Online Prediction Threshold Optimization Under Semi-deferred Labelling

Yorick Spenrath[1,*], Marwan Hassani[1] and Boudewijn F. van Dongen[1]

[1]Process Analytics Group, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands

Abstract

In supermarket loyalty campaigns, shoppers collect stamps to redeem limited-time luxury products. Having an accurate prediction of which shoppers will eventually redeem is crucial to effective execution. With the ultimate goal of changing shopper behavior, it is important to ensure an adequate number of rewards and to be able to steer promising shoppers into joining the campaign and redeeming a reward. If information from previous campaigns is available, a prediction model can be built to predict the redemption probability, possibly also adapting the prediction threshold to determine predicted the label. During a running campaign, we only know a subset of the labels of the positive class (the so-far redeemers), and have no access to the labels of any example of the negative class (non-redeemers at the end of the campaign). The majority of the examples during the campaign do not have a label yet (shoppers that could still redeem but have not done so yet). This is a *semi-deferred* labelling setting and our goal is to improve the prediction quality using this limited information. Existing work on predicting (semi-deferred) labels either focuses on positive-unlabelled learning, which does not use existing models, or updates models after the prediction is made by assigning expected labels using unsupervised learning models. In this paper we present a framework for Online Prediction threshold optimization Under Semi-deferred labelling (OPUS). Our framework does not change the existing model, but instead adapts the prediction threshold that decides which probability is required for a positive label, based on the semi-deferred labels we already know. We apply OPUS to two real-world datasets: a supermarket with two campaigns and over 160 000 shoppers.

Keywords

Prediction threshold, Online, Semi-deferred labels, Supermarket loyalty campaign

1. Introduction

Traditional supervised machine learning projects start with a set of labelled data. Specifically in binary classification, data belongs to one of two classes: positive and negative. Starting from a set of examples with their ground truth label, a predictor is created, which can assess the probability that an unseen example belongs to the positive class. Without optimization, an example is predicted to be part of the positive class if this probability is at least 0.5. One improvement is to select a different *prediction threshold* that distinguishes examples between the positive and negative class on their probability. In the presence of labelled data, such a threshold can be picked in a way that the predicted labels maximize a given metric. Provided that no concept drift occurs, the model with the adapted threshold can be used indefinitely without degrading prediction quality. In this scenario, we only use the 'offline training' (I) of Figure 1.

This assumption on the data is however not realistic since changes in the data distribution such as concept drift can reduce the quality of the model. A common technique is to retrain or update the model at a later point in time, when more recent labelled data is available [1] and potentially also updating the prediction threshold. This training is referred to as *online* training, where new labelled examples are received and can be used to update the model and/or prediction threshold. Such techniques assume that the label latency, that is the time between seeing an example and its true label, is small. Under these conditions prediction systems can react to concept drift as it occurs. This assumption may not be valid in all real-world scenarios, for example because it is computationally or labour intensive to do so [2].

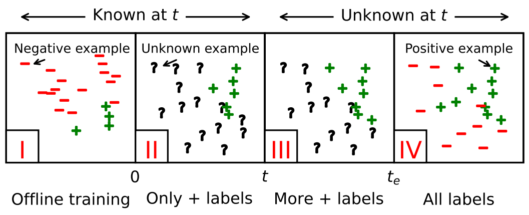


Figure 1: I) In an offline phase, all labels are present. II) In the online phase, this data is potentially outdated, but we do know a limited number of positive labels. III) The set of positive labels grows as time progresses. IV) At time t_e, all labels of the online phase are known. At time t we can only use the offline data (I) and the positive and unlabelled examples (II).

If the latency becomes too high, models can no longer properly be corrected for concept drift, or not even be updated at all if the latency is infinite [3].

In this paper we consider a special class of the latter, where we know for some examples that they are positive and do not know the label of the other examples (they may be either positive or negative). This is sketched in Figure 1. At time t, we have the (potentially outdated) offline training data. We have further seen new examples, of which we know some are positive. It is however not until time t_e that we know the actual labels of all examples, and up to that point we only receive the actual label of positive examples. We refer to this as *semi-deferred* labelling, as we have some labelled examples, but only from one class. This still does not allow us to update or retrain the model, neither does it allow us to set a new threshold that optimizes our target metric. Setting this new threshold is important, since we still want to make a prediction for all unlabelled examples. We therefore introduce OPUS, Online Prediction threshold optimization Under Semi-deferred labelling, which aims to set a better threshold for an existing prediction model, based on the limited available positive examples. Note that this is not the same as *imbalanced* training, as we have only a *single* class of up-to-date data and do not make assumptions

Published in the Proceedings of the Workshops of the EDBT/ICDT 2024 Joint Conference (March 25-28, 2024), Paestum, Italy
*Corresponding author.
y.spenrath@tue.nl (Y. Spenrath); m.hassani@tue.nl (M. Hassani); b.f.v.dongen@tue.nl (B. F. v. Dongen)
0000-0003-0908-9163 (Y. Spenrath); 0000-0002-4027-4351 (M. Hassani); 0000-0002-3978-6464 (B. F. v. Dongen)

about the possibly outdated data.

Even though OPUS can be generalized to other scenarios, we discuss a supermarket one in this paper. Shoppers in this supermarket make use of loyalty cards which identify them at each purchase they make. Once a year, the supermarket holds a so-called loyalty campaign [4]. During the campaign, shoppers may collect stamps. These stamps can be collected by spend (for example one stamp for every 10 monetary units) or in special promotions (an extra stamp for a certain product). If a shopper has collected enough stamps they can purchase a reward. Rewards are usually luxurious products available at a much lower (sometimes only symbolic) price. Shoppers are as such persuaded to participate in the loyalty campaign. In this work we consider campaigns with a limited time scope: shoppers can only collect the stamps and redeem rewards within the duration of the campaign. For such a campaign to work, the supermarket must accurately predict how many rewards are required, such that each participating shopper can actually get their rewards. Having too few rewards means disappointing shoppers, having too many means there will be unusable stock left. It is beneficial to make predictions about what shoppers will do during the rest of the campaign, in an effort to either steer their decisions or adapt the available rewards later on.

For our prediction model, we are interested in whether a shopper will make a redemption: shoppers have positive labels if they redeem a reward during the campaign, or negative labels otherwise. Consider two campaigns, p_1 and p_2. During p_2 we want to predict whether a shopper will redeem, based on a classifier we learned from the finished campaign p_1. We continuously receive new data and make new predictions and the end of every week. In practice this means we get new positive examples in batches, these are all consumers that first redeemed in the preceding week. Let t be the moment of prediction, since the start of p_2. We train a classifier and select the best threshold. To do so, we split the data of campaign p_1 into a train and validation set, and train several models on the former and adjust the threshold for each model to maximize a target metric using the latter. One of these models has the highest metric value and we select that as our prediction model. We retrain that model on all shoppers of campaign p_1, and test it on campaign p_2, using the corresponding learned optimal threshold. The problem with this approach is that the model trained on campaign p_1 might not work as well on campaign p_2. We also do not get new negative labels during p_2, which means that updating our prediction model or creating a new one is not feasible. The best we can do is to adapt the prediction threshold based on the limited labels we have available for p_2. This is the main contribution of this paper, OPUS is a framework that adapts this threshold, based on the new labelled data.

The remainder of this paper is structured as follows. In Section 2 we discuss existing solutions to the semi-deferred prediction problem. In Section 3 we formalize the prediction problem for loyalty campaign participation. In Section 4 we then discuss OPUS. We finally evaluate OPUS against alternative methods in Section 5 and conclude the paper in Section 6.

2. Related Work

For this problem there are roughly three existing categories of solutions. These are schematically presented in Figure 2. The first is retraining the classifier with an extended training set, where the new positive examples are added. This can either have limited effect if the number of positive labels is small or it can heavily bias the prediction model towards the positive class if the number of positive labels is high. The second category builds a model based on the positive and unlabelled state of the examples. The third is to ignore the positive labels, and assume all new data is unlabelled. All these techniques make a new prediction model or update the existing one. We discuss the second and third categories below, including their specific disadvantages in the semi-deferred labelling problem.

2.1. PU-learning

The original PU-learning solution was proposed in [5]. Here a classifier is build using positive and unlabelled examples, where the class of the unlabelled examples is not known. Semi-deferred labelling is less restrictive in two ways. The first is that we do have a, potentially outdated, set of labelled data. For the supermarkets, this is a previous loyalty campaign. The second is that we do not have a single set of positive examples but rather receive additional positive examples later on. Because Semi-deferred labelling is less restrictive, PU-learning can still be applied to our problem at different points in time, though it might not be as effective because it ignores part of the data.

A solution to PU-learning was originally proposed in [5]. Given a set of examples, of which only a fraction is labelled, and only from a single class (positive), the authors reason that the conditional probability that an example belongs to the positive class is equal to the conditional probability that the example is labelled, divided by the probability a positive sample is labelled. From this reasoning, one can construct a classifier on whether a sample is labelled and use that to predict the probability that positive samples are labelled (as average of the classifier outcome for the labelled examples). For the prediction of the unlabelled examples, the probability given by the classifier is divided by this latter value. An extension of this is discussed in [6], where some of its negative effects of misclassification are discussed. PU-learning usually assumes that the examples are labelled completely at random. In [7] this assumption is abandoned, where the probability of an example being labelled depends

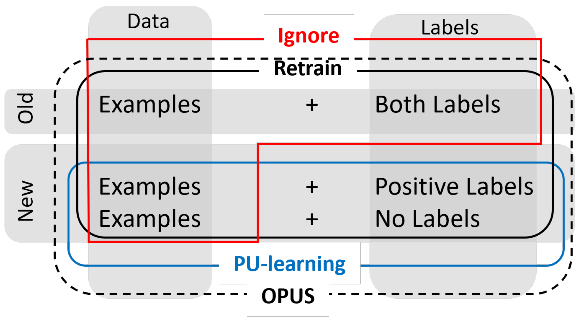


Figure 2: Various Semi-deferred labelling solutions and what data they use. 1) [Retrain] the model with the positive examples added to the existing dataset 2) [PU-learning] builds a model based on the positive and unlabelled state of the examples 3) [Ignore] the positive labels, only using the examples themselves. All three categories ignore part of the data, while OPUS uses all available data.

on its attributes. This is closer to our specific scenario, as consumers that are labelled as positive earlier in the loyalty campaign may have different characteristics than consumers labelled later in the loyalty campaign. In [8], the assumption that all labels are accurate is dropped: the authors propose a robust method of an ensemble of Support Vector Machines (SVMs) that can deal with the impurity of positive examples. For loyalty campaign participation prediction, this is not applicable, we have an exact definition of positive labels. In addition to these works, a more general overview of PU-learning is discussed in [9].

What all these techniques have in common is that existing predictions models are not used or changed. While this does not invalidate their use in semi-deferred labelling, their more restrictive assumption does not allow them to benefit from the existing data.

2.2. Ignoring new labels

A more restrictive assumption on the labels is that no labels are available, short of an initial training set. This would be comparable to ignoring newly converted consumers in the second loyalty campaign. Research dealing with such scenarios is often done in a streaming setting, either by predicting labels based on clustering techniques, or using semi-supervised learning methods. In [3], the authors propose a framework called SCARG. New examples are first classified by an existing classifier. After enough new examples are seen, new clusters are formed. The clusters are matched to existing clusters by their medoids and as such given a (new) label. Points in the clusters are then used to create a new prediction model. A similar approach using fuzzy clustering [10] is made in [11]. Instead of making predictions using an existing classifier, the COMPOSE framework discussed in [12] uses a semi-supervised approach. Assuming an initial set of labels, new unlabelled examples are classified in a semi-supervised manner. Next, only a selection of these examples are kept for the next timestep, at which new unlabelled examples are again classified using the semi-supervised learner. The main contribution of COMPOSE is the way in which the kept examples are selected: using α-shapes which are compacted until a desired number of examples remain.

The commonality in these approaches is that they focus on a *streaming* setting. At specific points in time, one or more existing models are updated using estimated labels from the respective solution. Although the authors show the effectiveness in dealing with streaming data by seeing many *new* examples at different timesteps, there are three key differences that make these methods unsuitable for our setting. First, we do not have a *constant* stream of *new* examples, instead we see new information about existing examples at discrete points in time. Second, we are only interested in improving a single model *once*, as we evaluate the existing model several weeks into the loyalty campaign. Third, we do not expect the drift to be gradual; we are starting a whole new loyalty campaign a year later, so our expected type of drift is more of the abrupt kind. If we were to apply SCARG or fuzzy clustering to predict loyalty campaign participation it would effectively be a more elaborate kNN classifier. COMPOSE is not applicable at all, since there is no way of knowing the true label of non-participating consumers until the end of the loyalty campaign.

Another solution to unavailability to unlabelled data is active learning, such as in [13]. Instead of having no labels, learners can query selected labels from domain experts. An example of dealing with such conditions in a multiclass setting is COCEL [2]. An ensemble of one-class classifiers is kept, each predicting only whether an example belongs to its respective class. If none of the classifiers recognizes the new example as one of their class, the example is added to a buffer. Clusters in this buffer are labelled by domain experts. Similar to COMPOSE, such solutions are not applicable in our scenario, as we cannot know the actual label until the end of the loyalty campaign.

3. Prediction Task

Before we explain how OPUS works, we first discuss the prediction task itself. All symbols used are summarized in Table 1. As introduced in Section 1, we want to solve the problem: "Given the data related to a consumer at time t during the loyalty campaign can we predict whether they will participate in the loyalty campaign?". This is a problem of *binary* classification: for each consumer we want to determine whether they are a member of one of two classes: the positive class (participant) or negative class (no participant). One way to do this is to use a *classifier* with a *prediction threshold.* A classifier is a function that assigns a probability to the encoding of a consumer. If the probability exceeds the prediction threshold, the consumer is predicted to belong to the positive class, otherwise they are predicted to belong to the negative class. While classifiers can usually be trained to make a prediction between more than two classes, we only consider binary classifiers in this paper which, without loss of generalization, predict the probability that a consumer is part of a positive class.

Let $M : \mathcal{S} \rightarrow [0, 1]$ be a function with $\mathcal{S}$ the set of all consumers. M is a *binary classifier* that estimates the probability that a consumer s belongs to the positive class. For a given *prediction threshold* $th \in [0, 1]$, consumer s is predicted to belong to the positive class if and only if $M(s) > th$. The binary classifier and the prediction threshold together assign a predicted binary class, this is the binary classification of the consumer. In this paper, the focus is on determining the threshold. However, before we do so, we explain what data is used to train the classifier for the prediction task. This is presented schematically in Figure 3.

Table 1
Symbols used for the prediction task

Symbol	Meaning
t	Time since start of loyalty campaign for which we train a model/make a prediction
p_i	Loyalty Campaign, where we make predictions during p_2 using a model trained on p_1
$\mathcal{S}_i$	Set of consumers that visited during p_i
$\mathcal{PS}_i$	Subset of $\mathcal{S}_i$ that redeemed during p_i
$\mathcal{NS}_i$	Subset of $\mathcal{S}_i$ that did not redeem during p_i
M_i^t	Binary classification model trained on p_i
th	Threshold to assign a label based on the probability predicted by a binary classification model
$\mathcal{CS}_i^t$	Subset of $\mathcal{S}_i$ that redeemed by time t in p_1
$\mathcal{US}_i^t$	Subset of $\mathcal{S}_i$ that not (yet) redeemed by time t
$oth_{i,j}^t$	Optimal threshold based on probabilities computed by M_i^t for shoppers in p_j
$metric_{i,j}^t$	Metric value computed using $oth_{i,j}^t$ and probabilities computed by M_i^t for shoppers in p_j

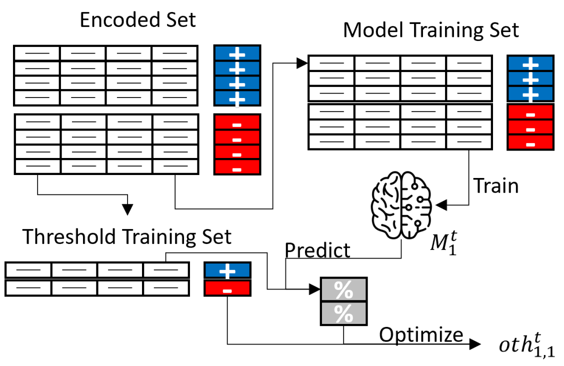


Figure 3: Overview of learning the model and optimal threshold in the first loyalty campaign. Since the first loyalty campaign has ended we know the labels and we can split the consumers in a positive set $\mathcal{PS}$ and a negative set $\mathcal{NS}$.

3.1. Training phase: the first loyalty campaign

We make a prediction at time t relative to the start of the loyalty campaign, which ends at t_e. We start by learning a classifier from a previous loyalty campaign. For p_1 we know which class each consumer that visited the store during p_1 belongs to, this is whether they redeemed a reward ($\mathcal{PS}_1$) or not ($\mathcal{NS}_1$). The 1 in the subscript indicates that these consumer sets belong to loyalty campaign p_1. We select a stratified sample of 75% from this set to train a model and later use the remaining 25% to optimize the threshold. Many different types of classifiers exist, but in general the training of a classifier involves finding a function that maximizes the predicted probability for consumers that belong to the positive class and minimizing it for consumers that belong to the negative class.

The classifier that is trained on data up to time t is referred to as M_1^t. The reason is that we want to use it at time t in p_2, so only data up to that time in p_1 should be used to train the classifier, to make the encoding of consumers as similar as possible. This is indicated by the superscript t. After training a classifier, we select a prediction threshold. If we set the threshold lower, more consumers will be predicted as participants, if we set the threshold higher, fewer consumers will be predicted as participants. For a given binary classification metric and a set of consumers, there is a threshold that can be considered the *best*, as it results in the best metric score. We call this the *optimal threshold* or $oth_{1,1}^t$. The consumers used to optimize the threshold come from the 25% that was not used in training. The double 1 subscript indicates that it is the optimal threshold for the model trained on p_1 and optimized using data from p_1. The value of the metric for this optimal threshold is denoted by $metric_{1,1}^t$.

Instead of training just one model and just one way of encoding, we can also train multiple different model types and different ways of encoding. We repeat the model training and threshold optimization for each combination of model type and encoding type. Each combination then results in a value for the metric and we select the combination with the highest value. This process is referred to as hyper-parameter optimization. We discuss the exact model types in Section 5, the different encoding types are beyond the scope of this paper. After selecting the best combination of model type and encoding type, we train a final model, which now includes *all* consumers from p_1, and keep the previously computed threshold of that combination. With this new model and the optimal threshold, we can make predictions in the following loyalty campaign, p_2.

Table 2
Symbols used by OPUS

Symbol	Meaning
$acp_{i,j}^t$	The average probability of $\mathcal{CS}_j^t$ predicted by M_i^t
$ancp_{i,j}^t$	The average probability of $\mathcal{US}_j^t$ predicted by M_i^t
$TPM_{i,j}$	Threshold prediction model trained on loyalty campaigns p_i and p_j

3.2. Inference phase: the second loyalty campaign

With the trained model, we can start making predictions *during* the second loyalty campaign. At point t in p_2 we only have data available up to point t, so that is what we use in the encoding of consumers. We can identify two sets of consumers. The first are those who have redeemed and have a positive label: $\mathcal{CS}_2^t$. The second are those who have not (yet) redeemed and may or may not still do so: $\mathcal{US}_2^t$. Both are indexed by t, as explained in Section 3 the sets change over time. Note that $\mathcal{CS}_2^t \subseteq \mathcal{PS}_2$, $\mathcal{NS}_2 \subseteq \mathcal{US}_2^t$ and $\mathcal{CS}_2^t \cup \mathcal{US}_2^t = \mathcal{S}_2$. Since we cannot distinguish between the latter two, we make predictions for them using M_1^t, and we assign the predicted labels using the learned $oth_{1,1}^t$. With the predicted and actual labels (which are available when p_2 ends), we can then compute the classification metric. This is visually presented in Figure 4 in the white area. As discussed previously, we are not able to update the prediction model M_1^t itself. However, we can change the threshold we use to assign the predicted class from the predicted probabilities which is the topic of Section 4.

4. OPUS framework

In this section, we sketch the idea behind OPUS and then formalize it. Table 2 summarizes the new symbols. An overview of how OPUS adds to the existing prediction problem is shown in Figure 4.

We start from $\mathcal{CS}_2^t$ and $\mathcal{US}_2^t$. As we do not know the labels of the latter, we cannot use these consumers to update M_1^t. We can however compute two characteristics about the current loyalty campaign p_2: the average probability computed by M_1^t to each of them. We refer to these as the *average converted probability* $acp_{1,2}^t$ and the *average nonconverted probability* $ancp_{1,2}^t$. The subscript 1 indicates that M_1^t is used for the model, the subscript 2 that $\mathcal{CS}_2^t$ and $\mathcal{US}_2^t$ are used for the consumers. We can do the same for p_1, using $\mathcal{CS}_1^t$ and $\mathcal{US}_1^t$ with M_1^t to compute $acp_{1,1}^t$ and $ancp_{1,1}^t$ respectively. In general, for loyalty campaigns p_i, p_j and time t we have $acp_{i,j}^t = \mu_{e \in \mathcal{CS}_j^t}[M_i^t(e)]$ and $ancp_{i,j}^t = \mu_{e \in \mathcal{US}_j^t}[M_i^t(e)]$. Note that these values do not necessarily say something about the model quality. On the one hand, we expect $acp_{1,2}^t$ to be high, since we *know* that it evaluates only positive consumers. On the other hand, we cannot make an expectation for $ancp_{1,2}^t$, as it evaluates negative and possibly also positive consumers. One thing

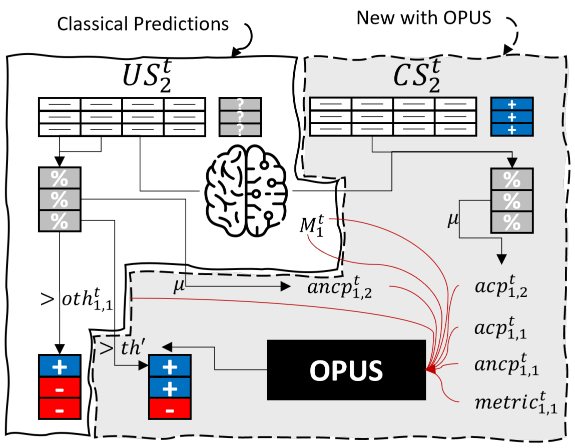


Figure 4: The original prediction task (white area with solid bounds) and how OPUS extends it (gray area with dashed bounds). The set of converted consumers $\mathcal{CS}$ is now used to tell something about the model.

Table 3
All threshold methods evaluated in this paper

Type	Name	Description
Base	Regular	No altered threshold (0.5)
	$oth_{i,i}^t$	Optimal threshold from training phase
Heuristic	$\varnothing acp$	$oth_{i,i}^t$ multiplied by the ratio of acp
	$\varnothing ancp$	$oth_{i,i}^t$ multiplied by the ratio of $ancp$
	Δacp	$oth_{i,i}^t$ increased by the difference of acp
	$\Delta ancp$	$oth_{i,i}^t$ increased by the difference of $ancp$
TPM	DT	$TPM_{i,j}$, with Decision Tree Regressor
	RF	$TPM_{i,j}$, with Random Forrest Regressor

to note is that for $acp_{1,2}^t$ we only evaluate consumers that have already converted by time t in p_2. It might be that for t early in the loyalty campaign we are including only a very specific subset of eager consumers. Such consumers may show different behaviour from consumers that converted later in the loyalty campaign. As such, $acp_{1,2}^t$ may not be an accurate representation of the average predicted probability of *all* positive consumers, $acp_{1,2}^{t_e}$. This is not a problem per se. First, the model is still trained on *all* consumers, it therefore also captures behaviour from the consumers that convert later. Second, we can compare $acp_{1,2}^t$ to $acp_{1,1}^t$. As such, both will be biased towards early redeemers if t is small. Therefore, we expect to compare the same type of consumers. We can therefore use these values to compare p_1 and p_2 and adapt the prediction threshold.

As we cannot change M_1^t, the best we can do is changing the prediction threshold. The best value is the one which is computed based on the labelled sets of consumers of the second loyalty campaign; $\mathcal{PS}_2$ and $\mathcal{NS}_2$. In line with the previous indexing, this value is $oth_{1,2}^t$. We can only know this value once p_2 ends, so during the loyalty campaign we need different ways of estimating it. Apart from the *Baseline* methods, we differentiate between *Heuristic* thresholds and *Learned* thresholds. All methods are presented in Table 3.

4.1. Baseline thresholds

For the baselines we consider values that at most use the training data, so we only require data from the first loyalty campaign. Without any alteration, a baseline of 0.5 is often used as standard. We further add the optimal threshold based on the training data, $oth_{1,1}^t$.

4.2. Heuristic thresholds

For the heuristic thresholds we consider values that also make use of the second loyalty campaign, and which can be used without further knowledge. For this, we consider four linear formulas. The first, $\varnothing acp$, multiplies the optimal thresholds from training, $oth_{1,1}^t$ with the ratio between the average converted probabilities, $acp_{1,2}^t/acp_{1,1}^t$. The idea behind this method is that if the average probability of the converted consumers has increased (or decreased), then the model likely overestimates (underestimates) the probability of a consumer being positive. Similarly, we can also use the average non-converter probabilities for this through multiplying $oth_{1,1}^t$ by $ancp_{1,2}^t/ancp_{1,2}^t$. We refer to this as the $\varnothing ancp$ method. Apart from the ratio, we can also take the difference, adding $acp_{1,2}^t - acp_{1,1}^t$ to $oth_{1,1}^t$, which we refer to as Δacp. The $\Delta ancp$ method is defined in a similar way. Note that in principle any of these four may result in a value above 1, and the difference based heuristic method may result in a value below 0. This does not invalidate the threshold, but it means that all entities will be predicted to be negative or positive, respectively.

4.3. Learned thresholds

For the learned thresholds we need full information about the second loyalty campaign, meaning we can only apply the learned thresholds to a third loyalty campaign. What the heuristic methods have in common is that they use one or more of the acp, $ancp$ and oth values to make an estimation about the target value ($oth_{1,2}^t$). For the final set of threshold estimation methods, we train a regression model, which we refer to as the *threshold prediction model*, or TPM. If needed, we refer to M_1^t as the consumer prediction model to distinguish between the two. TPM takes as descriptive space all known values at t: the (non-)converted probabilities $acp_{1,1}^t$, $acp_{1,2}^t$, $ancp_{1,1}^t$, $ancp_{1,2}^t$, the optimal training threshold $oth_{1,1}^t$, the training metric value $metric_{1,1}^t$, the model specification (type and encoding) and the point in time t. This is represented by the red lines Figure 4. As target space, we have the value of $oth_{1,2}^t$. We can sample these values for different values of t and for each of the different model specifications. Using two loyalty campaigns, we create a training set for TPM, with the target values coming from the second loyalty campaign and the descriptive features from both loyalty campaigns. We denote this TPM as $TPM_{1,2}$, without the superscript t as multiple values of t are used to train $TPM_{1,2}$. Note that we cannot use $TPM_{1,2}$ to estimate a threshold to use during p_2, as p_2 needs to be finished before $TPM_{1,2}$ can be learned. OPUS does not depend on the specific regressor used as a base model for TPM, in Section 5 we evaluate Decision Tree and Random Forest regressors.

The idea behind the learned thresholds is that $TPM_{1,2}$ learns how to adapt the prediction method (model and threshold) from one loyalty campaign to another. This means we could learn how the prediction method is best 'transformed' from p_1 to p_2, and then apply this learned transformation. Suppose that p_1 rewards glassware, p_2 rewards pans, and a third p_3 rewards kitchen knives. If we learn a transformation from glassware to pans, then we can assess 1) how well does it work on transforming the prediction method for a different set of consumers from glassware to pans and 2) how well does that transformation apply to the transformation from pans to kitchen knives. In the evaluation in Section 5 we limit the evaluation to the first due to the unavailability of additional data on a third loyalty campaign.

4.4. Existing Threshold Adaption techniques

OPUS is not the first framework that optimizes thresholds. In GHOST [14], the optimal threshold of any classifier is optimized on a validation set. The metric value for several thresholds is computed and averaged over several stratified subsets of a validation set. While this technique may prove effective for finding an optimal threshold for the original training and validation data, there is no update in a later stage, and such an update would require labelled data. As such, GHOST is not suitable for the problem at hand. Similarly to GHOST, [15] proposes a method to find the optimal threshold in a training set. The authors prove that for such a method only the probabilities of positive class are required when optimizing for F_1. While this technique potentially saves computation time, it is not universally applicable to all metrics. In this paper we optimize the thresholds by evaluating all predicted probabilities, and selecting the one that optimizes the target metric. Finally, [16] proposes a technique to adapt the separating hyperplane position of SVMs to improve its predictive quality. OPUS is different from these because it tries to optimize the threshold using the new, partly labelled data in an online fashion. OPUS is designed to improve the predictions under concept drift.

5. Experimental Evaluation

For the experimental evaluation[1] we use data from a real-world retailer containing a total of about 160 000 consumers shopping in exactly one of two consecutive loyalty campaigns, p_1 and p_2. We partition the consumers in ten separate datasets c_1 through c_{10}. For each dataset we build several prediction models based on a grid-search over hyper-parameters and at different timesteps relative to the start of the loyalty campaign.

Models are trained and tested at multiple points in time relative to the loyalty campaign, after $2, 4, 6, \ldots, 18$ weeks into the campaign, ending *before* t_e. The encoding of consumers is based on the data available from the pre loyalty campaign period and the data during the loyalty campaign up to t. We consider three hyper-parameters: the base model, the encoding technique, and a strategy. The base models are the consumer prediction models, for which we use k-nearest neighbours (kNN), decision tree classifiers (DTs), support vector classifiers (SVCs) and Adaptive Hoeffding Option Trees (AHOTs), all with their default parameters in Scikit-Learn for kNN, DTs, SVCs, and Scikit-Multiflow [17] for AHOTs, except for setting a constant seeding and training the SVCs on probabilities instead of labels. The encoding techniques define what type of features we use about a consumer. The features are based on individual visits to the store, using aggregates based on [18] and descriptive labels based on [19] We finally adopt two strategies: with and without adding the converted consumers in the test set to the training set. In either strategy, the prediction quality metrics are only computed on the non-converted consumers in the test set. For each timestep we select the hyper-parameter combination with the best performance during the training phase using $oth^t_{1,1}$ to report the results for all threshold estimation methods.

For each combination of hyper-parameters we compute $acp^t_{1,i}$, $ancp^t_{1,i}$, and $oth^t_{1,i}$ for $i \in \{1, 2\}$ and each timestep t. On the one hand we can use this to compute all threshold estimation methods discussed in Table 3. On the other hand we can use all combinations to train threshold prediction models. For each dataset c_x we compute two TPMs, one using a Decision Tree regressor, DT_x, and one using a Random Forest regressor RF_x. These $TPMs$ are then used in every other dataset c_y, $y \neq x$, to estimate optimal thresholds. We also add two results for comparison. The first uses the actual optimal threshold ($oth^t_{1,2}$) and the second uses a prediction model learned on 75% of the data of loyalty campaign p_2 and tested on the remaining 25% (Retrain). Both these methods violate the train-then-test principle but are added as a comparison to how close we can get.

During the training phase, models are trained on a stratified 75% split of the consumers that visited the store during loyalty campaign p_1. The remaining 25% are used as test set. During the inference phase, models are trained on all consumers that visited the store during loyalty campaign p_2. The consumers that visited the store by time t are used as test set. The entire test set is used to compute $oth^t_{1,i}$ using the model trained on the training set. The converted (non-converted) consumers by time t in the test set are used to compute $acp^t_{1,i}$ ($ancp^t_{1,i}$). The non-converted consumers by time t are used to compute the model performance for a given threshold estimation method.

Next to these methods, we also apply PU-learning defined by [5]. For this we use the same hyper-parameters as for the other experiments. We first find the best combination based on loyalty campaign p_1 and then report the score for that hyper-parameter combination in p_2, computed over all non-converted consumers at the respective point in time.

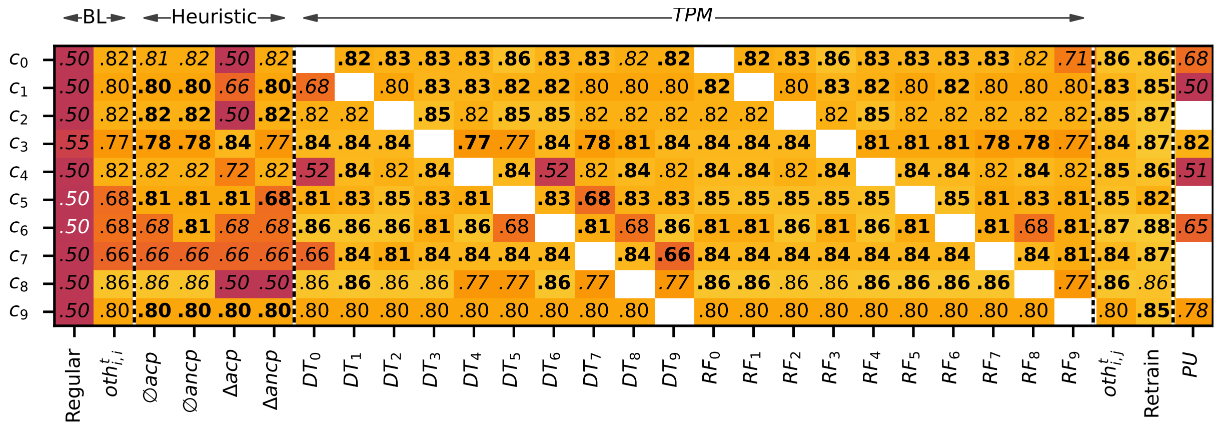
We apply the above for both the F_1 and Accuracy metrics. We present two types of results: those for timestep $t = 4$, separated per dataset, and those for all timesteps, aggregated over the datasets. The reason for separately reporting $t = 4$ comes from domain knowledge, this is the point in the loyalty campaign where additional rewards for the loyalty campaign can still be ordered by the supermarket.

5.1. Results for $t = 4$

Results are presented per split (one per row) and threshold estimation method (one per column) in Figure 5. We do not report the values of the combinations where a TPM is learned from the same split as the data it is tested on (the white diagonals).

Baseline methods For the baseline methods we see that using the regular value is not much better than always predicting True or False, almost all scores being 0.50, which is effectively the worst value one can get for with balanced accuracy in a binary classification. Using the optimal training threshold $oth^t_{i,i}$ works much better, with the exception of c_5, c_6 and c_7.

[1]For the implementation, see github.com/YorickSpenrath/opus



	Regular	$oth^t_{i,i}$	$\varnothing acp$	$\varnothing ancp$	Δacp	$\Delta ancp$	DT_0	DT_1	DT_2	DT_3	DT_4	DT_5	DT_6	DT_7	DT_8	DT_9	RF_0	RF_1	RF_2	RF_3	RF_4	RF_5	RF_6	RF_7	RF_8	RF_9	$oth^t_{i,j}$	Retrain	PU
c_0	*.50*	.82	*.81*	*.82*	*.50*	*.82*		**.82**	**.83**	**.83**	**.83**	**.86**	**.83**	**.83**	*.82*	**.82**		**.82**	**.83**	**.86**	**.83**	**.83**	**.83**	**.83**	*.82*	*.71*	**.86**	**.86**	*.68*
c_1	*.50*	.80	**.80**	**.80**	*.66*	**.80**	*.68*		.80	**.83**	**.83**	**.82**	**.82**	.80	.80	.80	**.82**		.80	**.83**	**.82**	.80	**.82**	.80	.80	.80	**.83**	**.85**	*.50*
c_2	*.50*	.82	**.82**	**.82**	*.50*	**.82**	.82	.82		**.85**	.82	**.85**	**.85**	.82	.82	.82	.82	.82		.82	**.85**	.82	.82	.82	.82	.82	**.85**	**.87**	
c_3	*.55*	.77	**.78**	**.78**	**.84**	*.77*	**.84**	**.84**	**.84**		**.77**	*.77*	**.84**	**.78**	**.81**	**.84**	**.84**	**.84**	**.84**		**.81**	**.81**	**.81**	**.78**	**.78**	*.77*	**.84**	**.87**	**.82**
c_4	*.50*	.82	*.82*	*.82*	*.72*	*.82*	*.52*	**.84**	.82	**.84**		**.84**	*.52*	.82	**.84**	.82	**.84**	**.84**	.82	**.84**		**.84**	**.84**	.82	**.84**	.82	**.85**	**.86**	*.51*
c_5	*.50*	.68	**.81**	**.81**	**.81**	**.68**	**.81**	**.83**	**.85**	**.83**	**.81**		**.83**	**.68**	**.83**	**.83**	**.85**	**.85**	**.85**	**.85**	**.85**		**.85**	**.81**	**.83**	**.81**	**.85**	**.82**	
c_6	*.50*	.68	*.68*	**.81**	*.68*	*.68*	**.86**	**.86**	**.86**	**.81**	**.86**	.68		**.81**	.68	**.86**	**.81**	**.81**	**.86**	**.81**	**.86**	**.81**		**.81**	.68	**.81**	**.87**	**.88**	*.65*
c_7	*.50*	.66	*.66*	*.66*	*.66*	*.66*	.66	**.84**	**.81**	**.84**	**.84**	**.84**	**.84**		**.84**	**.66**	**.84**	**.84**	**.84**	**.84**	**.84**	**.84**	**.84**		**.84**	**.81**	**.84**	**.87**	
c_8	*.50*	.86	*.86*	*.86*	*.50*	*.50*	.86	**.86**	.86	.86	*.77*	*.77*	**.86**	*.77*		*.77*	**.86**	**.86**	.86	.86	**.86**	**.86**	**.86**	**.86**		*.77*	**.86**	*.86*	
c_9	*.50*	.80	**.80**	**.80**	**.80**	**.80**	.80	.80	.80	.80	.80	.80	.80	.80	.80		.80	.80	.80	.80	.80	.80	.80	.80	.80		.80	**.85**	*.78*

Figure 5: Accuracy at timestep 4. Values shown in **bold** are strictly better than the $oth^t_{i,i}$, the ones in *italic* are strictly smaller. Higher values correspond to a brighter colour. The white diagonals are skipped since those the TPM would be trained on the same split we test it on. Some scores for PU-learning are left out; for these splits there were not enough converted consumers to train a meaningful prediction model.

Heuristic methods For the heuristic methods we have that, except for (c_8, $\Delta ancp$), the $ancp$ methods consistently performs as well or better than the baseline $oth^t_{i,i}$. The acp values do much worse however, especially for the Δacp. As explained in Section 4.2, if the difference between $acp^t_{i,i}$ and $acp^t_{i,j}$ is too high, this method might end up predicting all shoppers as redeemers or all shoppers as non-redeemers. This is what happens for splits 0, 2 and 8.

Learned methods With only a few exceptions, the learned methods always perform at least as good as the baseline and often beat it. Furthermore, for most splits the best values even come close to the results for $oth_{i,j}$, the value which we aim to estimate. Next to this, we see that the Random Forest TPM performs almost always as well as or better than the Decision Tree based TPM, compared on the same combination of splits. This is to be expected given the complexity of the Random Forest model and that it can as such capture more difficult relations.

PU-Learning One weakness of PU-learning is if the number of known positive labels is small. This results in a base prediction model that cannot predict positive points, resulting in an average probability of 0 for the known positive labels. In such a scenario, PU-learning fails to produce any result. This is what happens in 4 of the splits. Note that OPUS does not have this problem, as we are using the labelled data from last loyalty campaign. In the splits where there are enough known positive labels, it is still outperformed by OPUS, for a similar reason.

5.2. Results aggregated per timestep

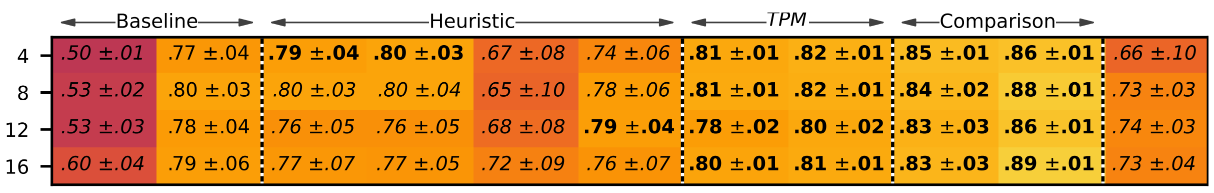
We next aggregate the results over the splits for each timestep in Figures 6a and 6b. We report the mean and 95% confidence interval of the metric for every timestep (each row). For most estimation methods (each column) this is the average over the 10 values from different splits, for the learned estimation methods we consider all combinations. In other words, the reported aggregated values for DT and RF are taken over all 90 combinations.

Accuracy Some results from Section 5.1 can be transferred to more timesteps. The learned thresholds seem to perform better than using $oth^t_{i,i}$. Note that, as we have averaged over both the splits used for the evaluation as well as the splits used for computing the TPM, this result describes the benefit of using the learned method in OPUS in general. We see what one can on average expect using *any* other dataset to train the TPM to estimate the optimal threshold is beneficial. What is more, mainly for the Random Forest TPM, we see that we get values that are close to using the actual optimal threshold $oth^t_{i,j}$. For the heuristic thresholds, we see that the ratio-based ones ($\varnothing$) are performing close to or better than the $oth^t_{i,i}$ baseline. For PU-learning we see that the results improve over time as more positive labels are known. This is to be expected, though it does not yet catch with OPUS methods.

$\mathbf{F_1}$ We see something similar to the accuracy results. This is expected, as we are penalizing in a similar way by comparing the actual with the predicted labels for each shopper. There are two important differences. The first is that we see lower values. This is because F_1 does not account for class imbalance as we did for the balanced accuracy score. Given that the redeemer class is much smaller than the non-redeemer class, the reported values for $oth^t_{i,j}$ and Retrain are reasonable. This is also seen by the non $oth^t_{i,i}$ baselines, which have a much lower score. The second is that we are further off from the optimal thresholds $oth^t_{i,j}$. This is likely because of a similar reason: the F_1 is much more penalizing for imbalanced datasets, and as such, being able to use the optimal thresholds/all labels allows for a much better score than estimating it using any of the methods from OPUS. Like accuracy, the results of PU-learning improve over time, eventually even improving over OPUS methods. The lower accuracy and higher F_1 for PU-learning is likely caused by the unbalanced dataset as well.

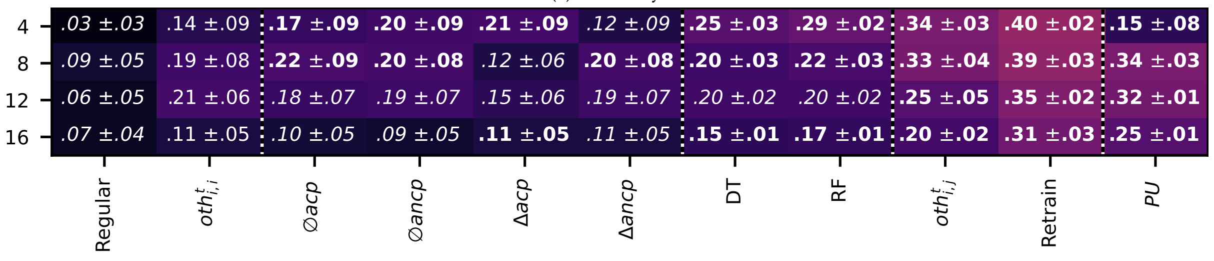
6. Conclusion

In this paper we have presented OPUS, a framework to estimate a better prediction threshold when only one class of the target labels is available. The advantage of such optimizations is that we can adapt existing prediction models without having to rely on fully labelled data. This is a more realistic assumption when the true label of one class is deferred. OPUS makes use of two types of threshold estimation methods, where learned methods are more effective in getting to an optimal prediction threshold at the cost of requiring more data (only being available at a third cam-



	Baseline		Heuristic				*TPM*		Comparison		
	Regular	$oth^t_{i,i}$	$\varnothing acp$	$\varnothing ancp$	Δacp	$\Delta ancp$	DT	RF	$oth^t_{i,j}$	Retrain	*PU*
4	*.50 ±.01*	.77 ±.04	**.79 ±.04**	**.80 ±.03**	*.67 ±.08*	*.74 ±.06*	**.81 ±.01**	**.82 ±.01**	**.85 ±.01**	**.86 ±.01**	*.66 ±.10*
8	*.53 ±.02*	.80 ±.03	*.80 ±.03*	*.80 ±.04*	*.65 ±.10*	*.78 ±.06*	**.81 ±.01**	**.82 ±.01**	**.84 ±.02**	**.88 ±.01**	*.73 ±.03*
12	*.53 ±.03*	.78 ±.04	*.76 ±.05*	*.76 ±.05*	*.68 ±.08*	**.79 ±.04**	**.78 ±.02**	**.80 ±.02**	**.83 ±.03**	**.86 ±.01**	*.74 ±.03*
16	*.60 ±.04*	.79 ±.06	*.77 ±.07*	*.77 ±.05*	*.72 ±.09*	*.76 ±.07*	**.80 ±.01**	**.81 ±.01**	**.83 ±.03**	**.89 ±.01**	*.73 ±.04*

(a) Accuracy



	Regular	$oth^t_{i,i}$	$\varnothing acp$	$\varnothing ancp$	Δacp	$\Delta ancp$	DT	RF	$oth^t_{i,j}$	Retrain	*PU*
4	*.03 ±.03*	.14 ±.09	**.17 ±.09**	**.20 ±.09**	**.21 ±.09**	*.12 ±.09*	**.25 ±.03**	**.29 ±.02**	**.34 ±.03**	**.40 ±.02**	**.15 ±.08**
8	*.09 ±.05*	.19 ±.08	**.22 ±.09**	**.20 ±.08**	*.12 ±.06*	**.20 ±.08**	**.20 ±.03**	**.22 ±.03**	**.33 ±.04**	**.39 ±.03**	**.34 ±.03**
12	*.06 ±.05*	.21 ±.06	*.18 ±.07*	*.19 ±.07*	*.15 ±.06*	*.19 ±.07*	*.20 ±.02*	*.20 ±.02*	**.25 ±.05**	**.35 ±.02**	**.32 ±.01**
16	*.07 ±.04*	.11 ±.05	*.10 ±.05*	*.09 ±.05*	**.11 ±.05**	*.11 ±.05*	**.15 ±.01**	**.17 ±.01**	**.20 ±.02**	**.31 ±.03**	**.25 ±.01**

(b) F_1

Figure 6: a) Accuracy and b) F_1 at each timestep, averaged over the splits with their 95% confidence interval. Values shown in **bold** are strictly better than the $oth^t_{i,i}$, the ones in *italic* are strictly worse (in terms of the reported mean). Better values correspond to a brighter colour.

paign), compared to heuristic methods that do not have these benefits or limitations. While the initial results are promising, we want to elaborate the experimental analysis, both by including more than two loyalty campaigns and by applying OPUS on a different use case inside the student learning domain. In the design of OPUS only characteristics that belong to the *current* timestep are considered, not of all previous timesteps. We argue that the progression of these characteristics, as well as the model performance over time, and how they compare to the same period in the previous loyalty campaign can be interesting to improve the framework in future work.

References

[1] W. Rizzi, C. Di Francescomarino, C. Ghidini, F. M. Maggi, How do I update my model? On the resilience of Predictive Process Monitoring models to change, KAIS 64 (2022) 1385–1416.

[2] C. Fahy, S. Yang, M. Gongora, Classification in Dynamic Data Streams with a Scarcity of Labels, IEEE TKDE (2021).

[3] V. M. A. Souza, D. F. Silva, J. Gama, G. E. A. P. A. Batista, Data Stream Classification Guided by Clustering on Nonstationary Environments and Extreme Verification Latency, in: SDM, 2015, pp. 873–881.

[4] N. J. Bombaij, S. Gelper, M. G. Dekimpe, Designing successful temporary loyalty programs: An exploratory study on retailer and country differences, International Journal of Research in Marketing 39 (2022) 1275–1295.

[5] C. Elkan, K. Noto, Learning classifiers from only positive and unlabeled data, KDD (2008) 213–220.

[6] J. Peeperkorn, C. O. Vázquez, A. Stevens, J. D. Smedt, S. Broucke, J. D. Weerdt, Outcome-Oriented Predictive Process Monitoring on Positive and Unlabelled Event Logs, ICPM workshops (2022).

[7] J. Bekker, P. Robberechts, J. Davis, Beyond the Selected Completely at Random Assumption for Learning from Positive and Unlabeled Data, LNCS 11907 LNAI (2020) 71–85.

[8] M. Claesen, F. De Smet, J. A. Suykens, B. De Moor, A robust ensemble approach to learn from positive and unlabeled data using SVM base models, Neurocomputing 160 (2015) 73–84.

[9] J. Bekker, J. Davis, Learning from positive and unlabeled data: a survey, Machine Learning 109 (2020) 719–760.

[10] P. De Abreu Lopes, H. De Arruda Camargo, FuzzStream: Fuzzy data stream clustering based on the online-offline framework, IEEE International Conference on Fuzzy Systems (2017).

[11] T. P. da Silva, V. M. A. Souza, G. E. A. P. A. Batista, H. de Arruda Camargo, A fuzzy classifier for data streams with infinitely delayed labels, LNCS 11401 (2019) 287–295.

[12] K. B. Dyer, R. Capo, R. Polikar, Compose: A semisupervised learning framework for initially labeled nonstationary streaming data, IEEE Transactions on Neural Networks and Learning Systems 25 (2014) 12–26.

[13] B. Settles, Active Learning Literature Survey, Technical Report, University of Wisconsin-Madison, 2009.

[14] C. Esposito, G. A. Landrum, N. Schneider, N. Stiefl, S. Riniker, GHOST: Adjusting the Decision Threshold to Handle Imbalanced Data in Machine Learning, Journal of Chemical Information and Modeling 61 (2021) 2623–2640.

[15] Q. Zou, S. Xie, Z. Lin, M. Wu, Y. Ju, Finding the Best Classification Threshold in Imbalanced Classification, Big Data Research 5 (2016) 2–8.

[16] H. Yu, C. Mu, C. Sun, W. Yang, X. Yang, X. Zuo, Support vector machine-based optimized decision threshold adjustment strategy for classifying imbalanced data, Knowledge-Based Systems 76 (2015) 67–78.

[17] J. Montiel, J. Read, A. Bifet, T. Abdessalem, Scikit-Multiflow: A Multi-output Streaming Framework, JMLR 19 (2018) 1–5.

[18] Y. Spenrath, M. Hassani, B. F. van Dongen, Online prediction of aggregated retailer consumer behaviour, ICPM Workshops (2021).

[19] Y. Spenrath, M. Hassani, B. v. Dongen, H. Tariq, Learning an efficient distance metric for retailer transaction data, in: PKDD, 2020, pp. 323–338.