Outcome-Oriented Predictive Process Monitoring on Positive and Unlabelled Event Logs

Jari Peeperkorn[1], Carlos Ortega Vázquez[1], Alexander Stevens[1], Johannes De Smedt[1], Seppe vanden Broucke[2,1], and Jochen De Weerdt[1]

[1] Research
Center for Information Systems Engineering (LIRIS), KU Leuven, Leuven, Belgium
[2] Department
of Business Informatics and Operations Management, Ghent University, Ghent, Belgium
{jari.peeperkorn, carloseduardo.ortegavazquez, alexander.stevens, johannes.desmedt, seppe.vandenbroucke, jochen.deweerdt}@kuleuven.be

Abstract. A lot of recent literature on outcome-oriented predictive process monitoring focuses on using models from machine and deep learning. In this literature, it is assumed the outcome labels of the historical cases are all known. However, in some cases, the labelling of cases is incomplete or inaccurate. For instance, you might only observe negative customer feedback, fraudulent cases might remain unnoticed. These cases are typically present in the so-called positive and unlabelled (PU) setting, where your data set consists of a couple of positively labelled examples and examples which do not have a positive label, but might still be examples of a positive outcome. In this work, we show, using a selection of event logs from the literature, the negative impact of mislabelling cases as negative, more specifically when using XGBoost and LSTM neural networks. Furthermore, we show promising results on real-life datasets mitigating this effect, by changing the loss function used by a set of models during training to those of unbiased Positive-Unlabelled (uPU) or non-negative Positive-Unlabelled (nnPU) learning.

Keywords: Process Mining · Predictive Process Monitoring · OOPPM · XGBoost · LSTM · PU learning · Label Uncertainty

1 Introduction

Outcome-Oriented Predictive Process Monitoring (OOPPM) refers to predicting the future state (labels) of ongoing processes, using the historical cases of business processes. Most recently, the literature in OOPPM has been focused on training machine and deep learning models on labelled historical data. However, to the best of our knowledge, no research has focused on training such models when the labels given to these historical cases are incomplete, uncertain, or even wrong. Accordingly, in this paper, a situation is investigated where part of the positive historical cases have been unnoticed and therefore mistakenly classified as negative in the data. A situation like this might be found when the positive label represents, e.g., detected outliers or fraud in a loan application, or customer (dis)satisfaction through a survey (which

might not always be filled out) for a production process [10,26]. Other examples can be found in medicine, e.g., when trying to predict the chances of complications which might go unnoticed, or when dealing with overall low-quality data and therefore uncertain labels [14,10]. Other examples can be found in multi-organisational processes, where information flow can sometimes be limited. This setting can be understood as one-sided label noise, in which some seeming negatives are actually positive [1]. One-sided label noise is a common interpretation of positive and unlabelled (PU) data. In this setting, data consists of positive and unlabelled instances, in which an unlabelled example can be either positive or negative. Consequently, this field of machine learning research is called PU learning. We focus on methods based on the Expected Risk Minimization (ERM) in the literature of PU learning. Particularly, we use unbiased PU learning and non-negative PU learning [7,11] because of the state-of-the-art performance. These two methods have been successfully utilised in other domains such as imbalanced learning [21], and graph neural networks [27].

In our experimental setup, we flip different percentages of the positive labels in the training logs to a negative label, replicating a real-world situation with missing positive labels. By training models on these different training sets and evaluating their performance on the untouched test set, we can evaluate the impact of missing positive training labels on the models' actual performance. The models in question are gradient boosted trees (more specifically eXtreme Gradient Boosting, known as XGBoost or XGB) [3] and the Long Short-Term Memory neural networks (LSTM) [9]. Furthermore, we investigate the impact of replacing the binary cross-entropy loss functions with functions inspired by the Positive and Unlabelled (PU) learning literature. This can be summarised in the following hypotheses:

Hypothesis 1 (H1): *Incorrectly labelling deviant (positive) behaviour as normal (negative), can have an important impact on the (future) performance of a predictive model.*

Hypothesis 2 (H2): *Using loss functions from PU-learning, the problem above can be (partially) mitigated.*

To investigate these hypotheses, our setup has been applied to a selection of nine real-life process event logs from the literature. The rest of the paper is organised as follows. We start by discussing some relevant related work in Section 2. Second, Section 3 introduces essential background information, followed by an introduction to PU learning in section 4. In Section 5, the experimental setup is described, before introducing the data and the hyperparameter search. This is succeeded by showing and discussing the results (Section 6). Finally, Section 7 provides a conclusion and some possible approaches for future research. The data, results, and code used and presented in this paper are available online[1].

2 Related Work

Predictive Process Monitoring (PPM) is concerned with many tasks such as predicting the remaining time [25], next activity [22] or the outcome of the process [24,12,4]. The

[1] https://github.com/jaripeeperkorn/PU-OOPPM

latter is also known as Outcome-Oriented Predictive Process Monitoring (OOPPM), a field of study that predicts the final state of an incoming, incomplete case. One of the pioneer studies in this field is [24], where they benchmark the state-of-the-art trace bucketing techniques and sequence encoding mechanisms used for different machine learning models. However, the use of classical machine learning models has been superseded by an avalanche of deep learning techniques. Here, the most frequent used predictive model in general predictive process monitoring literature has become LSTM neural networks, initiated by [22]. The introduction of this recurrent neural network has been motivated by the ability of this model to handle the dynamic behaviour of high-dimensional sequential data. In recent, many other sophisticated models have been benchmarked against this model in the field of predictive process monitoring, such as Convolutional Neural Networks (CNN) [16] or Generative Adversarial Networks (GAN) [23]. Some studies have already compared the predictive performance of different deep learning models [18,12,15].

Nonetheless, the predictions made by the majority of these works are based on data from past process instances, i.e. event logs, and therefore implicitly assume that the labelling made corresponds with the ground truth. Work on incremental predictive process monitoring [19,17] does provide a flexible alternative to deal with the rigidity of predictive models. Moreover, these incremental learning algorithms allow for the predictive model to deal with the variability and dynamic behaviour of business processes (i.e. different time periods have different characteristics [17]). Other recent work discusses a semi-supervised approach, also leveraging the power of deep neural networks, to handle scarcely labelled process logs in an OOPPM setting [8]. However, to the best of our knowledge, none of the related works has already used PU learning in the context of OOPPM, which incorporates that negatively labelled instances are possibly mislabelled.

3 Preliminaries

Executed *activities* in a process are recorded as an event in an Event Log L. Each *event* belongs to one *case*, indicated by its *CaseID* $c \in C$. An event e can also be written as a tuple $e=(c,a,t,d,s)$, with $a \in A$ the *activity* (i.e. *control-flow* attribute) and t the timestamp. Optionally, an event might also have event-related attributes (payload or *dynamic* attributes) $d = (d_1,d_2,...,d_{m_d})$, which are event specific and might evolve during a case. Other attributes do not evolve during the execution of a single case and are called case or *static* attributes $s=(s_1,s_2,...,s_{m_s})$. A sequence of events belonging to one case is called a trace. The outcome y of a trace is an attribute defined by the process owner. This attribute is often binary, indicating whether a certain criterion has been met [24]. We use the label *positive* when it is met, and call these cases *positive cases*. And *negative cases* otherwise. A prefix is part of a trace, consisting of the first l events (with l an integer smaller than the trace length). A prefix log L^* contains all possible prefixes which can be extracted from all traces in L.

XGBoost is an implementation of the gradient boosting ensemble method, which is constructed from multiple decision tree models. By adding additional trees to correct the prediction error from the prior iteration, an efficient yet powerful classifier

can be trained [3]. Recurrent Neural Networks (RNNs) are neural networks specifically designed to work with sequential data by letting information flow between multiple time steps. LSTMs are a specific variant of RNN, specifically designed to handle long-term dependencies [9].

Both of these models rely on a proper choice of loss function for training. The loss function is used to score the model on its performance, based on the predicted probability $p \in [0,1]$ and the actual label $y \in \{0,1\}$. Formally, in the Empirical Risk Minimisation (ERM) framework, the loss function L is utilised within the risk function when scoring a classifier $g(x)$:

$$R(g(x)) = \alpha \mathbb{E}_{f_+}[L^+(g(x))] + (1-\alpha)\mathbb{E}_{f_-}[L^-(g(x))], \tag{1}$$

where $L^+(g)$ and $L^-(g)$ are the losses for positive and negative examples; $\mathbb{E}_{f_+}$ and $\mathbb{E}_{f_-}$ are the expectation over the propensity density functions of the positive $f_+(x)$ and negative $f_-(x)$ instance space; and α is the positive class ratio or class prior as denoted in the literature. Usually for a binary classification problem, as often the case in OOPPM, the binary cross entropy loss function is used. The binary cross-entropy can be calculated from a data set when $L^+(g(x_i)) = -log(p_i)$ and $L^-(g(x_i)) = -log(1-p_i)$ in Equation 1:

$$BCE = -\sum_{i=1}^{N} (y_i \log(p_i) + (1-y_i)\log(1-p_i)) \tag{2}$$

4 PU Learning

Despite the popularity of binary cross-entropy in the standard classification setup in which labels are accurate and complete, some real-world applications suffer from label uncertainty. In such scenarios, the binary cross-entropy is no longer valid for model learning. We focus on the PU setting in which the training data consists of only positive and unlabelled examples; the labelled instances are always positive, but some positives remain unlabelled. In PU learning, the label status $l \in \{0,1\}$ determines if an example is either labelled or unlabeled. Formally, we assume that the positive and unlabelled instances are independent and identically distributed from the general distribution $f(x)$:

$$\begin{aligned} \mathcal{X} &\sim f(x) \\ &\sim \alpha f_+(x) + (1-\alpha) f_-(x) \qquad (3) \\ &\sim \alpha c f_l(x) + (1-\alpha c) f_u(x), \qquad (4) \end{aligned}$$

where $\mathcal{X}$ refers to the set of instances and the label frequency c is the probability of a positive example being labelled $P(l=1 \mid y=1)$. The general distribution can be formulated in terms of the positive distribution $f_+(x)$ and negative distribution $f_-(x)$ (see Equation 3). In the PU setting, the general distribution consists of the labeled $f_l(x)$ and unlabeled distribution $f_u(x)$ as shown in Equation 4. A proportion c of the positive instances of the data set is labelled, thus, a learner can only observe a fraction αc of instances with a positive label whereas the rest is unlabeled. Recent works have proposed methods based on the ERM framework, which are currently considered state-of-the-art [7,11,2]. These methods incorporate the information of the class prior (i.e., positive

class ratio) to weight the PU data within the loss function. The weighting allows the empirical risk from the PU data to be the same in expectation as in the fully labelled data. From Equation 1, we can transform the loss function into the PU setting as follows:

$$\begin{aligned} R_{upu}(g(x)) &= \alpha\mathbb{E}_{f_+}[L^+(g(x))] + (1-\alpha)\mathbb{E}_{f_-}[L^-(g(x))] \\ &= \alpha\mathbb{E}_{f_+}[L^+(g(x))] + \mathbb{E}_f[L^-(g(x))] - \alpha\mathbb{E}_{f_+}[L^-(g(x))] \\ &= \alpha c\mathbb{E}_{f_l}\left[\frac{1}{c}(L^+(g(x)) - L^-(g(x)))\right] + \mathbb{E}_f[L^-(g(x))] \\ &= \alpha c\mathbb{E}_{f_l}\left[\frac{1}{c}L^+(g(x)) + (1-\frac{1}{c})L^-(g(x))\right] + (1-\alpha c)\mathbb{E}_{f_u}[L^-(g(x))]. \end{aligned} \quad (5)$$

In the first step of Equation 5, we can substitute the term $(1-\alpha)\mathbb{E}_{f_-}[L^-(g(x))]$ with $\mathbb{E}_f[L^-(g(x))] - \alpha\mathbb{E}_{f_+}[L^-(g(x))]$ based on Equation 3: the negative distribution f_- is the difference between the general distribution f and the positive distribution f_+. In the second step, we substitute $\mathbb{E}_f[L^-(g(x))]$ with $\alpha c\mathbb{E}_f[L^-(g(x))] + (1 - \alpha c)\mathbb{E}_f[L^-(g(x))]$ based on Equation 4. Now the unlabelled instances are considered negative with a weight of 1. Also, all labelled examples are added both as positive with weight $\frac{1}{c}$ and as negative with $1-\frac{1}{c}$. The method is called unbiased PU (uPU) because the empirical risk for PU data (Equation 5) is equal in expectation to the empirical risk when data is fully labelled (Equation 1) [7]. The uPU can be used in modern techniques that require a convex loss function for training. However, the uPU method presents a weakness for flexible techniques that can easily overfit: the uPU risk estimator can provide negative empirical risks. This issue is problematic for powerful classifiers such as XGBoost [3] or deep learning models. Thus, the non-negative PU risk estimator is proposed that improves on uPU by adding a maximum operator [11]:

$$R_{nnpu}(g(x)) = \alpha c\mathbb{E}_{f_l}\left[\tfrac{1}{c}L^+(g(x))\right] + \max\left(0, (1-\alpha c)\mathbb{E}_{f_u}[L^-(g(x))] + \alpha c\mathbb{E}_{f_l}\left[(1-\tfrac{1}{c})L^-(g(x))\right]\right). \quad (6)$$

The maximum operator in Equation 6 prevents the issue of negative empirical risks. We can derive an appropriate loss function for PU learning that can substitute the binary cross-entropy based on Equation 6 and Equation 5. Hence, the unbiased PU cross-entropy and non-negative PU cross-entropy can be estimated from a data set:

$$uPU_{BCE} = -\sum_{i=1}^{N}\left(l_i\left[\frac{1}{c}\log(p_i) + (1-\frac{1}{c})\log(1-p_i)\right] + (1-l_i)\left[\log(1-p_i)\right]\right) \quad (7)$$

$$nnPU_{BCE} = -\textstyle\sum_{i=1}^{N}\left(l_i\left[\frac{1}{c}\log(p_i)\right] + \max\left(0, (1-l_i)\left[\log(1-p_i)\right] + l_i(1-\frac{1}{c})\log(1-p_i)\right)\right) \quad (8)$$

Unlike the binary cross-entropy, as shown in Equation 2, the ground-truth label y is not available but the label status $l \in \{0,1\}$. Notice that a labelled instance is always a positive example. We can, thus, use uPU_{BCE} and $nnPU_{BCE}$ as the loss function for a classification technique.

5 Experimental Setup

5.1 Setup

To address the hypotheses introduced earlier, two experimental setups were carefully constructed. Both share a similar setup, visualised in Figure 1. An event log is taken

and split into a training set and a test set. This is done $80-20\%$ *out-of-time*, i.e. every case with time activity timestamps before a certain moment is added to the training set and later cases are added to the test set (in way that approximately 20% of the cases ends up in the test set). However, since we do not want to discard too much data, it was opted to do a split without discarding the whole cases in an overlapping period and only remove the specific event with overlap to the test log period, in correspondence to other works in literature [24]. Subsequently, we look at the different positively labelled traces in the training, and flip different percentages (25, 50 and 75%) of these labels to a negative label, hereby replicating situations where different positive cases in the training set would not have been classified as such. The negative label should therefore better be called *unlabelled*. We also keep one *Original* log, for which no labels have been flipped. For each of these training sets, the prefix log is obtained, which is then used to train different models. The models used are each time an XGBoost classifier and an LSTM neural network, albeit with varying loss functions. In **Experiment 1** we solely want to investigate the possible negative effect of mislabelling positive examples. For this purpose, we opt to use the standard binary cross entropy. After training, the classifier predicts the labels of all prefixes in the prefix log of the test log, and these labels are compared to the true labels. As a score, we use the area under the ROC curve (AUC), which can be used to express the probability a classifier will give a higher prediction to a positive example than to a negative example. This was chosen due to it being threshold-independent and unbiased with imbalanced data sets. Notice that we did not flip any labels in the test log, as we want to test the model's actual performance. The different models (trained on logs with different label flip percentages) are compared.

In **Experiment 2** we also train classifiers with the *uPU* and *nnPU* loss functions introduced in Section 3. These classifiers are trained on the same training logs (only the one with label flips this time). By comparing the AUC on the test (untouched) examples, we can investigate the possible advantages of using PU learning loss functions over binary cross entropy. The XGBoost model is taken from [3] and the LSTM model is implemented by using the Python library Keras [2]. The *uPU* and *nnPU* loss function implementations designed for this work are also working on top of these libraries. The PU loss functions demand the user to give a *class prior*. In this work, we have used the percentage of label flips as input, to derive an estimate for the class prior. This is not a fully realistic setting, as in real-life you might not know how many positive cases you will have missed. However, often an adequate guess can be made based on expert knowledge or previous samples. The class prior derived from the flip ratio does not lead to the exact class prior as well, as it is based on traces and not prefixes. Longer traces create more prefixes in the training log since every activity in a trace (minus the last) is used as the last activity in a prefix. In addition, the positive class ratio of the training log is different from that of the test set. With this not-exact estimate of the class prior, we, therefore, deem our setup suitable to investigate Hypothesis 2.

[2] https://keras.io

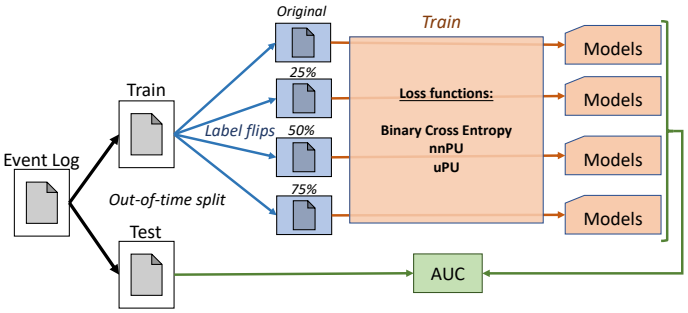

Fig. 1. Overview of the setup.

5.2 Event Logs

We have selected two sets of often used and publicly available event logs recorded from real-life processes. The outcomes are derived from a set of LTL rules similar as has been done in [13,4,24,20]. The first set of event logs, *BPIC2011* or *Hospital Log*, consists of four sublogs collected from the Gynaecology department of a Dutch Academic Hospital [6]. The different outcome LTL rules, and the accompanying trace cutting, are taken over from [24]. After collecting the patient's information, the patient's procedures and treatments are recorded. The *BPIC2015* event log consists of 5 different sublogs, each one having recorded a building permit application process in 5 different Dutch municipalities [5]. They share one LTL rule, checking whether a certain activity *send confirmation* receipt must always be followed by *retrieve missing data* [24]. The event logs' most important characteristics can be found in Table 1. Next to the number of traces in both train and test log, the minimum, maximum and median length of the traces can be found as well, together with the truncation length (prefixes longer than this length are not to be used). This can be due to computational considerations (cut off at 40 events) or earlier because the trace has reached all events determining its outcome. Also mentioned are the positive class ratio, $R(+)$ in both training and test set.

Dataset	Min Len	Med Len	Max Len	Trunc. Len	#Train	#Test	$R(+)$ Train	$R(+)$ Test
2011_1	1	25	1814	36	912	228	0.38	0.48
2011_2	1	54.5	1814	40	912	228	0.81	0.66
2011_3	1	21	1368	31	896	225	0.20	0.36
2011_4	1	44	1432	40	912	228	0.25	0.39
2015_1	2	42	101	40	555	140	0.22	0.26
2015_2	1	55	132	40	602	151	0.20	0.17
2015_3	3	42	124	40	1062	266	0.17	0.25
2015_4	1	42	82	40	460	116	0.17	0.13
2015_5	5	50	134	40	840	211	0.32	0.26

Table 1. An overview of the characteristics of the data sets used.

5.3 Encoding and Hyperparameters

The prepossessing pipeline of the XGBoost model is based on previous work discussing different machine learning approaches [24]. The adjustments to the preprocessing

pipeline to use the LSTM model are taken from [20]. To ensure proper training, some hyperparameters have to be carefully selected. For this purpose, we have done a hyperparameters search for each of the variations (different label flip ratios) of each log, and this for each different model (using different loss functions). The hyperparameter selection is performed with the use of hyperopt. For the LSTM models these are the *size* of the LSTM hidden layers, the *batch size dropout rate*, *learning rate* and *optimizer* (*Adam*, *Nadam*, *SGD* or *RMSprop*) used during training. For the XGB models these are *subsample*, *maximum tree depth*, *colsample bytree*, *minimum child weight* and the *learning rate*. The XGB models also use the *aggregation* encoding setting to encode the features of a prefix, taken over from [24]. Although this sequence encoding mechanism ignores the order of the traces, the study of k [24] shows that it works best for our selected data sets (and similar pipeline). For the LSTM models, the features are encoded in an embedding layer. The rest of the model consists of two bidirectional recurrent layers, with a dense output layer.

6 Experimental Evaluation

6.1 Experiment 1

The AUC on the independent test set is assessed for each of the models trained on the training logs with different ratios of flipping the positive examples. The results can be found in Table 6.1 and, as expected, overall we can see a decreasing trend in AUC when adding more and more label flips to the training set. What stands out is the relative bad AUC of the LSTM model as compared to the XGB. The decrease in AUC when adding positive label flips to the training set, often also seems sharper and more volatile (not always decreasing when more label flips are added) for the LSTM classifiers. The LSTM classifier trained on the original *'bpic_2011_3'* training set seems to score a particularly low score, and definitely stands out as an outlier. Another remarkable example can be found for data set *'bpic_2015_2'*, and *'bpic_2015_3'*, for which the relatively limited AUC decrease (for the XGB model) might be partially explained by this data set containing a lot of longer traces. The AUC results for the XGBoost model of [24] show that predictions for prefixes longer than length 15 are all almost 1. Intuitively, this boils down to the fact that the model is almost certain of the label prediction for prefixes with a minimal length of 15. In addition, the prefix log of the test set contains 63% prefixes of size larger or equal to 15, at which point the XGB model already has almost perfect predictions, such that the influence of the data flips in the training set has less influence on the overall AUC score. This is however only a partial explanation since it would not explain an actual increase of the XGB test performance when training on the training data with 25% of the positive labels flipped as compared to a model trained on the untouched training set. Possibly effects like the test set having a lower positive label ratio than the training set, or other data set-specific characteristics, might provide some extra explanation. Overall, we can confirm Hypothesis 1, however, the extent (and volatility) of the decrease can still be process dependent, or might even depend on which specific cases have a missing positive label.

		Flip Ratio			
Dataset	Method	0%	25%	50%	75%
bpic2011_1	LSTM	0.891	0.514	0.667	0.509
bpic2011_1	XGB	0.944	0.867	0.805	0.751
bpic2011_2	LSTM	0.882	0.520	0.783	0.494
bpic2011_2	XGB	0.972	0.962	0.905	0.785
bpic2011_3	LSTM	0.680	0.863	0.755	0.831
bpic2011_3	XGB	0.989	0.982	0.803	0.905
bpic2011_4	LSTM	0.873	0.680	0.680	0.736
bpic2011_4	XGB	0.865	0.855	0.813	0.720
bpic2015_1	LSTM	0.885	0.706	0.712	0.579
bpic2015_1	XGB	0.917	0.919	0.904	0.761
bpic2015_2	LSTM	0.937	0.854	0.803	0.807
bpic2015_2	XGB	0.947	0.952	0.914	0.909
bpic2015_3	LSTM	0.878	0.673	0.694	0.624
bpic2015_3	XGB	0.962	0.941	0.942	0.930
bpic2015_4	LSTM	0.858	0.784	0.715	0.465
bpic2015_4	XGB	0.917	0.898	0.837	0.847
bpic2015_5	LSTM	0.916	0.757	0.759	0.667
bpic2015_5	XGB	0.944	0.939	0.907	0.813

Table 2. AUC on an untouched test set for XGB and LSTM models, trained on training logs with different amounts of label flips.

6.2 Experiment 2

As mentioned earlier, in a second experiment we introduce the models using the *uPU* and *nnPU* loss functions, next to those using binary cross entropy (*CE*). We discard the original logs and only look at the logs for which positive labels have been flipped. The results of these experiments can be found in Table 6.2. Overall, an uplift can be seen in using the *nnPU* loss function over the *BCE*, for both LSTM and XGB. However, this is not always the case and the effectiveness of using PU learning seems to be log-dependent. Standing out again in the event log *'bpic_2015_2'*, for which the *nnPU* function seems not to be effective (even very flawed in the LSTM's case). This event log also showed only slight decreases in AUC when adding the label flips. Overall, using the *nnPU* loss function seems to lead to better scores than the *uPU*. Also in OOPPM the possibly negative risk values the *uPU* loss function can obtain, seem to have a negative impact on the learning. Depending on the process in question, using the *nnPU* loss function seems to be able to increase the real performance of a classifier, so Hypothesis 2 can be (partially) confirmed. Further research will be needed to understand when and why PU learning seems (not) to work well in OOPPM.

7 Conclusion and Future Work

In this work, we have introduced OOPPM models to a setting where our training log consists of positive and unlabelled traces. This kind of situation might arise when the labelling of your positive cases is uncertain, e.g. when it is hard for the process owner to obtain all the information or be sure. A key example application is

		LSTM			XGB		
Dataset	Flip	CE	nnPU	uPU	CE	nnPU	uPU
bpic2011_1	25%	0.514	**0.818**	**0.818**	0.867	**0.910**	0.897
bpic2011_1	50%	0.667	**0.736**	0.565	0.805	**0.889**	0.800
bpic2011_1	75%	0.509	0.505	**0.727**	0.751	**0.801**	0.684
bpic2011_2	25%	0.520	**0.752**	0.723	0.962	**0.963**	0.921
bpic2011_2	50%	0.783	**0.820**	0.662	0.905	0.922	**0.942**
bpic2011_2	75%	0.494	0.530	**0.612**	0.785	**0.827**	0.545
bpic2011_3	25%	**0.863**	0.838	0.750	0.982	0.975	**0.987**
bpic2011_3	50%	0.755	**0.773**	0.687	0.803	**0.925**	0.831
bpic2011_3	75%	**0.831**	0.779	0.707	0.905	**0.931**	0.911
bpic2011_4	25%	0.680	0.773	**0.775**	0.855	**0.868**	0.861
bpic2011_4	50%	0.680	**0.784**	0.734	**0.813**	0.812	0.718
bpic2011_4	75%	0.736	0.694	**0.840**	0.720	**0.797**	0.729
bpic2015_1	25%	0.706	0.804	**0.817**	**0.919**	0.916	0.917
bpic2015_1	50%	0.712	**0.803**	0.663	0.904	**0.918**	0.865
bpic2015_1	75%	0.579	0.609	**0.638**	0.761	0.631	**0.774**
bpic2015_2	25%	**0.854**	0.486	0.839	**0.952**	0.949	0.945
bpic2015_2	50%	0.803	0.594	**0.855**	**0.914**	0.902	0.867
bpic2015_2	75%	**0.807**	0.742	0.653	**0.909**	0.858	0.821
bpic2015_3	25%	0.673	**0.777**	0.592	0.941	**0.955**	0.947
bpic2015_3	50%	0.694	**0.715**	0.628	**0.942**	**0.942**	0.934
bpic2015_3	75%	0.624	**0.835**	0.583	**0.930**	0.904	**0.930**
bpic2015_4	25%	0.784	**0.821**	0.801	0.898	0.898	**0.923**
bpic2015_4	50%	**0.715**	0.615	0.678	0.837	**0.886**	0.844
bpic2015_4	75%	0.465	**0.664**	0.598	**0.847**	0.835	0.839
bpic2015_5	25%	**0.757**	0.710	0.684	**0.939**	0.937	0.924
bpic2015_5	50%	**0.759**	0.755	0.693	0.907	**0.921**	0.912
bpic2015_5	75%	0.667	**0.680**	0.576	0.813	**0.837**	0.777

Table 3. AUC on an untouched test set for models trained with different loss functions on training logs with different amounts of label flips.

fraud detection, but also in other areas, obtaining accurate labels for all cases might be costly or even impossible, such as labels based on customer feedback or labels to be obtained from other parties collaborating in a multi-organisational business process. By training different LSTM and XGB models on different variations of an event log, each time with an increasing number of the positively labelled traces' label flipped to negative (and therefore changing the negative label to unlabelled), a drop in the classifiers' performance could be noticed, hereby confirming Hypothesis 1. Furthermore, we investigated the potential use of loss functions from the field of PU learning to mitigate this issue and found that generally, on our example event logs, a model trained with the *nnPU* loss function would score higher in a situation where the training data had traces' positive labels flipped. This was generally true, but not for all event logs, so further investigations and fine-tuning might be interesting when applying this to data from other processes. This paper opens up a door for future research on OOPPM in positive and unlabelled settings.

In future work, a more extensive experiment with more event logs could be performed. Furthermore, creating multiple variations of the log for each random flip ratio,

as well as flipping labels of examples closer or further from the decision boundary might have an impact. A setting with 0% of the cases flipped has been excluded from experiment 2 since there would be little effect in changing the loss function (as the flip ratio was given). In future experiments on the sensitivity of having an (incorrect) class prior, this setting could be added, however. It would also be interesting to test this setup in data for which we know the labelling is uncertain by itself, in contrast to doing the ratio flips ourselves. One other limitation of this work is that our loss functions rely on knowing the *class prior*, and for this, we have used the flip ratio as an input. Because we purely wanted to investigate the potential use of the PU loss function (and because the class prior was still not the exact class prior of the training set), this was deemed acceptable. However, in future work, it might be interesting to investigate the impact of using different class prior values (or using class priors derived from different samples). Other future work on dealing with unreliable negative labels could be found in investigating options besides altering the loss function. The process behaviour itself may also reveal valuable information concerning which negative labels can be considered more certain.

References

1. Bekker, J., Davis, J.: Learning from positive and unlabeled data: a survey. Machine Learning **109**(4), 719–760 (2020). https://doi.org/10.1007/s10994-020-05877-5, https://doi.org/10.1007/s10994-020-05877-5 2
2. Bekker, J., Robberechts, P., Davis, J.: Beyond the selected completely at random assumption for learning from positive and unlabeled data. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 71–85. Springer (2019) 4
3. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. p. 785–794. KDD '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2939672.2939785 2, 4, 5, 6
4. Di Francescomarino, C., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. IEEE Transactions on Services Computing **12**(6), 896–909 (2019). https://doi.org/10.1109/TSC.2016.2645153 2, 7
5. van Dongen, B.B.: Bpi challenge 2015 (May 2015). https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1, https://data.4tu.nl/collections/BPI_Challenge_2015/5065424/1 7
6. van Dongen, B.: Real-life event logs - Hospital log (3 2011). https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54, https://data.4tu.nl/articles/dataset/Real-life_event_logs_-_Hospital_log/12716513 7
7. Du Plessis, M., Niu, G., Sugiyama, M.: Convex formulation for learning from positive and unlabeled data. In: International conference on machine learning. pp. 1386–1394. PMLR (2015) 2, 4, 5
8. Folino, F., Folino, G., Guarascio, M., Pontieri, L.: Semi-supervised discovery of dnn-based outcome predictors from scarcely-labeled process logs. Business & Information Systems Engineering (Apr 2022). https://doi.org/10.1007/s12599-022-00749-9 3
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997) 2, 4
10. Jaskie, K., Spanias, A.: Positive and unlabeled learning algorithms and applications: A survey. In: 2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA). pp. 1–8 (2019). https://doi.org/10.1109/IISA.2019.8900698 2

11. Kiryo, R., Niu, G., Du Plessis, M.C., Sugiyama, M.: Positive-unlabeled learning with non-negative risk estimator. Advances in neural information processing systems **30** (2017) 2, 4, 5
12. Kratsch, W., Manderscheid, J., Röglinger, M., Seyfried, J.: Machine learning in business process monitoring: a comparison of deep learning and classical approaches used for outcome prediction. Business & Information Systems Engineering **63**(3), 261–276 (2021) 2, 3
13. Leontjeva, A., Conforti, R., Francescomarino, C.D., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: International Conference on Business Process Management. pp. 297–313. Springer (2016) 7
14. Martin, N.: Data Quality in Process Mining, pp. 53–79. Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-53993-1_5 2
15. Neu, D.A., Lahann, J., Fettke, P.: A systematic literature review on state-of-the-art deep learning methods for process prediction. Artificial Intelligence Review pp. 1–27 (2021) 3
16. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Using convolutional neural networks for predictive process analytics. In: 2019 international conference on process mining (ICPM). pp. 129–136. IEEE (2019) 3
17. Pauwels, S., Calders, T.: Incremental predictive process monitoring: The next activity case. In: International Conference on Business Process Management. pp. 123–140. Springer (2021) 3
18. Rama-Maneiro, E., Vidal, J., Lama, M.: Deep learning for predictive business process monitoring: Review and benchmark. IEEE Transactions on Services Computing (2021) 3
19. Rizzi, W., Di Francescomarino, C., Ghidini, C., Maggi, F.M.: How do i update my model? on the resilience of predictive process monitoring models to change. Knowledge and Information Systems **64**(5), 1385–1416 (2022) 3
20. Stevens, A., De Smedt, J., Peeperkorn, J.: Quantifying explainability in outcome-oriented predictive process monitoring. In: Munoz-Gama, J., Lu, X. (eds.) Process Mining Workshops. pp. 194–206. Springer International Publishing, Cham (2022) 7, 8
21. Su, G., Chen, W., Xu, M.: Positive-unlabeled learning from imbalanced data. In: IJCAI. pp. 2995–3001 (2021) 2
22. Tax, N., Verenich, I., Rosa, M.L., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: International Conference on Advanced Information Systems Engineering. pp. 477–492. Springer (2017) 2, 3
23. Taymouri, F., Rosa, M.L., Erfani, S., Bozorgi, Z.D., Verenich, I.: Predictive business process monitoring via generative adversarial nets: the case of next event prediction. In: International Conference on Business Process Management. pp. 237–256. Springer (2020) 3
24. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Trans. Knowl. Discov. Data **13**(2) (mar 2019). https://doi.org/10.1145/3301300 2, 3, 6, 7, 8
25. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Teinemaa, I.: Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. ACM Transactions on Intelligent Systems and Technology (TIST) **10**(4), 1–34 (2019) 2
26. Wang, H., Wang, S.: Mining incomplete survey data through classification. Knowl. Inf. Syst. **24**, 221–233 (08 2010). https://doi.org/10.1007/s10115-009-0245-8 2
27. Wu, M., Pan, S., Du, L., Tsang, I., Zhu, X., Du, B.: Long-short distance aggregation networks for positive unlabeled graph learning. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management. pp. 2157–2160 (2019) 2