Multi-Armed Bandits

A simulation Gym for customer retention in a TelCo.

José Chumbo

Dissertation written under the supervision of Professor Miguel Godinho de Matos, with the collaboration of an industry partner in the telecommunications Industry

Dissertation submitted in partial fulfillment of requirements for the MSc in Business Analytics, at the Universidade Católica Portuguesa, 04/01/2023.

Multi-Armed Bandits

José Chumbo

April 05, 2023

Abstract

A indústria de telecomunicações altamente competitiva apresenta um desafio significativo para as TelCos na retenção de clientes. Para alcançar a retenção de clientes, as TelCos geralmente utilizam algoritmos de Aprendizado de Máquina (ML) para alinhar suas ofertas de produtos com as preferências dos clientes. No entanto, esses algoritmos têm limitações na exploração de interações passadas com clientes que podem conter viéses de diversas fontes. Esta tese investiga o potencial dos bandits multi-braços (MABs) para enfrentar esse desafio.

MABs são um tipo de algoritmo de aprendizado por reforço que maximiza as recompensas a longo prazo e tem sido utilizado em diversas indústrias para otimizar a alocação de recursos. Este estudo utiliza dados históricos de retenção de clientes proativa para construir uma academia, simulando as operações diárias de uma TelCo. A academia permite a avaliação de múltiplas políticas e cenários em um ambiente offline.

Os resultados do estudo mostram que os MABs podem equilibrar a exploração e a explotação e superar algoritmos clássicos em determinados casos. No entanto, quando o número de possíveis braços aumenta drasticamente, os algoritmos MAB mais simples podem enfrentar dificuldades. O estudo também mostra como podemos limitar estrategicamente esses braços para aumentar o desempenho sem mudar o algoritmo para um mais complexo.

Em geral, esta pesquisa aprimora a compreensão do potencial e das limitações dos MABs para a retenção de clientes nas TelCos e fornece insights para sua implementação e implantação bem-sucedidas.

Keywords: Multi-Armed Bandits, Retenção de Clientes, Systemas de Recomendação, Churn, Avaliação Offline de Políticas

Abstract

The highly competitive telecommunications industry poses a significant challenge for TelCos in retaining customers. To achieve customer retention, TelCos often utilize Machine Learning (ML) algorithms to align their product offerings with client preferences. However, these algorithms have limitations in exploiting past client interactions that may contain biases from various sources. This thesis investigates the potential of multi-armed bandits (MABs) to address this challenge.

MABs are a type of reinforcement learning algorithm that maximizes long-term rewards and has been used in various industries to optimize resource allocation. This study uses historical data from proactive customer retention to build a gym, simulating the daily operations of a TelCo. The gym allows for the evaluation of multiple policies and scenarios in an offline environment.

The study's findings show that MABs can balance exploration and exploitation and outperform classical algorithms in certain cases. However, when the number of possible arms increases dramatically, simpler MAB algorithms may struggle. The study also shows how we can strategically limit these arms to increase performance without changing the algorithm to one more complex.

Overall, this research enhances the understanding of the potential and limitations of MABs for customer retention in TelCos and provides insights for their successful implementation and deployment.

Keywords: Multi-Armed Bandits, Customer Retention, Recommendation Systems, Churn, Offline Policy Evaluation

Contents

1 Introduction **1**

2 Literature Review **2**

2.1 Recommendation Systems 2

2.1.1 Content-based filtering 2

2.1.2 Collaborative filtering 3

2.1.3 Graph based 3

2.1.4 Global Relevance 4

2.1.5 Deep Learning 4

2.1.6 Reinforcement Learning 6

2.2 Multi-Armed Bandits 6

2.2.1 Stochastic Bandits 7

2.2.2 Bayesian Bandits 8

2.2.3 Contextual Bandits 9

2.2.4 Bandits and Agents 10

2.3 Conclusion 12

3 Data and Descriptive Statistics **12**

3.1 Context 12

3.2 Descriptive Statistics 13

4 Methodology **14**

4.1 Candidate Model 15

4.2 Ranking Model 15

4.3 Sampling Model 16

4.4 Client Demand Model 17

4.5 Simulation Cycle 17

4.6 Simulation Scenarios 19

5 Results **20**

5.1 Candidate Model 20

5.2 Ranking Model 21

5.3 Simulation with no Candidate Model 22

5.4 Simulation with Candidate Model 23

5.5 Simulation with the introduction of new bundles 26

6 Discussion **28**

7 Limitations **29**

8 Conclusions **30**

A Appendix **32**

A.1 Wide And Deep Model Implemented 32

References **33**

List of Figures

1 Wide and/or Deep Models (Cheng et al. (2016)) 4
2 Wide and Deep Model (Cheng et al. (2016)) 5
3 Slate Sizes 13
4 Retention over Time 14
5 Two Tower Model 15
6 Rolling Window 19
7 Candidate Model Training 20
8 Ranking Model Training 21
9 Clients Saved with no Candidate Model 22
10 Clients Saved with no Candidate Model 24
11 Clients Saved with New Bundles 26
12 Distribution Change Over Time for one of the New Bundles 27
13 Wide and Deep model 32

List of Tables

1 Candidate Model Accuracy. 20
2 Percentage of Clients Saved with no Candidate Model 23
3 Candidate Model Accuracy. 23
4 Logistic and Linear Regression for Candidate Retrieval Model 24
5 Candidate Model Accuracy. 25
6 Generalized Gini Index. 25
7 Logistic and Linear Regression for new Bundles 27
8 Candidate Model Accuracy. 28

1 Introduction

Telecommunication companies (TelCos) face the continuous challenge of retaining customers. Customer retention is costly and time-consuming. Customer retention is complex and requires effectively matching product offers to client preferences.

Many firms use machine learning algorithms that are trained on historical data repositories to build models that learn consumer preferences and steer the offer strategies of client retention teams.

However, unknown biases often exist in historical data related to product offerings and client decisions, making it difficult to train machine learning algorithms on this data for product retention. These biases can arise from a variety of sources, including client choices, optimization decisions made by managers, and statistical models that were used to generate the historical product offerings.

Learning from historical data is a challenging task when it comes to product retention. While such algorithms can predict historical patterns in the data, they may not be useful in helping a firm change consumer behavior, which is the primary goal of a retention team. The ultimate objective is to persuade dissatisfied clients to remain with the firm, and this may require breaking away from the historical patterns and finding new strategies to retain customers. Therefore, debiasing these models and ensuring they are not solely relying on historical data is essential for improving the product retention models.

One promising approach to improve client retention model training strategies is using multi-armed bandits (MABs). MABs are a class of reinforcement learning algorithms designed to learn optimal policies through exploration and exploitation. The concept of optimal relates to maximizing some long-term reward (e.g. client lifetime value).

MABs have been applied to several classes of optimization problems. For example, there are applications of MABs in online advertising, clinical trials, and recommendation systems Beel et al. (2015).

So far, we are not currently aware of any study that used MABs for improving the process of training machine learning models in a client retention context in the telecommunications sector. It's very likely such systems have been developed and are in use but that information remained internal to the company.

In this thesis, we use state-of-the-art machine learning algorithms and MABs to design a product recommendation system and an algorithm training strategy that can be used to maximize client retention in the context of the telecommunications sector.

We provide a literature review on recommendation systems, an overview of MABs, their relevant background, and theoretical foundations. We examine the challenges and opportunities of using MABs for customer retention in TelCos.

Finally, we present the results of a simulation exercise calibrated using real-world data, in which MABs were implemented for customer retention in a TelCo, including an analysis of the

outcomes and a discussion of the implications for future research and potential implementation.

Overall, this thesis aims to contribute to a better understanding of the potential and limitations of MABs for customer retention in TelCos and to provide practical insights for their successful implementation.

2 Literature Review

To test MAB policies' performance in client retention, we start by implementing a product recommendation system that call-center agents can use to help them select what products to suggest to clients. We overview the field of recommendation systems and then go in-depth into the implementation chosen to interact with the MAB. In the second part, we detail the approaches present in the literature to solve the MAB problem that we will use to solve the problem posed in this thesis.

2.1 Recommendation Systems

Recommendation systems try to solve the problem of choice overload. When an agent can only choose a few options to suggest to a user from thousands or millions of possibilities, the agent will need a strategy to find a few needles in a haystack of choices to present to a user. Recommendation systems were developed to solve this filtering problem with varying techniques to help the agents recommend the best choices.

2.1.1 Content-based filtering

Content-based filtering (CBF) recommendation systems have been one of the most used and researched techniques (Ricci et al. (2011)). They work by modeling the user interests by inference.

These systems learn how to model the user's likes and dislikes by analyzing the user's interactions with items, i.e., the content. Interactions with these items are actions performed by the user, for example: downloading, buying, authoring a review, etc.

In content-based filtering algorithms, items are represented by a content model composed of the various features of the item. There are many types of features, for example, word-based tags or classes, numeric review scores, and flags signaling a presence of a particular characteristic. Features can also be extracted from more complex items such as images, videos, and text, etc.

CBF algorithms generate recommendations by comparing the user model with a list of potential recommendations for similarity.

CBF systems recommend items based on content similarity to other items the user liked in the past. CBF systems typically focus on content characteristics and ignore information like content popularity and quality of items (Dong et al. (2009)).

CBF systems have low serendipity and overspecialization. They recommend items as similar as possible to the ones in the user model (Ricci et al. (2011)).

The performance of these systems is dependent on the quality and quantity of features extracted from the items (Dong et al. (2009)).

2.1.2 Collaborative filtering

Collaborative filtering (CF) works by assuming a user likes what other like-minded users also like. Users are considered like-minded based on how similar their interactions with items are (Resnick et al. (1994)).

CF makes recommendations by identifying like-minded users and suggesting to the original user, items those like-minded users interacted positively with.

One of the strengths of CF systems is the fact that it is content independent, it does not rely on the quality of the feature extraction from item content (Schafer et al. (2007)), it relies on human quality assessment of items (Dong et al. (2009)).

CF systems can also make serendipitous recommendations since they don't depend on content similarity. They can recommend new items as long as one user of a like-minded group has interacted with it (Palopoli et al. (2013)).

The main problem with CF is the cold start problem. The cold start problem occurs when there are no user interactions to find like-minded users (Schafer et al. (2007)) or no user has interacted with a new item. One way to overcome cold starts is to infer user ratings (for example, a user finishing watching a movie would be positive, and if the user gave up midway, that would be negative) with the caveat that we are introducing a bias into the system in the form of an administrative rule, for example, a user might have finished watching a movie because he fell asleep from boredom biasing our positive classification.

CF is also vulnerable to rating manipulation, "review bombing", where agents might try to manipulate the quality of user interactions either to demote or promote an item (Mehta et al. (2007)).

CF systems are usually employed in the context where the ratio of users to items favors the number of users. In those cases, users have a reasonable number of interactions with items making it easier to find similar users, and new items almost always have at least a few interactions (Herlocker et al. (2004)).

2.1.3 Graph based

Graph-Based (GB) systems indirectly inspect users and items related to user-item pairs to predict whether an edge may exist between them (Osman and Barukub (2020)).

Representing the data as a graph serves to make the underlying relationships explicit. Focus on learning mechanisms that exploit this knowledge in order to get better performance for downstream tasks, like rating prediction. The goal is to learn better representations/embedding

of entities/nodes using neighborhood information. GB systems can map items, their attributes, users, preferences, and relations between users and items. By viewing the path taken through the graph, we can interpret how the system is making a recommendation.

2.1.4 Global Relevance

Global Reference (GR) systems adopt a one-fits-all approach (Beel et al. (2015)), recommending items with the highest global relevance. Relevance is not calculated per user. Instead, a global metric is used, such as how common an item is, how high it is rated or how much marketing exposure it has. These systems work on the assumption that users will probably want what most other users have or what is popular.

2.1.5 Deep Learning

Deep Learning (DL) systems are very flexible and have many different architectures depending on the task (classification, ranking, sequential recommendations, feature extraction, etc.). They can make recommendations by learning from features or embedding extracted from users or items. They have a few advantages, such as learning non-linear and non-trivial relationships. DL systems can be built using blocks, enabling previously learned patterns to be transferred into a new architecture. The main disadvantage comes from their opaqueness (they typically work as a black box).

One of the more popular architectures is the "Wide and Deep" (Cheng et al. (2016)) model which combines the strengths of linear models and deep neural networks for making recommendations in large-scale recommender systems. The model is based on the intuition that linear models are good at capturing simple feature interactions, while deep neural networks are better at modeling complex, non-linear feature interactions.

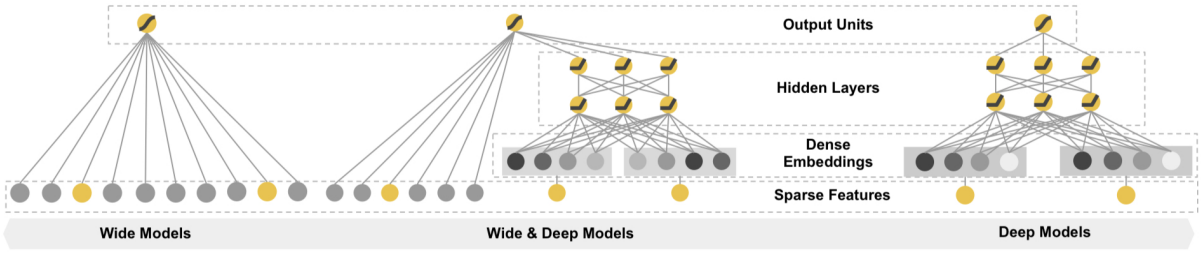

Figure 1: Wide and/or Deep Models (Cheng et al. (2016))

The architecture of the Wide Deep model consists of two parts: a wide part and a deep part. The wide part is a linear model that learns feature interactions between sparse features. The model uses cross-product interactions between all pairs of sparse features and learns a weight for each cross-product. This allows the model to capture high-order feature interactions.

The deep part of the model is a neural network that learns low-dimensional embeddings of dense features, such as numerical features. The neural network consists of several hidden layers

and is trained to predict the target variable, such as the probability of a user clicking on an item. The neural network is optimized using backpropagation and stochastic gradient descent.

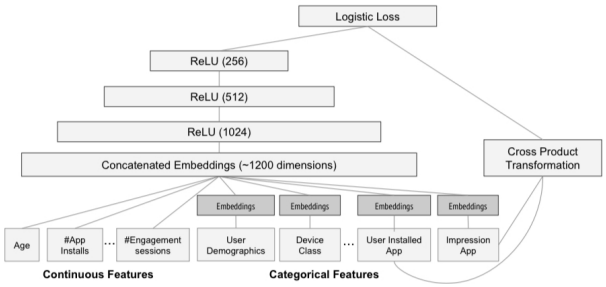

Figure 2: Wide and Deep Model (Cheng et al. (2016))

One advantage of the Wide and Deep model is that it can handle both sparse and dense features, which is important in many real-world recommender systems. The model is also computationally efficient because the wide part only needs to consider pairs of features, rather than all possible feature interactions. Additionally, the model is able to capture both simple and complex feature interactions, making it a powerful tool for making recommendations.

To use the "Wide and Deep" model for ranking, the model outputs a predicted probability for each candidate item, based on the user's past interactions and the features of the candidate item. The items are then ranked in descending order of their predicted probability. The model can also be used for personalized recommendations by incorporating user-specific features, such as user demographics or past purchase history, into the model.

One disadvantage of the Wide and Deep model is that it may require more training data than simpler models, due to its larger number of parameters. Additionally, the model may be more difficult to interpret than simpler models because of its combination of linear and non-linear components. However, the model's performance benefits often outweigh these drawbacks, particularly in large-scale recommender systems where accuracy and scalability are critical. To aid in scalability these algorithms are often paired with candidate retrieval models.

Ranking every possible combination can have exponential growth in processing time as the number of possibilities grows. Thus recommender systems separate this task in two stages. First a stage to retrieve candidates and a second to rank them. Having a candidate retrieval model is crucial in two-stage recommender systems because it enables the system to efficiently search and select a subset of items from a large item pool that are most relevant to the user's needs and preferences (Ma et al. (2020)). The candidate retrieval model essentially acts as a filter that reduces the search space and focuses the subsequent ranking model on a more manageable set of candidates. Without an effective candidate retrieval model, the ranking model would need to consider all the items in the pool, which can be computationally expensive and lead to suboptimal recommendations (Ma et al. (2020)). Therefore, a well-designed candidate retrieval model is essential for improving the overall performance of two-stage recommender systems, particularly when dealing with large item pools and high-dimensional feature spaces.

The candidate retrieval model is typically based on a representation learning approach such as the Two Tower model, which learns low-dimensional embeddings of users and items separately. Embedding can be shared with the ranking model. In addition, this model can be enriched by incorporating contextual information into the retrieval model, such as the context-aware Two Tower model, which extends the standard Two Tower model by incorporating contextual information such as time, location, and device information into the representation learning process.

2.1.6 Reinforcement Learning

Reinforcement Learning (RL) systems, or Deep RL systems, formulate recommendations as a sequential decision-making process.

The recommendations problem can be formulated as a Markov decision process that an RL algorithm can learn. An RL algorithm learns by receiving rewards/penalties through sequential user interactions and has four main components that need to be defined and built (state representation, policy optimization, reward formulation, and environment building).

RL systems excel in live environments, handle dynamic user interactions, and take long-term user engagement into account. Their main disadvantage comes from the difficulty in training in offline environments requiring a complex simulation environment, and training data might need to be synthetically generated from historical data.

2.2 Multi-Armed Bandits

This section of the literature review is based on the Paper Introduction to Multi-Armed Bandits by Slivkins (2019) and more detailed information can be found in it.

A multi-armed bandit is a reinforcement learning algorithm used to solve problems in which a decision must be made repeatedly over time to maximize a reward. The algorithm is called a "multi-armed bandit" because it is similar to a person sitting at a row of slot machines (also known as one-armed bandits) in a casino, trying to decide which machine to play to maximize their winnings. These algorithms are often used to explore unknown trends, gradually unbiasing historical data and minimizing regret/cost with various techniques.

In a multi-armed bandit problem, multiple "arms" or options can be chosen at each step, each with its probability of yielding a reward. The goal is to find the arm with the highest reward probability and to choose that arm as often as possible to maximize the total reward.

Multi-armed bandits are used in various applications, including online advertising, recommendation systems, and resource allocation. They are particularly useful in situations where it is not possible or practical to gather a lot of data about the rewards of each arm before making a decision, as the algorithm can explore and adapt as it goes along.

Performance can be evaluated by comparing regret. Regret measures how much an algorithm's decisions differ from the optimal decision. In other words, it is the difference between

an algorithm's reward and the reward it could have received if it had always chosen the arm with the highest expected reward. When dealing with offline data, this metric is trivial to calculate, but in practice, we seldom know the optimal choice. A more realistic metric for comparing these algorithms is their accumulated rewards.

2.2.1 Stochastic Bandits

In the realm of reinforcement learning, stochastic bandits present a classic problem where an agent must choose from a finite number of actions to maximize a reward. This involves a model of multi-arm bandits, where each arm represents an action and has an associated unknown reward probability distribution (Slivkins (2019)). The rewards for each action are independent and identically distributed, meaning that the probability distribution of rewards for one action does not depend on the rewards of any other action. The main objective of the agent is to learn about these unknown reward distributions and choose the action that maximizes the expected reward over time. To tackle this problem, researchers have developed various algorithms such as the epsilon-greedy, upper confidence bound (UCB), and Thompson sampling. These algorithms use different strategies to balance exploration and exploitation and can be applied to many reinforcement learning problems.

2.2.1.1 Uniform Exploration

Exploration is a key component in Multi-Armed Bandits (MAB) algorithms, where the goal is to balance exploration and exploitation to maximize reward (Slivkins (2019)). One simple approach to exploration is the Uniform Exploration algorithm. This algorithm involves exploring all the available arms uniformly, without considering previous observations, and then selecting the best arm based on the empirical evidence gathered during the exploration phase. One way to implement this is through an Explore-first algorithm, which allocates a certain percentage of rounds for exploration and the remaining rounds for exploitation. While this algorithm is easy to implement, it can suffer from the exploration-exploitation trade-off. Other exploration algorithms have been proposed in the literature, including Thompson sampling and UCB, which take into account the confidence in the estimated reward of each arm to balance exploration and exploitation. These algorithms have been shown to achieve better performance than Uniform Exploration in various applications, including recommendation systems and clinical trials.

2.2.1.2 Epsilon-Greedy

The Explore First algorithms are a class of algorithms that can suffer from poor performance during the exploration phase. To address this issue, the Epsilon-Greedy algorithm was proposed as a more balanced approach that explores arms at random while also exploiting the arm with the highest known reward probability (Slivkins (2019)). One important advantage of Epsilon-Greedy is that it allows for exploration to be spread out over time. This feature is particularly

useful when prior knowledge is available and there is a need to explore new spaces or adapt to changes in the reward probabilities of each arm. By fixing a constant ε, a non-adaptive rate of exploration can be achieved. However, to accelerate exploration, it is possible to start with a high ε_t value that decays over time. This approach helps to constrain regret within acceptable limits for each round.

for $T = 0, 1, 2, ..$ **do**
  Toss a coin with success probability ε_t;
  if *Success* **then**
    Explore: choose an arm uniformly at random
  else
    Exploit: choose the arm with the highest average reward prediction
  end
end

Algorithm 1: Epsilon-Greedy algorithm

2.2.1.3 UCB1

UCB1 is a popular algorithm for solving the Multi-Armed Bandit problem. It balances exploration and exploitation by calculating an Upper Bound Confidence (UCB) for each arm at each round. This UCB is calculated as the sum of the average reward $\bar{\mu}_t(a)$ and the confidence radius $r_t(a)$, i.e., $UCB_t(a) = \bar{\mu}_t(a) + r_t(a)$. The arm with the highest UCB is chosen in each round.

To calculate the confidence radius $r_t(a)$, we have two options: we can either use a scaling factor ucb_{scale} multiplied by the standard deviation σ_a divided by the square root of the number of times arm a has been pulled N_a, or we can use $\sqrt{2 * \log_{10}(T)/N_a}$, where T is the total number of rounds.

Try each arm once.
for $T = 0, 1, 2, ..$ **do**
  Pick arm a which maximizes $UCB_t(a)$;
end

Algorithm 2: UCB1 algorithm

UCB1 is particularly useful when we have prior knowledge about the problem and want to explore new spaces or adapt to changes in the reward probabilities of each arm. By adapting our exploration, we can constrain our regret bounds even further. A large $UCB_t(a)$ can indicate that the arm has not been explored much, or that it has a high expected reward. This balance between exploration and exploitation makes UCB1 an effective algorithm for solving the Multi-Armed Bandit problem.

2.2.2 Bayesian Bandits

Bayesian bandit algorithms are a type of multi-armed bandit algorithms that utilize Bayesian methods to model and update the probability distribution of rewards for each arm. This ap-

proach allows the algorithm to incorporate prior knowledge about the arms and make more informed decisions when selecting an action (Slivkins (2019)). Bayesian bandits are commonly used in scenarios where the rewards for each arm are uncertain or may change over time, and where exploration costs are high, such as online advertising, recommendation systems, and clinical trials.

Bayesian bandits can use various Bayesian. The choice depends on the type of reward distribution and the amount of prior knowledge available. In Bayesian bandits, the algorithm maintains a posterior distribution over the reward probabilities of each arm, which is updated using Bayes' rule after each round. This allows the algorithm to trade-off exploration and exploitation by exploiting arms with high posterior probability of being optimal while exploring arms with high uncertainty.

Bayesian bandits have several advantages over other multi-armed bandit algorithms. They can handle non-stationary reward distributions, incorporate prior knowledge, and converge to the optimal arm faster than other algorithms. However, they also have some limitations, such as the need for prior knowledge and the computational cost of updating the posterior distribution after each round.

2.2.2.1 Thompson Sampling

Thompson Sampling is a Bayesian algorithm that samples the expected reward for each arm to estimate its best choice. At each round t and arm a, Thompson Sampling computes the posterior probability that arm a is the best arm, then samples that arm a with that probability (Slivkins (2019)). This approach allows the algorithm to balance exploration and exploitation while also incorporating prior knowledge to make more informed decisions about which arm to select. Thompson Sampling has been shown to have superior performance compared to other popular algorithms such as epsilon-greedy and UCB1 in many applications.

for $T = 0, 1, 2, ..$ **do**
  Observe $H_{t-1} = H$, for some feasible $(t\text{ - }1) - history$ H;
  Sample mean reward vector $\bar{\mu}_t(a)$ from the posterior distribution P_H;
  Choose the best arm a_t according to $\mu_t(a)$.
end

Algorithm 3: Thompson Sampling algorithm

2.2.3 Contextual Bandits

To take into account the effect of the context on the rewards, Multi-Armed Bandit algorithms can use a separate algorithm for each context. This can be achieved by having a separate ALG_x (e.g., UCB1) for each context x, which provides a context-specific estimation of the expected rewards (Slivkins (2019)). In this way, the algorithm can make more informed decisions by taking into account the context in which the rewards are generated. The reward r_t in each round

t depends both on the context x_t and the chosen action a_t, which is denoted by $\mu(a|x)$. This approach is especially useful in contexts where the rewards associated with each action vary greatly across different contexts, such as in online advertising or recommendation systems.

2.2.3.1 Linear contextual bandits

In linear contextual bandits, the relationship between an arm's reward and the context is assumed to be linear. This means that the expected reward for selecting arm a given context x can be calculated by taking the dot product of the arm's feature vector x_a and a weight vector θ_a associated with that arm ($\mu(a|x) = x_a.\theta_a$). One approach to implementing this idea is through LinUCB, which functions in a similar way to UCB, but instead of constructing confidence intervals for the mean rewards of each arm, it calculates these intervals for the weight vector θ. The algorithm then uses this information to construct an upper confidence bound (UCB) on the mean reward of each arm, given a context x_t. Finally, it selects the arm with the highest UCB for that round. LinUCB is well-suited to problems where the number of features is small, but it may struggle to handle high-dimensional data.

for $T = 0, 1, 2, ..$ **do**
  Form a confidence region $C_t \subset \theta$ i.e. $\theta \in C_t$ with high probability; Observe context $x = x_t$
  for *arm* a **do**
    Compute $UCB_t(a|x) = sup_{\theta \in C_t} x_a.\theta_a$
  end
  Pick the arm a which maximizes $UCB_t(a|x_t)$
end

Algorithm 4: LinUCB algorithm

2.2.4 Bandits and Agents

In the context of recommendation systems, a common problem is the tension between exploiting known information and exploring new options. This is because agents are often incentivized to focus on exploitation rather than exploration since the benefits of exploration are usually not immediate. To address this issue, researchers have proposed a solution based on *Bayesian persuasion*, which involves designing a signaling scheme between a sender agent and a receiver agent that incentivizes the receiver to select an action that maximizes a particular objective and benefits the sender (Slivkins (2019)).

This problem can be formulated as a combination of two parts: an algorithmic part, such as Bayesian Bandits, and an economic part, which deals with knowledge and incentives. In this setup, a bandit algorithm (the principal) interacts with a decision-maker agent (the agent) over T rounds, where there are K possible actions or arms. The algorithm recommends an arm to the agent, who then selects an arm and receives a reward. The algorithm also observes the reward..

K arms, common prior P, reward distribution $D_x : x \in [0,1]$, mean reward vector $\mu \in [0,1]^k$
for $T = 0, 1, 2, ..$ **do**
  Algorithm recommends arm $rec_t \in [K]$
  Agent t receives rec_t and chooses an arm $a_t \in [K]$
  Agents reward $r_t \in [0,1]$ is realized as an independent draw from D_x, where $x = \mu_{at}$
  Action a_t and reward r_t are observed by the algorithm.
end

Algorithm 5: Incentivized Exploration

Before recommending an action, each Agent has knowledge of the parameterized reward distribution D_x and the prior P, but not the reward vector μ. The Principal rec_t has no knowledge of what previous Agents have done. Each Agent shares the same belief about the mean rewards, expressed by the prior P, which is correct in the sense that μ is drawn from prior P. To incentivize compliance with the Agent, we need to provide the Agent with the right amount of information. However, exploration recommendations are almost always worse in the short term than pure exploitation. If we give the Agent full information, the Agent will follow the *Bayesian-greedy* algorithm, which always exploits. The question, therefore, is how much information we should give to the Agent.

2.2.4.1 Repeated hidden exploration

One effective method for ensuring incentive compatibility in multi-armed bandits is to incorporate a small amount of exploration into a larger amount of exploitation. To achieve this, we can use a strategy where we observe the realization of an action $sig \in \Omega_{sig}$, and with a probability $\varepsilon > 0$, we recommend the arm we want to choose based on a target function $a_{trg} \to 1, 2$ (Slivkins (2019)). In this approach, we choose the arm $a_{trg} = 2$, and with the remaining probability, we exploit by selecting the arm that maximizes the expected reward given *sig* and past observations of rewards, denoted as $\mathbb{E}[\mu|sig]$. By using this method, we strike a balance between exploration and exploitation, which is essential for optimizing the long-term performance of the system.

probability $\varepsilon > 0$, function $a_{trg} : \Omega_{sig} \to \{1,2\}$
Input: signal realization $S \in \Omega_{sig}$
Output: recommended arm *tec*
if *explore, with probability* $\varepsilon > 0$ **then**
  $rec \leftarrow a_{trg}(S)$;
else
  $rec \leftarrow min(argmax_{a \in \{1,2\}} \mathbb{E}[\mu|sig = S])$
end

Algorithm 6: Hidden Exploration

To enhance the algorithm's exploration incentives, we can incorporate Hidden Exploration into a bandit algorithm (ALG) by repeatedly executing one call to ALG within the exploration branch. The exploitation branch then considers the history of all previous exploration rounds

to make its decision. This approach balances the need for exploration to gather information with the desire for exploitation to maximize immediate rewards. By conditioning exploitation on previous exploration rounds, we can ensure that the algorithm's recommendations align with the principal's objective while still providing incentives for exploration.

$N_0 \in \mathbb{N}$
In the first N_0 rounds, recommend arm 1.
for $T = 0, 1, 2, ..$ **do**
  if *explore, with probability* $\varepsilon > 0$ **then**
    Call ALG, let rec_t be the chosen arm, feed reward r_t to ALG;
  else
    $rec_t \leftarrow min(argmax_{a \in \{1,2\}} \mathbb{E}[\mu | sig = S_t])$
  end
end

Algorithm 7: Repeated Hidden Exploration

2.3 Conclusion

After evaluating the current literature on Recommendation Systems and Multi-Armed-Bandits we opted for an architecture currently in use at industry leaders in their field to evaluate their performance when applied to our context. We used use Two Stage-Recommender systems comprised of a candidate retrieval model based on the Two-Tower Model paired with the Wide and Deep Model For Ranking. To break the feedback loop of recommendation systems learning exclusively from their own recommendations we introduced a Multi-Armed-Badits Algorithm, in our case Epsilon-Greedy, to test multiple Policies and evaluate how well and how fast they can learn and adapt to new market conditions. This framework is generic enough to be able to accommodate different models and other policies that were not tested for time constraints.

3 Data and Descriptive Statistics

3.1 Context

We worked in collaboration with TELCO, a large telecommunications provider that serves more than one million households with TV, Internet, and telephony in the geography we analyzed. In this context, households may choose to cancel or renegotiate their service due to various reasons, including being dissatisfied with the service or being targeted by the competition to switch providers. To retain customers, TELCO's operations include using a risk assessment model that evaluates each client's probability of churning. This model, combined with the client's demographic and usage statistics, generates a set of recommendations for call-center operators to offer to clients who are considering leaving. The recommendations are generated

using a machine-learning process that analyzes historical data, which forms the basis of this thesis.

3.2 Descriptive Statistics

The data provided by the industry partner went from the 31st week of 2021 to the 22nd week of 2022. In this period a total of 278295 are present in the dataset having interacted with the retention team. On average each week we have 7538 (std. 615) in 2021 and 7164 (std. 1049) clients for 2022. For each client recommendation slates were built. In the data set the size of the slate is not fixed as we can see in Figure 3.

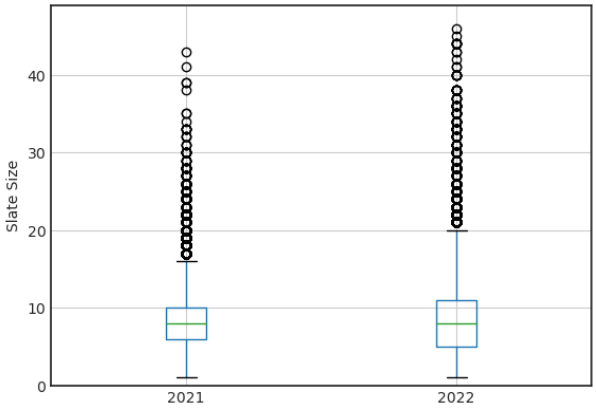

Figure 3: Slate Sizes

TelCo has several possible bundles it can offer its clients in the context of retention. We do not have information on the full offer available. We only know of the bundles that were part of recommendation slates. For the year 2021 we have 760 unique bundles and in 2022 we have 663 bundles. Of the 663, 159 bundles were not present in 2021. By the end of the period of data, we have 919 unique bundles.

We use TELCO's definition of retention throughout. A client is retained if she changes their product portfolio following an interaction with one of TELCO's retention agents. A change in client portfolio is determined by a change in the client's contract duration. Merely a change in intention to churn was not sufficient since we were informed that in most cases, clients either churned or re-engaged with TELCO to cancel their services within a few months. For the period of data, we have available, retention varies over time (Figure 4. For the year of 2021 the average retention is 51.57% and for the year 2022 the average is 46.58%.

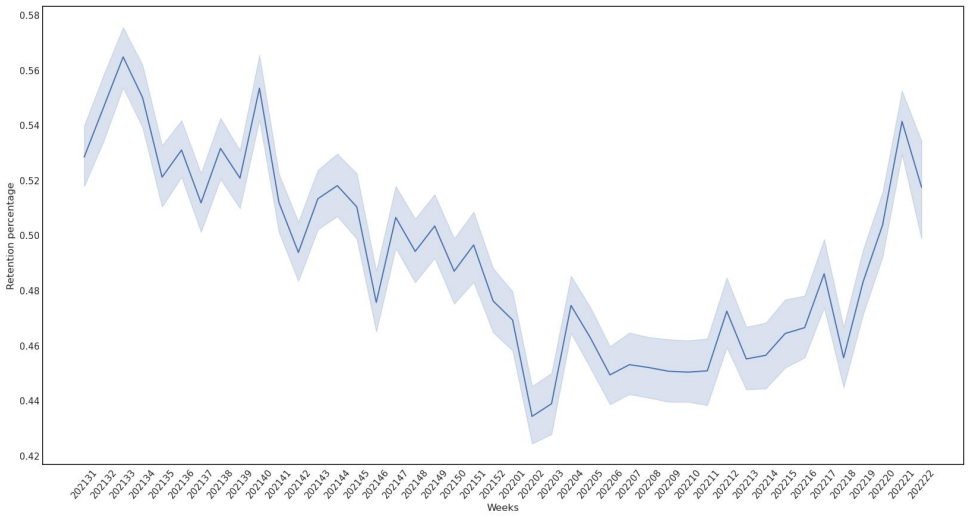

Figure 4: Retention over Time

4 Methodology

This thesis presents the implementation of a product recommendation system based on real-world data obtained from a retention process in the telecommunications industry. Our goal is to investigate how different recommendation delivery policies affect the learning outcomes of the system over time and improve overall retention outcomes. We accomplish this by developing a simulation environment that allows us to deliver recommendations to agents and test several multi-armed bandit policies. The environment will permit us to evaluate how well they learn user preferences, the learning speed as well as possible costs and trade-offs.

Initially, the recommendation system learns only from historical data, which may contain biases that do not accurately reflect client preferences. As such, we introduce exploration into the delivery of recommendations using a multi-armed bandit approach. This allows us to study how fast the system adapts to randomized and unbiased data.

While it would be ideal to test our product recommendations in a live setting, it was not feasible within the timeline of this thesis. Therefore, we created a simulation environment that replicates real-life scenarios. Each week, TelCo identifies a group of clients with a high risk of churn and sends them to a call center that implements a proactive customer retention strategy. In our simulation, the call center operators access bundle recommendations at multiple prices, which may be discounted. Our product recommendation system, a Wide and Deep Neural Network described in the next section, suggests a bundle and a price to a particular client, who then accepts or rejects the offer with a certain probability. The results of these interactions are continuously used to fine-tune product recommendations.

Finally, we present the different components of our simulation environment.

4.1 Candidate Model

The first component of our simulation is the candidate model, which is based on the two-stage recommendation system as described in the literature review. This system allows a simpler model to perform a retrieval operation by selecting a few candidates from a vast pool of possible recommendations, which are then ranked by a more complex and computationally expensive model.

For the first stage, we implemented the simplest possible Two Tower Model (Ma et al. (2020)), as shown in Figure 5. This models are widely used in live recommendation systems like Youtube, Pinterest, LinkedIn Ma et al. (2020), where speed, performance and scalability are crucial. The Query tower comprises an embedding[1], which encodes the current client bundle and serves as the entry point for querying the model for candidates. The second tower has the same architecture and encodes the possible bundles offered by call center operators. Although embedding layers are often shared between the Candidate and Ranking Models, we opted not to implement it due to time constraints and to keep the implementation as simple as possible. The two towers are combined by performing a dot product between the two embeddings, which yields a score.

To further improve the accuracy of the model, context pertaining to client demographics and behavior, as well as item features, can be incorporated into the query tower. Additionally, Deep Retrieval Models Gao et al. (2020) can be implemented to increase the complexity of each tower and the overall accuracy of the model, which brings it closer to a full-fledged Ranking Model.

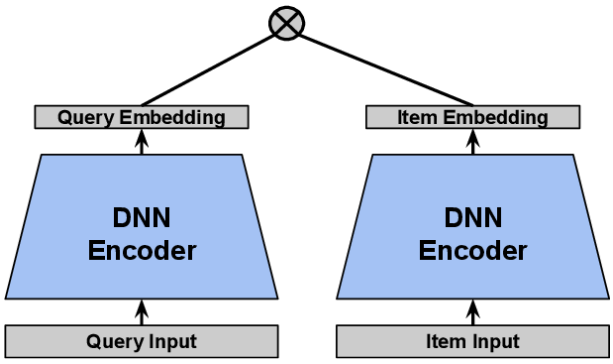

Figure 5: Two Tower Model

4.2 Ranking Model

The second component of our simulation is a ranking system which we implement as a Wide and Deep Neural Network (NN). We use this system to rank the bundles suggested to the call

[1] An embedding is a vector representation of a categorical variable, such as words or items, that allows it to be processed by machine learning algorithms. These vectors are learned through a process called embedding training, where the algorithm learns to map words or items to points in a high-dimensional vector space, where similar words or items are represented by vectors that are close to each other.

center operator.

The recommendation system is our best guess of how clients behave. Given a particular state (composed of client characteristics) and a particular product recommendation, the client may accept or reject the offer with a certain probability. In our pseudo-code, we denote a particular version of the recommendation system as the Agent.

In a Wide and Deep Neural Network model, we have multiple embeddings to encode categorical features that are inputs for the Deep part of the model as well as continuous features for the client behavior.The model we implemented has two embeddings, one to encode the current client bundle (TV, Net, Voice), and another to encode a possible Bundle the client can transition to (TV, Net, Voice).

Binary variables were used as inputs for the Wide part, since we do not have access to more information we cannot do cross-product interactions as described in the literature review. We were limited to flags that will memorize certain client behavior or other important markers such as the presence of certain technologies in the region or competitors.

We parameterize our model with 3 Deep Layers of 512, 256, and 128 nodes. We use ReLu activation function mixed with two 0.5 Drop Out layers. We use a final dense layer with a sigmoid activation function outputting the probability of a client switching to the proposed bundle-discount Pair. This architecture is an industry-standard (Guo et al. (2020)) with varying node sizes.

Figure **??** in the Appendix provides the detailed configuration of the neural network.

The Model will be used to rank sets of possible bundles to recommend to a client. Two sets will be tested, one containing all the possible bundles and a second one containing only the bundles retrieved by the candidate model in addition to new bundles unseen by the candidate model that will appear throughout the period of the simulation.

This Model was trained in a simulation environment analogous to the one described in the Simulation Cycle (without a Candidate Model) for the period of data available for 2021. The weights of the model were then frozen, and all recommendations and results were saved to be used as the starting point for all the tests and simulations run in 2022.

4.3 Sampling Model

The responsibility of the Sampling Model is to create a recommendation slate, of size K = 10, that can be presented to a client by a call center operator. This value was chosen at the request of the industry partner. In our simulation, the operator will offer all the recommendations in sequence to the client, stopping either when the client agrees to an offer or when the end of the slate is reached.

The slates are generated through a sampling process using an epsilon-greedy policy, which is explained in the literature review. For each slot in the slate, the Sampler selects one bundle using a probability of ε, either by randomly choosing a bundle from a set (Exploration) or

by sampling the top-ranked bundle from the Ranking Model (Exploitation). Alternatively, the whole slate could have been randomized. The for each client C with probability ε the Sampler would build a complete slate with random recommendations. This option would ensure that only unbiased data was present to the client and a Call Center operator wouldn't bypass them in favor of a more familiar Bundle. For this thesis, we only explored the former at the request of the industry partner since it is in theory safer. By applying the policy to each slot individually we can guarantee that all clients are presented with some amount of recommendations based on Exploitation data.

4.4 Client Demand Model

To accommodate a drift in preferences between what a model learns and a change in preferences, we developed the Client Demand Model. This will help us to simulate preferences not seen in the historical data, for example in a TelCo we can experiment with the introduction of a new product upsetting the demand for older products. We used the Wide and Deep Model, excluding the transition bundle embedding, and added a softmax layer. This layer produces a probability vector that represents a client's preferences for each possible bundle, with each position in the vector corresponding to a different bundle.

The Model generates a preferences vector for each client using their individual information. To simulate interactions between the Agent and the client, the component is queried with the id of a proposed bundle to be accepted by the client. The client Model will accept the offer with a probability of P that corresponds to the probability for the bundle id in the preferences vector.

We trained two client Demand Models, one for the period of 2021, which was used to interact with the Ranking Model in 2021, and another for the period of 2022, which was used in all the simulations.

4.5 Simulation Cycle

The pseudo-code for our simulation is detailed in algorithm 8.

The code simulates interactions between a TelCo agent and a customer to determine the most suitable bundle of services for the customer to accept, resulting in a reward for the agent. The simulation employs the Multi-Armed Bandit (MAB) algorithm to assist in making the decision that maximizes the reward, in our case maximizing retention.

The simulation runs from week 1 to week 22 of 2022 and involves loading all relevant models. To construct a recommendation slate for each client, the candidate model is fed with the client's information and returns a set of candidate bundles. New bundles, not previously seen by the candidate model, are included in the list. The candidate set is then ranked and ordered. For each slot in the slate, the sampling model follows a policy to fill these slots, with two options being tested in this thesis: a full exploit policy and an epsilon-greedy policy with

```
Window = first 10 periods (202142-202152)
RM = Load Ranking Model from 2021;
CRM = Load Candidate Retrieval Model from 2021;
CM = Load Client Model for 2022;
for P = 202201, 202202, 202203, ..., 202222 do
    for Client C in Period P do
        Candidates = CRM(C)+ Unseen Bundles that appear from period P onwards;
        Rankings = RM(Candidates);
        S = slate with N positions;
        for Slot N in Slate S do
            if explore, with probability ε then
                Explore: choose a bundle ID at random;
            else
                Exploit: choose the next top-rated bundle ID from Rankings;
            end
            S[N] = ID;
        end
        Preferences = CM.getPreferences(C)
        for Slot N in Slate S do
            Result = CM.offerClient(S[N], Preferences);
            if Result = Accepted then
                The Agent Witness a reward for Client C and adds the result to the
                 Window;
                Break;
            else
                The Agent Witness a penalty and adds the result to the Window;
            end
        end
    end
    Shift Window by removing Period P − 10;
    Fine Tune Weights of RM on the new Window;
    Retrain CRM on all data;
end
```

Algorithm 8: Simulation with Bandits

different epsilon values. The slate's recommendations are evaluated, and the client model is queried until a recommendation is accepted or the slate is fully processed.

After processing all clients, the results are added to the training window, which is then rolled forward by one week (IE. removing the oldest, Figure 6). We used a rolling window as a way of removing old, and possibly, biased data from which the Model will be fine-tuned. This is an optimization parameter, that should be the object of experimentation, but due to time constraints, we were not able to fine-tune it.

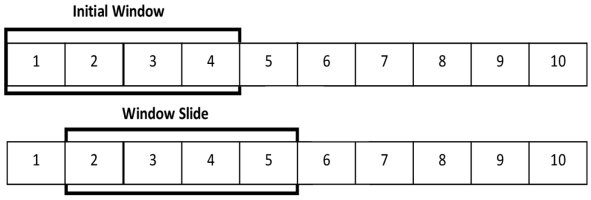

Figure 6: Rolling Window

We fine-tuned the weights of the ranking model on the new window and retrained the candidate model accordingly.

To establish a baseline, we set epsilon to zero and trained all models on the same simulated environment using historical data from 2021. We saved the models and used them for each simulation in 2022 to ensure they all started from the same point.

4.6 Simulation Scenarios

We conducted two simulations to test our approach. In the first simulation, we relied solely on historical data.

In the second simulation, we introduced four new bundles that had the potential to replace existing ones and affect their demand. To achieve this, we selected four bundles from 2022 that were not available in 2021 and calculated the distance [2] between each new bundle and all existing ones in 2021. Then, we identified the closest existing bundle and swapped their demand on the probability vector generated by the Client Model.

For this second scenario, we tested the Epsilon Greedy Policy using two different sets of arms. The first set included an arm with the Ranked Candidates (Exploit) and another arm with all possible bundles (Explore with a probability of Epsilon, sampled at random). The second set consisted of the Ranked Candidates (Exploit) and a limited set of new bundles (Explore from this set, which exclusively comprised the newly introduced bundles, sampled at random).

[2]Bundles in the data set were categorized by various continuous features forming a vector space. The spacial distance between these two vectors was calculated to determine similarities.

5 Results

In this section, we first assess the performance of each component of the simulation and then evaluate the results of each simulation.

5.1 Candidate Model

We evaluated the Candidate Model using 2021 data, which we divided into training and test sets comprising 80 and 20 percent of the total data, respectively. The model underwent 200 epochs of training, with early stopping enabled if no progress was observed in the factorized top 50 categorical accuracy after 20 epochs. The model ultimately ceased improving at Epoch 127, achieving the results illustrated in Figure 8.

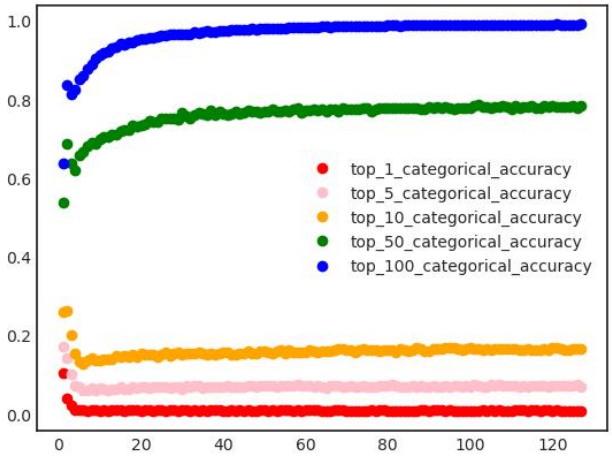

Figure 7: Candidate Model Training

As we examined the test set, we noticed a direct correlation between the size of the candidate pool retrieved and the model's performance, as depicted in Figure 1. As expected, we observed a notable increase in performance with larger candidate pools, specifically for k=50 and k=100. When considering k=50, we found that 73 percent of the included slates resulted in a client transition. This value increased significantly to 96 percent when considering k=100. Although we were limited by time constraints and could not evaluate both k values in the simulation, we ultimately decided to use k=50 as a compromise between accuracy and performance.

Factorized top K	k=1	k=5	k=10	k=50	k=100
Categorical Accuracy	0.43%	6.5%	13.75%	73.42%	96.3%

Table 1: Candidate Model Accuracy.

5.2 Ranking Model

We evaluated the Ranking Model by using 2021 data, which we divided into training and test sets comprising 80 and 20 percent of the total data, respectively. During training, the model underwent 500 epochs, with early stopping enabled if no progress in the binary cross-entropy loss was observed. We calculated loss with different weights during training, assigning a weight of 5 to the desired class denoting success (which was in the minority) and a weight of 1 to the negative class to account for the class imbalance. Currently, the initial training of the wide and deep model takes approximately 4 hours to converge, stopping at Epoch 175.

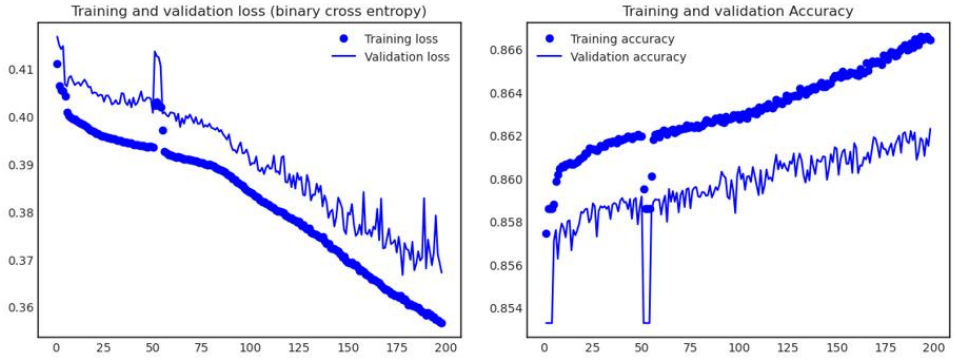

Figure 8: Ranking Model Training

The model demonstrated an AUC of approximately 0.72, with a PR-AUC of around 0.36. When the model was evaluated using the test set, it showed a recall of 0.07 for a threshold of 0.5, indicating the fraction of correct transitions over all transitions. In the out-of-sample test set, we found that 1.4 percent of validation clients opted to select a product from the top 5 recommendations provided (as compared to a random choice probability of 0.7 percent), while 4 percent of validation clients chose a product from their top 10 ranked offers (as compared to a random choice probability of 1.5 percent)

We found the performance of the model to be surprisingly poor. The model's poor performance was due to falsely labeled data provided by our industry partner, which categorized some recommendations as rejected even though they were never presented to the client because another recommendation was accepted first. This issue could be resolved by improving the data collection process to align with our methodology. We could also try to fine-tune the recommendation system using additional client data, which we did not for time constraints given the overall complexity of the problem at hand.

5.3 Simulation with no Candidate Model

We utilized data from weeks 1 to 22 of 2022 to conduct the simulation. Firstly, we evaluated the behavior of the Ranking model without the Candidate model to establish a baseline for comparison when it is introduced later. As per the methodology, each simulation run began with the saved Ranking Model from 2021 and the resulting data from the simulated customer retention process of 2021. We tested five different policies for the Sampling Model, including Full Exploit with no exploration, exactly as it ran in 2021, and Epsilon Greedy with four different Epsilons (1, 5, 10, and 100 percent). For the case of Epsilon 100 (i.e., Full Exploration), the simulations runs only for 10 weeks and revert to Full Exploit with the information it learned.

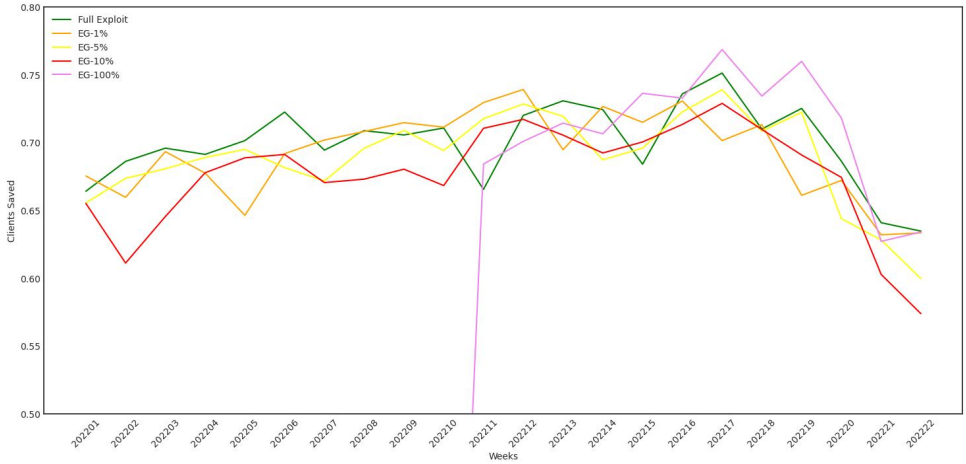

Figure 9: Clients Saved with no Candidate Model

Figure 2 shows that the Full Explore Model (discounting the initial 10 weeks of learning) outperformed the baseline, which corresponds to the Full Exploit Model, in terms of the overall percentage of clients that did not churn.

To assess the ability of each policy to capture the demand for 2022, we compared the distribution of the accepted bundles recommended (excluding Exploration) with the true demand generated by our Client Model. To quantify the difference between the two distributions, we calculated the error (RMSE) between the distribution of each specific bundle and the true demand. The results, as shown in the analysis, indicate that only the Full Explore policy was able to get closer to the true demand when compared to the baseline.

	Dependent variable:	
	Saved	
	logistic	*OLS*
EG1	-0.039^{***}	-0.008^{***}
	(0.012)	(0.002)
EG5	-0.049^{***}	-0.010^{***}
	(0.012)	(0.002)
EG10	-0.108^{***}	-0.023^{***}
	(0.012)	(0.002)
EG100	0.054^{***}	0.011^{***}
	(0.014)	(0.003)
Full_Exploit		
Constant	0.849^{***}	0.700^{***}
	(0.008)	(0.002)
Observations	317,175	317,175
R^2		0.001
Adjusted R^2		0.0005
Log Likelihood	$-195{,}665.000$	
Akaike Inf. Crit.	391,339.900	
Residual Std. Error		0.461 (df = 317170)
F Statistic		40.402^{***} (df = 4; 317170)
Note:		*p<0.1; **p<0.05; ***p<0.01

Table 2: Percentage of Clients Saved with no Candidate Model

Policy	Full Exploit	EG 1	EG 5	EG 10	EG 100
RMSE	.00888	.008964	.008969	.009102	.008541

Table 3: Candidate Model Accuracy.

5.4 Simulation with Candidate Model

We added the Candidate Retrieval Model as described in the methodology to enhance our simulation. Now, for each client of every week, the model retrieves a set of 50 candidates from all the previously seen Bundles. The list of candidates, along with ones that are new for that week, will be ranked, and the Recommendation Slate will be built as before.

Table 4 shows that the candidate model was able to improve the overall percentage of clients that did not churn, when compared to previous results. The comparison was made with the Full Exploit and Epsilon Greedy policies, with different exploration rates. It can be observed that the EG 5% and 10% policies, both using the candidate model, consistently outperformed all other approaches.

We compared the impact of the candidate model on the ability of the Ranking Model to

Figure 10: Clients Saved with no Candidate Model

	Dependent variable:	
	Saved	
	logistic	*OLS*
EG1	−0.039***	−0.008***
	(0.012)	(0.002)
EG1_CRM	0.043***	0.009***
	(0.012)	(0.002)
EG5	−0.049***	−0.010***
	(0.012)	(0.002)
EG5_CRM	0.041***	0.009***
	(0.012)	(0.002)
Full_Exploit		
Constant	0.849***	0.700***
	(0.008)	(0.002)
Observations	350,305	350,305
R^2		0.0003
Adjusted R^2		0.0003
Log Likelihood	−213,897.600	
Akaike Inf. Crit.	427,805.200	
Residual Std. Error		0.458 (df = 350300)
F Statistic		27.612*** (df = 4; 350300)
Note:		*p<0.1; **p<0.05; ***p<0.01

Table 4: Logistic and Linear Regression for Candidate Retrieval Model

capture the demand for 2022 by calculating the error (RMSE) between the distribution of the accepted bundles and the true demand. The results, presented in Table 5, demonstrate that the candidate model had a positive impact on the ability of the ranking model to learn the underlying demand.

Policy	Full Exploit	EG 1	EG 1 with CRM	EG 5	EG 5 with CRM
RMSE	.00888	.008964	.008722	.008969	.008708

Table 5: Candidate Model Accuracy.

To verify if the candidate model increased the inequality in the recommendation of bundles we calculated the Generalized Gini Index for each policy (Table 6.

Policy	Client Model	Full Exploit	EG1	EG1 CRM	EG 5	EG5 CRM
Gini Index	.006340	.016274	.016091	.014582	.013464	.013698

Table 6: Generalized Gini Index.

5.5 Simulation with the introduction of new bundles

For this simulation, we introduced 4 new bundles (as described in the methodology) by swapping their demand with the closest bundle. We retrained the client model for this new demand distribution. We chose this method to ensure demand for these new bundles was not a fixed probability across the board. But instead was contextual to each client. We tested the same Epsilon Greedy parameters as before and included the Candidate Model for all runs of the simulation. The difference comes from the set of bundles we will sample at random. For the baseline, the EG sampled from the full list of available bundles, for the new approach we limited this set to the newly introduced bundles.

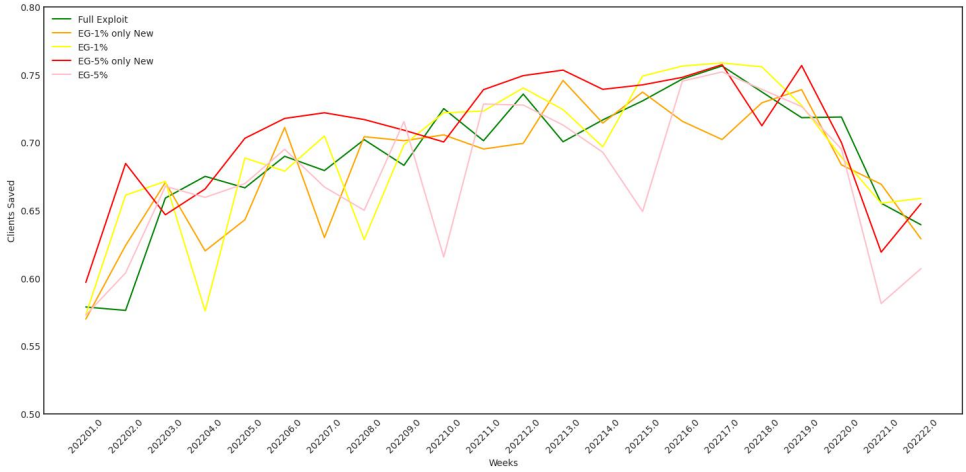

Figure 11: Clients Saved with New Bundles

Upon comparing the overall percentage of clients who did not churn with the introduction of new bundles, we observed a positive impact when utilizing epsilon greedy 5% sampling exclusively from the new bundles **??**.

We compared the distribution of the accepted bundles with the true demand by calculating the error (RMSE) between the distribution of each specific bundle and the true demand to determine if limiting the bundles sampled by the epsilon greedy policy affected the Ranking Model's ability to capture the demand for 2022. The results are presented in Table 8.

We calculated the learning rate by measuring the error at 4-week intervals. The results indicate that using the epsilon greedy policy of 5% only on new bundles stopped improving after week 8, outperforming all other policies. To illustrate this effect, we plotted the distribution change over time for one of the new bundles in Figure 12.

	Dependent variable:	
	Saved	
	logistic	*OLS*
EG1	0.005	0.001
	(0.012)	(0.002)
EG1_new	−0.031***	−0.007***
	(0.012)	(0.002)
EG5	−0.065***	−0.014***
	(0.011)	(0.002)
EG5_new	0.072***	0.015***
	(0.012)	(0.002)
Full_Exploit		
Constant	0.800***	0.690***
	(0.008)	(0.002)
Observations	350,305	350,305
R^2		0.0004
Adjusted R^2		0.0004
Log Likelihood	−217,048.200	
Akaike Inf. Crit.	434,106.400	
Residual Std. Error		0.463 (df = 350300)
F Statistic		38.335*** (df = 4; 350300)
Note:		*p<0.1; **p<0.05; ***p<0.01

Table 7: Logistic and Linear Regression for new Bundles

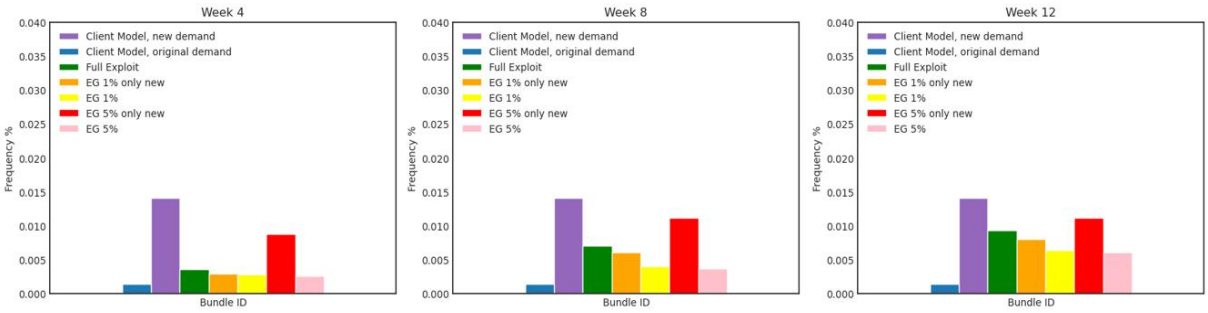

Figure 12: Distribution Change Over Time for one of the New Bundles

Policy	Full Exploit	EG 1	EG 1 only new	EG 5	EG 5 only new
RMSE week 4	.009460	.009302	.009252	.009421	.009130
RMSE week 8	.009192	.009173	.009187	.009232	.008885
RMSE week 12	.009140	.009087	.009111	.009126	.008888
RMSE week 16	.009096	.009066	.009063	.009069	.008908
RMSE week 22	.008976	.008958	.009008	.008983	.008889

Table 8: Candidate Model Accuracy.

6 Discussion

The results presented in this study provide valuable insights into the efficacy of different policies and models in the context of a customer retention strategy for a subscription-based service. The simulation was conducted on data from 2022, and the ranking model's behavior was evaluated without the candidate model to determine differences when it was later introduced. The study found that the Full Explore policy outperformed all other policies, including the Full Exploit policy, which was the baseline. The Full Explore policy was able to get closer to the true demand when compared to the baseline. We believe the poor performance of the epsilon greedy (with low epsilon) methods in this scenario is due to the big pool of possible candidates. Recommending at random has a very low chance of yielding a successful recommendation contributing to the already substantial class imbalance of negative observations introducing noise to the dataset with little information gain. This intuition is partially realized when we restricted the epsilon greedy policy to a small set of new bundles.

The candidate model was then introduced, and it was found to improve the performance of the ranking task. We found that both EG 5% and 10% with a candidate model outperformed all other approaches consistently. This indicates that the candidate model is effective in improving the overall performance of the retention strategy. Furthermore, the study found that the candidate model had a positive impact on the ability of the ranking model to learn the underlying demand. This finding confirmed the results found in the literature on two-stage recommendation systems that having a candidate model not only improves computational performance but also aids the Ranking model by removing noise. In terms of the Generalized Gini Index, the results were mixed. For EG 1% surprisingly the inequality in recommendations decreased when introducing the CRM, this unexpected result merits more research. For the EG 5% the inequality rose slightly as we would expect. The Ranking Model can rank never seen, or rarely-seen, bundles highly if they have similar characteristics to others, by first retrieving candidates based solely on the similarity of outcomes this behavior is mitigated. To ensure the CRM did not prevent the discovery of the demand for new bundles we add them to the ranking pool after the retrieval step as described in the methodology.

The study also explored the impact of restricting the bundles sampled by the epsilon greedy policy, to test the intuition described previously. The results showed that using epsilon greedy

sampling only from new Bundles had a positive impact on customer retention, but the difference was not statistically different from zero when compared to the baseline. However, the study found that restricting the bundles sampled by the policy had a positive effect on the learning rate of the Ranking Model for the EG 5%. The EG 5% policy only on new Bundles stopped improving at around week 8, outperforming all other policies. The EG 1% policy had surprisingly low performance not only compared with EG 5% but even with the baseline, we conjecture that once again 1% is too low an exploration rate to find positive results and gain information.

In summary, this study has demonstrated the effectiveness of Multi-Armed Bandits in developing effective customer retention strategies for subscription-based services. The findings have important implications for subscription-based services, as they provide insights into how to develop effective retention strategies that can ultimately lead to increased customer retention rates. By using a candidate model and adjusting the bundles sampled by the epsilon greedy policy, companies can improve their customer retention rates and gain a competitive advantage in the market.

7 Limitations

While the results presented in the previous section offer valuable insights into the effectiveness of different policies and models in the context of customer retention strategies for subscription-based services, several limitations to this study should be acknowledged. These limitations include the assumptions made in the simulation, the scope and representativeness of the dataset used, and the assumptions when generating new demand for Bundles Introduced in the market. In this section, we discuss these limitations in detail and provide recommendations for future research to address them.

One major assumption we made in building the simulation environment may not reflect the real-world scenario. Specifically, we assumed that Call center agents present offers to clients in a sequential order, whereas in reality, agents make the decision independently. While they may evaluate the recommendations in order before presenting them to the client, the historical dataset does not even contain information on which recommendations were offered to the client. This means that the dataset may contain false information, and recommendations that are classified as rejected from the full slate could have been accepted if presented. In our simulation, presenting recommendations in order eliminated this issue. However, it's important to acknowledge that this assumption may have led to an overestimate of the effectiveness of the policies and models we evaluated.

Another limitation of our dataset was that the bundles we analyzed are often paired with other add-ons and promotions that we did not have access to construct a final offer to be ranked. Therefore, we had to limit ourselves to the main Bundle. Future research is needed to validate the findings of this study in this area. It's possible that including the additional promotions could have a significant impact on the effectiveness of the retention strategies.

We could have also texted different Multi-Armed Bandits policies that have demonstrated to be more effective in certain scenarios in exploring the unknown space of possibilities. By leveraging context future research could test the effectiveness of policies such as UCB-1 described in the literature review for example.

Finally, another known limitation of the Sampling Model is related to the composition of the slate, which is not always complementary. The slate is built from the top-ranked bundle to the lowest, meaning that the ranking model may identify users who respond well to bundles with more internet speed, for example. As a result, all the top-ranked bundles have that feature in common and fill up the recommendation slate. However, it may be the case that two recommendations of this type are enough, and the rest of the slate should be filled with complementary options in other areas. Applying available research in this context could help to derive more accurate conclusions about the effectiveness of different retention strategies.

8 Conclusions

In this thesis, we investigated the effectiveness of different policies and models in the context of a customer retention strategy for a subscription-based service. We simulated the customer retention process using multi-armed bandit algorithms and evaluated the performance of different policies and models. In this section, we summarize the key findings of our study, discuss the limitations, and suggest areas for future research.

Our study found that introducing a candidate model improved the performance of the ranking task, and that restricting the bundles sampled by the epsilon greedy policy had a positive effect on the learning rate of the underlying demand and uncovered the speed at wich the model learns. Our results indicate that multi-armed bandits can be effective in the context of a customer retention strategy, and that having a candidate model not only improves computational performance but also aids the ranking model by removing noise.

Our study was subject to several limitations. We assumed that the call center agent presents offers to the client in sequential order, which may not be realistic. The historical dataset we used contained some false information and recommendations that are classified as rejected from the full slate could in theory have been accepted if presented. Additionally, we did not have access to construct a final offer that included add-ons and promotions paired with the main bundle, which may have limited the scope of our analysis.

Our study suggests several areas for future research. One direction is to explore the impact of including add-ons and promotions in the final offer, as this may have a significant impact on customer retention rates. Additionally, further research is needed to evaluate the optimal approach in building recommendation slates in this context.

In conclusion, our study has shown that multi-armed bandits can be effective in the context of a customer retention strategy for subscription-based services. Our results indicate that the Epsilon Greedy policy with a restricted number of offers outperformed other policies and that

having a candidate model improves the performance of the ranking task. However, our study was subject to several limitations, including assumptions about how offers are presented and limitations in the historical dataset. Future research in these areas is needed to further validate our findings and extend our understanding of the effectiveness of multi-armed bandits in customer retention strategies.

A Appendix

A.1 Wide And Deep Model Implemented

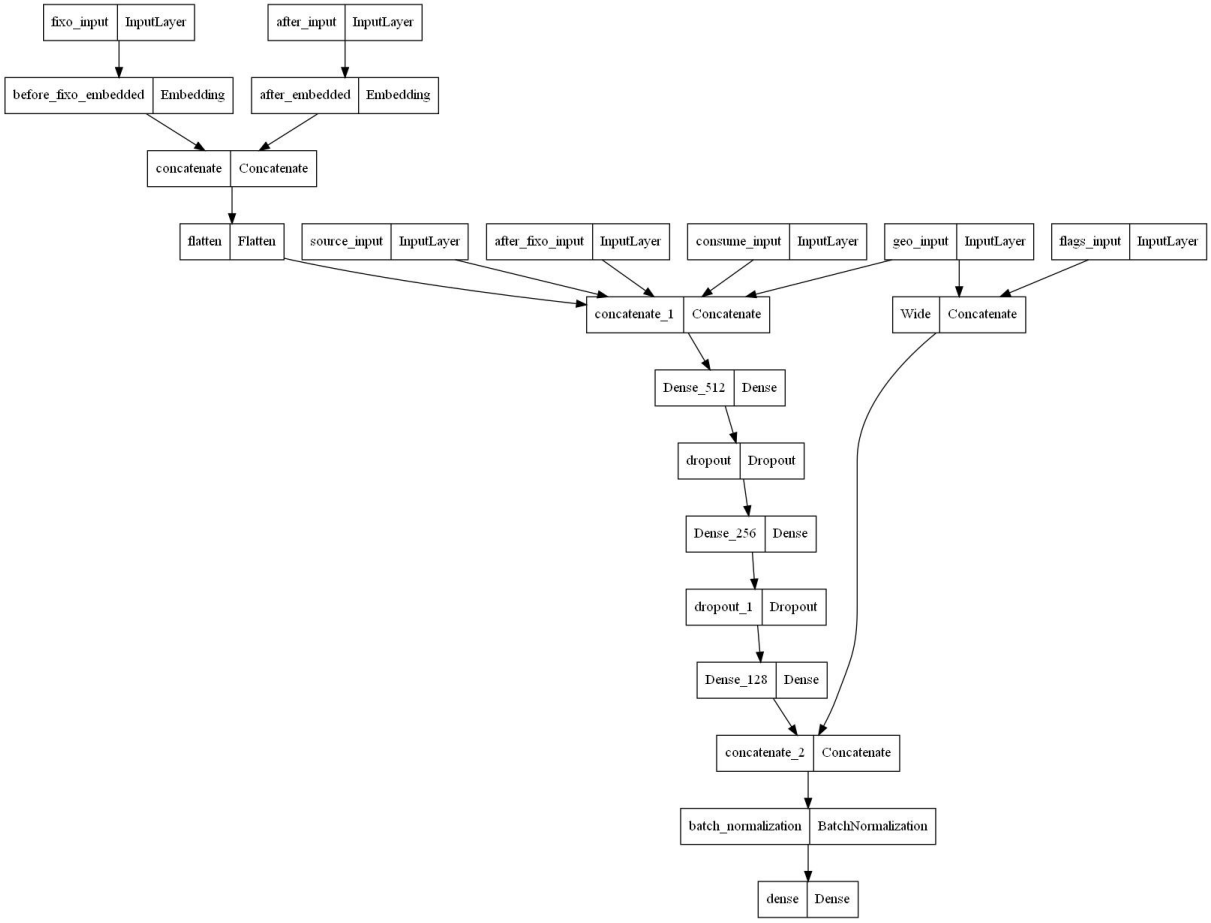

Figure 13: Wide and Deep model

References

Beel, J., Gipp, B., Langer, S., and Breitinger, C. (2015). Research-paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Anil, R., Haque, Z., Hong, L., Jain, V., Liu, X., and Shah, H. (2016). Wide and deep learning for recommender systems.

Dong, R., Tokarchuk, L., and Ma, A. (2009). *: Digging Friendship: Paper Recommendation in Social Network*. In: Proceedings of Networking and Electronic Commerce Research Conference (NAEC 2009).

Gao, W., Fan, X., Wang, C., Sun, J., Jia, K., Xiao, W., Ding, R., Bin, X., Yang, H., and Liu, X. (2020). Deep retrieval: Learning a retrievable structure for large-scale recommendations.

Guo, D., Ktena, S. I., Huszar, F., Myana, P. K., Shi, W., and Tejani, A. (2020). Deep bayesian bandits: Exploring in online personalized recommendations.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). : Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst. (TOIS)*, 22(1).

Ma, J., Zhao, Z., Yi, X., Yang, J., Chen, M., Tang, J., Hong, L., and Chi, E. H. (2020). Off-policy learning in two-stage recommender systems. In *Proceedings of The Web Conference 2020*. ACM.

Mehta, B., Hofmann, T., and Fankhauser, P. (2007). *: Lies and propaganda: detecting spam users in collaborative filtering*. In: Proceedings of the 12th international conference on Intelligent user interfaces.

Osman, A. H. and Barukub, O. M. (2020). Graph-based text representation and matching: A review of the state of the art and future challenges. *IEEE Access*, 8:87562–87583.

Palopoli, L., Rosaci, D., and Sarné, G. M. (2013). *: A multi-tiered recommender system architecture for supporting E-Commerce, Intelligent Distributed Computing VI*. Springer.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). *: GroupLens: an open architecture for collaborative filtering of netnews*. In: Proceedings of the 1994 ACM conference on Computer supported cooperative work.

Ricci, F., Rokach, L., Shapira, B., and Kantor (2011). *Recommender Systems Handbook*. P. B. (eds).

Schafer, J., Frankowski, D., Herlocker, J., and Sen, S. (2007). *Collaborative filtering recommender systems*, volume 4321. Lect. Notes Comput. Sci. 4321.

Slivkins, A. (2019). Introduction to multi-armed bandits.