MVA Internship Report at Ecole Normale Paris-Saclay

A contextual bandits approach to a personalized debt collections policy

Gasparini Fiuza do Nascimento, Henrique
Professor: Alessandro Lazaric
Supervisor: Thierry Silbermann

October 6, 2019

Abstract

We study decision making in environments where the reward is only partially observed and can be modeled as a function of an action and an observed context, known as contextual bandits. We study a practical application of this setting in the problem of personalizing debt collections.

We investigate contextual bandits algorithms and counterfactual learning techniques and present our design choices when developing our own personalization system. We explore both regression-based and cost-sensitive-classification-based algorithms to construct personalized policies. We perform offline evaluation of these policies and develop new feature explaining tools for this problem. We also emphasize the importance of reducing common technical debts in contextual bandits systems when developing ours, by logging the probabilities of actions selection, allowing reproducibility and counterfactual offline evaluation, and making our learning model adaptable to non-stationary data.

Finally, we present the initial results of our personalization system during its first month of usage in a practical use case: personalize the communication to be sent to a customer who is 34 days late in his bills aiming at maximizing payments in the following 5 days.

Contents

1 Introduction

This internship took place at Nubank's Collection team, as part of their initiative to use advanced personalization tools for selecting actions of high impact.

Nubank, as a 13-million customers financial services company focused on issuing credit cards, inevitably has a significant number of customers who are late in their bills. In fact, hundreds of thousands of clients have had their credit cards blocked during the month of July after staying 8 days late in their bills.

In particular, during a customer collections life-cycle, which is the period they stay late in their bills, we usually perform several actions, from sending communications (email, sms, push notifications) to blocking their credit cards and reporting them to credit card bureaus. Since each of these actions impacts on a significant number of customers, it was understood that Nubank's mathematical tool-set for decision making should not be limited to basic A/B tests and segmented A/B tests when deciding which actions to perform.

This need eventually led to my internship project: implementing a proof-of-concept policy with the goal of personalizing the communications sent in the 34th day of customers in collections, which optimized a business metrics of short-term payments.

In this report, I will attempt to organize my experience respecting the following road-map: a formal definition of the problem and the considered approaches; an overview of the design, architecture, and implementation of the solution; a thorough description of the experiment setup, development, and results; and finally a conclusion section.

1.1 Prior work

In their "A contextual bandit bake-off" paper, Bietti et al[1] summed up several guidelines based on their evaluation of contextual bandits algorithms on multiple datasets, which concerned the evaluation metrics, the hurting impact of forced uniform exploration, the choice of cost encodings, and the

variability of results according to the size and the difficulty of the dataset.

Our choice of using an ϵ-greedy algorithm was strongly influenced by the work of Agarwal et al[2], where they describe Microsoft's internal contextual bandits system, which has recently been released as a cloud-based service in Azure. In their paper, they propose many insightful design choices that we adapted in our system to allow, for example, reproducibility, offline evaluation, and ability to adapt to temporal changes.

Once we decided to use an ϵ-greedy algorithm, we concentrated our efforts on performing efficient counterfactual learning, which has often been referred as the logged contextual bandits problem. In this problem, one wants to learn a well-performing policy using past logged data, which may contain probabilities of performing each action given the context at the moment of decision. An interesting benchmarking work was performed by Lefortier et al[3], who compared the performances of multiple counterfactual learning methods on very large datasets. It proved to be a good resource of useful algorithms, although they studied very different data from the data in our experiment.

Using reinforcement learning techniques for debt collection is not a novelty of this work. In 2011, Naoki et al[4] proposed an approach for tax collecting based on constrained Markov Decision Processes, which estimated actions (and sequences of actions) expected returns according to context and also considered the cost of each action when automatically deciding the action to be sent. Some differences from our work are the fact that they only used linear models, as well as the problem definition, for we only model the impact of one action by client, while their work also considered sequence of actions.

2 Problem definition and approach

Let $\mathcal{X}$ be an input space and $\mathcal{A} = 1, 2, ..., k$ a finite action space. A contextual bandit problem is specified by a distribution over pairs $(x, \vec{r})$ where $x \in \mathcal{X}$ is the context and $\vec{r} \in \mathbb{R}^k$ is a vector of rewards. The input data has been generated using some unknown policy, hereby called π_0, (possibly adaptive and stochastic), as follows:

1. The world draws a new example $(x, \vec{r}) \sim \textit{Distribution}$. Only x is revealed.
2. The policy chooses an action $a \sim p_{\pi_0}(a|x)$
3. Reward r_a is revealed. Other rewards $r_{a'}$ with $a' \neq a$ are not observed.

Given a dataset $S = (x, a, r_a)$ collected as above, we are interested in 2 tasks: policy evaluation and policy optimization. In policy evaluation, we are interested in estimating the value of a stationary policy π, defined as:

$$V^{\pi} = \mathbb{E}_{(x,\vec{r})\sim \textit{Distribution}}\left[r_{\pi(x)}\right].$$

In policy optimization, we look for a policy with maximum value: $\pi^* = \arg\max_{\pi} V^{\pi}$.

2.1 Temporal dimension and long-term value

One important aspect in our problem is the temporal dimension. In fact, we collect new data everyday and update the decision policy based on more recent observations. For this reason, a more formal definition of the problem would consider datasets $S_0, S_1, S_2, \cdots, S_d, \cdots$ and policies $\pi_0, \pi_1, \cdots \pi_{d-1}, \cdots$. In this definition, the policy of day d is constructed from the observations in S_{d-1}, the knowledge of previous policies, and a construction rule π. We could write $\pi_d = \pi(d, S_{d-1}, \pi_0, \cdots \pi_{d-1})$.

In this framework, each dataset is a function of the previous dataset and the policy of the previous day:

$$S_d = S_{d-1} \cup S(d-1, \pi_{d-1})$$

The *value* of a sequence of policies can be analogously defined for the first D days as:

$$\sum_{d=0}^{D} \mathbb{E}_{(x,\vec{r})\sim Distribution(d)} \left[r_{\pi_d(x)} \right] = \sum_{d=0}^{D} \mathbb{E}_{(x,\vec{r})\sim Distribution(d)} \left[r_{\pi(d,S_{d-1},\pi_0,\cdots\pi_{d-1})}(x) \right] \quad (1)$$

Maximizing a sequence of policies is not equivalent to maximizing the value of each policy, as the choices made by a policy π_d impact on the dataset S_d, and therefore on the information we haven on the rewards distribution. A policy which maximizes value on the day it is executed may generate little valuable information for the construction of the following policies. This situation can be viewed in the frame of the exploration-exploitation dilemma.

Another factor that needs to be considered is that the distribution of rewards is not the same every day. In fact, in our experiments, we observed that it significantly depends on the day of the week and the day of the month (customers are more willing to pay us in the beginning of the month and cannot pay in the weekend). This can be addressed by adding date-based features.

The distribution of rewards also significantly changes over longer periods. In fact, the individuals profiles vary a lot over years as Nubank's customer-base has exponentially increased in its 5 years of existence. This was addressed by reducing the number of days used for training to a period of 60 days.

2.2 The construction rule - the contextual bandits algorithm

In their paper, Bietti et al[1] proposed some guidelines for choosing the contextual bandit algorithm, here also referred by the name of construction

rule. They highlighted the importance of using a good loss estimation metric and the hurting effect of using uniform exploration. One interesting result that they observed was that a completely greedy algorithm can perform quite well for most of the evaluated datasets.

In our use case, we evaluated using several metrics and opted for an ϵ-greedy approach: for a fraction ϵ of the individuals, we perform random actions (uniformly), while for the remaining individuals we perform the action with the estimated optimal value. In other words:

$$\pi(d, S_{d-1})(x) = \begin{cases} \text{Uniform}(1, \cdots, k) & \text{with probability } \epsilon \\ \hat{\pi}^*_{S_{d-1}}(x) & \text{with probability } 1 - \epsilon \end{cases}$$

An ϵ-greedy algorithm has some advantages over the best-performing algorithms: it allows for simple offline evaluation, which is the ability to evaluate how a new policy would perform by using past data and counterfactual learning techniques (described in the next subsection); it also allows for continuous exploration of the actions space, which is necessary for a non-stationary problem.

Knowing from Bietti et al[1] that a greedy algorithm can perform quite well (compared to other algorithms), we can stay reassured that an ϵ-greedy algorithm will perform well for the majority of the individuals.

Lastly, regarding the non-stationarity of our problem, it is important to note that all the datasets that were used in Bietti et al study were constructed under a hypothesis of stationarity (often with shuffling in the beginning), which does not hold in our problem.

2.3 Counterfactual evaluation

The main challenge when estimating policy value, given a dataset $S_{\pi_0} = (x, a, r_a)$ is that we have partial information of the reward. We cannot directly simulate our new policy π on S_{π_0}, as we do not know what would have been the reward for actions that π_0 did not perform.

Here we compile a list of approaches commonly used in the literature for

evaluating a deterministic policy π:

- *Direct Estimator*: let $\hat{\rho}(x, a)$ be an estimate of the expected reward given a context and action. One can estimate the policy value as:

$$\hat{V}^{\pi}_{DE} = \frac{1}{|S_{\pi_0}|} \sum_{x \in S_{\pi_0}} \hat{\rho}(x, \pi(x))$$

- *Inverse Propensity Score*: first compute estimates of $p_{\pi_0}(a, x)$, the probability of π_0 picking the action a given context x, or simply observe the logged probability if these probabilities have been logged. Then estimate the policy value as:

$$\hat{V}^{\pi}_{IPS} = \frac{1}{|S_{\pi_0}|} \sum_{(x,a,r_a) \in S_{\pi_0}} \frac{r_a 1(\pi(x) = a)}{\hat{p}_{\pi_0}(a, x)}$$

- *Self-Normalized Inverse Propensity Score*: very similar, but estimate the policy value as:

$$\hat{V}^{\pi}_{SNIPS} = \frac{\sum_{(x,a,r_a) \in S_{\pi_0}} \frac{r_a 1(\pi(x) = a)}{\hat{p}_{\pi_0}(a, x)}}{\sum_{(x,a,r_a) \in S_{\pi_0}} \frac{1(\pi(x) = a)}{\hat{p}_{\pi_0}(a, x)}}$$

- *Doubly Robust Estimator*: uses both $\hat{\rho}(x, a)$ and $\hat{p}_{\pi_0}(a, x)$ to estimate the policy value as:

$$\hat{V}^{\pi}_{DR} = \frac{1}{|S_{\pi_0}|} \sum_{(x,a,r_a) \in S_{\pi_0}} \frac{(r_a - \hat{\rho}(x, a)) 1(\pi(x) = a)}{\hat{p}_{\pi_0}(a, x)} + \hat{\rho}(x, \pi(x))$$

- *Self-Normalized Doubly Robust Estimator*: uses ideas from all previous methods to estimate the policy value as:

$$\hat{V}^{\pi}_{SNDR} = \frac{1}{|S_{\pi_0}|} \sum_{x \in S_{\pi_0}} \hat{\rho}(x, \pi(x)) + \frac{\sum_{(x,a,r_a) \in S_{\pi_0}} \frac{(r_a - \hat{\rho}(x, a)) 1(\pi(x) = a)}{\hat{p}_{\pi_0}(a, x)}}{\sum_{(x,a,r_a) \in S_{\pi_0}} \frac{1(\pi(x) = a)}{\hat{p}_{\pi_0}(a, x)}} \quad (2)$$

An interesting bias and variance analysis was performed by Dudik et al[5], who showed that the expected value of the doubly robust estimator will be close to the the policy value if either $\hat{p}(a, x)$ or $\hat{\rho}(x, a)$ have close to zero bias. In addition to that, they concluded that if these two estimators have close to zero bias then the doubly robust estimator will outperform the direct estimator and the inverse propensity scores estimators. On the variance analysis, they concluded that the direct estimator should have the smallest variance, while the doubly robust estimator should have a smaller variance than the inverse propensity score, provided that the magnitude of the error of $\hat{\rho}(x, a)$ is sufficiently small.

Considering the self-normalized inverse propensity score, it was proposed by Kong at [6] and according to Owen[7] it can have either smaller or larger variance than the basic inverse propensity score estimator.

The self-normalized doubly robust estimator was proposed by Thomas and Brunskill [8] as part of their work on offline policy evaluation of Markov decision processes. In their paper, they showed that this estimator is a strongly consistent estimator of the policy value, in spite of being biased.

Considering that there is no clear winner, we decided to use all these metrics (except for the direct estimator) in our analysis and check if their values differed too much in our experiments.

Finally, it is worth mentioning that we only evaluated on the samples coming from uniformly random decisions, so we always have $\hat{p}_{\pi_0}(a, x) =$ 1/number of actions. Since, in the experiment, we did not send random actions to every customer (most of them received actions coming from some personalization policy), we needed to adjust this formula by a correction factor of 1/fraction of clients receiving uniformly random actions.

2.4 Modeling - Counterfactual Learning

2.4.1 When is counterfactual learning possible?

Known impossibility results for counterfactual evaluation using data from a previous policy π_0 also apply to counterfactual learning. The reasoning is simple: if some actions are never explored by π_0 for some contexts, it is impossible to know their outcomes would have been and therefore we

cannot learn if π should pick those actions.

From importance sampling theory[7], what matters most for effective counterfactual learning is how well π_0 explores the actions with favorable rewards.

Since we opted to use ϵ-greedy policies with relatively large ϵ in our experiment, we are always exploring all possible actions, so we know we will be able to perform counterfactual learning.

2.4.2 Indirect approach - regression oracle

The probably more natural approach for policy optimization would be to first build a predictive model that considers the context and an action as input, and then use it to construct an optimal policy which performs the action with the highest prediction for each context, as outlined in detail at algorithm 1.

One important particularity of our approach was to consider as training data only the fraction ϵ of data that correspond to uniformly randomly chosen actions. By doing so, we avoid creating a training dataset that is biased by previous action choices.

Algorithm 1 Selecting the action with the largest predicted reward

```
1:  procedure SelectActions(history, contexts)
2:      random_history ← filter only actions picked randomly history
3:      X_train ← context and actions of random_history
4:      Y_train ← observed rewards of random_history
5:      model  ← create and train a regression model on X_train, Y_train
6:      selected_actions  ← empty list
7:      for x ∈ contexts do
8:          for a ∈ 𝒜 do
9:              Compute model prediction for x, a, model
10:         add action with highest prediction to selected_actions
        return selected_actions
```

Due to some technical limitations in our production environment, we opted to use this approach in production. The specific reasons will be detailed further in the next section.

Since we knew in advance that were going to use this approach, we explored it in more detail.

2.4.2.1 Feature selection

A feature is relevant if the fact that we observe it can lead us to perform an action with a larger expected reward. In fact, one could estimate the value added by a feature F as:

$$\mathbb{E}\Big[\max_{a\in\mathcal{A}}\mathbb{E}\big[r_a|F\big] - \max_{a\in\mathcal{A}}\mathbb{E}\big[r_a\big]\Big] = \mathbb{E}\Big[\max_{a\in\mathcal{A}}\mathbb{E}\big[r_a|F\big]\Big] - \max_{a\in\mathcal{A}}\mathbb{E}\big[r_a\big]$$

One common way of estimating this impact is to segment the feature by bands and determine the action with the highest observed average reward in each band. Then use them to compute the first term of the above formula. The second term can be estimated more easily by simply determining the action with the highest average over all historical data.

Consider now the problem of evaluating feature importance of a model that uses multiple features. Consider a reward prediction model $\hat{\rho}(x, a)$ and let a^* be the action with the largest observed average reward.

One can estimate the value added by personalizing the policy using available features for an individual x as:

$$\widehat{Lift}(x) := \max_{a\in\mathcal{A}}\hat{\rho}(x, a) - \hat{\rho}(x, a^*)$$

When understanding feature importance for a personalization agent, one possible approach is to compute feature importance for a model that estimates lift for an individual. In fact, the features that impact the most on the decisions of this agent are the ones that impact the most on the prediction result of the lift predictor above defined.

An advantage of doing this is the fact that a lot of frameworks already exist for explaining feature importance for predictive models. In our analy-

sis, as detailed in the Experiment section, we used Shapley values[9] as our feature explanation tool.

2.4.2.2 Considerations on model choice

Due to technical limitations in the architecture, we were not able to perform online learning. For this reason, we need a model that was not too computationally heavy, considered it would have to be retrained from scratch every day.

Another important observation was that for most samples the regression target was zero and that most tested models tended to over-fit to the training data. For this reason, we preferred using an ensembled model.

For these reasons, we decided to use random forest regression models in our experiments.

2.4.2.3 Considerations on parameters selection

A common problem that arises in many machine learning problems is overfitting. When performing parameter-tuning, one often has to choose the amount of regularization to apply in order to avoid overfitting and at the same time do not make the model too simple.

Considering random forest models, the usual parameter that is used to add regularization is the maximum depth of the decision trees that are constructed as part of the random forest model. However, in our case, this parameter was not the best choice.

In fact, since our predictive model is trained on more data every day, we observed that models with smaller maximum depths (stronger regularization) led to policies which performed better in the beginning of the experiment and worse after some days of the experiment (due to their inability to express more complex relations). On the other hand, models with larger maximum depths (weaker regularization) had the opposite effect. Since we were looking for models that could be good in all the days of the experiment, we opted to fine tune the minimum number of points in the leafs of the decision trees. By doing so, our agent could be express very complex relations

if provided enough data and still have some guarantees on the degree of uncertainty of the observed relations.

This argument for smart parameters selection applies to any situation in which a model is constantly retrained on more data.

2.4.3 Direct approaches - cost sensitive classification oracle

The main idea here is to make use of direct loss minimization[10] techniques to optimize a policy evaluation metric. This idea was first proposed by Dudík et al [5]. In these approaches, the reward is modelled as a differentiable function of the context and the loss to be minimized is minus the evaluation metric.

In formal terms:

Let the policy π be constructed on top of $f_1, \cdots, f_k$, k reward-prediction differentiable functions, parametrized by k weight vectors $\theta_1, \cdots, \theta_k$. Let π be defined as $\pi(x) := \arg\max_{a \in \mathcal{A}} f_{a,\theta_a}(x)$.

Let $\hat{V}$ be any of the estimates defined in the policy evaluation subsection. Then $-\hat{V}(\pi)$ is a differentiable loss on the weight vectors and can be optimized using gradient descent for points from the training set. Although this loss is very likely not convex, gradient descent still can be applied and may produce interesting performances for some local optimums.

For the sake of having a more regular loss function, one can replace the indicator function in the numerator by a softmax probability $\frac{\exp f_{a,\theta_a}}{\sum_{a' \in \mathcal{A}} \exp f_{a',\theta_{a'}}}$, in which case π is constructed as a stochastic policy.

2.4.3.1 Counterfactual Risk Minimization

In counterfactual risk minimization, which was the state-of-the-art of counterfactual learning[11] as of 2015, one first estimates the inverse propensity score using a biased estimator with low variance and then adds a term that measures the standard deviation of the IPS estimator. In mathematical terms:

Define $u_\pi^{(i)} := r_i \min\left(\frac{\exp f_{a,\theta_a}(x_i)}{p_{\pi_0}(a_i|x_i)}, M\right)$, $\bar{u}_\pi := \frac{1}{|S|}\sum_i u_\pi^{(i)}$, and $Var_\pi(u) = \frac{1}{|S|-1}\sum_i (u_\pi^{(i)} - \bar{u}_\pi)^2$.

According to the counterfactual risk minimization principle, the optimal performance can be estimated as:

$$\hat{\pi}^{CRM} = \arg\min_\pi \bar{u}_\pi + \lambda\sqrt{\frac{Var(u)}{|S|}}$$

for fine-tuned values M and λ.

This algorithm also has a self-normalized version with interesting results.[12]

2.4.3.2 Considerations on method choice

As in model selection in machine learning prediction problems, the best method choice will depend on the dataset of the problem we are solving.

In Lefortier et al work[3], they evaluated the indirect approach, the IPS and doubly robust estimator direct loss minimization, and the counterfactual risk minimization algorithms on a counterfactual learning task in a large dataset of more than 100 million advertisement displays. Their results showed that counterfactual risk minimization achieved the best performance, but not by a large distance from direct loss minimization of the doubly robust estimator.

Since the doubly robust estimator-based counterfactual learning method was already implemented in Vowpal Wabbit[13], an open-source machine learning system library developed at Microsoft Research, and therefore more easily available, we opted to use it as our gold-standard method when performing offline evaluation.

In addition, one concerning result of their analysis was that the direct loss minimization approach performed significantly better than the indirect approaches. For this reason, we could not know before performing the offline evaluation of both strategies on our data if the indirect approaches would be able to obtain similar performances to those from the direct loss minimization approach.

3 Architecture and implementation overview

To apply the personalized policy, we needed to implement an interactive learning system. We outline it at figure 1:

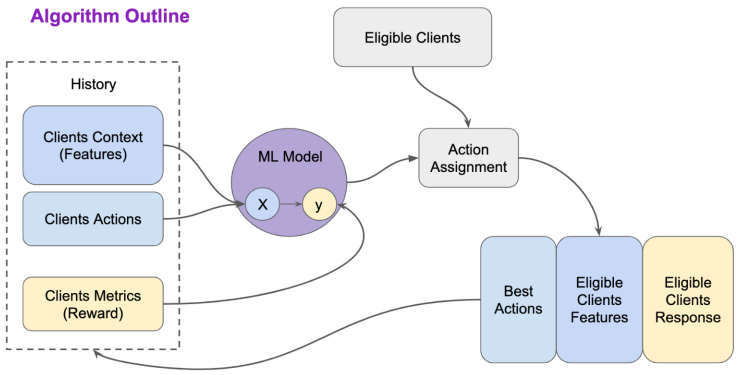

Figure 1: Interactive system outline

3.1 Overview of Nubank's data infrastructure

Nubank's data infrastructure is split in two data warehouses: the first one contains real-time data and is consumed by the micro-services which communicate with our clients; the second one is used for analytics and is constructed using data from the previous day by our ETL (Extract, Transform, Load) pipeline. The data in this data warehouse is more easily queriable while the data in the first one is less mistake-prone, and its history is entirely recorded.

When constructing a policy that may change depending on individuals' contextual information, one can access this information more easily in the analytics warehouse. For this reason, we construct policies which depend on customers contexts in this warehouse. In practice, the policies generate datasets with lists of actions by individual in the analytics warehouse, which are later read by a micro-service who will actually execute them. In particular, this simplifies the machine learning pipeline, as models do not need to be served: they are trained (or loaded from a pickle file) and used to make predictions only once per day.

For this reason, any time we say "action" it should be read as "intention to perform action", for the micro-services do not always perform the actions in theses lists. This may happen for several reasons: micro-services failures, real-time checks on customer status (for example, we stop performing collections actions from the moment a customer has paid his debt. He/she may not have paid when the policy decided to perform the action, but he/she might have paid between the action decision and its execution. We also avoid sending communications for people that "promise" to pay their debts in a conversation with a customer assistant).

Another particularity comes from the fact that the ETL takes around 18 hours to run. For this reason, the actions taken by the policy need to be computed the day before their execution.

Finally, since we decided to compute the personalized policy as part of the ETL pipeline, which is implemented with Spark in Scala, we were forced to also implement our model in Spark. This choice reduced significantly the number of counterfactual learning approaches that could be used in production for the first experiments, but allowed us to use many existing features from the ETL to help reducing the technical debt of the project.

3.2 State-of-the-art architecture

In their paper, Agarwal et al[2] explore some common technical debts in interactive learning systems and propose some design goals to overcome them. We worked to follow as many of these designs as possible in our architecture.

First, some of the mentioned technical debts that such systems may have were:

1. *Partial feedback and bias*: a system that does not explore will reinforce its own biased decisions. In fact, counterfactual evaluation is not possible if some actions for certain contexts are never explored by π_0. And counterfactual learning will not lead to the best policy for a context if the best actions is never explored by π_0. In fact, counterfactual learning will tend to create a policy which selects the best action for a context among those that were likely to happen under policy π_0. Some bias may also be introduced by the feedback evaluation method: the

feedback of an action is observed only in the short-term. The system will therefore tend to select actions with better short-term feed-backs, although from a business point of view we are also concerned about long-term impacts.

2. *Incorrect data collection*: depending on external complexities of the system, there might be a significant number of inconsistencies in the data over time. In particular, individuals' contexts change, so it is important to log their context as of the moment when the decision is made. A common mistake would be to log the context as of the moment when the reward is observed.

3. *Changes in the environment*: real environments are non-stationary. The model must adapt to environment changes over time in order to maintain performance.

4. *Weak monitoring and debugging*: the system must have the ability to reproduce an online run offline. In other words, we need to be able to perform counterfactual evaluation of new policies offline, which is much less expensive. We also need to be able to reproduce the results of online runs, which is a fundamental part of debugging eventual failures.

The authors proposed the following design goals, which we followed as strictly as we could. We emphasize the parts of our architecture, shown in figure 2, where we implemented the design goal

1. *Logging at the point of decision*: in order to be able to perform offline counterfactual evaluation later, we need to log the action and its probability at the point of decision. This is all done at the `CollectionsRL` dataset, and it is later stored as a metadata in the history dataset, `CollectionsRLHistory`. More in detail, the actions proposed by `CollectionsRL` are appended to the set of all actions to be executed by our micro-service, `CollectionsUnion`, with the probability as a metadata. This dataset is archived as `UnionCleanArchive`, a dataset that contains all actions executed the day before today. This dataset is used to construct `CollectionsRLHistory`, which also contains the probability as a metadata.

2. *Experimental unit for joining*: set a pre-set duration for reward observation before new data is appended to the training set, `CollectionsRLDataset`,

so that rewards have a consistent definition. If no reward is observed, append 0 as a default reward. In our case, we set a duration of 5 days. We also monitor the rewards in a time window of 15 days to assess possible biases towards short-term rewards.

3. *Continuous learning*: the system must continue learning indeterminately to capture environment changes over time. In our system, we handle this by retraining the model everyday (as part of the ETL run) using the previous training dataset appended with the results from actions executed 5 days earlier, and considering only data from the previous 60 days.

4. *Reproducibility*: important for debugging. One should be able to fully reproduce an online run offline when debugging. Nubank's ETL datasets were already constructed to be fully reproducible in different dates. In fact, the entire ETL consists of deterministic operations that take as an initial argument the date on which it is supposed to run. However, the Spark model training and prediction unfortunately cannot be reproduced, as these operations are not deterministic. Just specifying a seed did not work as distributed training is not really performed deterministically by Spark. We are currently working on addressing this by storing the model in S3 and making it available for loading if one needs to debug it.

3.2.1 Complete description of project architecture

Here we detail each step of figure 2:

1. Data in the micro-services get extracted to contracts that are available on the analytics data warehouse

2. Datasets combine and transform the contracts and other dataset to analytics needs

3. Collections Policies uses data available across Nubank to make collections decisions

4. All collections policies get merged into the `CollectionUnion` policy dataset. This dataset serializes the actions in a way easy for the services to parse

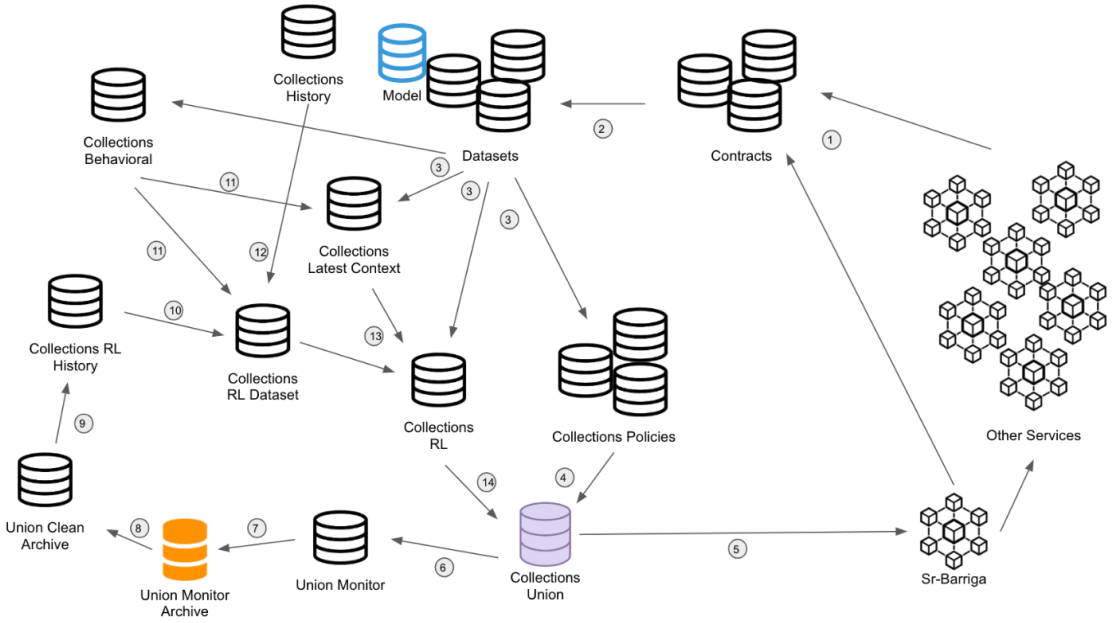

Figure 2: Project interactive learning system

5. `CollectionUnion` is propagated to the micro-service actions, which schedules and execute the actions specified by the policies

6. `CollectionUnion` gets parsed to a human friendly format at `UnionMonitor`

7. `UnionMonitor` is archived. This ensures that all actions taken by the collections policies is stored

8. The archived dataset `UnionMonitorArchive` is read with data from the previous day (ex: actions taken on day 2019-01-01 will only be visible on the day 2019-01-02). The archived is cleaned from known bugs and deduplicated. The result is a cleaned 1-day delayed version of the collections union monitor

9. `CollectionsRLHistory` implements a daily dataset for collections customers. It uses the clean archive to get the actions executed for the customers each day. This dataset is also delayed by a day, since its based on clean archive

10. `CollectionsRLHistory` gets passed to `CollectionsRLDataset`, which

implements the training data of the RL model. `CollectionsRLHistory` gives the information of previously performed actions to the model

11. `CollectionsBehavioral` implements the contextual features used by the RL model. It is passed to the `CollectionsRLHistory` (the training dataset) and the `CollectionsLatestContext` dataset (which is the testing/production dataset)

12. `CollectionLHistory` is used to define the target of the RL model, which is then defined in the `CollectionsRLDataset` (the training dataset)

13. `CollectionsRLDataset` (the training dataset) and `CollectionsLatestContext` (the test/production dataset) gets passed to `CollectionsRL`, which implements the contextual bandits model

14. `CollectionsRL` trains on using the tripled (x, a, r) from the `CollectionsRLDataset`. Then, it explodes the production dataset `CollectionsLatestContext` to contain one extra line for each action from the actions space. It then runs the model reward prediction function on this production dataset (x, a), where a is a simulated action. The model than selects the best performing action with probability $1 - \epsilon$ and a random action with probability ϵ. This action is finally passed to the `CollectionsUnion` dataset

Remark: The apparent circular dependencies on the dataset is broken by the archive.

4 Experiment

4.1 Experiment setup

We worked on a proof-of-concept policy to personalize communications sent to late customers after 34 days late. During the first month of the experiment, we sent one of 18 actions with uniform probability to each customer in day 34. After this period, we started using a policy implementing the indirect approach from section 2.3.2 to decide the actions for 50% of our customers. We kept 50% of our actions randomly, which provided a continuous source of data to be used in offline counterfactual evaluation. Twenty days after releasing the first personalized policy, we started testing 8 other policy types, applying each personalized policy to 8% of our clients and keeping 28% of our clients with the uniformly random policy.

The choice of sending the communications after 34 days late represented a good trade-off between having a significant number of customers which the policy acted on every day (which decreases as we go later in the collections timeline) and not suffering too much from eventual noise at the operations team (actions early in the collections in the timeline have a larger impact on the operations team, as they increase the number of tickets and calls coming from a larger number of customers).

Our policy impacts on an average of about 2.500 customers per day, which is the number of customers who get to exactly 34 days late in their bills every day.

We set the reward of the experiment to be short-term payments. More specifically, payments in the 5 days that follow the communication. Other business metrics could work too, such as number of debt cures and number of tickets, as well as some more complex metrics which approximated the customer net present value (NPV).

We sent 18 different actions: 15 of them with soft content and the other 3 with harsher content:

- Soft actions:
  - Emails: contact_us, contact_us_2, payment_options, avoid_interest,

unblock_your_card, payment_options_2,

  - Pushed notifications: payment_options, avoid_interest, contact_us, late_reminder_1, unblock_your_card, late_reminder_2
  - SMSs: unblock_your_card, keep_your_account
  - Do nothing action (named in graphs as *drop_drop*)

- Harsh actions:
  - Email: extra_judicial_threat
  - Pushed notification: extra_judicial_threat
  - SMS: internal_delinquent_3

As initial evidence of different impacts of these actions, we observed the average payments for each of them over the first month of the experiment, which we show in figure 3.

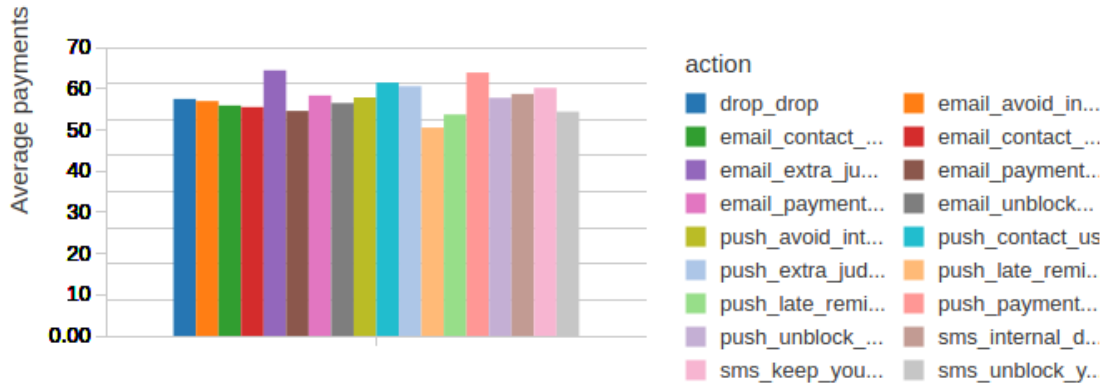

Figure 3: Average short-term payments for each action. Each average is computed on around 4000 observed samples.

As initial evidence of the potential of personalization, we observed that the most effective communications change significantly depending on customers' risk levels and their ages, as in figures 4 and 5.

In figure 4, we can observe that riskier customers react differently to communications than less risky customers. In fact, some of the best performing communications for low risk customers perform below average for high risk customers.

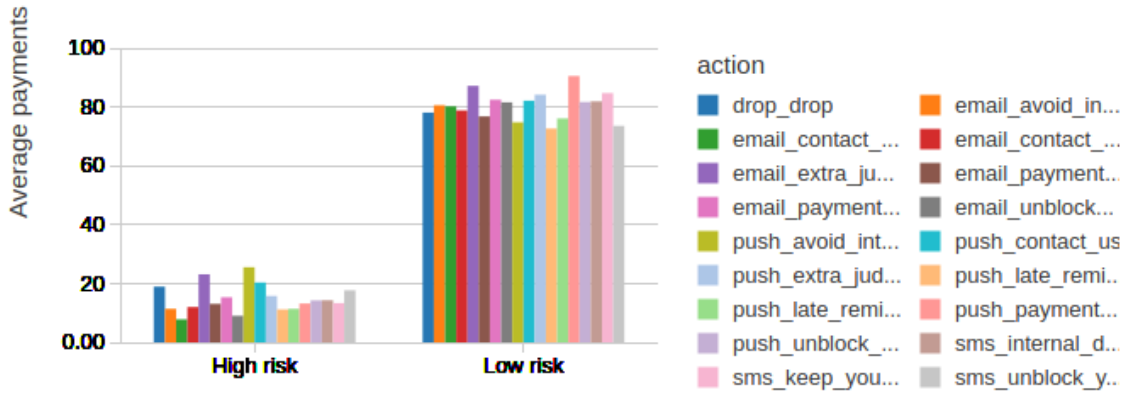

Figure 4: Average short-term payments for each action for two risk groups.

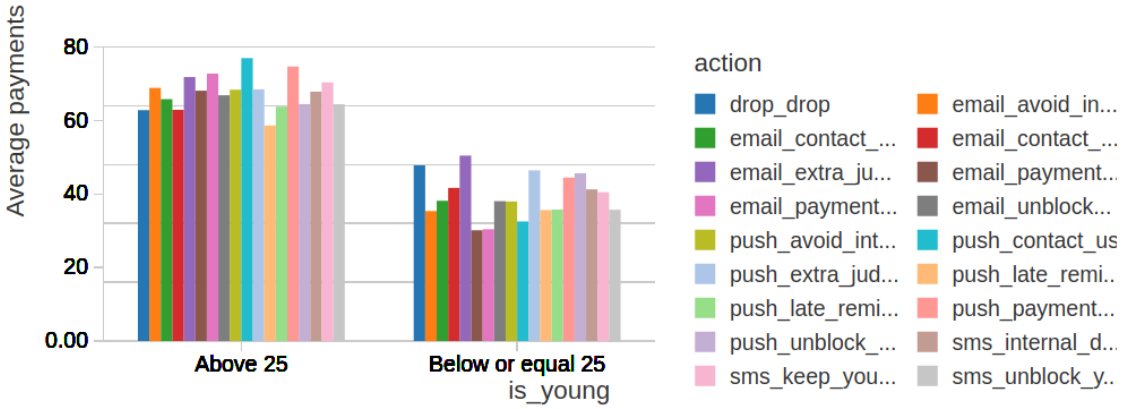

Figure 5: Average short-term payments for each action for two age groups.

In figure 5, we can observe that, to some degree, young customers react better to harsher communications (in particular, those which mention an extrajudicial letter), while older customers react better to softer communications, which mention payment options and ask them to contact Nubank.

4.1.1 Some statistical tests

4.1.1.1 Comparing actions average impacts

We ran a Kruskal statistical test to evaluate if there was any action such that the payments following that action stochastically dominated those from another action, using all past data from our experiment (around 90000 samples).

We obtained a p-value of 0.34, which does not allow us to reject the null hypothesis. Putting very simply, this shows that probably there is no action that, in median, is better than another action for all situations.

If we had obtained a smaller p-value, then we would have been able to say that the impact of some action is better than the one from another action, in average. It is disappointing not to be able to say this directly. Nevertheless, this is far from sufficient to say that a personalized policy has no value. At best, it says that a policy that always sends the same action might not beat a random policy.

Additionally, we checked for the significance of the actions impact by fitting a linear model with the action as the single context feature and using fixed time effects. This allowed us to estimate one coefficient by action and compute confidence intervals for each action coefficient. One interesting result was that some actions had a p-value of 0.20 (for the null hypothesis that that coefficient is zero for the considered action).

4.1.1.2 Exploring the impact of implementing a personalized policy

We wanted to construct a personalized policy that performed better than a greedy policy with some statistical significance. For this reason,

we estimated the lift of a policy that splits clients by some feature in two groups and performs the best performing action for each group, compared to a greedy policy that performs the same action for all clients.

We inspired ourselves by the formula from the problem definition section: $\mathbb{E}\Big[\max_{a\in\mathcal{A}}\mathbb{E}\big[r_a|F\big]\Big] - \max_{a\in\mathcal{A}}\mathbb{E}\big[r_a\big]$.

We estimated the gain of a segmented policy (by the risk feature *healbot 3 prediction*, in 2 groups) against a greedy policy without any segmentation. After controlling these estimates by risk and late balance to estimate the impact of the best action in each group compared to the overall best action, we could obtain a p-value of 0.10 for the null hypothesis that assigning the personalized action according to this segmentation has no impact compared to assigning the overall best action to an individual. We used a fixed effects model with time effects, fitted on data from each group separately: $y_{i,t} \sim c_t + \texttt{risk}_i + \texttt{late balance}_i + 1(\texttt{assigned personalized action})_i$.

4.2 Offline evaluation: methodology and results

Before sending any new policy to production to make actual decisions, we performed offline evaluation to understand how they would have performed in the past.

Then, after observing the offline evaluation results, we sent the best performing strategies to production and observed their actual performance when taking decisions.

4.2.1 Methodology

Since our interactive learning system updates the policy everyday, we were not really interested in evaluate a specific policy, but a strategy for constructing optimal policies everyday. A strategy can be defined as the construction rule that we use to construct the policy that will be used today provided data from past policies executions.

An example of strategy is to use the indirect approach defined in section 2.3.2 using a random forest with 200 estimators and a maximum depth of

4 as the rewards prediction model. In this strategy, this model is retrained everyday with more recent data, the policy changes when the trained model changes, but the rule used to construct it stays the same.

Our evaluation methodology looks to estimate how a strategy would have performed if we had applied it since the beginning of the experiment.

We split the data in a training set and a test set. For each day in the training set, we perform an offline evaluation of a policy constructed using all the data from before that day. We evaluate all the metrics described in section 2.2 using algorithm 2. We then compute a weighted average of these results (the weights are the number of examples of each day). This number is an estimate of how the policy would have performed if it had been deployed since the beginning.

We use this estimate to determine the best hyper-parameters for our strategy (e.g., if it is based on a random forest model with a fine-tunable max depth, we select the max depth value that produces the largest estimated strategy value).

To assess how the fine-tuned strategy will perform after it is deployed, we go through a similar process using the data from the test set. We perform offline evaluation on each day of the test set. This time, for each day from the test set, we use all the available data from before that day, including data from both the training and the test set.

In the tables reported in this section, we put these results in columns *Train estimated value* and *Test estimated value*.

Algorithm 2 Offline evaluating a strategy over a period of days data

```
1: procedure OfflineEvaluate(strategy, data)
2:     offline_results ← empty list
3:     dates ← get dates which appear in data
4:     for date ∈ dates do
5:         training_data ← filter data from before a certain date data date
6:         evaluation_data ← filter data from a certain date data date
7:         policy ← construct policy from strategy, training_data
8:         offline_result ← offline evaluate policy, evaluation_data
9:         add offline_result to offline_results
       return offline_results
```

4.2.2 Baseline strategies

Our personalized policy should be able to beat some simple baselines:

1. *Do nothing strategy*: this strategy always picks the action which consists of not sending any communication at all
2. *Random strategy*: this strategy picks any of the 18 actions with equal probability for each individual
3. *A/B test strategy*: this strategy looks at past data to compute the average of observed payments for each action. Then, it picks the action with the largest average for all clients
4. *Segmentation strategy*: in this strategy, we choose a feature and a number k of quantiles in the beginning. Then, we split the feature in k quantiles and observe the average payment for each action for each quantile. The constructed policy consists of determining which quantile the value of the feature from a customer is in and then performing the action that has the largest observed average payment in that quantile.

4.2.3 Feature selection

We considered the following features to construct our model:

- *External agency*: categorical - the name of the external agency hired to help collection this customer's debt
- *Due installments before the collection*: - continuous - total due installments of the client one day before entering collections
- *Installments issuer before the collection* - continuous - late due installments of the client one day before entering collections
- *Renegotiated debt* - continuous - if an older debt was renegotiated, this feature contains the value of that debt. Otherwise it is null.

- *Revolving* - continuous - if the customer has only paid the minimum required amount in the previous month, this feature contains the value of the remaining debt. Otherwise it is null
- *Boa vista consult score* - continuous - contains the credit score in range $[0, 1]$ computed by Boa Vista SCPC, a credit bureau
- *Boa vista estimated income* - continuous - contains the customer's income estimated by Boa Vista SCPC
- *Collection initial debt* - continuous
- *Collection index* - discrete - 1 if it is the first time the customer is late in their bills, 2 if it is the second time, and so on
- *Customer reported income* - continuous - (unverified) income reported by the customer
- *Customer shipping address state* - categorical
- *Customer age* - continuous - age up to the sixth decimal case
- *Healbot 2 prediction* - continuous - estimated probability that the customer will pay his debt in the next thirty days, according to our risk prediction machine learning-based model
- *Healbot 3 prediction* - continuous - similar, but using a more recent version of the risk prediction model
- *If quantity* - discrete - number of financial institutions which the customer holds credit products at. Reported by the central bank
- *Late balance* - continuous - current debt amount
- *NuConta gross balance* - continuous - balance of the bank account that the customer has at Nubank
- *NuConta status* - categorical - whether the customer has a bank account at Nubank. It is false for most customers.
- *Ops quantity* - discrete - number of credit products held by the customer at Brazilian financial institutions. Reported by the central bank
- *Promise status at date* - binary - whether the customer has communicated his intention to pay his debt to one of our agents

Feature	Train estimated value	Test estimated value
Previous due installments	53.59	70.71
Previous installments issuer	55.74	73.69
Renegotiated debt	54.06	70.97
Boa vista consult score	59.02	50.08
Boa vista estimated income	61.81	57.27
Collection initial debt	54.18	63.62
Collection index	61.99	81.06
Customer reported income	63.21	69.84
Customer age	58.86	71.03
Healbot 2 prediction	63.84	67.23
Healbot 3 prediction	58.82	65.33
If quantity	55.50	64.25
Late balance	57.31	65.40
NuConta gross balance	56.69	71.67
Ops quantity	64.38	68.67

Table 1: Median results of segmentation strategies based on each numerical feature. Used SNIPS as the evaluation metric

4.2.3.1 Feature singly importance

We evaluated the performance of a segmentation strategy for each feature and report here their performances, which we obtained according to algorithm 2. We evaluated strategies using from 2 to 10 quantiles for all features and computed the median of their performances for each feature, reported at table 1. In this analysis, we only considered only numerical features.

We expect that the learning algorithm (specially if it as tree-based algorithm) will be able to learn the best number of quantiles to use, which is quite analogous to learn where to create a split.

We also evaluate feature singly importance by estimating the performance of model-based strategies that use exclusively the evaluated feature. We report at table 2 the average of the results of such a strategy using 200 estimators and 2048, 4096, 8192, 16384, and 32768 as the minimum leaf

Feature	Train estimated value	Test estimated value
Previous due installments	59.51	67.94
Previous installments issuer	62.92	60.14
Renegotiated debt	62.25	54.66
Boa vista consult score	59.21	65.29
Boa vista estimated income	66.69	72.40
Collection initial debt	58.59	57.67
Collection index	60.48	60.41
Customer reported income	61.47	59.88
Customer age	61.47	61.26
Healbot 2 prediction	63.65	66.87
Healbot 3 prediction	63.29	62.03
If quantity	60.58	58.12
Late balance	61.19	60.60
NuConta gross balance	60.70	71.83
Ops quantity	61.34	70.67

Table 2: Average results of random forest strategies based on each numerical feature alone. Used SNIPS as the evaluation metric

node size in the ensembled decision trees when creating a random forests model. These hyper-parameters were selected because the median of the performances of strategies using those hyper-parameters (but different single features) was considerably higher in the training set.

4.2.3.2 Feature importance

To understand feature importance, we trained a rewards-prediction model using all these features and then used Shapley values as a feature explainer tool, as put in more detail in section 2.3.2.1.

In fact, this rewards-prediction model allows us to construct a lift prediction function by simply computing the highest reward prediction among all possible actions and then subtracting the prediction for the do nothing action.

In figure 6, we can observe each feature's contribution for the predicted lift for each point in the dataset, when using a random forest with 200 estimators and maximum depth 4 as the rewards-prediction model.

The results are more interesting if we use a maximum depth of 6, although the strategy based on this model actually performs quite worse. They are in table 7.

It is worth mentioning that, after conducting the same analysis to the strategy based on a random forest model that uses 4096 as the minimum number of points per leaf node, we could observe that its model actually gives 0 importance to context features, which means that it is equivalent to the random strategy.

4.2.4 Offline evaluation of strategies

Besides the baseline strategies, we evaluated model-based strategies that implement the indirect approach as well as strategies that implement direct loss minimization algorithms using Vowpal Wabbit.

We report in table 3 the observed results for each of the tested strategies.

In this table, we report the performances for the following fine-tuned hyper-parameters for each strategy:

1. *Segmentation*: segment on *Ops quantity* in 6 bands.
2. *Random forest*: a random forest model with a minimum of 8192 points per leaf node in each of its 200 ensembled decision trees
3. *Random forest with one feature*: a random forest model that used *Healbot 2 prediction* as its single context feature and forced a minimum of 16384 points per leaf node in each of its 200 ensembled trees
4. *Direct loss minimization with Vowpal Wabbit*: a linear model that minimizes doubly robust loss metric, using L1 regularization (0.001), a learning rate of 1.0, and a minibatch of 8

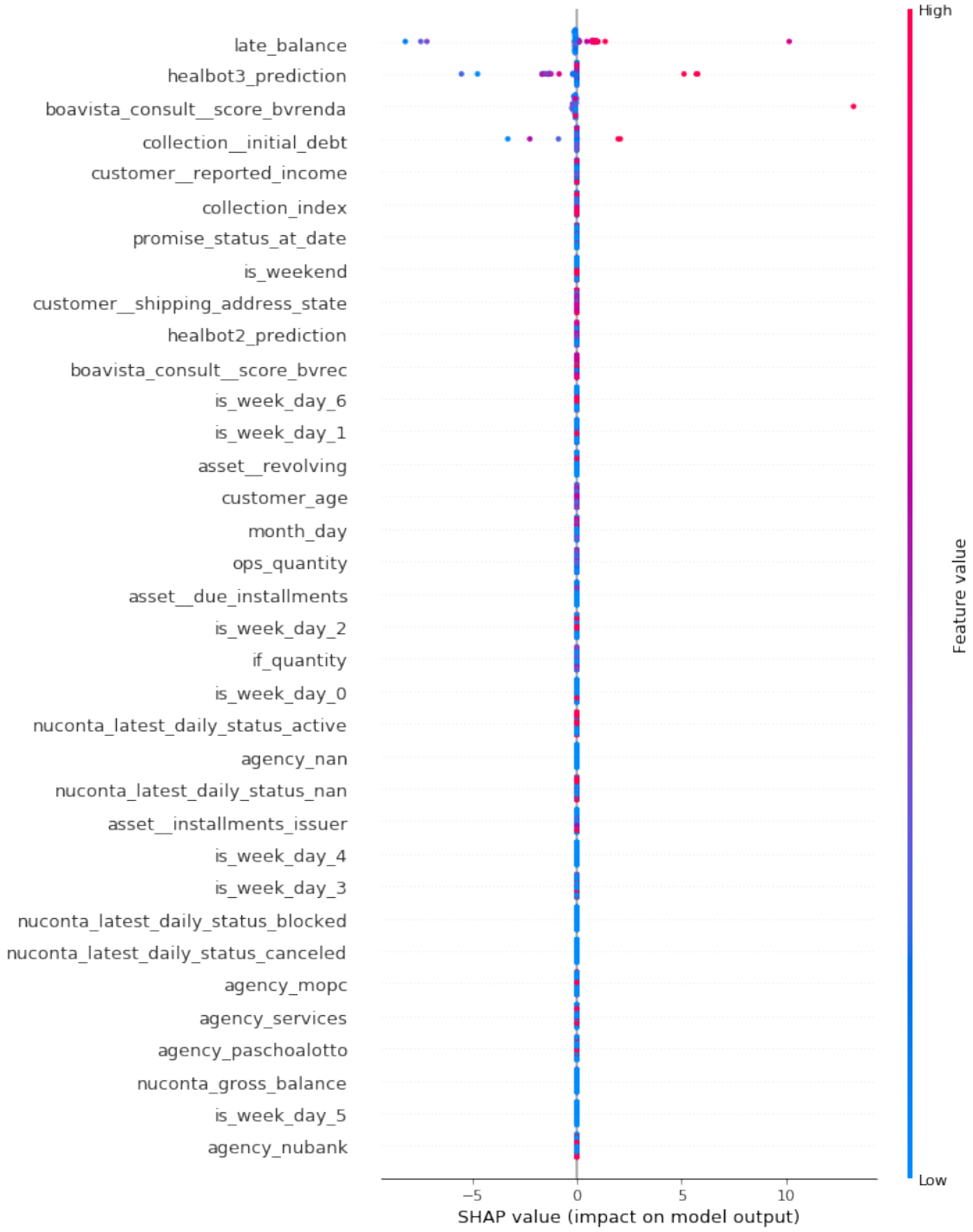

Figure 6: Visualization of feature importance for lift prediction - Shapley values

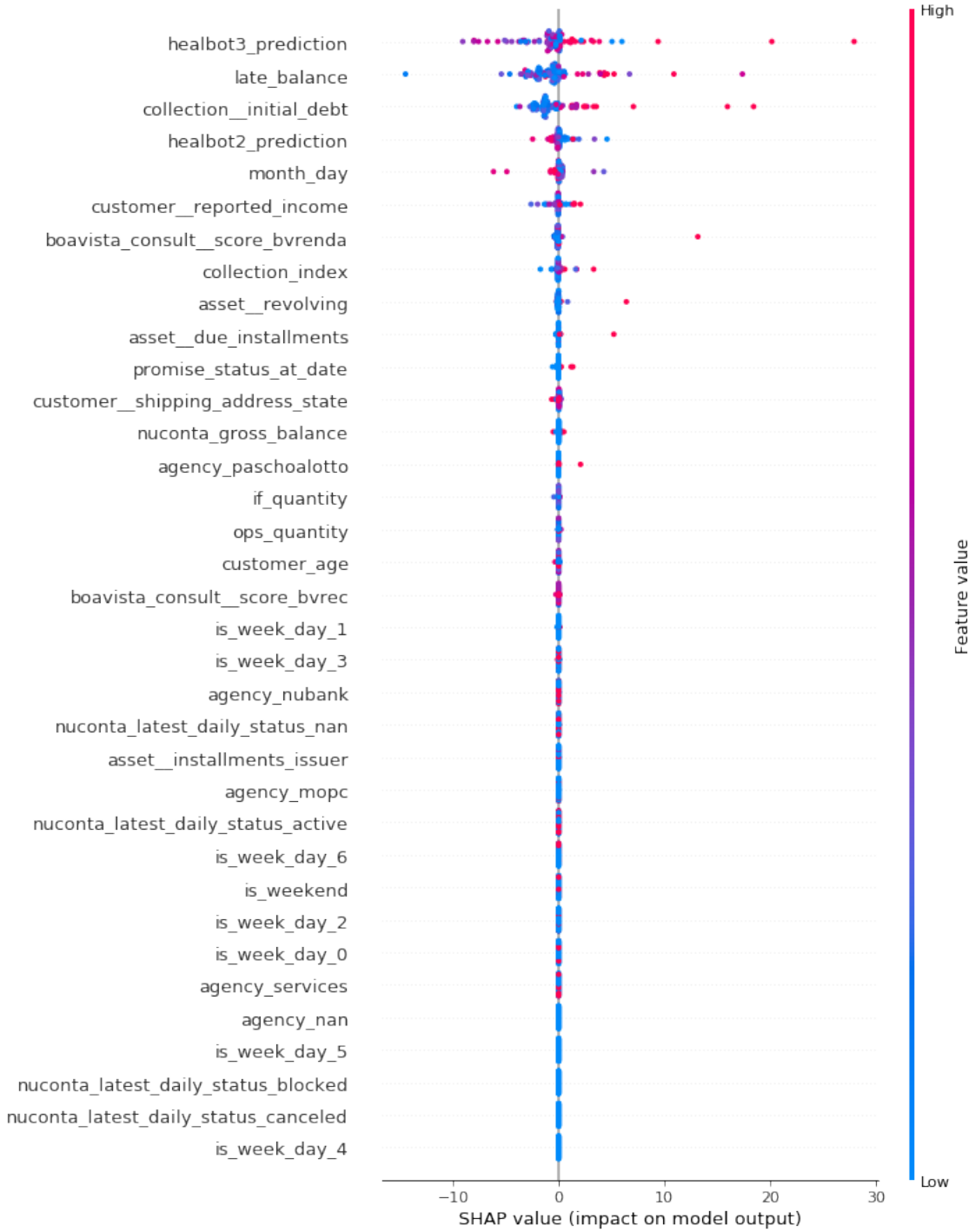

Figure 7: Visualization of feature importance for lift prediction - Shapley values

Baselines		
Strategy	Train estimated value	Test estimated value
Do nothing	61.34	60.39
Random	61.63	57.99
A/B test	56.37	63.87
Segmentation	72.40	64.36
Tested strategies		
Strategy	Train estimated value	Test estimated value
Random forest	66.99	76.83
Random forest- single feature	70.97	84.63
DLM - Vowpal Wabbit	61.43	60.39

Table 3: Offline evaluation for each policy. Used SNIPS as the evaluation metric

4.2.5 Comparing metrics

We compared the results using each of the evaluated metrics (IPS, SNIPS, DE, DR, SNDR, as described in section 2) in table 4, in which we evaluated the random forest based strategy with the fine-tuned hyper-parameters (200 estimators and at least 8192 points per leaf node).

Our understanding is that the metrics produced similar results, except for the direct estimator, which was expected. The self-normalized inverse propensity score (SNIPS) produced estimates that were not too far from the other estimates (less than 2% far).

This result was quite reassuring, as it confirmed that we could use SNIPS to compare all the strategies, including those for which DE, DR, and SNDR are not so easily defined. In fact, for strategies that are not based on a rewards-prediction model, there is no inherent rewards-prediction model that we can use to compute those estimators.

Dataset	DE	IPS	SNIPS	DR	SNDR
Training data	68.59	65.40	66.99	65.96	66.59
Test data	64.80	74.40	76.83	75.14	76.20

Table 4: Offline evaluation for the same policy, with different evaluation metrics

4.2.6 Uncertainty

We could observe that there is considerable uncertainty in the results obtained through offline evaluation. In fact, much of the uncertainty comes from the fact that both IPS and SNIPS metrics are by definition computed using only a small fraction of the available data. More precisely, on 1/18 of the data.

To better visualize this phenomenon, we plot in figure 8 a histogram of the results of estimating the SNIPS of a random policy using our evaluation methodology. We could also observe similar histograms for the random forest-based policy when varying the model's seed in its implementation.

For this reason, we needed to evaluate our strategies on real data, which we report in the next sub-section.

4.3 Real-life results

In the production set up, we removed some of the features, either because they were not so important based on their Shapley values or because they were too prone to breaking, which would by consequence break our model.

We ran 10 different strategies according to the following proportions: every day, 28% of clients received a uniformly random communcation, while, for each of 9 strategies, 8% of the clients received a communication according to that strategy. The above explained methods are represented in at least one strategy, with the selected hyper-parameters (the hyper-parameters can differ from the results above, considering that they were determined in an earlier stage of the project).

In table 5, we report the average returns of each of these strategies from

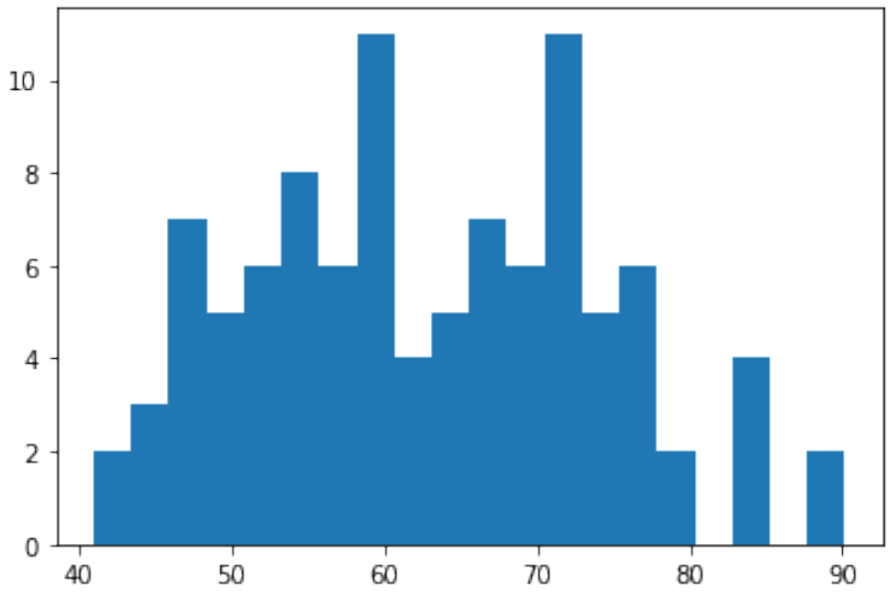

Figure 8: Histogram of estimates of the performance of the random strategy on the test set

August 15th to August 28th. The results are relatively worrying considering that the best personalization strategy (a random forest with decision trees with at least 4096 instances per leaf node) is known to be equivalent to a uniformly random strategy (from the feature importance analysis). It is less worrying if we remember that we are not considering a sufficient number of days in this analysis, specially because our analysis did not include days in the beginning of the month, which is the period when we observe more payments.

In the beginning of the experiment, we ran only two strategies: a uniformly random policy for 50% of the clients and a greedy policy based on a random forest model with max depth 4 for the other 50%. We report in table 6 the average returns of each of these strategies by week from July 25th to August 28th.

Baselines		
Strategy	Hyper-parameters	Return
Random	n.a.	48.56
A/B test	n.a.	41.87
Segmentation	7 bands of risk	47.64
Segmentation	10 bands of customer age	45.51
Segmentation	10 bands of late balance	**53.80**
Tested strategies		
Strategy	Hyper-parameters	Return
Random forest	max depth 4	48.90
Random forest	max depth 6	44.05
Random forest	min instances per node 2	46.83
Random forest	min instances per node 4096	51.14
Random forest - one feature	max depth 4, risk	42.30

Table 5: Average returns after 5 days from August 15$^{\text{th}}$ to August 28$^{\text{th}}$

Baseline		
Strategy	Hyper-parameters	Return
Random	n.a.	52.76
Tested strategies		
Strategy	Hyper-parameters	Return
Random forest	max depth 4	**54.13**

Table 6: Average returns after 5 days from July 25$^{\text{th}}$ to August 28$^{\text{th}}$

Baselines			
Strategy	Hyper-parameters	5 days	15 days
Random	n.a.	47.24	105.96
A/B test	n.a.	41.13	90.86
Segmentation	7 bands of risk	43.70	110.28
Segmentation	10 bands of customer age	42.63	116.67
Segmentation	10 bands of late balance	50.21	102.91
Tested strategies			
Strategy	Hyper-parameters	5 days	15 days
Random forest	max depth 4	45.98	104.07
Random forest	max depth 6	41.74	101.10
Random forest	min instances per node 2	44.16	**117.53**
Random forest	min instances per node 4096	47.45	**118.58**
Random forest - one feature	max depth 4, risk	38.23	102.15

Table 7: Average returns after 5 and 15 days from August 15$^{\text{th}}$ to August 19$^{\text{th}}$

4.3.1 Medium-term impact

We also wanted to understand the medium-term impact. In fact, it would be a little disappointing if we observed that the communications only had an advancing effect on the paid amount. Quite the contrary, the communications effect measured as payments over 15 days proved to be larger than the effect measured on 5 days, as reported in tables 7 and 8.

In fact, when we look at the larger time space with less strategies, we observe that the difference in payments average between the random forest and the random strategy increases from 0.57 to 1.95 (more than doubles) when we increase the window that we use to observe payments.

For similar reasons as the ones presented in the last analysis, we prefer to disregard the results of strategies evaluated in the shorter time space.

	Baseline		
Strategy	Hyper-parameters	5 days	15 days
Random	n.a.	51.92	130.82
	Tested strategies		
Strategy	Hyper-parameters	5 days	15 days
Random forest	max depth 4	52.49	**132.77**

Table 8: Average returns after 5 and 15 days from July 25$^{\text{th}}$ to August 19$^{\text{th}}$

4.3.2 A statistical test

Comparing the returns of the random policy and the random forest based policy with a Kruskal test on 25000 individuals that received each of the policies, we obtained a 0.66 p-value. This means that, with the data that we have now, we cannot conclude that the personalized policy is better than the random one using this test.

Using a better test that takes into account contextual information of individuals, we could obtain a much better p-value. By fitting a fixed time effects linear model on our data, controlling by risk and late balance, we could estimate the impact of assigning an individual to receive actions determined by the personalized policy (compared to assigning to the random one). We could obtain a p-value of 0.13 for the null hypothesis that this impact is zero. In mathematical terms, we fitted the model: $y_{i,t} \sim c_t + \texttt{risk}_i + \texttt{late balance}_i + 1(\texttt{assigned to personalized policy})_i$

4.3.3 Next steps

Although the personalized policy that we constructed has been able to generate more payments than the random policy, we could only deny the null hypothesis of our statistical test using a relatively high significance, which means that this result might have happened only by chance.

As part of the next steps of the experiment, we certainly should consider reducing the number of possible actions: with less actions, we will be able to better estimate the impact of each of the possible actions, as we will have more samples for each action. This will allows us to do better use of the

offline evaluation techniques and have better statistical guarantees.

Considering the policy construction rule, we plan to, as a next step, use also the data coming from individuals who received actions from a greedy policy.

5 Conclusion

In this study, we presented a practical use case of contextual bandits algorithms for constructing personalized policies. We covered both the theoretical and the implementation aspects of the development of such an interactive learning system, outlining the step by step procedure for the development of its architecture and the design choices, the offline evaluation of possible strategies, and the results of the selected strategies since they started being used in production.

Despite the relatively large variance of the observed targets, we believe that the increased debt payments observed by our personalized policy during a period of more than a month is motivating enough for developing new personalized policies in this domain. In fact, the increase of 1.95 reais per customer corresponds to an increase of 156.000 reais (or 37.000 dollars) per month, which pays off the effort that was put constructing the system.

Thirdly, we believe that the design choices of our experiment permitted some valuable features, which include: the development and evaluation of good predictive models, the ability of performing counterfactual analysis (from new policies evaluation to context features importance explaining), the reproducibility of the actions performed by the system, and the ability to adapt to non-stationary data. These features were key in obtaining successful experiment results when measuring observed payments.

References

[1] Alberto Bietti, Alekh Agarwal, and John Langford. A contextual bandit bake-off. *arXiv preprint arXiv:1802.04064*, 2018.

[2] Alekh Agarwal, Sarah Bird, Markus Cozowicz, Luong Hoang, John Langford, Stephen Lee, Jiaji Li, Dan Melamed, Gal Oshri, Oswaldo Ribas, et al. Making contextual decisions with low technical debt. *arXiv preprint arXiv:1606.03966*, 2016.

[3] Damien Lefortier, Adith Swaminathan, Xiaotao Gu, Thorsten Joachims, and Maarten de Rijke. Large-scale validation of counterfactual learning methods: A test-bed. *arXiv preprint arXiv:1612.00367*, 2016.

[4] Naoki Abe, Prem Melville, Cezar Pendus, Chandan K Reddy, David L Jensen, Vince P Thomas, James J Bennett, Gary F Anderson, Brent R Cooley, Melissa Kowalczyk, et al. Optimizing debt collections using constrained reinforcement learning. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 75–84. ACM, 2010.

[5] Miroslav Dudík, John Langford, and Lihong Li. Doubly robust policy evaluation and learning. *arXiv preprint arXiv:1103.4601*, 2011.

[6] Augustine Kong. A note on importance sampling using standardized weights. *University of Chicago, Dept. of Statistics, Tech. Rep*, 348, 1992.

[7] Art B. Owen. *Monte Carlo theory, methods and examples.* 2013.

[8] Philip Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. In *International Conference on Machine Learning*, pages 2139–2148, 2016.

[9] Lloyd S Shapley. A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317, 1953.

[10] Tamir Hazan, Joseph Keshet, and David A McAllester. Direct loss minimization for structured prediction. In *Advances in Neural Information Processing Systems*, pages 1594–1602, 2010.

[11] Adith Swaminathan and Thorsten Joachims. Counterfactual risk minimization: Learning from logged bandit feedback. In *International Conference on Machine Learning*, pages 814–823, 2015.

[12] Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. In *advances in neural information processing systems*, pages 3231–3239, 2015.

[13] Vowpal Wabbit. https://github.com/VowpalWabbit/vowpal_wabbit.