate. The conversion rate refers to the frequency a user selects a recommended item. The test is often used to compare two algorithms, where we randomly selected a sample of users for both the control group (A) and the treatment (B) [3]. According to [1] the selection of users should be as similar as possible between the two segments. At the end of the process, we compare the conversion rate obtained in each group. One major concern with this type of evaluation is the sample size. To have statistically significant results results, we must have

a large sample of users interacting with our recommender systems. Otherwise, analyzing over small sample sizes may yield noisy results and, consequently, incorrect analysis.

2.2 Machine Learning - General Overview

Machine Learning can be encompassed as a field of Artificial Intelligence. According to [10] it is the science (and art) of programming computers so they can learn from data. This definition highlights an important concept of this field, which is the learning process. Hence, this field that aims to create algorithms capable of producing models that can generalize to unseen scenarios. In other words, the models are capable of applying what they learn from the training data to unseen data.

We can have four types of learning settings, supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.

- Supervised Learning

  In the Supervised Learning context, the data used to train the model has the corresponding labels, each label can be viewed as the target response. The tasks typically performed under supervised learning are classification and regression.

- Semi-Supervised Learning

  In Semi-Supervised Learning, we have both labeled data and unlabeled data. In this scenario, the strategies employed involve using supervised and unsupervised learning techniques. Nonetheless, a dedicated chapter on this topic will be presented due to the importance of this theme for the project developed.

- Unsupervised Learning

  We can also have a situation of Unsupervised Learning, in which no data is labeled, the focus of this setting is to let the algorithm learn the patterns and relations of the underlying data. Thus, tasks such as clustering, dimensionality reduction, and association rule mining are performed under unsupervised learning contexts.

- Reinforcement Learning

  Finally, in the case of Reinforcement Learning, which distinguishes from the other approaches, we have an agent that performs actions in a defined environment. In the case of actions that are aligned with the defined goals, it is provided a reward. Otherwise, it is penalized in the form of a negative reward. The goal is to learn the best strategy, called policy, to collect most of the rewards over time.

2.3 Model Ensembles

Model ensembles refer to a group of machine learning techniques that involve combining more than one model. This combination of models focuses on constructing diverse predictive models based on reweighted or resampled training data. Then, it applies techniques of voting or weighted voting for classification problems, or averaging for regression tasks, which combine their predictive power [9].

2.3.1 Random Forest

One of the most common and powerful ensemble methods is the Random Forest. These types of models rely on a method called Bagging, an abbreviation for bootstrap aggregating. It consists of training n models with n random samples from the training data, sampled uniformly with replacement. This will create diverse models that will compose our ensemble.

Bagging offers some advantages over simple methods, such as it reduces the variance of predictions, this is especially useful for models that suffer from high variance, such as decision tree models. Thus, bagging techniques are more robust to overfitting than a decision tree. It provides a dataset that can be used to assess the performance of the model ensemble, called out-of-bag. This refers to the observations that were not used to train the model. Usually, the performance obtained in the out-of-bag set correlates well with the performance of the model with other unseen data [18].

Based on bagging it surges the Random Forest, which is a model ensemble based on bagged trees. With this aspect, we perform sampling over the training data and the feature space. Thus, each tree will be trained with a random sample described by a random sample of columns [9]. Sampling the column space aims to reduce the problem of tree correlation, which refers to having trees that share similar structures, a common problem with bagged trees. Thus, introducing the sampling of features adds another random factor that will force the trees to have different structures. The selection of features is done at the split level of the tree. Hence, the tree correlation will be lessened [18].

Due to this property of the Random Forest, this algorithm tends to be used for feature selection, and one of the most common methods is called Boruta. The Boruta algorithm according to [19] consists of the following steps:

- Copy the original features and the replicated variables are denominated shadow features.
- Train several Random Forests with the original and shadow features, where the replicated features have, their values being randomized to remove correlation with the target.
- Compute the feature importance and a feature is considered important for a single run if its importance is higher than the maximal importance of all shadow features.
- Perform two-sided equality test over all attributes where the null hypothesis states that the importance of a variable is equal to the maximal importance of the random attributes. We count the number of times a variable recorded an importance superior to the shadow features. A feature is considered important when its number of hits is higher than the expected value and unimportant when it is significantly lower than the expected value.
- Remove the unimportant variables and features that were neither rejected nor accepted, and considered considered undetermined.

2.3.2 Gradient Boosting Trees

Another ensemble technique is boosting, which is composed of weak learners, which are models in complexity in terms of their hyperparameter space. They are trained sequentially way, with the goal of reducing the error obtained by the previous model [10].

A popular boosting method is Gradient Trees, this model ensemble seeks to produce an additive model capable of minimizing a loss function. Therefore, the model has predictors that try to fit the next model to the residual error made by the previous predictor. Thus, at each iteration, the residuals of the previous model will be the target response for the next model. Thus, the final model can be summarized by the following formula:

$$f_M(x) = f_0(x) + \Sigma_{m=1}^{M} \eta \cdot h_m(x), \tag{2.6}$$

where $f_0(x)$ refers to the predictions of the first model, M refers to the total number of estimators, and η corresponds to the learning rate, a positive real number, that has a shrinkage effect. This means that it reduces the effect of subsequent models to avoid overfitting [18], and $h_m(x)$ is equal to the subsequent predictions of the models that were fitted with the residuals of the previous predictors.

2.4 Semi-Supervised Learning

In a Semi-Supervised Learning (SSL) scenario we have unlabeled and labeled data. Therefore, with a data set D containing a subset of labeled data, D_l, and a subset of unlabeled data, D_u. Instead of only using the labeled data, we aim to leverage information contained in the unlabeled examples to train a better-performing model [26].

The SSL models only work under some assumptions about the structure of the data, which are:

- Smoothness Assumption

  If two observations belong to the same cluster, a high-density region of the input space, then their corresponding outputs need to be close. On the opposite, if two points are separated by a low-density region, the outputs must be distant from each other.

- Cluster Assumption

  It states that points belonging to the same cluster are likely to be of the same class. Therefore, this assumption can be viewed as the low-density separation assumption: The decision boundary should lie in the low-density regions.

- Manifold Assumption

  The data can be mapped on a low-dimensional manifold, allowing it to solve a simplified version of the SSL task regions.

SSL methods are categorized into the following:

- Consistency Regularization

If a realistic perturbation was applied to the unlabeled data points, then the prediction should not change significantly. So, there is a function f_θ capable of giving consistent predictions for similar data points. Hence, given an unlabeled data point $x \in D_u$ and its perturbed version $\hat{x}_u$, the objective is to minimize the distance between the two outputs, $d(f_\theta(x), f_\theta(\hat{x}))$.

- Proxy-label Methods

  These methods use a trained model on the labeled set to produce additional training examples by labeling instances of the unlabeled set based on some heuristics. An example of proxy-label methods is self-training, where a model is first, trained on labeled data. Then, iteratively, a portion of the unlabeled data is labeled using the trained model and added to the training set for the next training iteration. Also, co-training is another example of a proxy-label method, where a data point x is represented by two conditionally independent views $v_1(x)$ and $v_2(x)$, and each view is sufficient to train a good model. Then, two models are trained on each specific view on the labeled set D_l. An unlabeled point is added to the training set if the confident score is higher then a defined threshold.

2.5 Positive-Unlabeled Learning

PU Learning is a variant of binary classification where the training data consists of positive and unlabeled examples. Therefore, it assumes that an unlabeled observation could belong to either a positive or a negative class. PU learning naturally arises in many applications. For instance, the absence of a patient diagnosis of a patient is not indicative of the absence of the disease, or the lack of a film review by a user does not mean that the user has not liked the movie [2].

In PU learning, the goal is the same as in binary classification-training a classifier to distinguish positive from negative examples. The difference lies in the learning phase, we only have access to some labeled positive examples and unlabeled observations, that can belong to either class. Hence, PU dataset is represented as a set of triplets (x, y, s), being x a vector of features, y the class and s indicating if it is a labeled or unlabeled example. It is worth mentioning, that the class y is not observed, but some conclusions can be derived from the labeled s. If an observation received the labeled 1, $s = 1$, then it belongs to the positive class, $P(y = 1|s = 1) = 1$, but an example with $s = 0$, it can belong to either class.

The PU data can originate from a single training set scenario, where the positive and unlabeled data examples belong to the same dataset, which is an i.i.d sample from the real distribution. In contrast, the case-control scenario assumes that the positive and unlabeled examples come from two independent datasets, and the unlabeled dataset is an i.i.d sample from the real distribution.

The observations to be labeled are selected according to a probabilistic labeling mechanism. Thus, each positive example x will have a probability of being selected, called the propensity

score, $e(x) = P(s = 1|y = 1, x)$. From this, results a biased version of the labeled distribution of the positive distribution given by the following Probability Density Function (PDF):

$$f_l = \frac{e(x)}{c} f_+(x), \tag{2.7}$$

where $f_l(x)$ and $f_+(x)$ correspond, respectively, to the PDFs of the labeled and positive distributions. The constant c refers to the label frequency, which is the fraction of positive examples that are labeled, $c = \mathbb{E}_x[e(x)] = P(s = 1|y = 1)$:

$$c = P(s = 1|y = 1) = \frac{P(s = 1, y = 1)}{P(y = 1)} = \frac{P(s = 1)}{P(y = 1)}. \tag{2.8}$$

In the situation of a single-training-set scenario:

$$c = \frac{P(s = 1)}{\alpha}. \tag{2.9}$$

The process of learning with PU data is not a trivial task, due to the uncertainty associated with unlabeled examples. In fact, an unlabeled example corresponds to an observation, which was not selected by the labeling mechanism. Thus, there arises the need to define assumptions regarding the labeling mechanism and the class distributions.

- Selected Completely At Random

  The Selected Completely at Random (SCAR) assumption lies at the basis of most PU learning methods. It assumes that labeled examples are selected completely at random, regardless of their attributes, from the positive distribution. Therefore, the propensity score is constant, which is equal to the label frequency. Hence, under the SCAR assumption, the probability for an example to be labeled is directly proportional to the probability for an example to be positive. This property enables the use of non-traditional classifiers, being classifiers that predict $P(s = 1|x)$, which are learned by considering the unlabeled examples as negative, $P(s = 1|x) = cP(y = 1|x)$.

- Selected At Random

  The Selected at Random (SAR) assumption, is the most general assumption regarding the labeling mechanism. It states that the selection of positive examples to be labeled depends on their attributes' values, being defined by the propensity score, $e(x) = P(s = 1|x, y = 1)$.

Regarding the data distribution assumptions, they are:

- Negativity

  The negativity assumption, also known as the closed-world assumption, states that all unlabeled examples belong to the negative class. This enables the use of standard supervised machine learning methods. However, this assumption does not hold in practice.

- Separability

  It implies that the two classes of interest are naturally separated. Therefore, there exists a function f in the regarded hypothesis space that maps the negatives and positives examples into values that are lower or higher to a threshold τ.

- Smoothness

  If two observations x_1 and x_2 are similar, then, $P(y = 1|x_1)$ and $P(y = 1|x_2)$ will also be similar.

Moreover, assumptions about the class prior are needed to make it identifiable. Therefore, the separable classes entail that the positive and negative distributions do not overlap. On a less restrictive side, the Positive subdomain only requires a subset of the instance space defined by the partial attribute assignment to be purely positive. A more general version of the positive subdomain assumption is the Positive function. It states that the subdomain can be defined by any function instead of being limited to partial variable assignments. Finally, the irreducibility assumption assumes that the negative distribution cannot be a mixture that contains the positive distribution.

For measuring performance in PU scenarios, computing the standard evaluation metrics, such as the precision or the F1-score, is not an obvious task. The most commonly used metric in PU Learning is a metric similar to the F1-score, which is defined as:

$$F_1(\hat{y}) = \frac{r^2}{P(\hat{y} = 1)}, \tag{2.10}$$

where r refers to the recall, under the SCAR assumption, this is equal to $P(\hat{y} = 1|s = 1)$, and it measures the model's ability to identify correctly the positive samples. Thus, the recall can be estimated using the set of positive labeled examples P_l, and the correctly predicted labeled examples as positive, TP_l, hence [14]:

$$\frac{TP_l}{|P_l|} = \frac{1}{|P_l|} \sum_{x \epsilon P_l} f(x) = \frac{\sum_{s=1} \hat{y}}{\sum_{s=1} s}. \qquad (\tag{2.11}$$

The Formula 2.10 ensures that higher values happen when both, the precision and recall are also higher, similar to the standard F1-score. Thus, we can interpret this modified F1-score as a measure of the model's accuracy.

Nonetheless, standard evaluation metrics can be used by leveraging the SCAR Assumption, which allows to estimate the class prior. By doing this, it is possible to estimate the size of the positives on the dataset and latent positives on the unlabeled dataset. Due to the SCAR assumption, we can assume that the rank distributions of labeled and unlabeled positives should be similar. Therefore, it is possible to compute the total number of positive samples below or above a given rank, giving the necessary information to build a confusion matrix.

The methods used in PU Learning follow a logic of incorporating mechanisms to deal with the uncertainty in learning with positive and unlabeled data.

- Two-Steps Techniques

  These methods consist of, in the first step identifying the reliable negative examples and, in the second step, training a model with the labeled positive and reliable negatives.

Therefore, the two-step technique assumes the existence of the separability and smoothness assumptions on the data distribution. Regarding the first step, identifying the reliable negatives means analyzing their similarity with the positive examples and selecting the most different. Hence, several methods of the first step apply distance techniques [2].

- Bias learning

  The biased PU learning methods treat the unlabeled examples as negative examples associating a constant value of noise, making use of the SCAR assumption. The noise can be taken into account by placing higher penalties on misclassified positive examples or tunning hyperparameters based on an evaluation metric that is suitable for PU data.

- Weighted approach to PU learning

  In this approach, weights are assigned to each unlabeled sample. These weights represent the likelihood that an observation x belongs to the positive or negative set. Then, a standard learning algorithm uses the weighted unlabeled samples as either constantly or weighted negative samples or weighted positive and negative examples concurrently [13].

- Methods that incorporate the class prior

  These types of techniques consist of using the class prior, and leveraging the SCAR assumption. There are three types of methods, postprocessing, preprocessing and method modification. The postprocessing approach refers to the training of a non-traditional probabilistic classifier by considering unlabeled data as negative and the output probabilities are modified. Inversely, the preprocessing methods focus on changing the data set by using the class prior. Finally, the method modification approach incorporates the class prior on the training stage [2].

- PU-Bagging

  Model ensemble techniques have proved to be highly reliable and achieved good performances, being Bagging one of them. Thus, extending the benefits of this technique to a PU Learning setting [25] presented a bagging method where we sample from the unlabeled dataset. The goal of this approach is to leverage the capabilities of a model ensemble, by having each predictor capable of distinguishing the positive samples. In addition, it argues that PU problems have specific factors that produce instability in models. Consequently, this can be explored by bagging techniques that tend to work well when there is high variance between models. It is worth mentioning that by sampling from the unlabeled set, we will train different predictors with distinct positive contamination rates, each will produce diverse classifiers, reducing the variance.

Regarding the use of PU Learning methods on imbalance data sets, to the best of our knowledge, not much has been explored under this topic, as many of the PU Learning methods are applied to benchmark datasets, that are balanced, not reflecting much of the real-world cases where these methods can be applied [30].

Several methods, such as performing under-sampling and/or assigning weights to the minority class, are common approaches used to deal with imbalanced datasets. Nonetheless, they cannot be

directly applied to a PU Learning scenario. This occurs because there is not any negative data available, thus under-sampling cannot be used, for methods that weight the risks of positive data by the class prior, their effect on learning is reduced. Furthermore, using cost-sensitive methods requires assigning a cost matrix accurately, which might be impossible due to the lack of domain knowledge.

Nonetheless, some methods are presented, for instance [30] proposes a general re-weighting strategy for imbalanced PU learning, by designing the weights for the risks for Positive and Unlabeled data. Furthermore, [16] presents a novel method called ProbTagging to deal with extremely imbalanced datasets. The key idea of this technique is to find the maximum number of positive observations in the unlabeled data set. Thus, it generates multiple positive-negative datasets from the PU data, by labeling each unlabeled example positive or negative with an assigned probability according to the similarity to other labeled positive sample. Then, several models will be trained with the generated datasets and combined into an ensemble model.

2.6 Deep Learning

Nowadays, deep learning approaches have been widely adopted to solve different tasks, such as image and speech recognition, generate content, and they have seen their performance consolidated and achieved state-of-the-art status.

2.6.1 The Perceptron

One of the earliest algorithms to rely on a neural network approach was the perceptron. The perceptron is a linear binary classifier capable of finding the optimal decision boundary that separates the classes. The algorithm will iterate over the training data until all examples are correctly classified. Nonetheless, the convergence of the algorithm is dependent on the assumption of linear separability. If this condition is not verified on the data the algorithm will not be able to converge and, consequently, not able to correctly classify all examples [9]. Being a linear model, the perceptron captures the linear relations between the dependent variable y and the independent variables x_i, through the following formula:

$$y = w_0 + w_1 x_1 + w_2 x_2 + \dots w_n x_n, \tag{2.12}$$

where w_n are the weights adjusted by the linear model.

In the case of the perceptron these weights are adjusted by the update rule:

$$w' = w + \eta y_i x_i, \tag{2.13}$$

where η the learning rate, x_i the ith observation, and y_i the corresponding label of that example. Intuitively the perceptron adjusts and finds the optimal parameters, through each iteration updating the weights' vector of each misclassified observation. The update consists of the product between the vector of features represented by $\boldsymbol{x_i}$ and the corresponding label. It is important to note that the learning rate regulates how much the weights are updated. Thus, the learning rate affects how fast the algorithm converges.

Nonetheless, linear models such as the perceptron are only useful when the data is linearly separable. However, in real-world problems this condition is not verified, consequently, it arose the need to create other approaches capable of dealing with non-linear relations represented by the data.

2.6.2 Deep Feedforward Networks

Deep feedforward networks or Multilayer Perceptron (MLP) are the foundations of deep learning. The goal is to approximate some function *f*, by defining a mapping between the target y and the features x, expressed as $y = f(x; \theta)$. Therefore, the goal is to learn the values of θ that best approximate the function f [11]. In Figure 2.2 we have a deep forward neural network architecture representation:

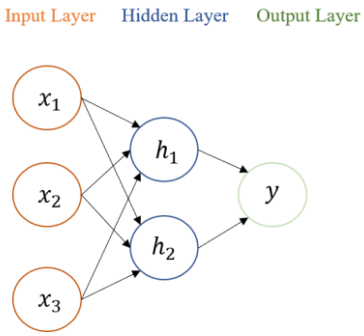


Figure 2.2: Example of a feedforward network.

In a deep feedforward network, we have layers composed of neurons/perceptrons, which are functions. In each layer the information flows from the input layer to the output layer. Hence, if we have three functions f^1, f^2, f^3, they can be chained such as, $f(x) = f^3(f^2(f^1(x)))$. Nonetheless, having a MLP without performing any non-linear operation would still result in a linear model. Therefore, the strategy applied is to use an activation function, that transforms the linear combination of features into a non-linear description of them. The most commonly used activation function is called Rectified Linear Unit (ReLU), which yields a non-linear transformation. However, because this function is almost linear it preserves many proprieties found in linear models, such as the easiness to optimize with gradient-descent methods and generalization capabilities found in linear models.

Similar to any machine learning model, a neural network represents a probability distribution, $p(y|\mathbf{x}; \theta)$, and it applies the principle of maximum likelihood. This consists of computing the cross-entropy between the target distribution and the approximated distribution by the model, which is given by the expression, where $-\mathbb{E}_{x,y\sim\hat{P}data}$ represents the negative average between the pairs of (x,y) sampled from the approximate data distribution $\hat{P}data$, and $\log p_{model}(y|x)$ represents the logarithm of the estimated probability of the model.

$$L(\theta) = -\mathbb{E}_{x,y\sim\hat{P}data} \log p_{model}(y|x), \tag{2.14}$$

Nonetheless, the non-linearity of the neural networks causes most loss functions are non-convex. Therefore, the learning process of a neural network is gradient-descent based, applying the backpropagation algorithm. So, in each iteration the weights are updated by subtracting

the computed gradient, usually a proportion of this gradient that is defined by the learning rate. This will allow at each iteration to take incremental steps that minimize our loss function [10]. However, because of the nonconvex propriety, convergence to the global optimum is not guaranteed and is very sensitive to the initial values of the parameters. As a rule of thumb, the weights should be initialized to small random values [11].

One interesting property of neural networks is their ability to approximate any continuous function [11], this is captured by the universal approximation theorem. This theorem states that any feedforward network with a linear output layer and at least one hidden layer with an activation function can approximate any Borel measurable function from one finite-dimensional space to another with any desired nonzero amount of error, assuming that we have sufficient hidden units.

2.6.3 Autoencoders

The autoencoder is a specialized type of neural network whose task is to learn a representation of the input and copy it as an output. Thus, an autoencoder is composed of an encoder, $h = f(x)$, tasked to learn and represent the inputs on a latent space. Whereas the decoder, $r = g(h)$ has, the objective of learning a reconstruction of the object starting from the encoding space. The goal is to force the model to learn important features and aspects of the input. Hence, the representation learned by the encoder is an approximation of the inputs, Figure 2.3 illustrates the components that constitute an autoencoder.

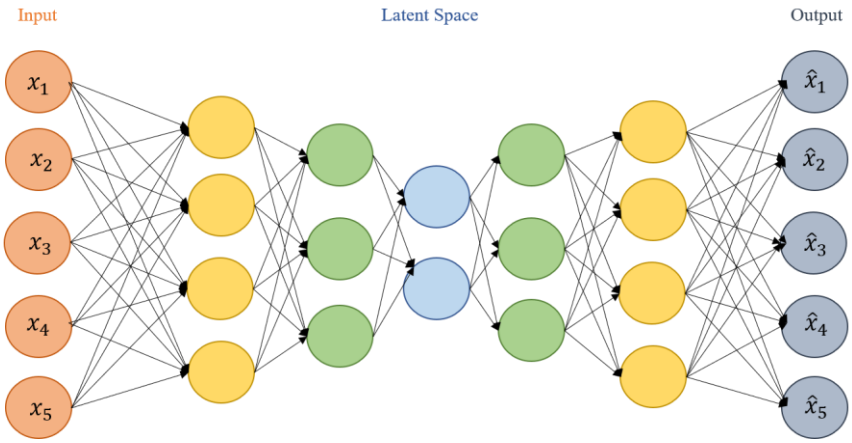


Figure 2.3: General structure of an autoencoder.

In other words, the autoencoders assume that the data lies around a low-dimension manifold or a small set of manifolds. The machine learning algorithms that rely on this concept try to learn the distribution of the data on such manifold, but the autoencoders' goal is to learn the structure of the manifold. To achieve such an aim the autoencoders assume that at a point x on a d-dimensional manifold, the tangent plane of that plane has d basis vectors that span in all local variations allowed on the manifold, representing the possible changes of x. In addition, the autoencoder focuses on learning a representation of x on a latent space h such that the decoder can approximately reconstruct it from h. On top of that, we impose constraints and/or regularization terms to ensure that the model can learn the important features and has generalizing capabilities. Consequently, the principle is that an autoencoder can represent the necessary variations that are needed to reconstruct the input.

These are the variations tangent to the manifold around x and they need to correspond to the changes in $h = f(x)$. Thus, the autoencoder learns a representation of x that is only sensitive to the variations allowed on the directions of the tangent plane of the manifold. However, it is insensitive to orthogonal changes on the manifold.

Moreover, in most cases, the learning task is assuring that the encoder is able to capture useful and relevant features of the input. Hence, we reduce the dimension of the latent space. Applying such constraint enable us to force the autoencoder to learn the relevant information.

Additionally, because an autoencoder is a feedforward network the advantages of having a deep network prevail. According, to the universal approximator theorem stated before, an autoencoder with just one hidden layer will be able to learn the identity function along the domain of the data. Nonetheless, the representation of the input on the latent space will be shallow. So, having at least one additional layer will allow the deep autoencoder to learn a representation of any mapping from the inputs to the encoded space. Furthermore, adding depth to the autoencoder can bring advantages such as less training time and reducing the computational cost of representing some function. This latent representation or embedding, h, of the input, x is typically given by low-dimensional vectors. In some learning algorithms each training example represents an embedding. In general cases, the goal of the model is to represent any given input into a latent space. Therefore, the embeddings are data representations in a latent space. This introduces the notion of how a different representation of the data may lead to better performance and allow different operations on the data.

This concept has been applied by [24], known as Word2Vec, each word maps to a dense vector of contiguous numeric values. This encoding captures linguistic patterns and relations between the words, and it benefits from linear transformations such as addition and subtraction of vectors that yield meaningful results.

2.6.4 Deep Learning applied into Recommender Systems Context

Deep Learning has shown its value and capacity to overcoming formerly complex problems in areas of natural language processing, visual computing, and robotics, among others. According to [33] deep learning can capture and learn, the non-linear and complex user-item relationships. Enabling the representation of this information into latent spaces. Moreover, this type of model allows to work with different data types such as contextual, textual, and visual information.

Regarding telecommunication recommendations, [28] argues that the traditional methods used on recommendations such as collaborative filtering and content-based fall short of producing effective recommendations in a scenario of multi-dimensional data. In addition, in telecommunication scenarios a common problem is data sparseness.

Furthermore, [34] delves into the advantages of the use of Autoencoders in recommender systems, such as encoding complex user-item relations in latent representations and tackling the data sparsity problem by incorporating further features. Therefore, we enumerate the advantages of using neural networks in a recommender system setting:

- Nonlinear Transformations: As stated before, neural networks can model non-linear relations, due to the activation functions such as ReLU. Thus, they capture the complex non-linear relations that exist between users and items. This is an advantage over matrix factorization models, which can only model linear relations.
- Representation Learning: Neural Networks can learn a representation of heterogeneous data (images, text, audio, among others) and their underlying explanatory factors;
- Sequence Modelling: Neural networks have been successful in modeling sequence tasks, such as speech recognition, and time-series, among others. Therefore, they are also capable of modeling the temporal dynamics associated with the user behavior regarding with its interactions with the items;
- Flexibility: Due to the advent of frameworks such as Keras [7], and Tensorflow [22], among others, developing methods based on neural networks has become much easier and faster to develop.

Nonetheless, this approach offers many advantages, but it has some limitations, such as:

- Interpretability and Explainability: Deep Learning models are known to be black-boxes and providing explain- able predictions is a hard task. This is particularly sensitive to recommender systems, in which is desired to have the explanation for such produced recommendations for determined user;
- Data Requirement: Deep Learning is known for being data-hungry, in the sense it needs sufficient data to support all the parameters associated with a neural network;
- Extensive Hyperparameter Tuning: Deep Neural Networks are known for their extensive and careful hyperparameter tuning, and decisions such as the size of hidden layers, the number of hidden layers, activation functions to be used, batch sizes, learning rate, among others, present a wide space of hyperparameters to tune, each in some cases may not be feasible to tune all of them at the same time searching for the best combination of the values.

2.6.4.1 Two-Tower Model

A challenge faced by the recommender systems is called the Cold Start Problem. This refers to the situation when for a new user we do not have any rating (explicit data) or action performed (implicit data) by the user on our platform. Therefore, we cannot perform any recommendation to the user because we do not have information about their preferences. Nonetheless, an emergent approach based on the dual encoder framework of language models proposes to learn the latent representations of the words, which is particularly effective in learning similarities and relations between words [20]. Thus, an approach inspired by the same methodology emerged called two-tower neural networks which are composed of two deep neural networks that act as encoders of both users/queries and items/candidates [31]. Hence, the model learns deep representations on a low-dimensional embedding space of the user and item features, allowing us to compute a scoring function of similarity like the dot product between user and item embeddings. From these scores, we can select the top k pairs of $\langle u(x,\theta_1), v(x,\theta_2)\rangle$ producing recommendations for the user. Therefore, these embeddings are the translation of high-dimensional vectors into low-dimensional

vectors. Ideally, an embedding should be capable of capturing some semantics of the input by placing semantically similar inputs close together in the embedding space. Notice that these embeddings are learnable and can be reused across models as input features.

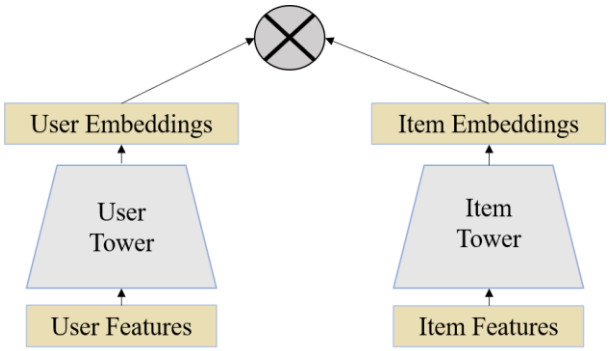


Figure 2.4: Two-Tower Model Architecture.

As illustrated in Figure 2.4, we aim to develop parameterized embedding functions u and v, the corresponding towers, that maps respectively, the query and candidate features into a latent space of dimension k based on model's parameters θ. So, u and v are two deep neural networks and the output of the model will be the dot product between two embeddings:

$$s(x, y) = \langle u(x, \theta_1), v(x, \theta_2) \rangle. \tag{2.15}$$

Based on this, if we only pass as features the user ID and item ID respectively to the user and item towers, we have a latent factor model. However, the power of this approach relies on being able to input more features that describe the user and the item spaces. Therefore, the Two-Tower Model Architecture applied to the recommender systems can be viewed as a Hybrid Recommender System, specifically according to the classification in [4], it can be classified as a feature combination type, by merging data from different sources.

In the context of a retrieval model, we have a multi-class classification problem. Specifically in Deep Neural Networks (DNN), we use the softmax function to address this type of classification. The softmax function will output a probability vector of equal size to the number of items present on the corpus, representing the probability of interaction between the user and each item.

Nonetheless, training with all corpus of items may be in many cases impossible, and training with a large number of negatives may lead to inefficiency. Thus, we perform the batch softmax optimization, where the item probability is computed over all items in a random batch. However, this type of sampling induces bias, as popular items are severely penalized due to the high probability of being sampled as negatives in the batch. To overcome this, we perform a log Q correction [32], which for the model's score is done by subtracting the probability of sampling an item j in a random batch as shown in the equation below.

$$s^c(x_i, y_i) = s(x_i, y_i) - \log(p_j). \tag{2.16}$$

Chapter 3

Related Work

In this section we provide an overview of similar works that have applied PU learning and Semi-supervised methods to the recommender systems setting.

Regarding the use of semi-supervised techniques in the context of recommender systems, [23] applied these methods to the setting of stream-based collaborative filtering to tackle the sparsity problem. The approaches consisted of combining co-training and self-learning strategies into a model ensemble. Therefore, these approaches exploit the unlabeled data highly available in recommender systems problems. The authors tested on five datasets, Movielens, Epinions, Flixster, and Netflix dataset, and they concluded that the co-training approach achieved significant improvements over the approaches that did not use semi-supervised techniques. Nevertheless, this method has the disadvantages of being time-consuming in terms of computations and, for potentially infinite streams, the co-trainers may approximate each other as they receive the same instances.

Additionally, [8] presents a semi-supervised learning clustering approach for building movies' recommender systems that combine content and collaborative information. The proposed clustering algorithm is called MK-means (mass K-means), it assumes that each unlabeled data point has an intermediate-mass value. This value remains constant during the iteration and each point is drawn towards a centroid of a determined cluster. In the case of labeled points these belong to a determined class and their mass value is set to one. However, it might occur that a cluster aggregates all of the unlabeled points.

In [6], the authors applied PU learning techniques into recommender systems, in the context of movie recommendation. The challenge consisted of having a model learning with implicit historical data, whether a user watched a movie or not. Thus, if a user watched a movie, it is labeled as positive, and unlabeled otherwise. Therefore, the authors framed the problem as a binary classification task. It is worth mentioning, under PU Learning setting that the success of a model being capable of generalizing well is largely influenced by the pool of the positive labeled examples. To overcome this challenge, the authors proposed a framework that combined PU Learning methods. It consisted of selecting the most helpful observations from the unlabeled data set and implement an active learning strategy. This technique is promising in contexts where labeled data is insufficient. Nonetheless, this approach falls short amidst cold start problems or dealing with a large unlabeled data set.

Furthermore, [5] developed a content-based recommender system of documents for Parliament members, using PU Learning methods. The authors implemented a two-step technique approach, by first extracting the reliable negatives using a new modified K-Means clustering algorithm. This modified version consisted in having the constraint of grouping all positive examples in one cluster. The data points that were grouped on other clusters were considered reliable negatives. Then, the identified reliable negatives and labeled data were used to train a binary classifier for each Parliament member. The authors showed that this approach surpassed other methods, such as considering all unlabeled data as negative and the PU Learning method supported by the Naïve Bayes.

Moreover, [34] proposed a new framework based on Generative Adversarial Networks (GAN) for recommendations that uses a PU sampling strategy. The framework developed focus on training an unbiased positive-unlabeled discriminator and training a generator on the embedding space of users and items. In more detail, the discriminator is a binary classifier that focuses on predicting a probability (i.e., a relevance score.)

Concerning the deep learning approaches in recommender systems focusing on telecommunication settings, [20] proposed a method that consisted of exploring the similarities between products. It focuses on computing the items' embeddings. The data used consisted solely on customer interactions with the products, which were mobile tariffs. In this study, a tariff is a word, and a sentence corresponds to the historical transitions a customer has done between tariffs. So, the proposed method relied on a modified version of the commonly known approach Word2Vec, presented in [24]. This strategy consisted of embedding the items in a latent space, allowing to capture relations between products without considering details about the product or the end-users.

Additionally, [28] presented an approach that combines the methods of Word2Vec, Embeddings, Deep Cross Network (DCN), and Deep Belief Network (DBN) applied to the scenario of a telecommunication package recommender system. The authors argue that viewing the telecommunication packages as items allows to re-frame the problem as a task of improving the probability of a user clicking on a recommended item. Thus, the model is composed first by a Word2Vec operation over the Textual features. Then, the dense, sparse, and converted textual features will be the input of an embedding layer, and the output of this layer will serve as input for the DCN and DBN that work in parallel. The DCN will explore feature interactions, whereas the DBN will focus on finding high-order non-linear relations within the data. According to the results presented their combined approach of different methods recorded a better performance when compared with each method separated.

Moreover, [27] proposed a collaborative filtering approach using the Alternating Least Square, to produce telco recommendations, text messages, voice and mobile data, based on implicit feedback. The data was composed by an historic of six months of user-item interactions, that consisted of monthly purchasing these products. The authors proceeded with the segmentation of their clients, where they experimented with two segmentation methods, the first based on the revenue generated and services included in their purchased offers. Then, they proceeded with the application of the algorithm chosen. The authors concluded that their approach was fast and provided quick recommendation.

Nevertheless, the methods and strategies mentioned above did not solve the problem of this project. As we have a rich user feature space applying approaches that cannot integrate this type of information means that we are discarding valuable information to solve our problem. Besides, the solution to our problem should be simple and fast-running, as it is expected to be integrated into the daily operation of the company.

Chapter 4

Methodology and Data

This chapter presents the methodologies adopted for the project, PU Learning and recommender systems, and an Exploratory Data Analysis (EDA) for both datasets used in each strategy. Also, for the general framework, we describe how the NBO project currently works. The project developed presents two different methodologies that aim to address the stated problem and improve the recommender system of the NBO project. Since these are two distinguished approaches, we have two different datasets with similarities regarding features but capture different information. For the PU Learning approach, we use a dataset where each observation corresponds to the proposal made through the outbound channel. Each proposal maps a client to an offer and can have two possible outcomes: adhesion or refusal. Contrarily, the dataset used for the recommender system approach is composed of the adhesions that occured in all selling channels of the company.

The PU Learning's goals is to produce a binary classifier that predicts whether an assigned offer is an adhesion or a refusal. Inversely, for the recommender system approach, we focus on developing a retrieval model that for a given user retrieves and recommends the most similar items. The following sections provide more in-depth information regarding the datasets and the methodologies adopted.

4.1 General Framework - Next Best Offer Project

The NBO is the company's use case that has the goal of implementing a recommender system capable of producing offers that:

- are aligned with the preferences and needs of the client;
- increase the profit of the company;
- increase the adherence rate.

We achieve these offers by combining the power of machine learning and analytics with business knowledge. Therefore, the NBO is a complex project, constituted by a multidisciplinary team, that leverages the expertise of each team member to achieve the goals proposed. Also, to the present day, the NBO only produces recommendations for outbound campaigns, which are telemarketing campaigns where the operator reaches the client to propose an offer. Worth mentioning, that these campaigns work based on commercial cycles. Currently, the recommendations are produced by

two models. The first is the heuristic model, which works by business rules. The second is the analytical model, based on a machine learning approach.

Furthermore, in terms of steps that compose the NBO working pipeline, we have the following stages:

1. Generate a pool of possible offers to be recommend to each NBO client;
2. Select the top-3 offers based on the propensity score and offer value;
3. Deliver the top-3 recommendations to each client to the operation.

At each business cycle, a table containing all possible assigned offers for a client is updated with the newly generated proposals for that cycle. This is a process based on heuristic rules and corresponds to the first stage of the NBO pipeline. This process ensures that the generated offers are aligned with the business interests and strategies and do not contain any service downgrade. Concerning the selection of offers, it is important to mention that from the around 125,000 clients, only around 60,000 are suitable for analytics campaigns. Consequently, these are clients eligible to receive a recommendation produced by the analytical model. The remaining clients receive recommendations based on the heuristic model.

Moreover, focusing on the analytical model of the NBO we have three stages:

- Data Stage - Generating the train and inference datasets;
- Modeling Stage - Training and fine-tuning the analytical model;
- Cockpit Stage - Ranking and selection of the offers to be sent to the operation;

On the first stage (the data stage), the training and inference datasets are created. Due to the constant streaming of new data generated in each cycle, the model is re-trained with the new training panel. This dataset is created based on a database where we filter out, through an outcomes field, all observations that we are not certain whether an offer's presentation occurred. So, we train the model with observations that we are certain that the operator presented an offer to the client. Thus, a line of the training panel will correspond to a client with a presented offer. Regarding the features, the panel encompasses the client's portfolio and historic variables about the customer regarding topics and consumption. Also, it contains attributes describing the offers' content, the corresponding telco components upgrades, and the difference between the monthly payment and the effort price, designated as effort delta. Moreover, the inference dataset is created for the specific inference cycle. It will have as many lines as assigned offers for each client. So, each line of the inference dataset corresponds to a client and a mapped offer.

The modeling phase trains an adhesion's propensity model following a binary classification approach. The positive label refers to an adhesion of a recommended offer. Consequently, the model will output a propensity score of the customer adhering a given mapped offer. In the first phase the model consisted of a Random Forest, latter it was adopted a Gradient Tree Boosting approach. This modeling step produces a trained model, which will be used for the inference stage, where the ranking and selection of the top-3 offers is done by the cockpit stage.

The cockpit stage is responsible for ranking and selecting the top-3 offers to be recommended to the user on the operation side. It also ensures the required business restrictions so that the recommendations align with the strategies that the business decides. Based on the model's score it will be computed the Expected Value (EV) metric defined by the following formula:

$$EV = ew_1 + pw_2. \tag{4.1}$$

Where e is the normalized offer's effort and p is the normalized model's score. The EV will be used as a ranking score to choose the top 3 offers to be recommended for each client. Nonetheless, this ranking is subject to some restrictions to ensure the compliance with business rules and strategies. Such restrictions are guaranteeing that the offer's effort increases as the rank of offers increases. Thus, top one recommendations will have the offers with the higher efforts, followed by top two and top three offers having, respectively, minor effort values. Also, all offers must have different component upgrades.

Afterward, the recommendations will be delivered to the operation side which will be tasked with allocating the campaigns and reaching the clients via phone calls to propose the recommendations.

Figure 4.1 illustrates the training and inference processes described above, which shows how each procedure is related to others and the outputs produced in each stage.

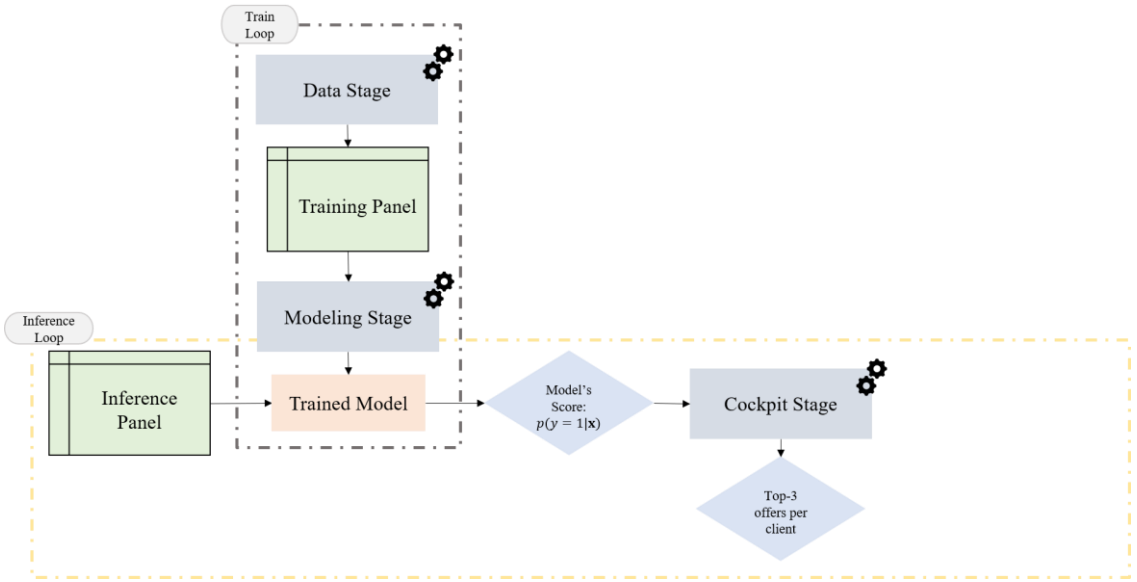


Figure 4.1: Schema illustrating the stages that compose the NBO Project.

Based on Figure 4.1 the adopted methodologies in this thesis focus on a PU Learning approach which impact:

- Modelling Stage.

Also, we adopt a recommender system approach, which have influence on:

- Data Stage;

- Modelling Stage;
- Cockpit Stage.

4.2 EDA in PU Learning Appoach

The training panel has a row dimension of 200,000 observations and a column space of 90 features. It consists of a binary supervised dataset, where label 1 means an accepted assigned offer and label 0 is a refusal of that offer. Nonetheless, as explained before, these negative examples in several cases, will be noisy negatives, as the tracking of the offer is not possible. The validation of an adhesion is done at the end of each cycle, in which we check if the client's portfolio has changed and if the changes correspond to the upgrades offered. The panel encompasses the clients' portfolio changes, which are verified throughout three cycles in the future. It is worth mentioning, that the training panel only have observations coming from the outbound channel.

The first step of the EDA, we checked the percentage of missing values across the variables. Most of the features did not present null values, being the highest percentage of null values around 32%. Since it is a value below 75%, these features retained information for the model. Hence, we opted for applying the imputation of missing values strategies instead of dropping the columns.

Furthermore, Figure 4.2 shows the number of observations per business cycle, allowing us to check for any data fluctuation and detect an outlier period. Analyzing Figure 4.2, we observe that the period with fewer records was cycle two, whereas cycle five recorded the highest number of observations. Nonetheless, we do not observe high deviations from the mean across cycles.

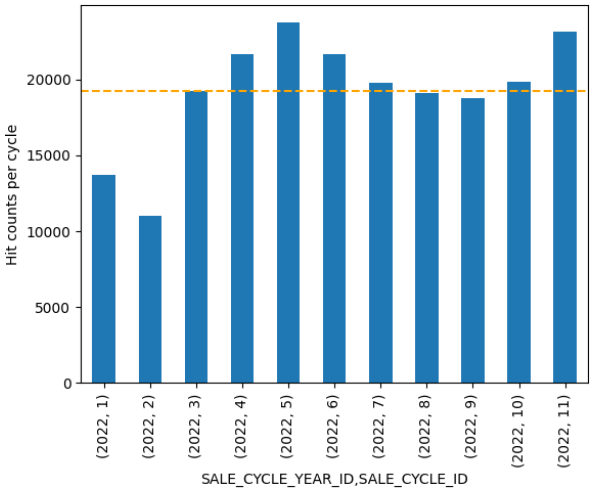


Figure 4.2: Distribution of data size across business cycles, yellow line represents the mean.

Figure 4.3 shows a general view of the clients' portfolios at the start of a cycle. In terms of bundle typology more than 40% of the clients have a 4P, followed by a close value of users with 3P as product. Regarding Television (TV) rank, more than 30% of the customers have rank 2, then more than 25% have rank 1. Regarding the internet download speed, we observe that more than 30% of the clients have an Internet speed of 200 Megabits per second (MBPS), followed by more than 25% with 500 MBPS. Note that 20% of the users have an Internet speed of 100 MBPS,

and around 5% of the clients have the maximum internet speed offered of 1024 MBPS. Regarding, of Mobile Data (MD) plafond quantity, almost of 50% of the users do not have MD. Contrarily, more than 10% of the costumers have 10 GB as MD and around 10% of the users have 6 GB of DM. As for quantity of Mobile Voice (MV) cards, more than 40% of the clients do not have MV cards assigned to the telco package. On the opposite, around 20% of the clients have 2 MV cards followed by more than 15% have one MV card.

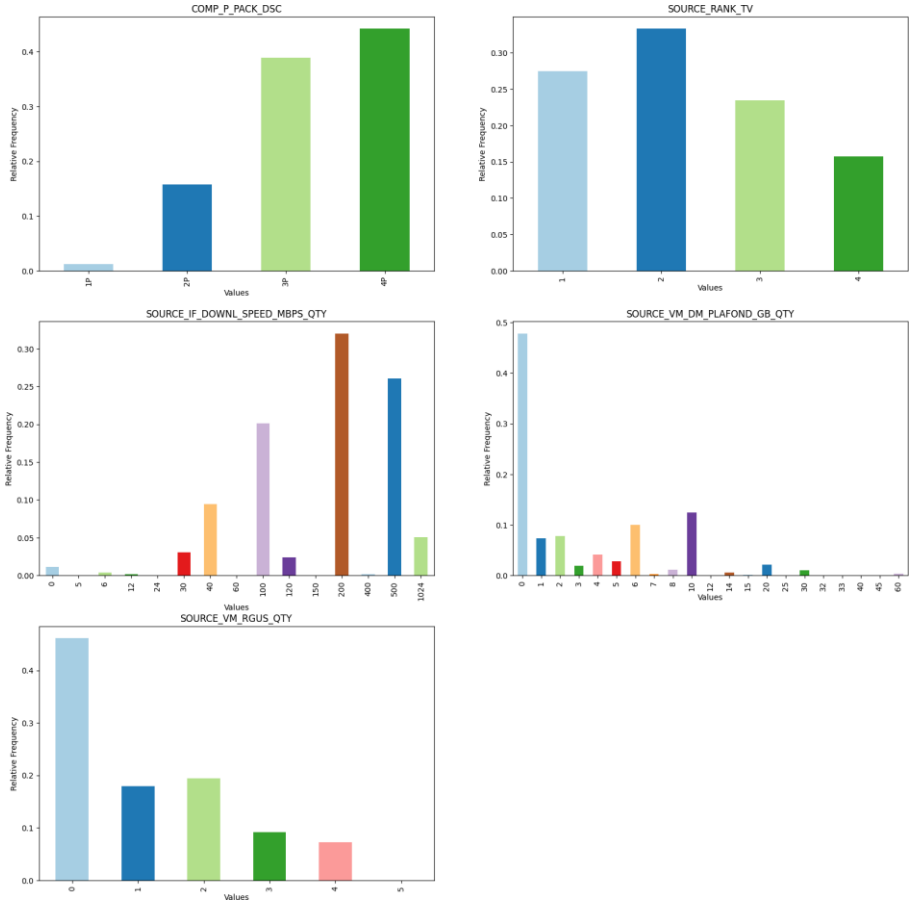


Figure 4.3: General characterization of the clients of the outbound channel.

Figure 4.4 shows offers' effort distribution. We observe that most of the offers have an effort on the interval of [1,12]. Above 12, we verify a decrease in of the number of offers with such values.

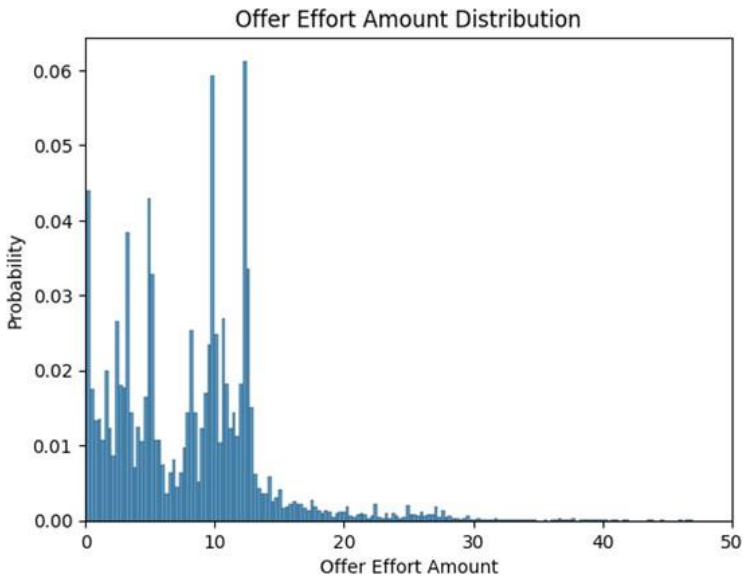


Figure 4.4: Distribution of the Effort of Offers.

Figure 4.5 shows the offered upgrades or jumps at each telco service, which we conclude that around 75% of the offers do not possess TV upgrades, then below 25% of the offers propose 1 jump and only a small proposals percentage have more than 1 jump. Considering the Fixed Internet (FI) [1], more than 20% of the observations have 1 upgrade of this component. On the opposite, around 60% do not propose any FI jump. As for MD Jumps, we conclude that more than 50% of the offers own at least 1 upgrade, recording a small percentage of offers allowing 6 jumps in MD. Concerning MV jumps around 60% of the offers have 1 jump in MV and 40% do not offer any upgrade.

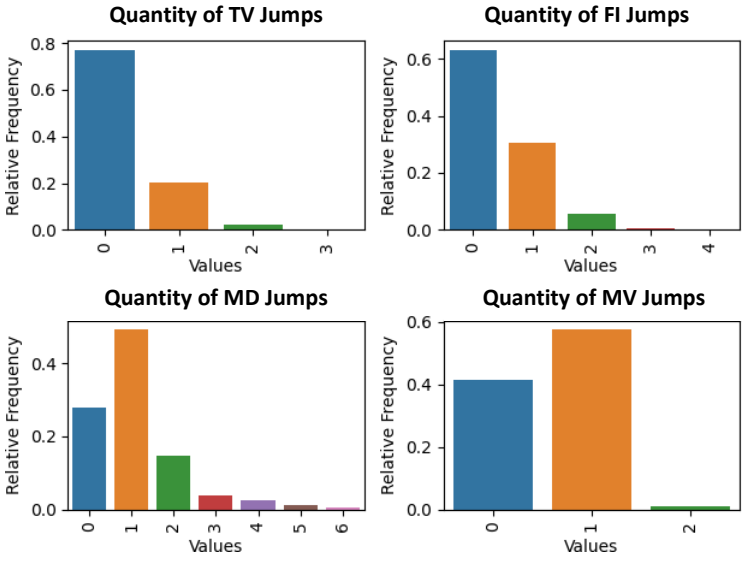


Figure 4.5: Jumps Quantity per Component.

Regarding the target distribution, we have an imbalanced dataset, with 95% of our observations being recorded as negatives and 5% being positive, as illustrated by Figure 4.6.

[1]Home Internet

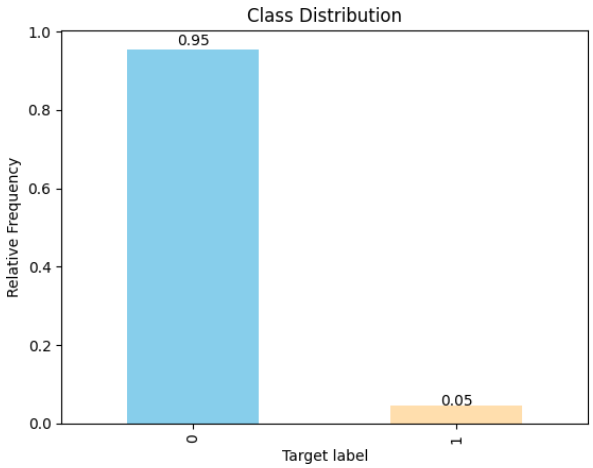


Figure 4.6: Distribution of classes.

Figure 4.7 allows us to conclude that the cycle with the highest adhesion rate was cycle six followed by cycles nine and two. Inversely, cycles four, five, and eleven recorded the lowest adherence rate highlighting the eleven cycle as worst in the available data period.

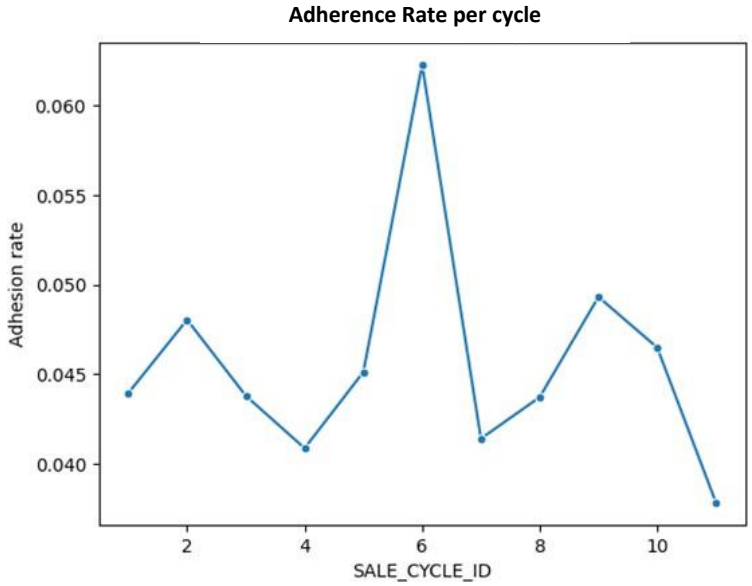


Figure 4.7: Adherence Rate per cycle.

Figure 4.8 shows the adherences per component (TV, FI, MV and MD) across business cycles. We observe that offers with FI or MD upgrades tend to have higher adherences rates. Whereas offers with MV or TV jumps tend to have lower adhesion rates. Furthermore, the adherence per feature, we see that the cycles two, six, and ten recorded the highest adherence rate across all components. Contrarily, cycles seven, eight and eleven were decreasing periods in terms of adhesion rate for all components, except for MD on cycle eleven increased.

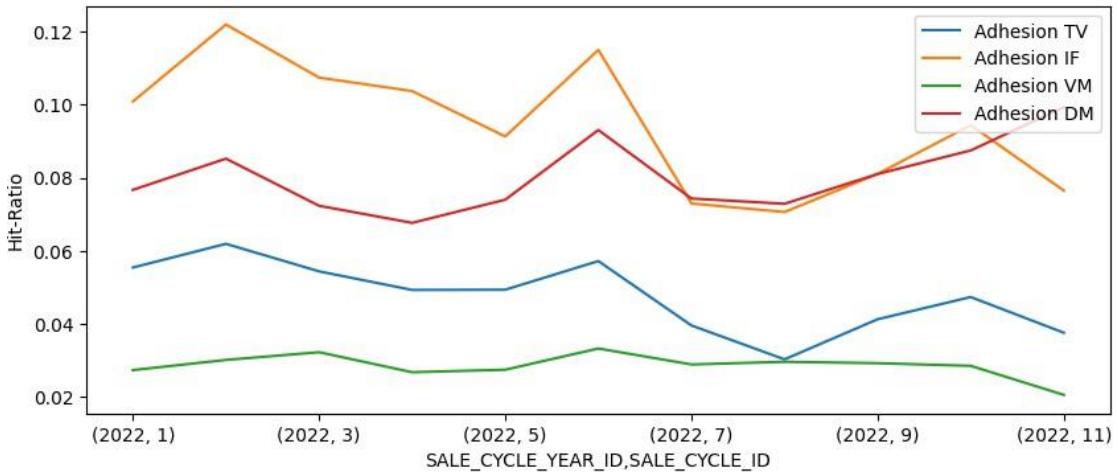


Figure 4.8: Adherence Rate across business cycles per product component.

Regarding the adherence rate per jump of component, Figure 4.9 shows that, in terms of TV jumps the adherence is higher for 2 TV upgrades. For FI jumps, the hit rate is greater for 2 jumps followed by 3 upgrades. As for MD jumps, the adherence rate is higher for 4 upgrades, followed by 6 jumps and 1 upgrade. For most offers the MV is higher when not offering any MV upgrade.

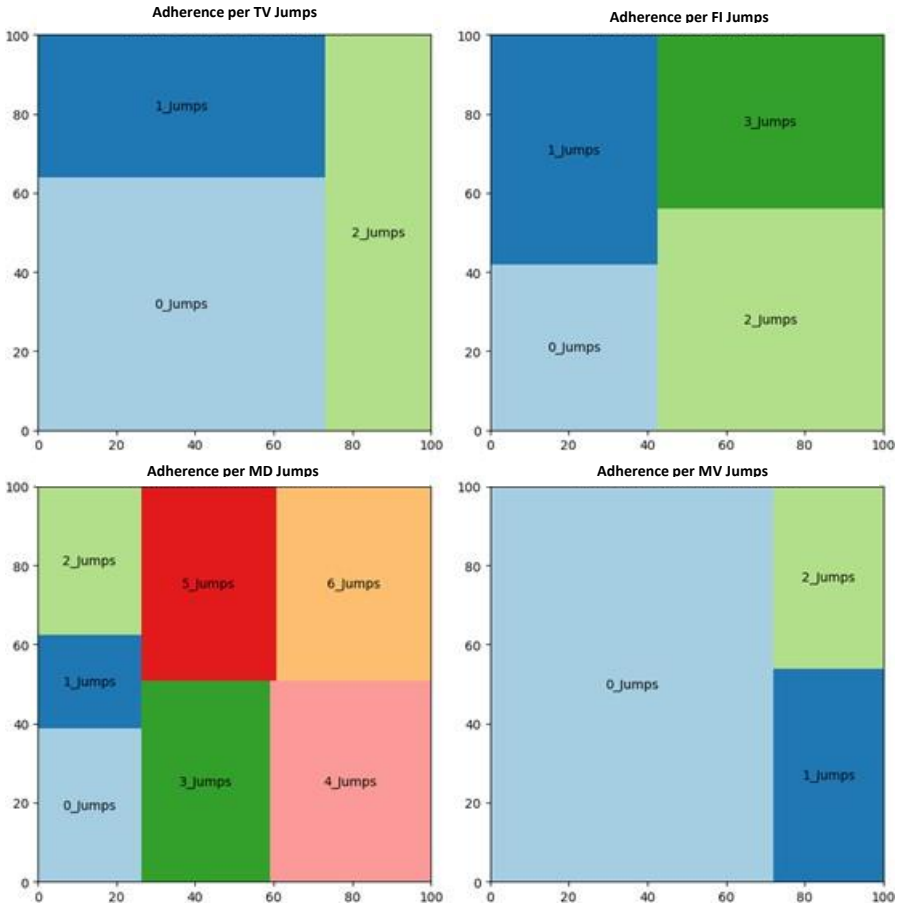


Figure 4.9: Adherence Rate per jump feature quantity.

Overall, we do not have strong correlations, and we can consider all of them to be weak correlated with the target (see Appendix A). For the features OF_EFFORT_PERC_QTY, OF_VM_RGUS_JUMPS_QTY, and SOURCE_RANK_TV record the highest negative correlations. As for the OF_EFFORT_PERC_QTY, the offer's effort is inversely related to the adherence rate since a client will prefer to pay less for the same service. Furthermore, the variable OF_VM_RGUS_JUMPS_QTY is negatively correlated with the target as offers that have MV jumps are more expensive. For the case of SOURCE_RANK_TV, the negative correlation is explained by the fact that higher ranks of TV mean the client can no longer upgrade or the quantity of jumps is lesser. As shown in Figure 4.9 this rate is higher for 2 jumps, which means we are seeing adhesions for customers with TV ranks of 3, in other words, clients without TV service.

For the case of the features with the highest positive correlation, we observe these are SOURCE_TOTAL_DAY_AMT and SOURCE_VM_RGUS_QTY. For the feature SOURCE_TOTAL_DAY_AMT, a client with a higher monthly fee will also have a higher adherence rate. The same can be concluded for SOURCE_VM_RGUS_QTY as customers with more VM Revenue Generating Unit (RGU) tend to have a higher adherence rate.

4.3 Adopted Methodology in PU Learning

4.3.1 Training Stage

The Training Panel used on the PU Learning approach is the same used on the NBO project. Figure 4.10, encompasses the client's profile and the offer's profile. Regarding client's features, they are related to the customer's portfolio, such as the TV, F1, MD and MV ranks, and the total monthly payment for the contracted telco services. Furthermore, the panel has features related to the consumption of TV services, such as the number of hours of TV watched and recorded. Also, it possesses features related to topics/issues signaled by the client, such as how many non-resolved issues the customer has. Besides, the dataset has client historical features in terms of consumption, topics, among others. Finally, the panel encompasses features related to the business activity performed by the customer, which describes the client in terms of its business, the size of the company, and how much time is operating, among others.

As for the offer's profile, the dataset has features that describe the offer, such as how much download internet speed, data mobile plan, and quantity of mobile cards, among others. It contains information regarding the upgrades that result from the deltas between what the client has and what is being offered to the customer. Alongside, this the respective effort, which corresponds to the delta between the current monthly payment and the payment proposed by the offer.

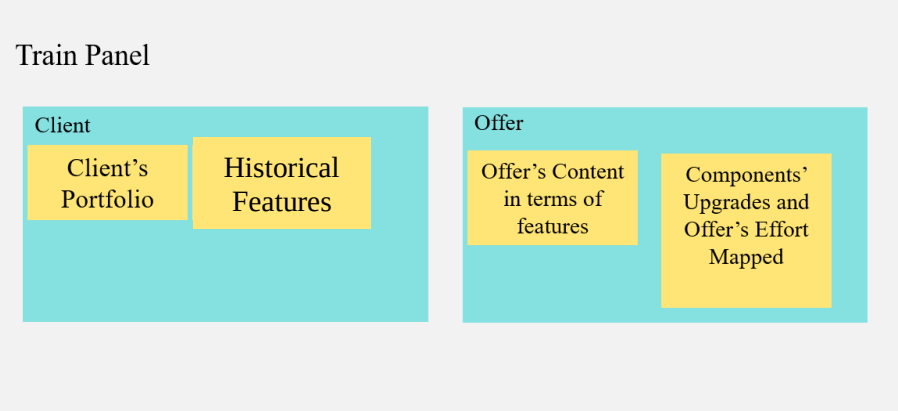


Figure 4.10: Schema illustrating the profiles that comprise the Training painel.

The Methodology adopted consisted of testing several PU methods that followed the two-step approach, such as KNN, hierarchical clustering, and k-means used as the first step to select the reliable negatives. Under all the methods experimented with we selected the Spy technique [21]. This approach consists in:

1. Select a sample of positive labeled examples and change the label to unlabeled, which will be called spies;
2. Train a binary classifier;
3. A reliable negative is selected if its probability of being positive is below the probability's first decile assigned to the spies by the model.

So, a reliable negative is considered when $RN = P(x|y = 0) < D_1[P(x|y = 1)]$, being D_1 the first decile of the model score assigned to the sample of spies' observations.

Due to the imbalance of classes, during the stage of finding reliable negatives, we trained the model with data partitions containing the same percentage of positives and unlabeled data. Moreover, the fraction of spies selected corresponded to 20% of the positive samples in each data partition. Thus, to sum up the steps taken with this algorithm, we have:

1. First-step
    1.1. Partition data, so that we have equal proportions of positive labeled and unlabeled observations;
    1.2. Select 20% of the positives as spies;
    1.3. Train a binary classifier with this dataset
    1.4. Use the trained model to select the reliable negatives following the spy method;
    1.5. Repeat the steps above until there is no more data to iterate.
2. Second step

    Train a binary classifier with reliable negatives and positive labeled examples.

Moreover, we also proceeded with the method of PU-Bagging. Thus, we selected two methods to be tested and compared with our baseline, the propensity model implemented on the NBO.

Figure 4.11 summarizes the steps of the training stage of the Spy method. We divide our Training panel into three datasets, training, validation, and test dataset. On these datasets, we perform the imputation of missing values and the encoding of categorical variables. Then, we perform select of reliable negatives on the training set using the Spy technique. Next, with the selected reliable negatives and positive examples, we optimize the hyperparameters of the model and train it with the best combination found. We can train directly the model without running the optimization process. In that case, the default or inputted hyperparameters are used. Afterward, we evaluate the model using the defined evaluation metrics, the PUF-SCORE, and the Area Under the Receiver Operating Characteristic Curve (AUROC).

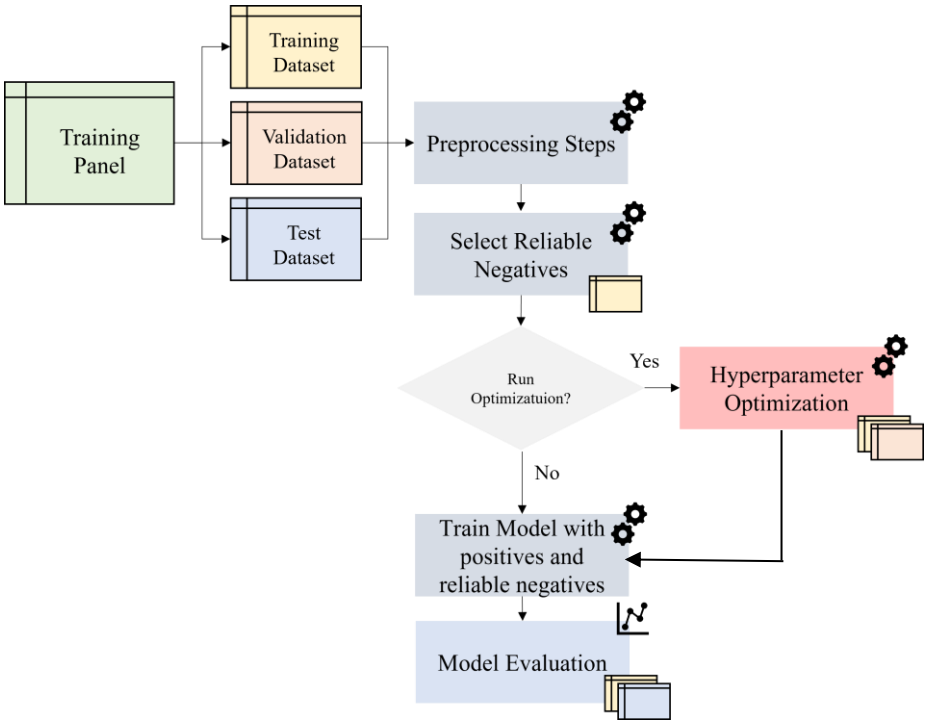


Figure 4.11: Diagram showing the steps that compose the training process of the Two-Steps Approach (Spy- method).

Figure 4.12 illustrates a sample of 1000 spies and 1000 reliable negatives selected on a 3D scatter plot. For visualization purposes, we performed the Principal Component Analysis (PCA) operation to reduce the dimensions and we plotted the three first principal components. Nonetheless, the dimension reduction was not used for the training of the model. Although not being the most accurate representation of our data, we observe two major clusters that tend to group the spies and the reliable negatives, showing a separation between these clusters.

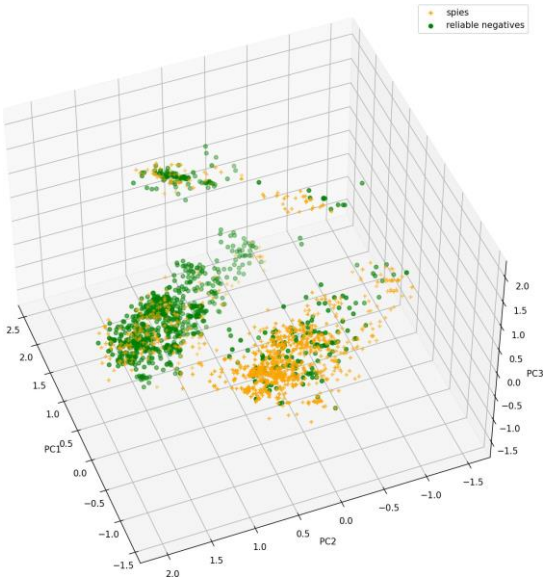


Figure 4.12: 3D representation of a sample of 1000 spies and 1000 reliable negatives, described by three first principal components.

Figure 4.13 presents the training and evaluation stage of the PU-Bagging model. In terms of splitting the training panel, preprocessing and running the optimization are the same as the steps taken for the two-step approach. Then, each base estimator is trained with the positive labeled data and a random sample of unlabeled examples. Finally, we perform evaluate the model using the PUF-SCORE and the AUROC.

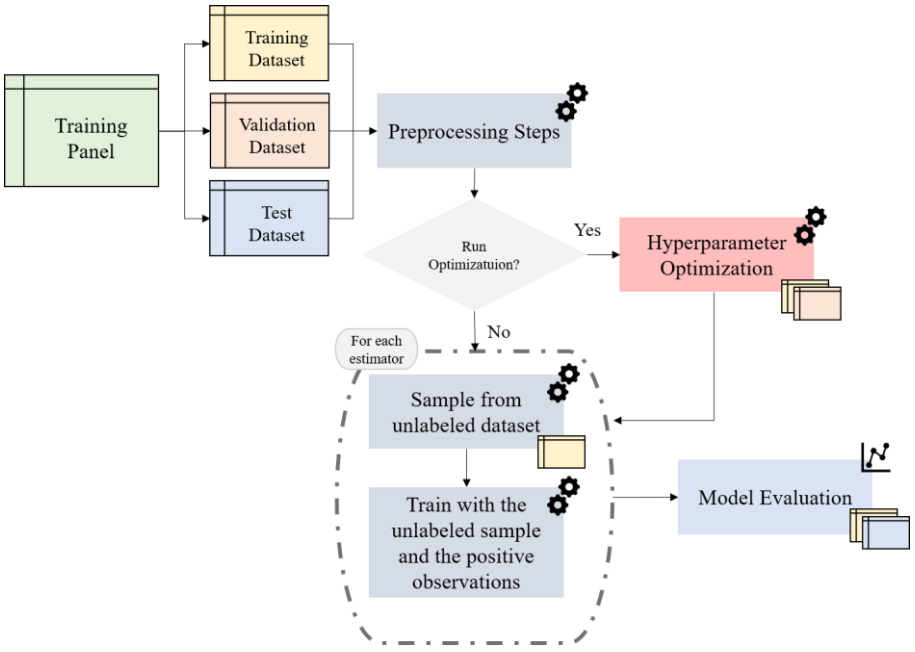


Figure 4.13: Diagram that illustrates the training process of the PU-Bagging.

4.3.2 Preprocessing Data

As for the preprocessing stage, we removed the duplicated records, imputed missing values, and encoded categorical variables. Concerning the encoding of categorical features, we tested different encoding types and recorded the AUROC and PUF-SCORE obtained as shown in Figure 4.14. We concluded that, in terms of AUROC, the best was the ordinal encoder, and regarding the PUF-Score, it was the target encoder. Nonetheless, we chose the target encoder, which corresponded to the encoding performed already by the model implemented on the NBO.

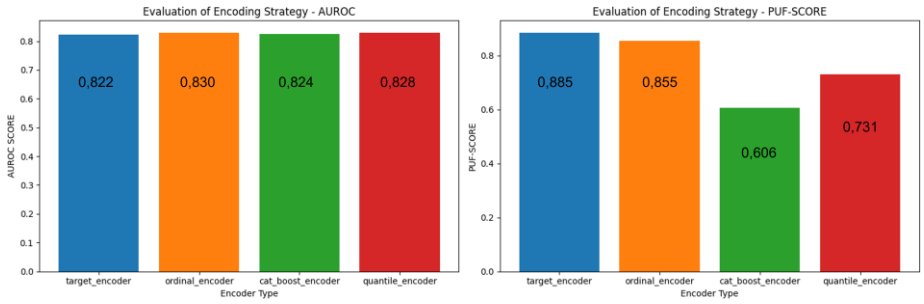


Figure 4.14: Evaluation of the different types of encoding.

We also tested feature selection using the Boruta method and recorded the AUROC and PUF-SCORE, as illustrated in Figure 4.15. From analyzing this figure we can conclude that not performing feature selection leads to a better result in both metrics. Therefore, we proceeded without feature selection.

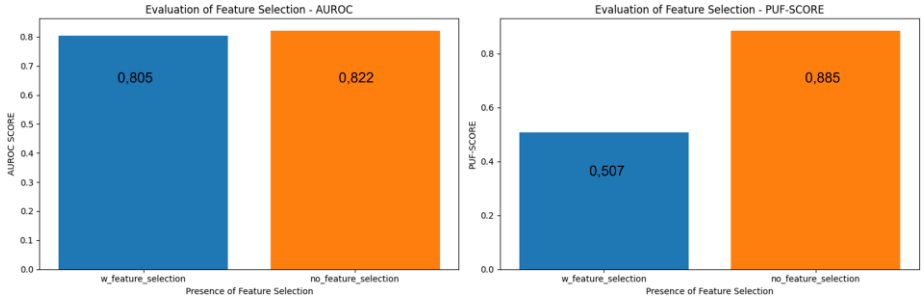


Figure 4.15: Evaluation of the model with and without feature selection.

It is important to note that, across all models, we used gradient boosting trees, recurring the library lightgbm [17]. The only exception was for the Spy technique in which a Random Forest was used to select the reliable negatives.

4.4 EDA in Recommender Systems Approach

In this section we present the EDA performed over the training panel used on the recommender systems approach. The training panel is composed by 39,000 observations and 55 features. It has 3,605 unique items and 30,789 unique users. This dataset comprises the portfolio upgrades that occurred per cycle. It is important to note that we are covering all activations that occurred onsite and in online stores, the inbound/outbound channels, and the agents (for high value clients that need closer assistance), not only the outbound channel as it was the case for the PU Learning approach.

Moreover, we verified the missing values and observed that 15 of the 55 features presented missing values being the highest value recorded around 46% of null values in some variables. Therefore, the imputation of missing values was performed. We also check the distribution of observations across business cycles, as shown in Figure 4.16. We can observe that across all cycles until cycle 2 of 2023, there is not any major variation. Then, for the cycle four of 2023 we see a major peak due to an increase of standalone MVs sold. Inversely, for cycle five of 2023, we observe the inverse behavior, a very accentuated decrease in of the portfolio's changes.

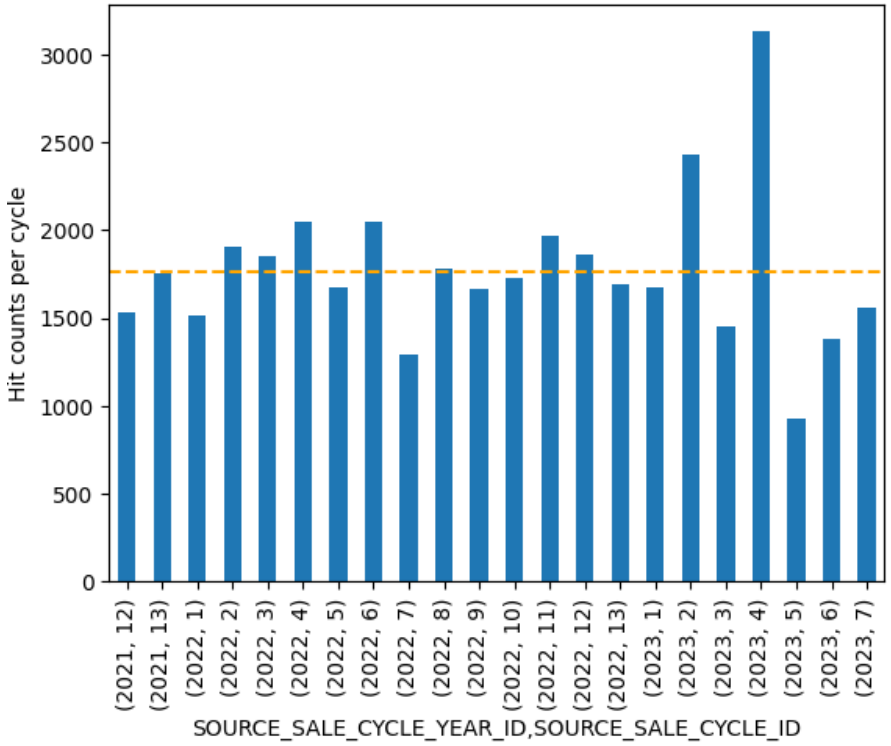


Figure 4.16: Distribution of interactions across business cycle.

Figure 4.17 shows a general characterization of the client's portfolio before occurring a change. We observe over more than 50% of clients have a 4P package, followed by more than 30% of the clients having a 3P. In terms of TV ranking, almost 40% of the customers have rank 1, hence, among these clients is not possible to upgrade the TV. Besides that, nearly 35% of clients have a rank 2 allowing these clients to have 1 jump. In terms of the download speed of FI, more than 30% of the users have a download speed of 200 MBPS, followed by 30% having an internet speed of 500 MBPS. Only 10% of the clients have the max download speed offered with 1024 MBPS. Also, a small percentage of the customers do not have internet. As for the MD plan, more than 40% of the users do not have a MD plan. On the opposite side, nearly 20% have 10 GB as a MD plan followed by 10% with 6 GB. Note that a small percentage of the clients have unlimited GB, marked as 998. As for MV cards, less than 60% have at least 1 MV card. As nearly 20% of the customers have 2 cards, followed by more than 15% of the clients who have 4 cards.

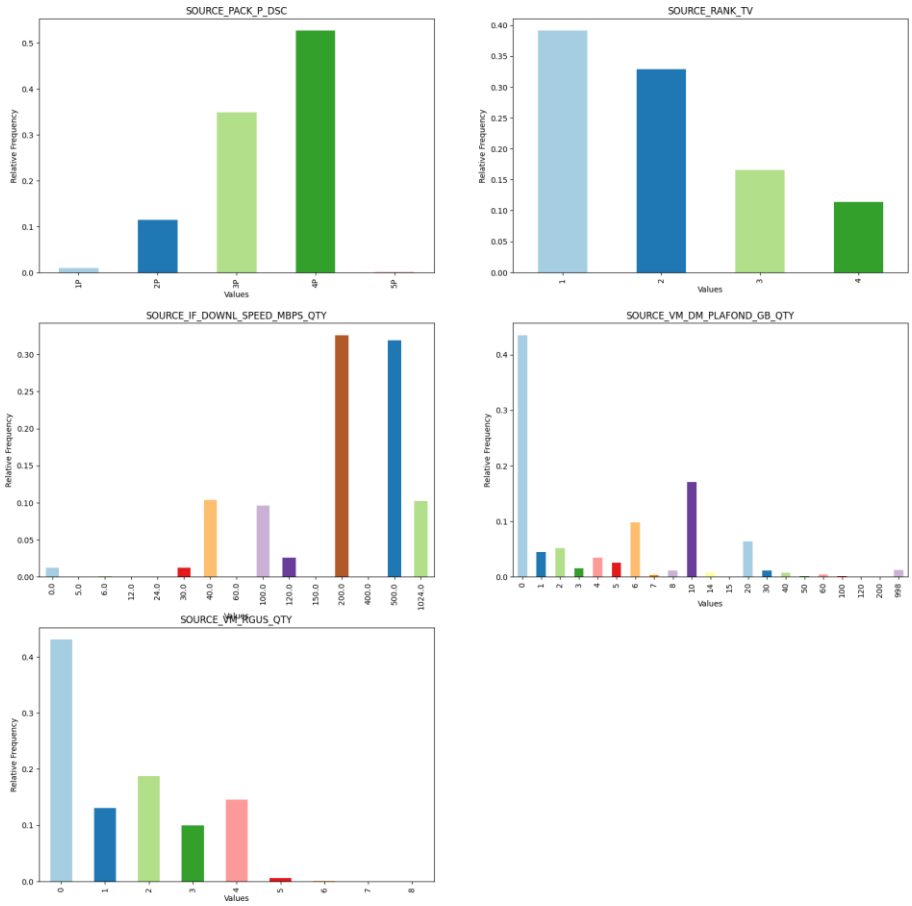


Figure 4.17: General characterization of park of clients (all channels).

Figure 4.18 illustrates the distribution of the effort of the offers accepted. We can observe that for the majority of proposals, the effort is located at the lower values of this feature, until 12. Then, for effort values higher than this, we observe a more steady, gradual decrease in the number of adhered offers.

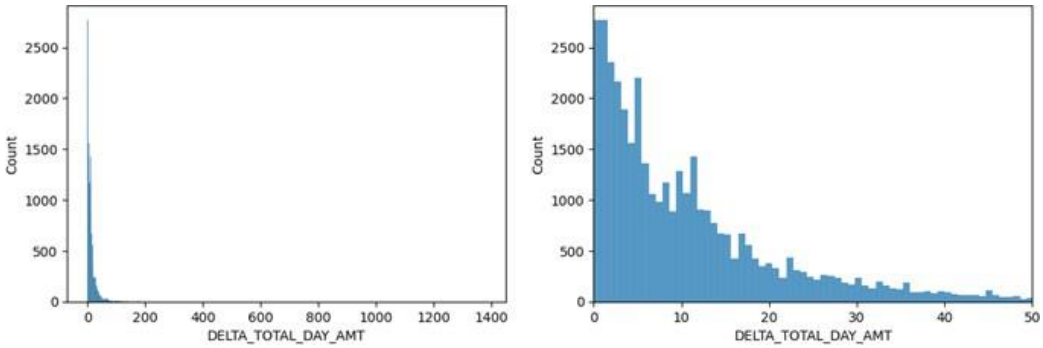


Figure 4.18: Distribution of effort

Figure 4.19 shows the number of adherences recorded per item, which can be interpreted as the item's popularity. We observe that the most popular item corresponds to the item with one jump of MV and an interval of effort higher than 4% and lower or equal to 8%. From the

distribution we notice we have a long-tail problem, which, for a large percentage of items, we only have one interaction. In fact, it accounts for around 51% of our observations is composed of only one interaction per item.

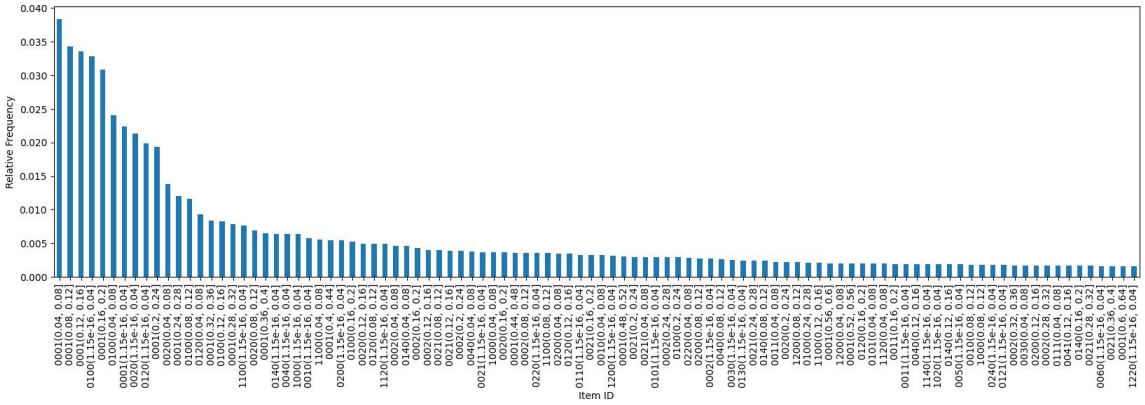


Figure 4.19: Top 100 popular items.

In Figure 4.20 we observe the number of times a user appears in our dataset, and we see that more than 80% of our customers only adhered once time to the proposed offer. This leads to a dataset where we lack historical information about the users' behavior and their preferences.

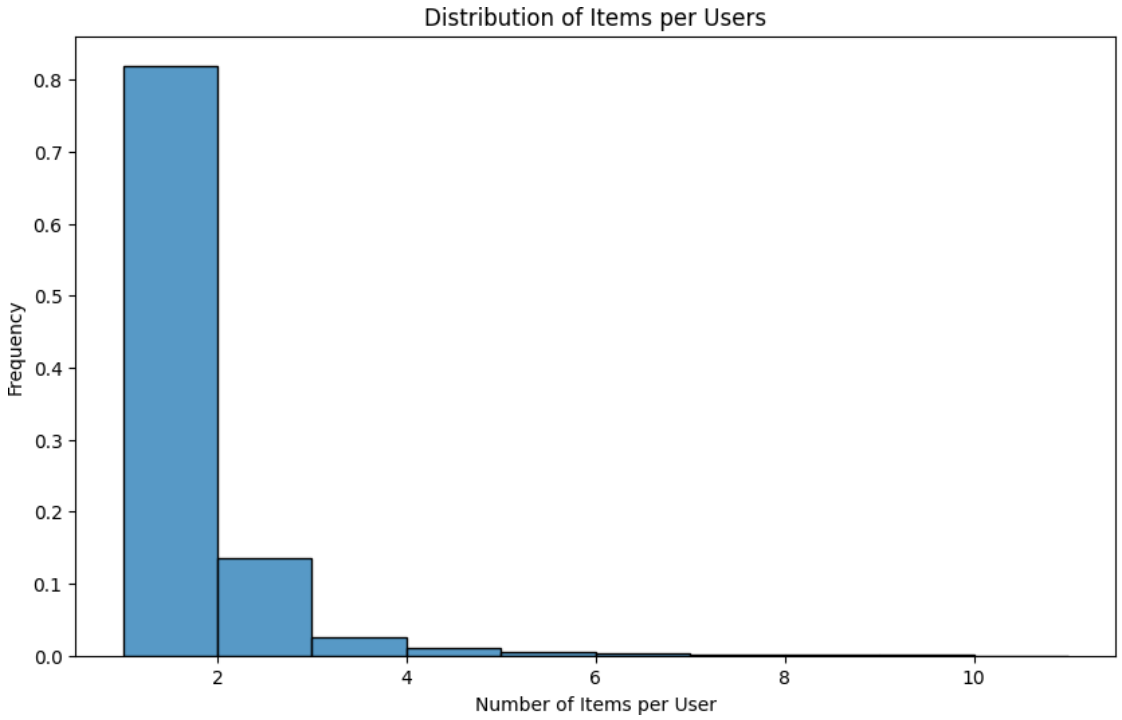


Figure 4.20: Distribution of the number of items per user

In Figure 4.21, we can observe that for TV jumps, below 20% of the clients perform 1 TV upgrade or more. Moreover, in terms of FI jumps, we observe that around 30% perform 1 FI upgrade and more than 10% do 2 FI jumps. As for MV jumps, we observe that 40% of the clients perform 1 MV jump, which means they add one more MV card. In general, more than 50% of the clients do at least one MV jump. Finally, for MD jumps around 20% perform 2 jumps of MD and 10%

perform 4 DM jumps. In general around 50% perform at least one DM jump.

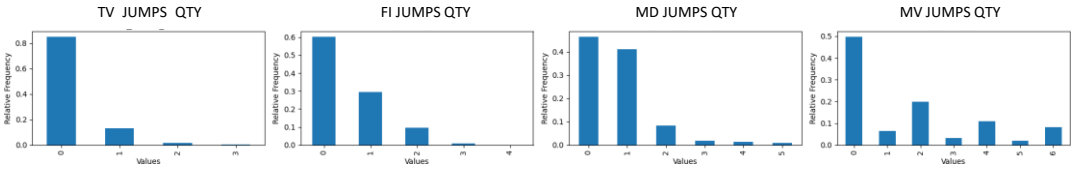


Figure 4.21: Quantity of jumps per component.

Additionally, in Figure 4.22 we observe that in terms of TV Jumps around 20% of the customers with a CAE equal to RESTAURANTE (Restaurant) and ATIVDESPORTIVA (Sports Activity) perform TV jumps. This makes sense because restaurants and cafes rely on sports content as a way to attract customers to their establishments. As for FI Jumps, the sectors INDSAUDE (Health) and RESTAURANTE are the industries that have more companies adhering to FI upgrades with more than 50% of the clients' performing upgrades on this component. For MV jumps, the clients with CAE as OUTATIVID (Other Activity) 100% perform MV jumps. Inversely, the restaurants are the clients that perform fewer MV jumps, as in general, these are small companies, whose employes do not need mobile service. Finally, for the case of MD jumps, 60% of the clients with CAE OUTATIVID execute DM upgrades followed by companies with an activity classified as INDSAUDE.

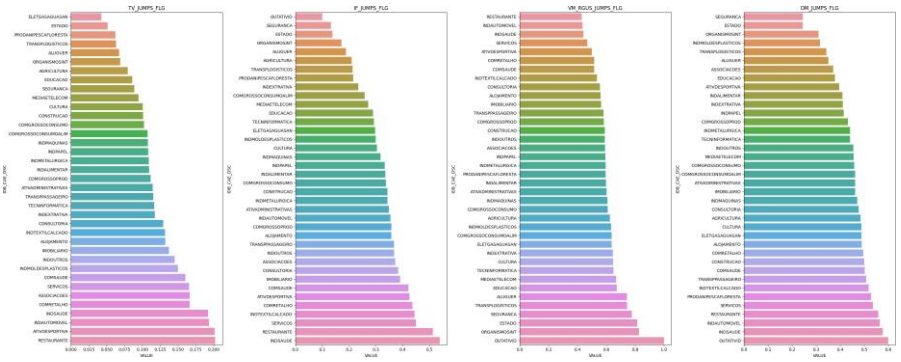


Figure 4.22: Distribution of Components' Upgrades per CAE.

In terms of correlation with TV jumps, in Figure 4.23 we do not observe high positive or negative correlations between features. For higher values on the TV rank the customer tends to perform higher TV upgrades. The same pattern happens for the MD rank and FI rank but with lower correlations. In the case of negative correlations with TV jumps we observe that the case of the feature SOURCE_TV_RECORD_HOURS_QTY has almost a negative correlation of 25%.

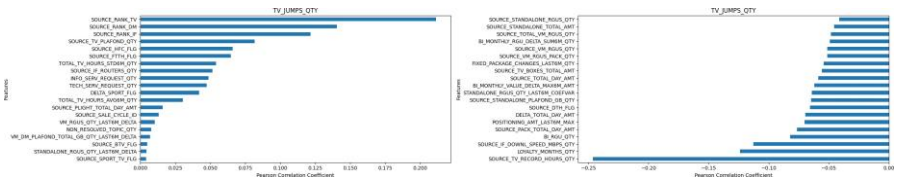


Figure 4.23: Top-20 most (positive and negative) correlated features with the variable TV_JUMPS_QTY.

In Figure 4.24 we notice the FI rank has a positive correlation with the variable FI jumps of around 30%. Also, we observe a negative correlation with the feature download speed quantity of the FI with a value above 30%.

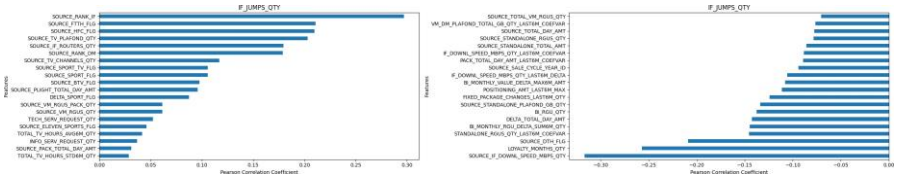


Figure 4.24: Top-20 most (positive and negative) correlated features with the variable IF_JUMPS_QTY.

In the case of features correlated with MV Jumps, we observe in Figure 4.25 a positive correlation with the offer's effort. Due to the price of the MV Cards as these types of services increase the price of effort. As for the negatively correlated features, we do not note any moderately or strong correlation, being the features with a higher correlation the SOURCE_VM_RGUS_QTY and SOURCE_VM_RGUS_PACK_QTY with a value above the 20%.

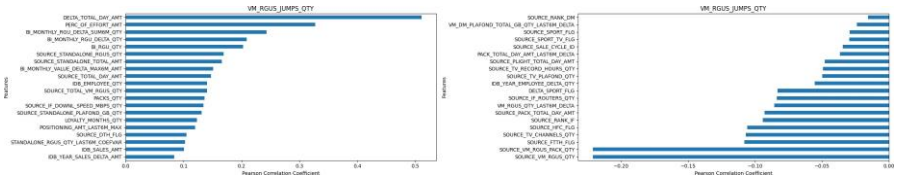


Figure 4.25: Top-20 most (positive and negative) correlated features with the variable VM_RGUS_JUMPS_QTY.

In Figure 4.26 we see that the Rank of MD is the most highly correlated feature with the MD jumps recording a value of around 35%. For negative correlated features, we observe a weak correlation with the variable SOURCE_STANDALONE_PLAFOND_GB_QTY being around -18%.

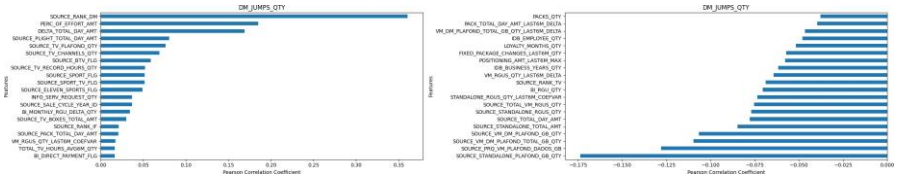


Figure 4.26: Top-20 most (positive and negative) correlated features with the variable DM_JUMPS_QTY.

4.5 Adopted Methodology in Recommender System Approach

In this subsection we present, the recommender system methodology adopted, which consisted in a retrieval model based on the two-tower model architecture.

Retrieval models originate from the Information Retrieval theory, which focuses on de- veloping systems capable of searching and retrieving the relevant information for a given query from a large corpus. Based on this, several retrieval models have been developed, such as the vector space models. These algorithms convert text inputs into vectors and each word represents a dimension.

Therefore, the result will be a very high-dimensional vec- tor. To retrieve the suitable document or documents for a given query, a scoring similar function is computed between the pair query-document vectors [29].

So, combining the retrieval paradigm with the two-tower model architecture, we have an embedding model that performs the retrieval of recommendations over a learned latent space, as shown in Figure 4.27. Typically, retrieval models can search and retrieve candidates over a large corpus of items using techniques to approximate nearest neighbor search.

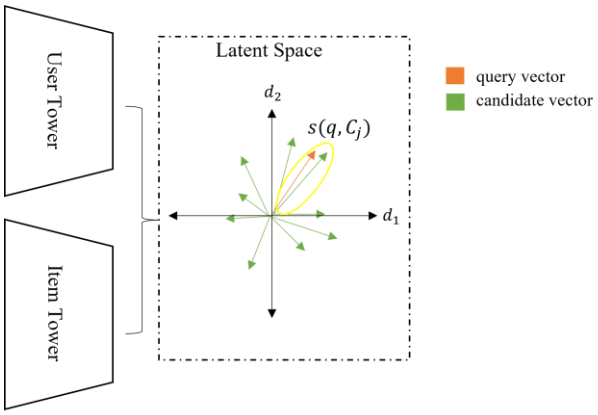


Figure 4.27: Illustrative schema of the functioning of a retrieval model in a 2D latent space.

4.5.1 Item Definition

Regarding the item definition, as exposed in the Related Work section, telco packages are products that differ from any other product recommendation. Since they aggregate several services or products there it arises the need to create an item definition that describes what is being offered.

In our problem, a product can be identified by the composite key: PRICE_INTEGRATION_ID + VM_INTEGRATION_ID + VM_RGUS_QTY + IM_RGUS_QTY + IM_INTEGRATION_ID + IM_RGUS_QTY + PACK_MATRIX_ID, the Table 4.1 presents the definition of each variable.

Feature Name	Definition
PRICE_INTEGRATION_ID	Identifies the fix services (TV, IF and VF) of the packages and corresponding price
VM_INTEGRATION_ID	Identifies the VM integrated in packages
IM_INTEGRATION_ID	Identifies the IM integrated in packages
VM_RGUS_QTY	Defines the number of VM cards offered
IM_RGUS_QTY	Defines the number of IM cards offered
PACK_MATRIX_ID	Identifies the price of the mobile part (VM and IM)

Table 4.1: Definition of the features that compose the item definition.

Nonetheless, this definition produces many unique items that are very similar in terms of offered components and price, combined with a small dataset. Hence, we ended up with a very sparse dataset, where in most of the cases we only have one interaction recorded for that specific item. Therefore, we decided to define the item from a perspective of the number of upgrades per s and the percentual effort of the offer discretized in bins.This led to a new item definition: OF_TV_JUMPS_QTY + OF_IF_JUMPS_QTY + OF_DM_JUMPS_QTY + OF_VM_RGUS_JUMPS_QTY + OF_PERCENTUAL_EFFORT_BINNED. This definition allowed us to tackle the problem of sparsity observed on our dataset. Additionally, by defining an item as the combination of jumps' quantity with the binned percentual effort, the model produces complete recommendations that are insightful regarding the portfolio movements that are more likely to be preferred by the user, including their magni- tude (how many jumps) and their value.

Nevertheless, determining the effort bins is required to define the effort step. Thus, we proceeded with testing different values for the effort step ranging between 0.01 and 0.05. As evaluation methods to assess the best value, we used the evaluation metric recall@k and analyzed the inference results for a sample of customers.

	recall@1	recall@3	recall@10	recall@50	recall@100
pct_effort_step@0.01	0.012	0.037	0.105	0.332	0.443
pct_effort_step@0.02	0.025	0.069	0.199	0.460	0.577
pct_effort_step@0.03	0.039	0.103	0.270	0.544	0.657
pct_effort_step@0.04	0.058	0.148	0.330	0.591	0.701
pct_effort_step@0.05	0.078	0.185	0.365	0.629	0.730

Table 4.2: Evaluation of the effort step for different values.

Analyzing Table 4.2, as expected, a higher effort step leads to an increase if the recall across the ks selected. Since having a larger effort step leads to wider effort intervals, clustering more items. This results in a reduction of the sparsity of the dataset and the error. We only test these effort steps as we argue that having wider effort bins may cause the model to be less sensitive to smaller effort values.

Nonetheless, selecting the right effort step was not obvious. Thus, we simulated an inference scenario for each model trained with the different effort steps. In Figure 4.28, we observe the results of inference for a random client. We have assigned different groups of offers that share the same number of upgrades but vary in effort. The expected is for the same offered services' upgrades that the model should assign a lower score for the proposals with higher effort values. We observe that when the effort step is equal to 0.05 the offers with MD upgrades located on the interval between 5 and 10 are assigned higher scores.

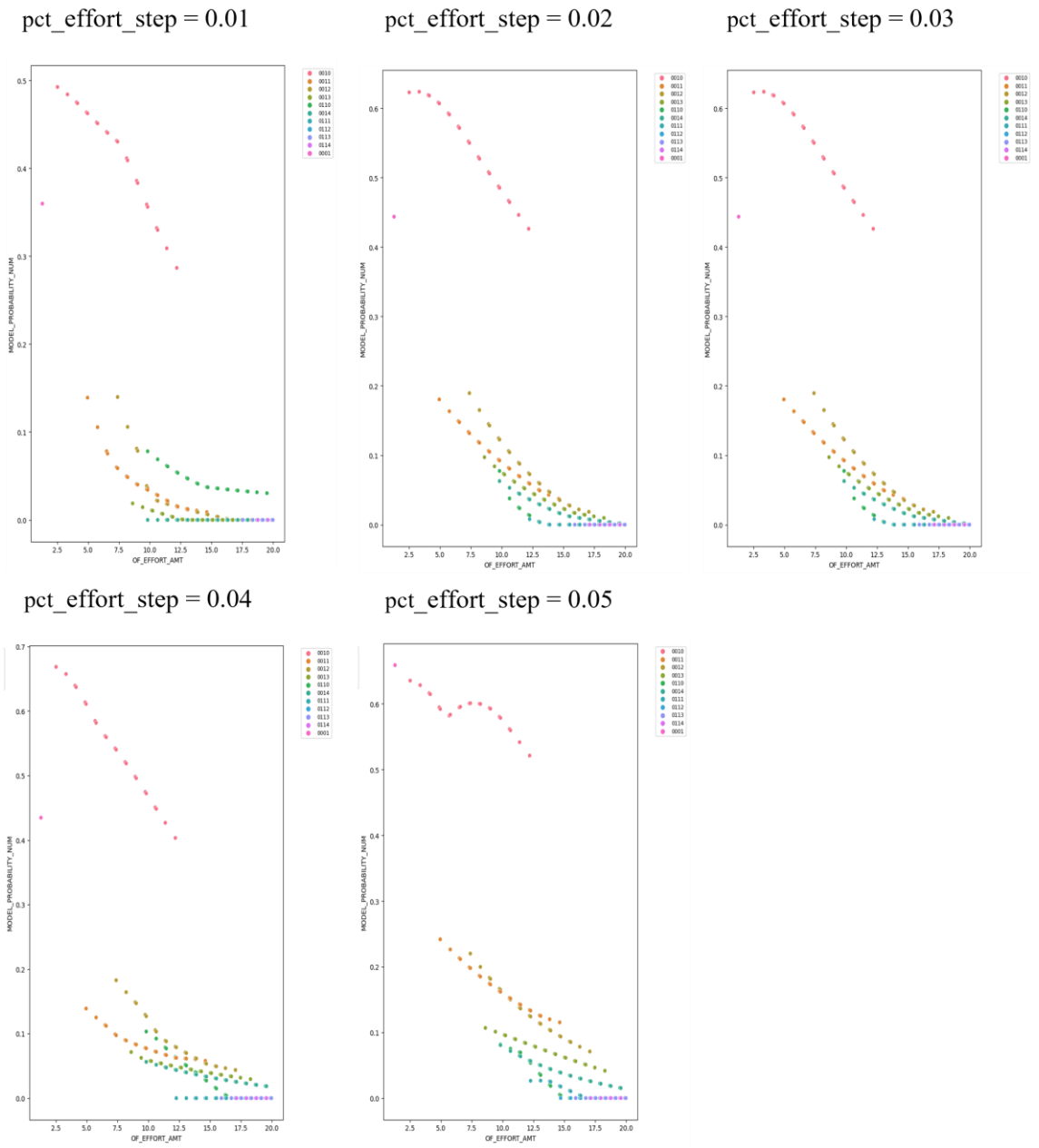


Figure 4.28: Different scenarios of percentual step effort.

Therefore, from analyzing several inference examples supported by the recall@k results, we concluded that the better effort step was 0.04. Thus, we selected that step to define the width of the effort bins. Nevertheless, the decision made was based on the experiments explained above. Moreover, we argue that choosing 0.05 would lead the model, within the same bundle, to select higher offers, which would not correspond to the customer's behavior in a purchasing situation. Finally, with the definition followed we obtained 3,605 unique items in our dataset.

4.5.2 Training Panel

Regarding the building of the Training Panel, this is accomplished through a query, as shown in Figure 4.29. This query takes several substeps, namely:

- Computes the portfolio's changes in subsequent months for each client;

- Enriches with additional information (information about the company, information about the client, historical features, Topics and TV Usage);
- Finally, only the observed changes in the portfolio are kept. Thus, the final dataset corresponds to all portfolio changes of each cycle.

Nonetheless, many portfolio changes can correspond to downgrades of services, and training the model with such observations would result in recommendations of downgrades. Thus, to ensure that the model is only trained with interactions that result in an upgrade, we took some preprocessing steps to be align with the business interests. Furthermore, to have richer item space, we did some feature engineering by creating additional features, such as ratios, derived from the effort and the number of upgrades.

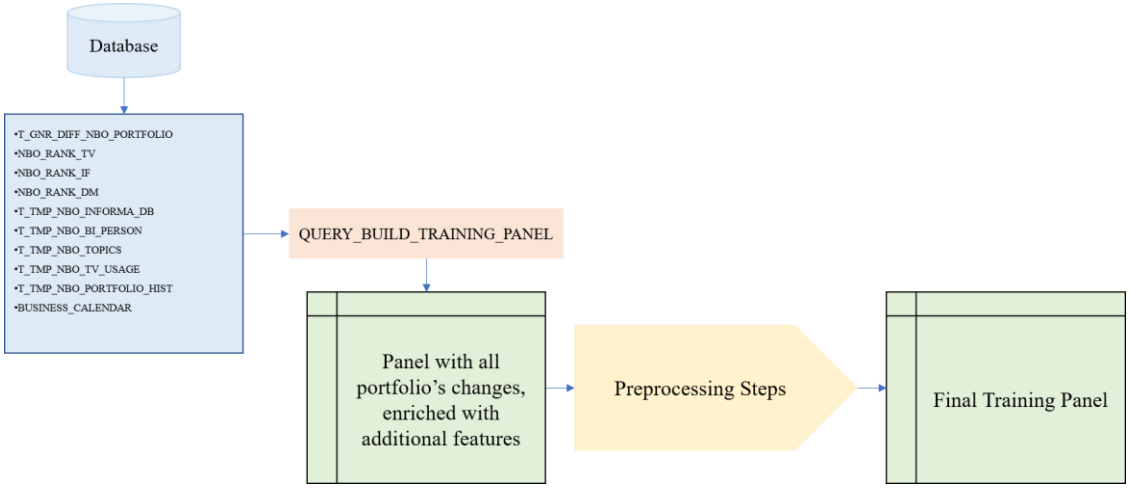


Figure 4.29: Diagram that illustrates the steps taken until the generation of the training panel.

The preprocessing steps consisted of:

1. Removal of observations/interactions that result in a downgrade in one of the components (TV, FI, MD or MV), a downgrade of the typology of the package, (change from a 4P to a 3P), or reduction of the monthly payment;
2. For the observations that offered more than 5 MV cards we limited to 5;
3. Removal of clients that are in debt;
4. Creation of the item definition following the steps mentioned above;
5. Addition of new features resulting of feature interaction between the effort's offer and the jumps' services.

Therefore, the training panel is composed by interactions between users and items, where an interaction corresponds to an upgrade. So an upgrade in our problem is defined as:

- No downgrade is recorded in any of the components (TV, FI, MD or MV);
- It increased or kept the same product typology (e.g. a 3P client kept a 3P or increased to a 4P or 5P);

- At least one jump in one of the components (TV, FI, MD or MV);
- It increased their monthly payment value.

The training panel illustrated in Figure 4.30 has the features related to the customer/user dimension and product/item profile. The client dimension is composed of the client's portfolio. These variables are related to information describing the type of product the client has in its origin, such as the internet download speed, type of box TV, and the quantity of Gigabyte (GB) in data mobile, among others. Moreover, tables *Informa DB* and *BI Person* compose our user feature space. These tables contain features that describe the client from a business perspective, containing attributes such as the Código Atividade Económica (CAE), address, number of employees, and quantity of services, among others.

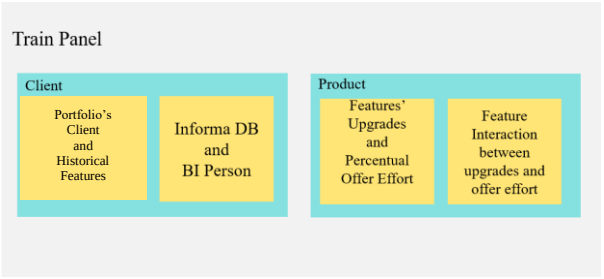


Figure 4.30: Schema of the dimensions that compose the training painel.

The item feature space is described by the services' upgrades, offers' effort, and the ratios resulting from the interactions between the offers' effort and services' jumps. With this, a feature a engineering step allowed us to extract more information and have a richer item dimension.

4.5.3 Model Architecture Adopted

As stated before, the model developed used the two-tower model architecture. This approach required the fine-tuning of the architecture of both towers to produce the best model possible. Thus, we tested different sizes and depths for both towers. The tests were produced with 5-fold Cross-Validation (CV) using the recall@k as an evaluation measure. The architecture that produced the best recall@k from our tests was composed of three hidden layers on the user tower with the sizes of 32, and 16 neurons and on the side of item was two hidden layers both with size 16 (see Appendix B). The final architecture of the model is shown in Figure 4.31.

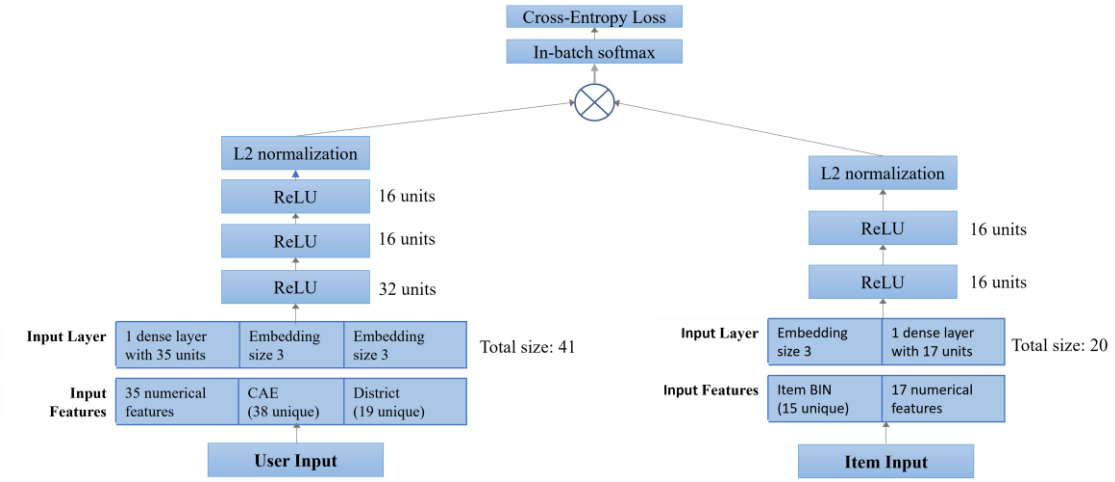


Figure 4.31: Architecture of the implemented model.

Regarding the user tower, the input layer is composed by one dense layer with equal size to the numeric features, 35, and two embedding layers both with size three. The output of these three layers will be concatenated along the first axis. Then, the output of this layer will be inputted into the hidden layers of the user tower, and the L2 Normalization is performed as final operation over the output of the final hidden layer. As for the item tower, the input layer has a dense layer with size 17 and an embedding layer with size 3. Similar to the user tower, the output of these two layers is concatenated along the first axis. This will be the input for the subsequent hidden layers, and the L2 Normalization is computed at the output of the final hidden layer. It is worth mentioning that the L2 Normalization is performed before the computation of the dot product, as [32] explains that the dot product is sensitive to the magnitude of vectors. Thus, this operation improves the recommendations and the results. In both towers we choose as the activation function the ReLU function, incorporating the capacity of the model to capture non-linear relations on the data.

4.5.4 Baseline Model

We implemented a baseline model following a content-based approach. The goal was to establish a base comparison with our Two-Tower Model. Thus, we sought to observe how much increment following the hybrid approach would bring in regards to a simpler method.

The Content-based model was a KNN with the cosine distance as the similarity metric. The recommendations consisted in returning for each query an array of scores with the size of the number of unique items. We sorted this array by descent order to select the top-k items recommended by the model and to compute the recall@k metric. We should stress that we only passed the item features to the model, allowing us to compute the similarity between items that the user interacted with.

4.5.5 Training Stage

Figure 4.32 illustrates the training and evaluation steps. We start by dividing the Training Panel into three datasets, the training, validation, and test datasets. Then we perform the

following preprocessing steps:

1. Transform all values of the feature BI_DISTRICT_DSC to lower case, to eliminate equal values written with different letter cases. Also, we impute for the cases that the value was 'desconhecido' a change to a missing value;
2. Do imputation of missing values for some features related to the client's portfolio we impute 0 because it corresponds to a scenario of non-subscription to a service. For categorical features, we impute missing as the default value, and for the numeric columns we impute the median of the given column;
3. Change data type of numeric and categorical features to reflect the accepted types by the library TensorFlow;
4. Scale the User numeric features using a RobustScaler. We are not performing on the item features because we observe that this leads to worse results on the model;
5. Compute the sampling probability of selecting an item;

When we execute hyperparameter tuning, we save the best combination of values on a JSON file to use at the training step. Otherwise, the default values are used. Then, we train the model, perform the evaluation, and save the model to be used in the inference stage.

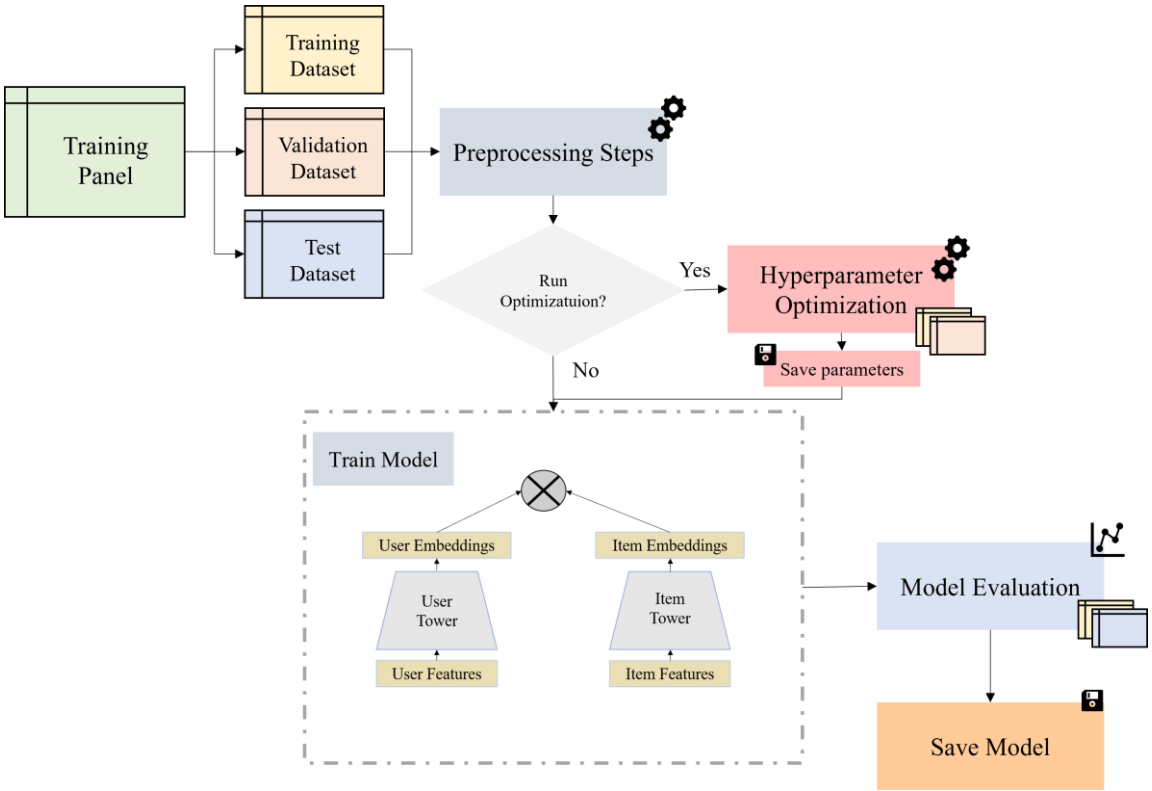


Figure 4.32: Diagram describing the steps that comprise the training stage of the model.

4.5.6 Inference stage

Figure 4.33 shows the steps needed to create the Inference Panel, similar to the training panel. In the inference panel, we will have, for each client, a set of possible offers

originating from a business source. So, for the inference stage, we will use our model to output the similarity score between each pair user-item. As preprocessing steps, we created the additional features for the item tower.

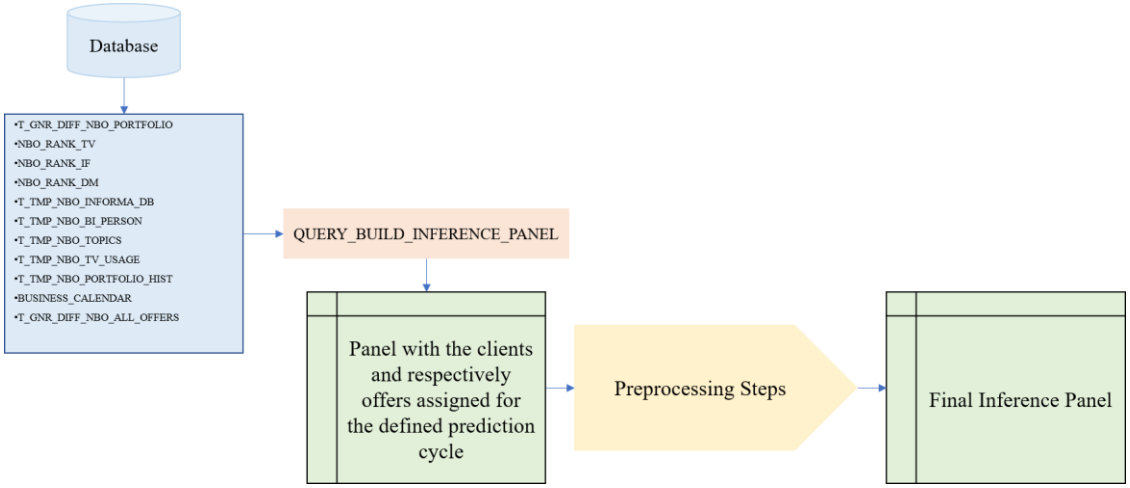


Figure 4.33: Diagram describing the process of building the inference panel.

Figure 4.34 shows the steps that form the inference stage, which are:

1. We partition the Inference Panel according to the loyalty and package typology that the client has;
2. For each partition:
   2.1. We infer the dot product for each pair of users and offer;
   2.2. Compute the expected value, which in this case will be equal to the score of the model;
   2.3. Run the cockpit to rank and select the top-3 offers for each client, ensuring the business restrictions;
   2.4. We write this output on a specific table of the Data Lake;
3. Then, we analyze the results obtained in terms of services' upgrades and the effort distribution across the selected offers having in regard the loyalty period, the package typology, and the rank of the offer, first, third or third recommendation. The decision on whether the results are satisfactory lies on the business side. If the business validates the results, we follow with the next steps. Otherwise, we run again the previous steps, with the tuning of the cockpit.
4. Next, we ensure the business restriction.
5. Finally, we write the final table with the information regarding the clients with the selected offers assigned and the corresponding model score and the ranking of the offer (first, second, third).

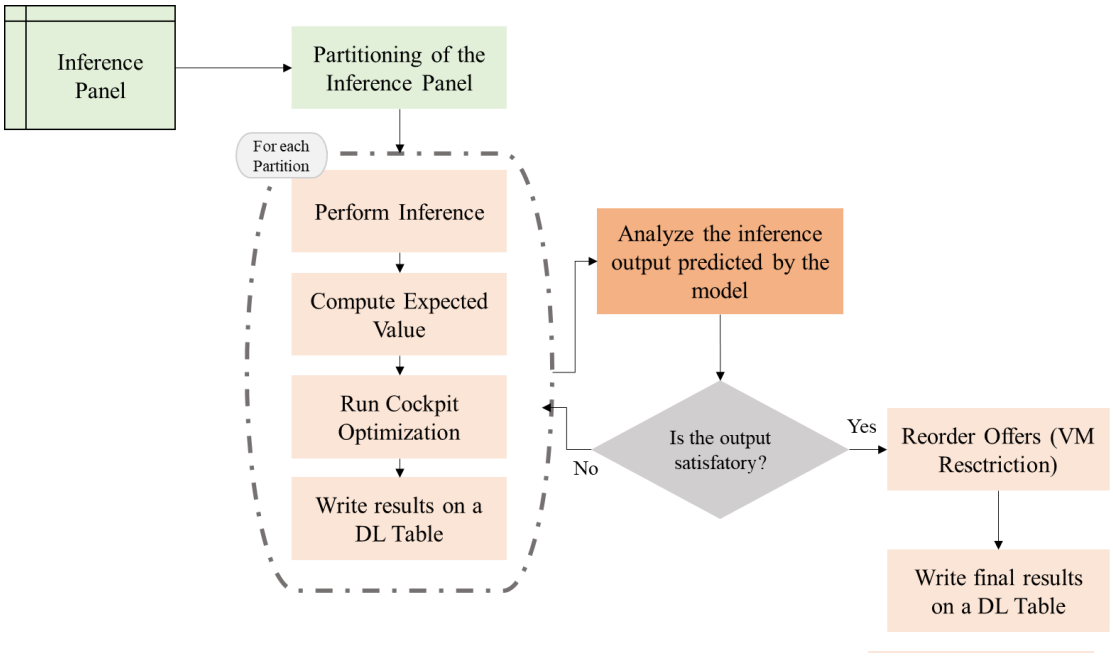


Figure 4.34: Diagram describing the steps that comprise the inference stage.

Chapter 5

Results and Discussion

In this chapter, we present the experimental and A/B testing results for each approach along with a discussion of the outcomes.

5.1 PU Learning Approach Results and Discussion

Regarding the PU approach, we proceeded with several experiments, where we tested our models in distinct scenarios. These experiments varied in noise degree by filtering the dataset using the outcomes field. Also, we tested the methods with data coming from different selling strategies, as the offers' content may be very different between strategies. In total we tested in 5 different conditions, being each situation defined as:

- 1st Experiment: Training with outcomes of all commercial strategies, Cross-Sell and Loyalty, identified as success, callback or failure with offer presented and no offer presented;
- 2nd Experiment: Training with outcomes of all commercial strategies identified as success or failure with offer presented;
- 3rd Experiment: Training with outcomes of all commercial strategies identified as success or client contacted;
- 4th Experiment: Training with outcomes of Loyalty strategy identified as success or client contacted;
- 5th Experiment: Training with outcomes of Cross-Sell strategy identified as success or client contacted.

Regarding Table 5.1[1], we observe that across all experiments the 3 models did achieve similar performances in terms of AUROC. Nonetheless, the best model across all scenarios was the PU-Bagging, obtaining an increment slightly better than the baseline. These increments varied from the 0.6%, for the second and fourth experiments to the 0.8%, for the fifth one. For the case of the Spy method, it did not prove to be better than the baseline, in some cases was slightly worst, in other cases only recorded an increment of 0.1%, which is not significant.

[1]The cross validation performed in the experiments corresponded to splitting the training and validation dataset based on random splits, not considering the time-based dimension associated to the data

	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment	5th Experiment
Model	AUROC	AUROC	AUROC	AUROC	AUROC
Baseline (LGBM)	81.4%	81.3%	81.4%	82.7%	83.1%
PU-Bagging (LGBM)	82.1%	81.9%	82.1%	83.3%	83.9%
Two-Steps (LGBM)	81.5%	81.3%	81.5%	82.4%	83.0%

Table 5.1: AUROC results with the 5 Experiments, with 3-Fold CV, higher AUROC is better.

Considering the Table 5.2, we also notice that the PU-Bagging shows to be the best across all scenarios in terms of PUF-Score. Specifically in the fourth case where it achieved a score of 1.97, whereas the baseline achieved 1.87 and the two-step recorded an even lower score of 1.85.

	1st Experiment	2nd Experiment	3rd Experiment	4th Experiment	5th Experiment
Model	PUF-SCORE	PUF-SCORE	PUF-SCORE	PUF-SCORE	PUF-SCORE
Baseline (LGBM)	1.73	1.79	1.79	1.87	2.14
PU-Bagging (LGBM)	1.82	1.87	1.83	1.97	2.23
Two-Steps (LGBM)	1.72	1.78	1.80	1.85	2.17

Table 5.2: PU-Score results with the 5 Experiments, with 3-Fold CV, higher PUF-Score is better.

Table 5.3 shows the results obtained with the final dataset. With 3-Fold CV performed, where the split of the fold was time-based. The PU-Bagging continued to be the best model, achieving the best results over the other models, as well less overfitting. For the AUROC obtained on the test set, he PU-Bagging achieved a score of 78.74%, and if recorded a PUF-Score of 1.44.

	AUROC Train	AUROC Test	PUF-SCORE Train	PUF-SCORE Test
Baseline (LGBM)	86.28%	78.09%	2.34	1.39
PU-Bagging (LGBM)	85.60%	78.74%	2.07	1.44
Two-Steps (Spy + LGBM)	86.75%	78.26%	2.54	1.38

Table 5.3: Final Results obtained with 3-fold time-based CV.

In Figure 5.1 we that the observe the confusion matrix, recorded with the PU-Bagging, has a higher hit rate of true positives, of 3.85%, against 3.75% of the baseline and 3.68% of the Spy technique. Also, it has a better recall than the other models, which have a lower rate of false negatives of 4.27%, while the other models obtained 4.37% and 4.44%, respectively, for the Baseline and Two-Steps. For the case of the two-steps, it showed to have a better rate of true negatives of 80.48% and a lower rate of false positives of 11.40%. Thus, we argue that having a better recall and, consequently, having a lower false negative rate could be more beneficial for

the business. A higher rate of false negatives could harm the business with missed selling opportunities.

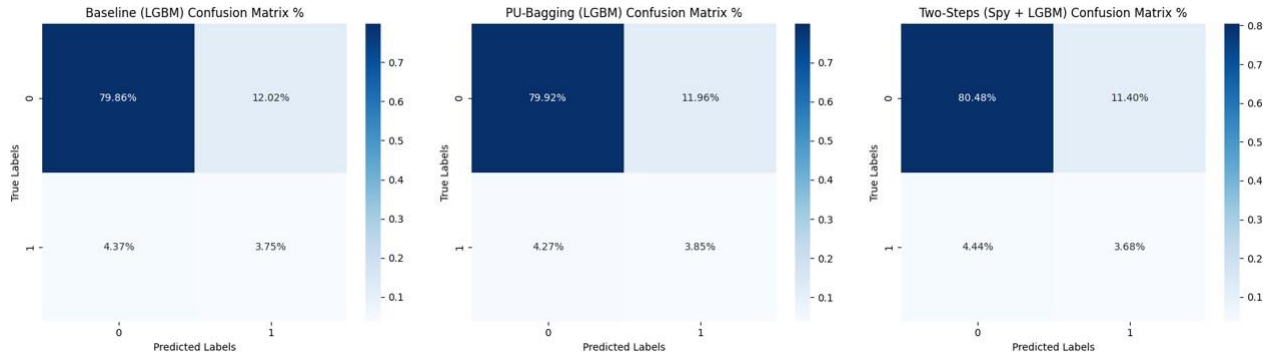


Figure 5.1: Confusion Matrices for each method

In Figure 5.2 illustrates for a sample of 100 clients, the distribution of the models' scores over a range of effort values for the same offer. We observe that for the baseline the model can predict lower scores for higher offers' effort values until approximate 5€. Then, it starts losing some sensitivity, assigning higher scores for offers with efforts over the 7€ until the 12€. Afterward, for offers with values higher than 20€ the model assigns lower scores but cannot distinguish offers with efforts higher than 20€. For the Pu-Bagging, we notice a better behavior, where the model for offers with efforts until 20€ will assign lower scores. However, it behaves similarly to the baseline for efforts over 20€ and cannot distinguish these offers. Notwithstanding, this behavior is expected as it exists few observations with such effort ranges. As for the two-step model it shows a very similar behavior to the baseline.

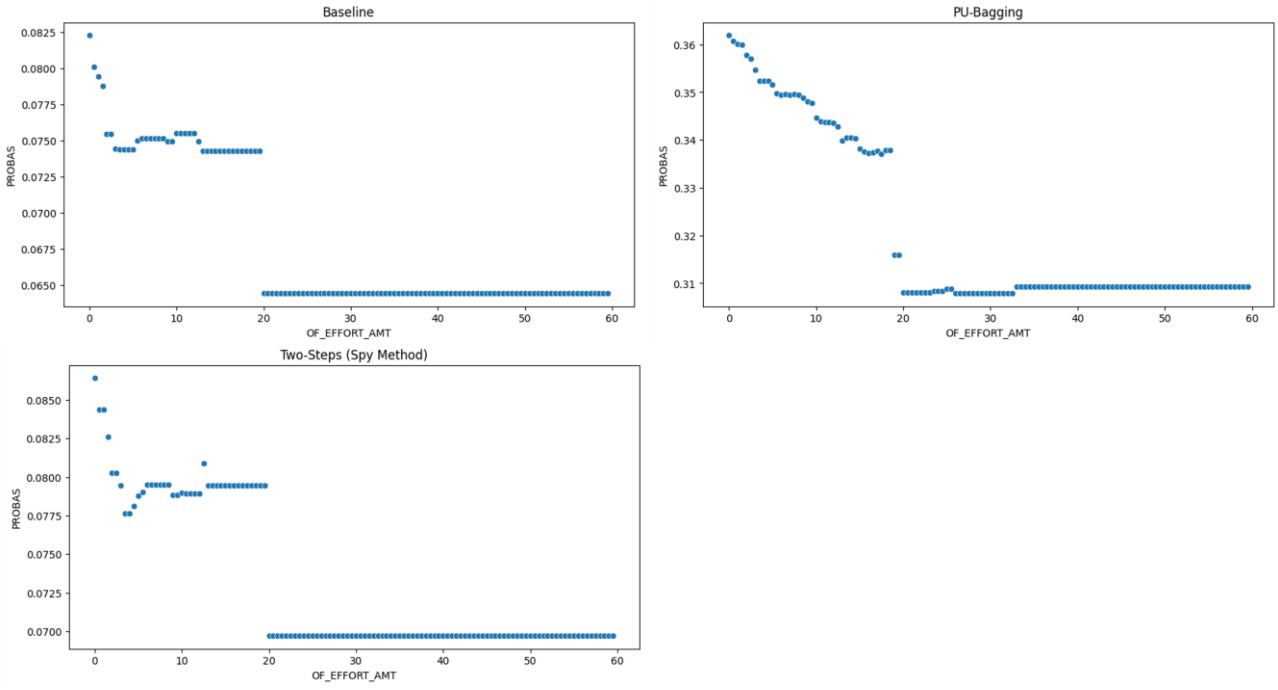


Figure 5.2: Scatter Plot that represents the model score and different offer efforts for the same type of offer. Each plot corresponds to each tested method.

Thus, based on these results obtained over the evaluation metrics and inference we opted for testing the PU-Bagging approach with an A/B testing.

The selection of the top-3 offers to recommend was based on the weighted combination of the propensity model's probability and the offer's effort. Consequently, the output of this combination corresponds to the final ranking score that defines the top-3 recommended offers. Because we are using the model's probability to rank our offers, we need to calibrate this score to achieve a more precise probability.

In Figure 5.3 we compare the uncalibrated and calibrated probabilities of the PU-Bagging model. The calibration was achieved by using the method Platt Scaling. The diagonal line rep- resents the scenario of a perfectly calibrated model, where the score predicted corresponds exactly to the probability of that event. Therefore, scores that tend to be below this diagonal line indicate an optimistic model, because the predicted score corresponds to a lower probability. For predicated scores situated above the diagonal line, we have a scenario of a pessimistic model, where the actual probability is higher than the predicted score. We observed that the calibration had good results, ensuring that most predicted probabilities were close to the diagonal line.

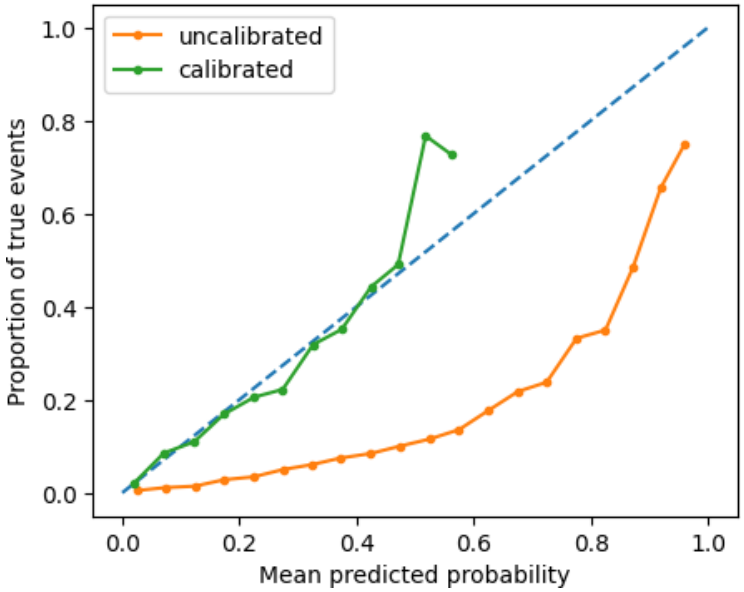


Figure 5.3: Calibrated probas.

Concerning explainability, we used the SHapley Additive exPlanations (SHAP) method to obtain insights into the PU-Bagging Model. We see, in Figure 5.4, that the variable OF_VM_RGUS_JUMPS_QTY tends to have a negative influence on the target. It shows that higher values of this feature lead to more refusals. So, offers with few MV upgrades tend to have a higher chance of being accepted. This may be explained due to the price of a VM card, which increases the final price of the offer. Note that a similar behavior happens for the feature OF_TV_JUMPS_QTY. As for the variable SOURCE_TOTAL_DAY_AMT, which refers to the monthly payment, we observe that lower values lead to a negative response and higher values have

tend to accept the offer. Also, for the case of the attribute OF_EF- FORT_PERC_QTY, we notice that lower values influence the model to predict an activation. The same case happens for the feature OF_EFFORT_AMT. Finally with lower values for the number of services subscribed, BIP_RGU_QTY, the model tends to predict a refusal or, with higher values, an adherence.

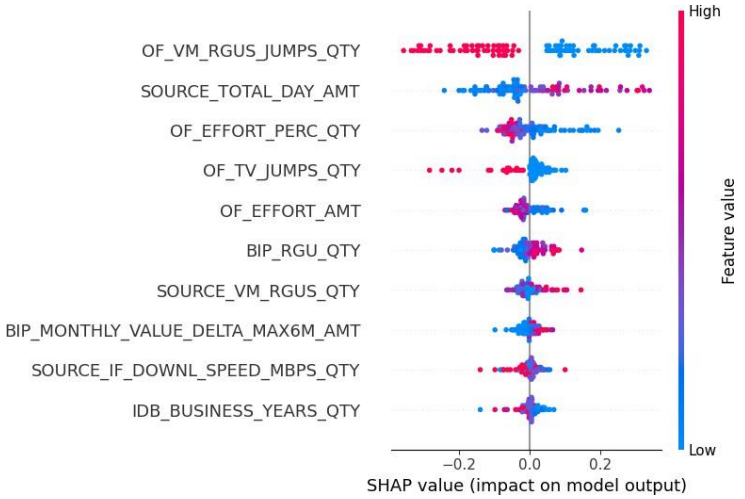


Figure 5.4: Shap Plot for the PU Bagging Model.

So, we decided to perform A/B testing with the PU-Bagging on the second cycle of 2023, C2. We defined 3 groups, the control group that corresponded to the heuristic approach already implemented on the project, the baseline which corresponded to the present model implemented on the NBO and the PU group.

In Figure 5.5 shows the results obtained on the A/B testing per group. On the left, we compare the hit-rate/adherence rate. We observe that the PU-Bagging recorded the best adherence rate among the three groups, reaching a hit rate of 3.41%, when compared with the control group, representing an increasing of 32.17% of adherences.

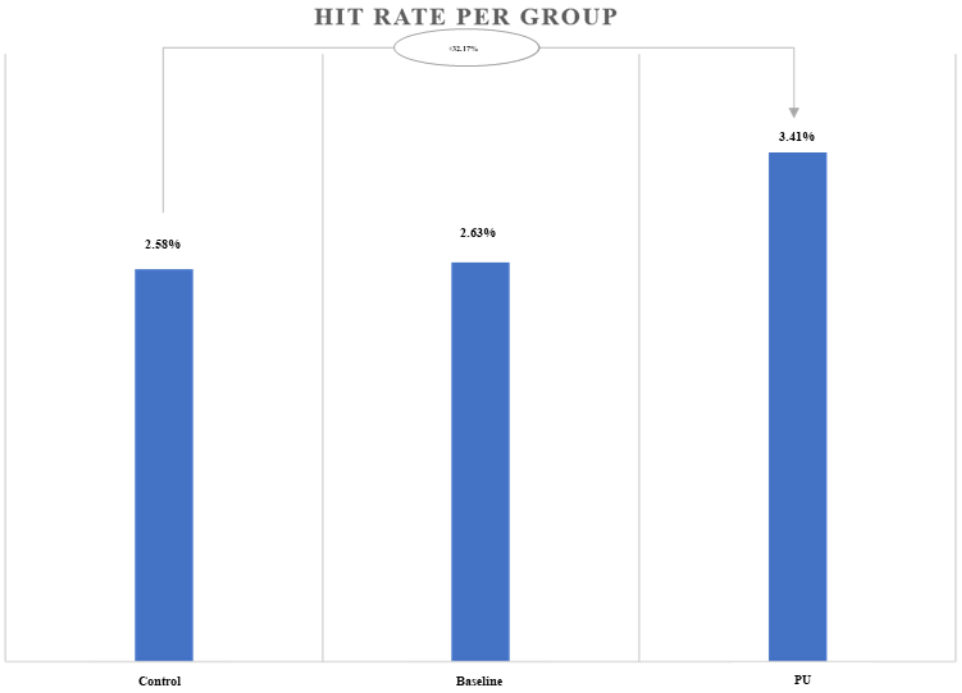


Figure 5.5: A/B testing results obtained on C2 per group.

The Figure 5.6 shows the results obtained on the A/B testing per group discriminated per strategy, loyalty and cross-sell. In both strategies, we observe that the PU group recorded the best hit rate over the other groups. Regarding the loyalty strategy the PU group attained an adherence rate of 2.67% representing an increase of 44.32% adhesions over the control group. The same pattern is recorded for cross-sell strategies, achieving 3.70% of adherence rate being an increase of 32.62% adherence over the control group.

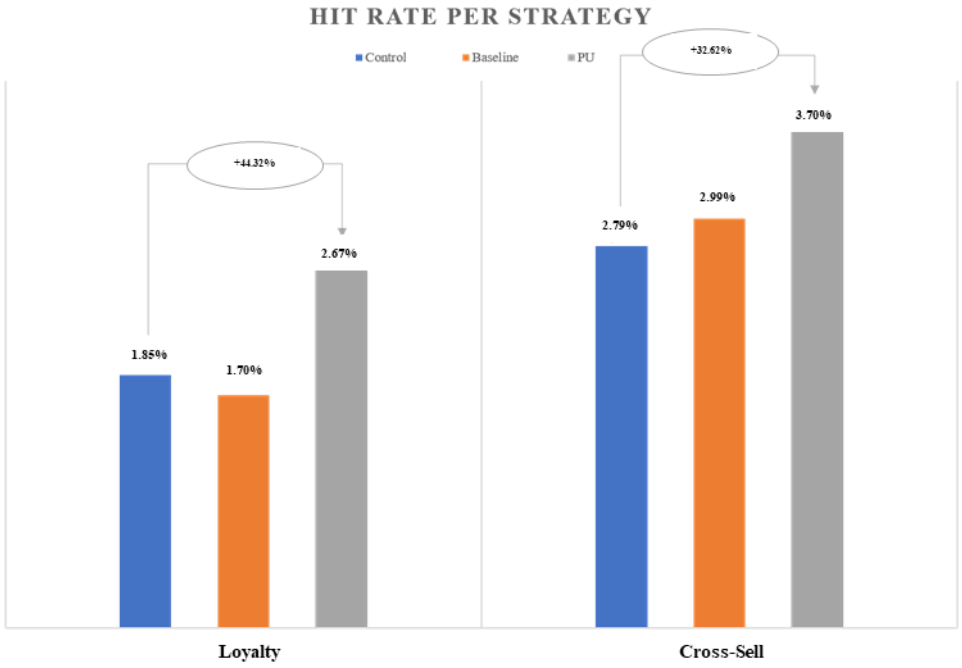


Figure 5.6: A/B testing results obtained on C2 per group, discriminated by strategy.

5.2 Recommender System Approach Results and Discussion

This section presents the results obtained with the recommender systems approach regarding experimental results and A/B Testing.

Table 5.4 exposes the experimental results obtained with the baseline and the Two-Tower model. We conclude that the Two-Tower model achieved a better performance across all recall@k. Where in a recall@1 it recorded a score of 0.074, whereas the Content Model achieved a value of 0.034 and on recall@100 it recorded a score of 0.793, while the Content-Based approach obtained a score of 0.415. Therefore, the Two-Tower model compared with the baseline proved to be better.

	recall@1	recall@3	recall@10	recall@50	recall@100
Content-Based model (kNN)	0.034	0.056	0.098	0.338	0.415
Two-Tower model	0.074	0.200	0.397	0.677	0.793

Table 5.4: Comparison of recall between Content-Based model and Two-Tower model.

Table 5.5 shows the recall@k obtained the train and test set with the Two-Tower model. We notice that, overall, the model did not show signs of overfitting.

	recall@1	recall@3	recall@10	recall@50	recall@100
Train set	0.078	0.218	0.424	0.705	0.801
Test set	0.074	0.200	0.397	0.677	0.793

Table 5.5: Recall@k for training and test sets.

Figure 5.7 illustrates the inference results for the model score assigned for different groups of offers that have different effort values. Each scatter plot corresponds to a client. Hence, we can observe that across all plots, different offers with higher effort tend to be assigned lower scores. When looking within the same offer group, we notice the same behavior of assigning lower scores to subsequent offers with higher effort values.

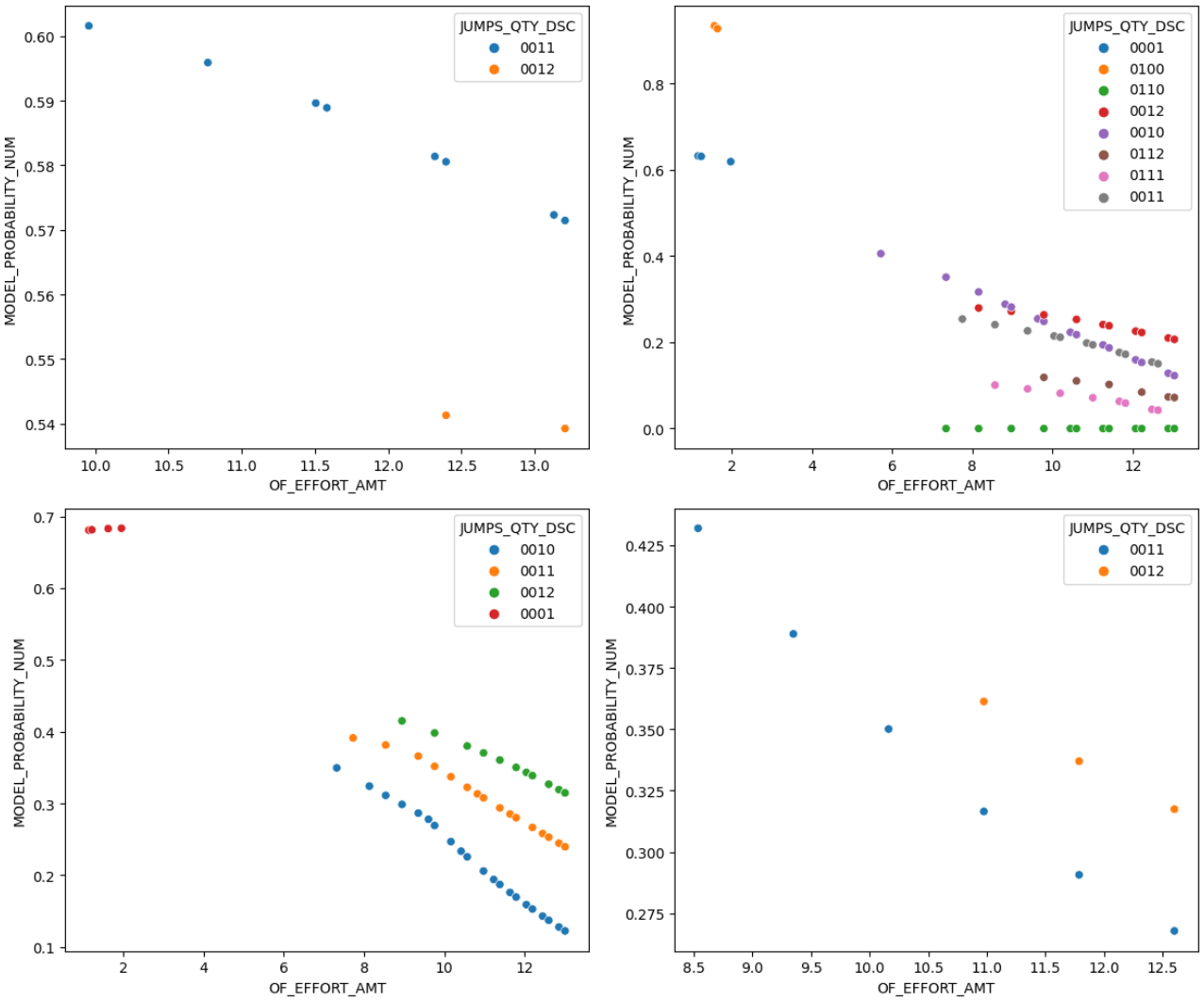


Figure 5.7: Scatter Plot that represents the model score and different offer efforts for the same type of offer.

In Figure 5.8 we observe the model's score variation regarding the offer's effort across the three offers recommended by the model during the inference stage for clients with a loyalty period above the 9 months and clients with a loyalty period less or equal or less to 9 months. As a business requirement, the first offer should have effort values always higher than the second and third offers, as observed in Figure 5.8. Nonetheless, around 30% of the first offers for customers with a loyalty above the 9 months of have proposals on the range or below of the 2€. For the case of second offer the number of offers with that price tag increases as for the third offers. For clients with loyalty less or equal than 9 months, these efforts tend to be lower, especially for the second and third offers. For the case of the first offer it also tend to be lower, with more than 40% of the clients having an offer with an effort below 2€.

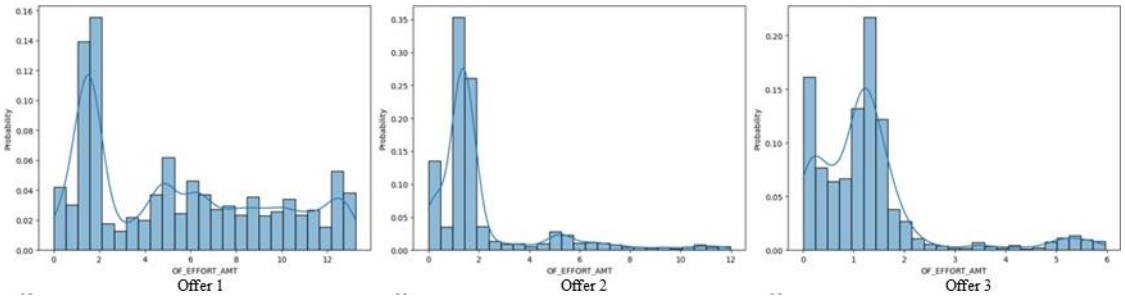


Figure 5.8: Offer Effort distribution for first, second and third recommender offers for clients with loyalty greater than 9 months.

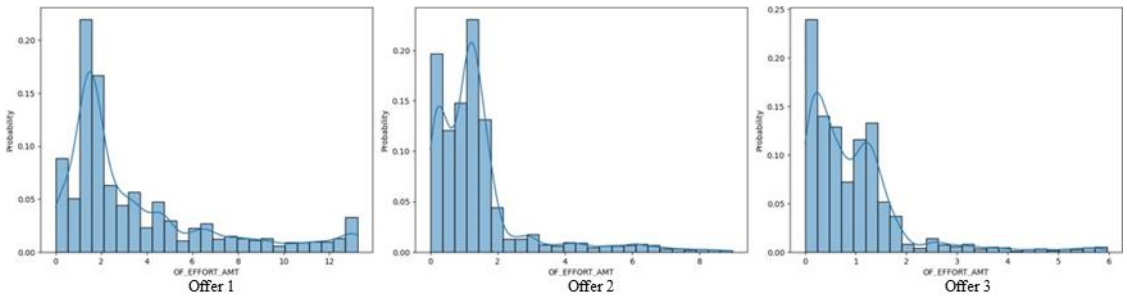


Figure 5.9: Offer Effort distribution for first, second and third recommender offers for clients with loyalty less or equal to 9 months.

The A/B ocurred place on the eighth business cycle and consisted of three groups. The control group, which refers to the heuristic model, the "as is" approach, corresponded to the current propensity model implemented on the NBO. Then, we had the Two-Tower group that consisted on the Two-Tower model to be tested.

In terms of general statistics, it was allocated 17,747 clients, from which 10,052 the company could contact. Over approximately 10,000 clients, around 37% reached a decision, recording an overall hit rate [2] of around 9%. In a more detailed view, the control group achieved a decision rate of 39,1%, the "as is" group recorded a decision rate of 33,3%, and the Two-Tower group attained a decision rate of 35,3%.

As for the hit rate obtained in each group, we observe Figure 5.10, which shows that the recommender system approach achieved a better hit rate over the other groups, of 10,57%, recording an increase of around 18,75% of adhesions over the control group.

[2]The adhesion or hit rate is computed as the number of successes over the number of decisions.

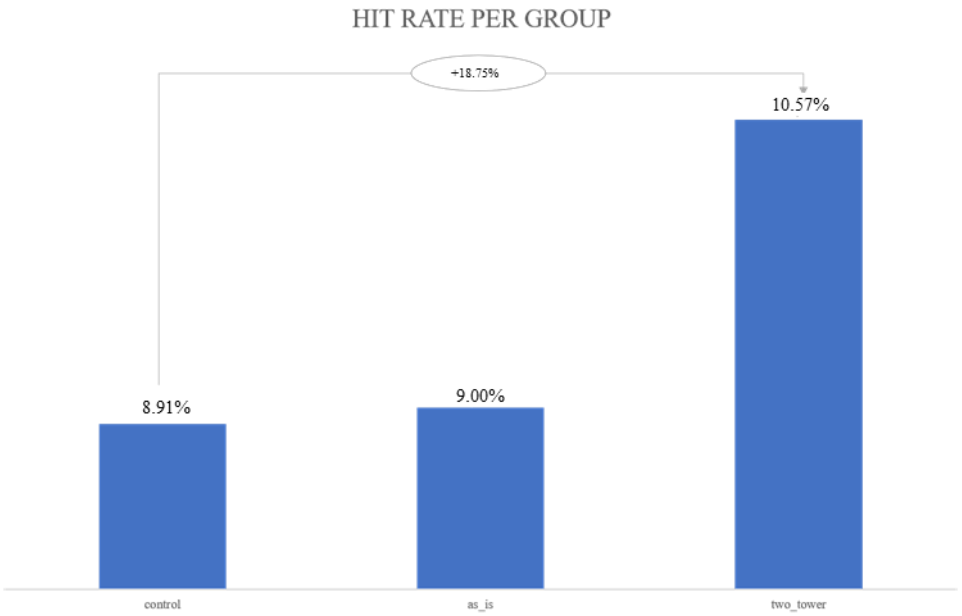


Figure 5.10: Adhesion rate obtained from the A/B testing per each group in C8.

Regarding the hit rate obtained in each campaign, we observe from Figure 5.11 that the Two-Tower group recorded an adherence rate superior to the control group. Also, two of the three campaigns were better than the current NBO model. Thus, the Two-Tower model recorded an adhesion rate of 11.30%, on the campaign O2O | Add More Card and a 10.99% of hit rate on O2O | Loyalty. It is worth mentioning, that for the campaign O2O | Loyalty, which refers to the loyalty's campaign, the recommender system approach recorded an increasing of 73.34% of adherences over the "as is" group.

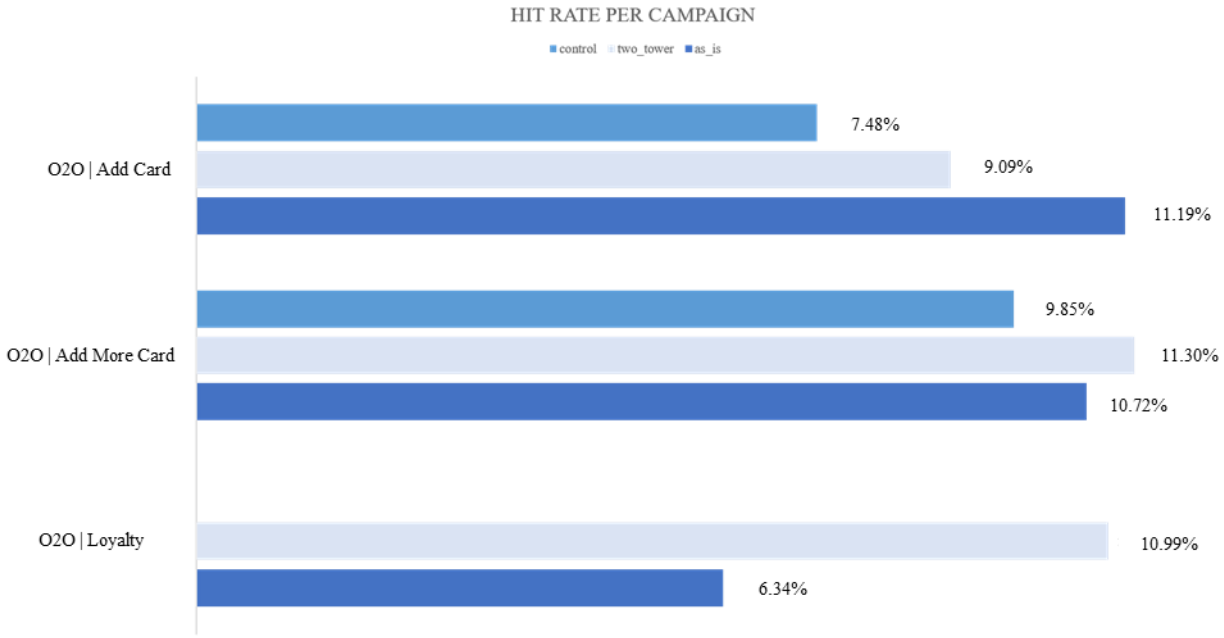


Figure 5.11: Adhesion rate obtained from the A/B testing per each group in C8, discriminated per campaign.

Finally, we conclude that the recommender system approach revealed great potential. Regarding adhesion rate, it was the best model of the three groups. It allows the company to learn and model the costumers' preferences and needs, letting the organization recommend and propose offers more aligned with the customers' tastes. Consequently, this translates into an increase in profit. Furthermore, this type of model allowed us to work with a dataset with some issues like the data sparsity and the small size, but due to learning user and item representations, we could overcome this.

Chapter 6

Conclusion and Future Work

The following chapter presents the conclusion of the project and the future work as improvement guidelines.

6.1 Conclusion

Regarding the PU Learning approach, we concluded that the PU-Bagging method proved to be a better method in terms of theoretic results and A/B testing. This improvement might be explainable, by the bagging technique that allows reducing the variance inherent to PU problems and combining the power of powerful models. Furthermore, it also addresses the unbalanced dataset problems. By sampling only from the unlabeled dataset, we can to train with all the positives and a fraction of unlabeled observations. Regarding the two-step approach, concerning the experimental results we did not observe any improvement over the baseline. We argue that the reason was that machine learning methods require high amounts of data, and with this method, we are reducing the dataset size. The next step would be to repeat this experiment with a karget dataset.

For the Two-Tower model, we conclude that in terms of experimental results, it achieved a satisfactory performance on the recall@k metric. As for A/B testing, the Two-Tower achieved good results for the hit rate, outperforming the control group in all campaigns and the current analytical model in two of the three campaigns. Also, we highlight the performance attained on the fidelization campaign, which shows the potential of this model to be used in loyalty strategies.

6.2 Future Work

For Future Work, one of the main challenges encountered with the Two-Tower Model approach was the small dataset. Therefore, using techniques of transfer learning might mitigate the impact of this problem. The transfer learning could be done by training a first model on the Business-to-Consumer (B2C) segment side, which operates within the universe of big data, with millions of clients. This first model could then be used as a starting point for the B2B model.

In addition, the model used did not consider the sequential dimension present in the data. In general terms, a client adheres solely to a telecommunication bundle subscribing for a loyalty period

of 2 years. Therefore, each time a purchase occurs it represents a new state of the customer, which can be used as a sequential model of the user-item relationship.

Moreover, as we have seen the cockpit has the functions of ensuring business rules, and maximize the revenue. Therefore, the recommendations proceeded by the model should not only focus on recommending items more aligned with the preferences of the customers, it also should also try to optimize the expected revenue. Consequently, an approach based on a multi-objective recommender system could be developed to answer this question.

Bibliography

[1] Charu C. Aggarwal. “An Introduction to Recommender Systems”. In: Recommender Systems The Textbook. Ed. by Charu C.Editor Aggarwal. 1st ed. Springer Cham, 2016, pp. 1–28.

[2] Jessa Bekker and Jesse Davis. “Learning from positive and unlabeled data: a survey”. In: Machine Learning 109 (4 Apr. 2020), pp. 719–760. ISSN: 15730565. DOI: 10.1007/s10994-020-05877-5.

[3] Pavel. Berkhin et al. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2007, p. 1060. ISBN: 9781595936097.

[4] Robin Burke. In: User Modeling and User-Adapted Interaction 12.4 (2002), pp. 331–370. DOI: 10.1023/a:1021240730564.

[5] Luis M. de Campos et al. “Positive unlabeled learning for building recommender systems in a parliamentary setting”. In: Information Sciences 433-434 (Apr. 2018), pp. 221–232. ISSN: 00200255. DOI: 10.1016/j.ins.2017.12.046.

[6] Jia Lue Chen et al. “PU Active Learning for Recommender Systems”. In: Neural Processing Letters 53 (5 Oct. 2021), pp. 3639–3652. ISSN: 1573773X. DOI: 10.1007/s11063-021-10496-9.

[7] Francois Chollet et al. Keras. 2015. URL: https://github.com/fchollet/keras.

[8] Christina Christakou et al. A Movie Recommender System Based on Semi-supervised Clustering. 2005. URL: www.imdb.com.

[9] Peter A. Flach. “Chapter 7 - Linear Models”. In: Machine learning: the art and science of algorithms that make sense of data. Cambridge University Press, 2017, pp. 194–230.

[10] Aurélien Géron. Hands-on machine learning with scikit-learn, keras and tensorflow: Concepts, tools, and techniques to build Intelligent Systems. 2nd. OReilly, 2019.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. http://www.deeplearningbook.org. MIT Press, 2016.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. “Collaborative Filtering for Implicit Feedback Datasets”. In: 2008 Eighth IEEE International Conference on Data Mining. 2008, pp. 263–272. DOI: 10.1109/ICDM.2008.22.

[13] Kristen Jaskie and Andreas Spanias. “Positive And Unlabeled Learning Algorithms And Applications: A Survey”. In: 2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA). 2019, pp. 1–8. DOI: 10.1109/IISA.2019.8900698.

[14] Kristen Jaskie et al. Positive Unlabeled Learning Optimization and Evaluation. 2021.

[15] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. "Comparison of Implicit and Explicit Feedback from an Online Music Recommendation Service". In: Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems. HetRec '10. Barcelona, Spain: Association for Computing Machinery, 2010, pp. 47–51. ISBN: 9781450304078. DOI: 10.1145/1869446.1869453. URL: https://doi.org/10.1145/1869446.1869453.

[16] Liwei Jiang et al. "Improving Positive Unlabeled Learning: Practical AUL Estimation and New Training Method for Extremely Imbalanced Data Sets". In: (Apr. 2020). URL: http://arxiv.org/abs/2004.09820.

[17] Guolin Ke et al. "Lightgbm: A highly efficient gradient boosting decision tree". In: Advances in neural information processing systems 30 (2017), pp. 3146–3154.

[18] Max Kuhn and Kjell Johnson. Applied predictive modeling. Springer, 2013.

[19] Miron B. Kursa, Aleksander Jankowski, and Witold R. Rudnicki. "Boruta - A system for feature selection". In: Fundamenta Informaticae 101 (4 2010), pp. 271–285. ISSN: 01692968. DOI: 10.3233/FI-2010-288.

[20] Nikolaos Lamprou and Giovanna Miritello. "A neural embedding approach to recommender systems in telecommunication". In: POSTER (2017), p. 63.

[21] Bing Liu, Philip S Yu, and Xiaoli Li. Partially Supervised Classification of Text Documents Wee Sun Lee. 2002.

[22] Martn Abadi et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[23] Pawel Matuszyk and Myra Spiliopoulou. "Stream-based semi-supervised learning for recommender systems". In: Machine Learning 106 (6 June 2017), pp. 771–798. ISSN: 15730565. DOI: 10.1007/s10994-016-5614-4.

[24] Tomas Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality. 2013. arXiv: 1310.4546 [cs.CL].

[25] F. Mordelet and J. P. Vert. "A bagging SVM to learn from positive and unlabeled examples". In: Pattern Recognition Letters 37 (1 Feb. 2014), pp. 201–209. ISSN: 01678655. DOI: 10.1016/j.patrec.2013.06.010.

[26] Yassine Ouali, Céline Hudelot, and Myriam Tami. "An Overview of Deep Semi-Supervised Learning". In: (June 2020). URL: http://arxiv.org/abs/2006.05278.

[27] Cong Dan Pham et al. "A recommendation system for offers in telecommunications". In: Institute of Electrical and Electronics Engineers Inc., Jan. 2021, pp. 302–306. ISBN: 9781728154701. DOI: 10.1109/ICCE48956.2021.9352111.

[28] Congming Shi et al. "Recommender System for Telecom Packages Based on the Deep & Cross Network". In: (2021). DOI: 10.21203/rs.3.rs-889572/v1.

[29] Amit Singhal et al. "Modern information retrieval: A brief overview". In: IEEE Data Eng. Bull. 24.4 (2001), pp. 35–43.

[30] Guangxin Su, Weitong Chen, and Miao Xu. Positive-Unlabeled Learning from Imbalanced Data. 2021.

[31] Ji Yang et al. “Mixed Negative Sampling for Learning Two-tower Neural Networks in Recommendations”. In: 2020.

[32] Xinyang Yi et al. “Sampling-Bias-Corrected Neural Modeling for Large Corpus Item Recommendations”. In: Proceedings of the 13th ACM Conference on Recommender Systems. RecSys ’19. Copenhagen, Denmark: Association for Computing Machinery, 2019, pp. 269–277. ISBN: 9781450362436. DOI: 10.1145/3298689.3346996. URL: https://doi.org/10.1145/3298689.3346996.

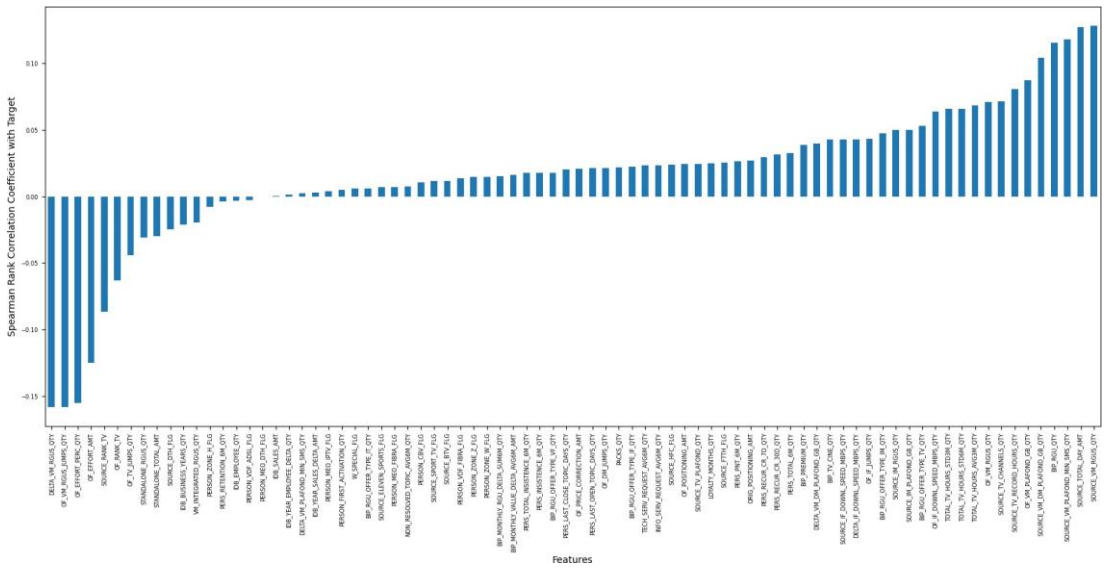
[33] Shuai Zhang et al. “Deep Learning based Recommender System: A Survey and New Perspectives”. In: (July 2017). DOI: 10.1145/3285029. URL: http://arxiv.org/abs/1707.07435%20http://dx.doi.org/10.1145/3285029.

[34] Yao Zhou et al. “GAN-based Recommendation with Positive-Unlabeled Sampling”. In: (Dec. 2020). URL: http://arxiv.org/abs/2012.06901.

Appendices

Appendix A

Correlation Between Dependent Variables and Independent Variable



Appendix B

Recall@k at 5-Fold CV for different architecture configurations of each tower

Batch Size	Candidate	Embedding size	N ReLU layers User	N ReLU layers Item	Hidden Layer Units User	Hidden Layer Units Item	train_recall@1	test_recall@1	train_recall@5	test_recall@5	train_recall@10	test_recall@10	train_recall@50	test_recall@50	train_recall@100	test_recall@100
2104	128	3	2	2	[16,16]	[16,16]	7,33%	7,18%	18,59%	18,09%	38,76%	38,19%	64,23%	63,15%	74,57%	73,71%
2104	128	3	2	2	[32,16]	[16,16]	7,70%	7,50%	18,88%	18,44%	39,08%	38,27%	63,78%	62,95%	74,21%	73,73%
2104	128	3	3	2	[32,16,16]	[16,16]	7,82%	7,58%	19,13%	18,68%	39,27%	38,64%	64,32%	63,35%	74,71%	73,73%
2104	128	3	3	3	[32,16,16]	[16,16,16]	7,59%	7,40%	18,91%	18,31%	38,64%	37,88%	63,95%	63,27%	74,30%	73,48%
2104	128	3	3	3	[32,16,8]	[16,16,8]	6,70%	6,49%	17,54%	17,10%	36,14%	35,65%	60,02%	59,13%	71,30%	70,61%
2104	128	3	3	2	[32,16,8]	[16,8]	7,19%	6,89%	18,11%	17,68%	37,20%	36,67%	61,38%	61,06%	72,31%	71,75%

Appendix C

Comparison between baseline, PU-Bagging and Two-Steps

MODEL	TRAIN_SIZE	NEGATIVE_TRAIN_SIZE	POSITIVE_TRAIN_SIZE	POSITIVE_TRAIN_RATE	NEGATIVE_TEST_SIZE	POSITIVE_TEST_SIZE	POSITIVE_TEST_RATE	AUROC_TRAIN	AUROC_TEST	PUF_SCORE_TRAIN	PUF_SCORE_TEST
Two-Steps (Spy + LGBM)	79577	73483	6094	7,66%	20837	1758	7,78%	87,85%	80,90%	2,79	1,61
Two-Steps (Spy + LGBM)	93216	86288	6928	7,43%	17780	1847	9,41%	86,36%	78,28%	2,38	1,36
Two-Steps (Spy + LGBM)	103663	95811	7852	7,57%	22513	1797	7,39%	86,04%	75,61%	2,45	1,15
PU-Bagging (LGBM)	81757	75663	6094	7,45%	20837	1758	7,78%	86,02%	81,29%	2,12	1,73
PU-Bagging (LGBM)	93816	86888	6928	7,38%	17780	1847	9,41%	85,58%	78,70%	2,10	1,35
PU-Bagging (LGBM)	104352	96500	7852	7,52%	22513	1797	7,39%	85,18%	76,22%	2,01	1,24
Baseline (LGBM)	81757	75663	6094	7,45%	20837	1758	7,78%	86,79%	81,02%	2,42	1,62
Baseline (LGBM)	93816	86888	6928	7,38%	17780	1847	9,41%	86,18%	77,89%	2,37	1,31
Baseline (LGBM)	104352	96500	7852	7,52%	22513	1797	7,39%	85,85%	75,36%	2,23	1,26

Appendix D

Model Summary for User and Item Towers

Model: "sequential_70"

Layer (type)	Output Shape	Param #
user_model_10 (UserModel)	multiple	1440
dense_71 (Dense)	multiple	1344
batch_normalization_30 (BatchNormalization)	multiple	128
re_lu_30 (ReLU)	multiple	0
dense_72 (Dense)	multiple	528
batch_normalization_31 (BatchNormalization)	multiple	64
re_lu_31 (ReLU)	multiple	0
dense_73 (Dense)	multiple	272
batch_normalization_32 (BatchNormalization)	multiple	64
re_lu_32 (ReLU)	multiple	0
l2_norm_layer_20 (L2NormLayer)	multiple	0

Total params: 3,840
Trainable params: 3,712
Non-trainable params: 128

Model: "sequential_67"

Layer (type)	Output Shape	Param #
item_model_9 (ItemModel)	multiple	354
dense_68 (Dense)	multiple	336
batch_normalization_28 (BatchNormalization)	multiple	64
re_lu_28 (ReLU)	multiple	0
dense_69 (Dense)	multiple	272
batch_normalization_29 (BatchNormalization)	multiple	64
re_lu_29 (ReLU)	multiple	0
l2_norm_layer_19 (L2NormLayer)	multiple	0

Total params: 1,090
Trainable params: 1,026
Non-trainable params: 64

Appendix E

Results of inference Two-Tower Model discriminated by Package Type, Loyalty Period and Offer number recommended

COMP_P_PACK_DSC	LOYALTY_MONTHS_QTY_BUCKETS_DSC	MODEL_OFFER_ID	N_CLIENTS	PCT_CLIENTS	OF_EFFORT_AMT	OF_TV_JUMPS_FLG	OF_IF_JUMPS_FLG	OF_DM_JUMPS_FLG	OF_VM_RGUS_JUMPS_FLG	Nr Total de Saltos	Esforço p/ salto
2	<=-4	1	1694	1	5,60	17,65%	56,79%	48,76%	49,17%	1,8	3,15
2	<=-4	2	374	0,221	2,17	19,25%	71,12%	35,83%	21,66%	1,8	1,22
2	<=-4	3	76	0,045	1,18	48,68%	32,89%	80,26%	52,63%	2,6	0,50
2	]-4, 4]	1	1124	1	5,67	22,42%	40,93%	62,01%	63,26%	2,0	2,79
2	]-4, 4]	2	293	0,261	2,43	40,61%	49,49%	62,12%	33,11%	2,5	1,05
2	]-4, 4]	3	73	0,065	0,97	45,21%	32,88%	73,97%	28,77%	2,8	0,43
2	]4, 9]	1	1112	1	4,96	21,67%	51,80%	56,92%	44,78%	1,9	2,44
2	]4, 9]	2	294	0,264	3,28	43,88%	11,90%	77,55%	44,90%	2,3	1,65
2	]4, 9]	3	145	0,13	2,30	67,59%	12,41%	68,28%	49,66%	2,3	1,20
2	]9, 15]	1	1959	1	5,49	18,12%	36,19%	66,62%	54,26%	2,0	2,94
2	]9, 15]	2	839	0,428	3,83	42,43%	29,20%	63,05%	48,27%	2,4	1,75
2	]9, 15]	3	271	0,138	2,45	41,33%	16,61%	69,00%	60,15%	2,6	1,05
2	>20	1	2215	1	6,02	0,00%	21,94%	71,83%	70,88%	1,7	3,42
2	>20	2	496	0,224	1,64	0,00%	37,10%	55,24%	17,14%	1,1	1,43
2	>20	3	96	0,043	0,83	0,00%	15,63%	18,75%	77,08%	1,1	0,80
2	]15, 20]	1	1665	1	4,86	0,00%	32,43%	59,16%	60,12%	1,6	3,01
2	]15, 20]	2	408	0,245	1,62	0,00%	37,50%	47,79%	35,29%	1,3	1,28
2	]15, 20]	3	62	0,037	0,86	0,00%	25,81%	24,19%	85,48%	1,4	0,74
3	<=-4	1	4634	1	2,39	47,24%	72,64%	41,99%	31,70%	2,2	1,24
3	<=-4	2	2009	0,434	0,88	36,29%	49,88%	48,18%	34,49%	2,2	0,50
3	<=-4	3	452	0,098	0,62	31,86%	61,73%	60,40%	46,90%	2,8	0,29
3	]-4, 4]	1	2839	1	3,28	25,54%	45,83%	58,93%	49,63%	1,9	1,84
3	]-4, 4]	2	1098	0,387	1,03	53,64%	41,62%	27,87%	15,39%	1,7	0,76
3	]-4, 4]	3	173	0,061	0,80	36,99%	40,46%	57,80%	40,46%	2,1	0,44
3	]4, 9]	1	3085	1	4,22	23,11%	48,14%	57,54%	46,00%	1,9	2,31
3	]4, 9]	2	1146	0,371	1,89	42,67%	31,59%	37,00%	26,44%	1,6	1,34
3	]4, 9]	3	222	0,072	1,62	51,35%	14,41%	63,06%	49,55%	1,9	0,99
3	]9, 15]	1	4067	1	4,37	14,61%	42,39%	61,57%	50,09%	1,9	2,33
3	]9, 15]	2	1420	0,349	2,10	36,41%	29,72%	50,07%	38,52%	1,8	1,29
3	]9, 15]	3	310	0,076	1,54	31,61%	22,90%	72,26%	63,55%	2,2	0,83
3	>20	1	3800	1	6,33	0,55%	14,05%	84,61%	84,32%	1,9	3,32
3	>20	2	1031	0,271	1,55	50,73%	29,78%	23,18%	9,89%	1,2	1,43
3	>20	3	139	0,037	1,03	32,37%	46,04%	25,90%	28,06%	1,4	0,96
3	]15, 20]	1	2946	1	5,43	3,53%	22,17%	73,22%	73,93%	1,8	3,01
3	]15, 20]	2	756	0,257	1,42	46,96%	25,93%	34,26%	28,04%	1,5	1,19
3	]15, 20]	3	68	0,023	0,87	29,41%	52,94%	42,65%	60,29%	2,2	0,67
4	<=-4	1	2435	1	2,66	3,78%	51,01%	95,15%	33,59%	2,9	1,22
4	<=-4	2	1393	0,572	0,88	14,43%	34,24%	89,88%	45,94%	2,8	0,38
4	<=-4	3	629	0,258	0,55	37,36%	40,06%	78,54%	65,50%	2,9	0,23
4	]-4, 4]	1	2515	1	2,74	1,87%	32,05%	82,86%	43,34%	2,2	1,60
4	]-4, 4]	2	1330	0,529	1,02	17,74%	35,11%	80,83%	25,86%	2,3	0,53
4	]-4, 4]	3	584	0,232	0,71	39,55%	30,14%	63,01%	44,52%	2,2	0,37
4	]4, 9]	1	3547	1	3,84	1,83%	34,25%	80,01%	28,84%	1,8	2,34
4	]4, 9]	2	1819	0,513	2,18	21,00%	23,20%	61,96%	38,54%	1,7	1,33
4	]4, 9]	3	662	0,187	1,26	40,48%	29,15%	22,66%	50,30%	1,5	0,95
4	]9, 15]	1	6510	1	4,71	2,73%	32,52%	63,73%	37,82%	1,8	2,95
4	]9, 15]	2	2787	0,428	2,34	22,64%	14,24%	64,55%	30,79%	1,6	1,35
4	]9, 15]	3	857	0,132	1,22	34,07%	23,92%	24,50%	45,16%	1,4	0,98
4	>20	1	3963	1	6,48	1,56%	10,24%	48,12%	72,70%	1,5	5,01
4	>20	2	1199	0,303	2,02	20,27%	12,34%	58,80%	28,11%	1,2	1,50
4	>20	3	299	0,075	1,00	44,15%	10,37%	13,04%	38,13%	1,1	0,99
4	]15, 20]	1	4451	1	6,02	1,28%	12,96%	45,27%	70,46%	1,4	4,82
4	]15, 20]	2	1374	0,309	1,74	17,54%	13,25%	60,63%	31,30%	1,3	1,32
4	]15, 20]	3	341	0,077	0,91	38,12%	14,37%	9,09%	52,79%	1,2	0,88