Sparse Mixtures of Shallow Linear Experts for Interpretable and Fast Outcome Prediction

Francesco Folino, Luigi Pontieri, and Pietro Sabatino

Institute for High Performance Computing and Networking (ICAR-CNR)
via P. Bucci 8/9C, 87036 Rende (CS), Italy
name.surname@icar.cnr.it

Abstract. In Process Mining research, Outcome Prediction entails predicting a discrete, outcome-related property of an unfinished process instance from its partial trace. Various outcome predictors discovered via Machine Learning (ML) methods, like rule/tree ensembles and (deep) neural networks, have achieved top accuracy performances. However, their opaqueness makes them unsuitable for scenarios necessitating understandable outcome predictors. Aligning with recent efforts to mine inherently interpretable predictors, we suggest training a sparse Mixture-of-Experts, with the "gate" and "expert" sub-nets being Logistic Regressors. This ensemble of specialized predictors is trained end-to-end while restricting the number of input features used in the sub-nets, in the place of typical multi-step/objective mining pipelines (e.g, global feature selection steps followed by an ML one). This enables different experts to focus on varied input features for predicting the outcomes of instances within their competency regions. Test results on benchmark logs confirmed the ability of this approach to reach a compelling trade-off between accuracy and interpretability, compared to current solutions.

Keywords: Process Mining · Machine Learning · XAI · Green AI

1 Introduction

Predictive Process Monitoring (PPM) methods [7] aim at extending process monitoring mechanisms with the ability to forecast properties of ongoing process instances. This is an active line of research in the area of *Process Mining* (*PM*), owing to the advantages that such methods can bring in terms of run-time decision support and optimization. In this context, we specifically address the *(Process) Outcome Prediction* problem [19], i.e., the problem of predicting the outcome of an unfinished process instance, based on its associated *prefix trace* (i.e., the partial sequence of events available for it).

Recently, different supervised learning approaches to this problem were proposed, which allow for discovering an outcome prediction model from labeled traces. Outstanding performances in terms of prediction accuracy have been achieved by two classes of powerful predictors: (i) big ensembles of decision rules/trees discovered with random forest or gradient boosting algorithms [19]; and *Deep Neural Networks* (*DNNs*) (see [10] for a recent survey and benchmark) consisting, for instance, of multiple feed-forward layers with sequence-oriented RNNs [18] or CNN [8]. In particular, the DNNs quickly became popular in this field thanks to their ability to automatically

learn hierarchies of abstract features from raw data without requiring the analyst to preliminary bring the log data into a tabular form based on some suitable trace encoding method [19]. However, the function-approximation power of these models comes at the cost of an opaque internal decision logic, which makes them unfit for real-life settings where decision-makers want explainable predictions and/or interpretable predictors.

The call for transparent outcome prediction gave rise to a first wave of proposals that simply reused model-agnostic post-hoc explanation methods (like LIME and SHAP) [5,3,11,13] or explanation-friendly DNN-oriented solutions (e.g., attention modules, gradient/LRP -based feature attribution) [20,17]. More recently, concerns on the reliability of attention/attribution-based explanations [2] and the faithfulness, stability, consistency and efficiency of post-hoc explanations [14,21] pushed PM researchers to face the discovery of inherently-interpretable predictors [16,15,9].

Related works and their limitations Two kinds of interpretable outcome predictors, both leveraging *Logistic Regression* (*LR*) models as a building block, were recently used in [16,15] (where an evaluation framework for interpretable outcome predictors was introduced), starting from flattened (through aggregation encoding [19]) log traces: (i) *Logit Leaf Model* (*LLM*), a sort of decision tree where each leaf hosts an LR submodel; and (ii) *Generalized Logistic Rule Model* (*GLRM*) where a single LR model is built upon the original feature and novel features, defined as conjunctive rules over subsets of the former and derived via column generation. Both LLM and GLRM models were shown to improve plain LR predictors, thanks to their ability to capture some non-linear input-output dependencies. An approach leveraging a neural implementation of fuzzy logic, named *FOX*, was proposed in [9], which can extract logical outcome-prediction rules from aggregation-encoded traces. Its result specifically consists of easy-to-interpret IF-THEN rules, each containing a fuzzy set per input feature and a membership score per outcome class. The domain of each data feature is split into three "linguistic" fuzzy sets, so that 3^k rules are learnt by training this model over flat k-dimensional data, after selecting the best k features with an MI-based method.

In principle, learning an LLM consisting of multiple LRs, as proposed in [16,15] LLM could help find an interpretable predictor. However, if using no mechanisms for limiting the number of data features and the size of the decision tree, cumbersome models can be obtained, as noted in [15], which hardly allow the user to grasp a full and precise understanding of model predictions. On the other hand, as noted in [15], the GLRM method is very demanding in terms of training examples (owing to its sensitiveness to outliers and the sparsity of its transformed space) and computation time (because of its column generation step). Finally, we are afraid the global feature selection and (3-way) feature binning performed in [9] allow the user to control the size of each prediction rule, but at the risk of losing some information and prediction accuracy.

Goal and contribution Our research work was aimed at directly seeking a (locally) optimal ensemble of specialized (and complementary) LR models by training a Mixture of Experts (MoE) [6] neural network, which consists of multiple "experts" (the LR models) and a sparse "gate" module assigning any data instance to one of the experts. The proposed approach shares some conceptual and technical features with a method presented in [1]. Specifically, in [1] several MoE variants, differing in the forms of

the gate (black-box vs. interpretable) and of the experts (only interpretable vs. also including a DNN), are trained to predict a numerical target, using a combined loss function including many regularization terms (e.g., linked to the prediction accuracy of both the MoE and its gate module and the degree of load balance among the experts).

Our approach to discovering an interpretable MoE-based outcome predictor, called *MoE-OPM*, relies upon several ad hoc design choices and novel technical solutions:

- For the sake of interpretability and scalability, the gate and expert models play as LR classifiers (in fact, all these modules are one-layer neural networks with linear activations and a final softmax/sigmoid link function); this allows for directly learning several local LR models and an associated interpretable assignment function like in LLM models [16,15].
- Differently from [1], we let the user control the complexity (and thus the interpretability) of the model by fixing the maximum number $kTop$ of features that the gate and each expert sub-net can use, as well as the desired number m of experts. However, instead of resorting to a preliminary global feature selection step as in [16] and [9], the model is trained using all the original data features, while using L1-based regularization terms to encourage model sparsity; the parameters of each trained sub-net (i.e. either the gate or an expert) are then pruned in a "feature-based" way, by zeroing all but the $kTop$ most important input features of the sub-net. Each expert can thus use a specific subset of input features when making predictions for the data instances assigned to it.
- We prefer to avoid enforcing expert load balancing through an explicit loss term, as in [1], to prune redundant experts and further reduce the model size.

Tests on benchmark logs confirmed that the approach achieves compelling accuracy scores (w.r.t. state-of-the-art interpretable outcome-prediction models [9,16,15]), and supports compact and fully faithful prediction explanations (in the form of feature-attribution scores) –see Section 4.3 for an example from a real healthcare process.

2 Background and problem statement

As usual, assume that, for every execution instance of the process under analysis (a.k.a. *process instance*), a distinguished *trace* is stored, which consists of a (temporally ordered) sequence of (log) *events*, plus several instance-level attributes that do not vary during the process execution. Each event is a tuple representing a process execution event, which usually stores information on the execution of a process activity (e.g., which activity was executed, the executor, a timestamp, etc.).

At run time, during the unfolding of a process instance, a trace is recorded incrementally to store the events observed for that instance. Such a trace is called a *prefix trace* and represents a kind of pre-mortem log data. Let us denote as U the universe of all possible prefix traces produced by the business process under analysis.

Problem OPM *discovery* This work aims to discover a predictive model for (probabilistically) forecasting the outcome class of a running process instance. As we want to exploit neural networks to address this problem, standard data transformation methods

are used to pre-process the tabular trace representation described so far: the value of each categorical attribute, including the outcome class, is eventually represented in a one-hot encoding (we do not employ dense embeddings, for the sake of explainability), whereas any numerical attribute is just made undergo min-max normalization.

Let us assume that each trace $\tau \in U$ can be turned into a real-valued vector $x \in \mathbb{R}^d$ (for some suitable $d \in \mathbb{N}$), and its class label is encoded as a one-hot vector $y \in [0,1]^{c-1}$, where c is the number of outcome classes. For the sake of simplicity, and w.l.o.g, let us focus from now on on the representations obtained with the *aggregation encoding* of [19] in the case of two outcome classes (i.e., $c = 2$).

Under this perspective, an *OPM* can be re-defined as a neural network N that encodes a function $f_N : \mathbb{R}^d \to [0,1]$ mapping the vectorial representation x of any (prefix) trace in U to an estimate of the probability that x belongs to the second class (i.e., the one associated with label 1). Then, the inductive learning problem faced in this work, named *OPM* `discovery` from now on, amounts to training a neural net N of this form out of a given collection of class-labeled prefix traces, once turning them into a pair (X,Y) of tensors that store (numerical representations of) the propositional encodings of these traces and their associated outcome labels, respectively.

3 Solution approach

Similarly, to [1], we rephrase the problem of learning of a *OPM* as discovering a special (extremely sparse) kind of *Mixture of Experts* (*MoE*) ensemble, that consists of multiple specialized outcome-oriented classifiers, named "experts", plus a "gate" module that routes each data sample to one of the experts. For the sake of readability, the conceptual architecture of a standard MoE is sketched in Figure 1.

Definition 1 (*MoE-OPM*). *A* Mixture-of-Experts-OPM *(*MoE-OPM*) is a neural net of the form* $\mathcal{N} = \langle \mathcal{N}_g, \mathcal{N}_1, \ldots, \mathcal{N}_m \rangle$ *that assembles two kinds of sub-nets: (i)* $\mathcal{N}_1, \ldots, \mathcal{N}_m$*, called* experts*, which encode different (local)* OPM *functions* $f_1, \ldots, f_m \in [0,1]^{(\mathbb{R}^d)}$*, each mapping any trace encoding to an estimate of the probability that the respective process instance will belong to the second outcome class; and (ii)* $\mathcal{N}_g$*, named* gate*, which encodes a routing function* $g : \mathbb{R}^d \to \Delta_m$ *(with* Δ_m *denoting the probability simplex over* $\{1, \ldots, m\}$*) that maps any trace encoding to a categorical probability distribution over the (indexes of the) experts. As a whole,* $\mathcal{N}$ *is itself an* OPM *that encodes the function* $f : \mathbb{R}^d \to [0,1]$ *defined as follows:* $f(x) \triangleq f_k(x)$ *such that* $k = \arg\max_{k \in \{1,\ldots,m\}} g(x)[k]$ *and* $g(x)[k]$ *is the k-th component of the probability vector returned by g when applied to x.* □

The experts and the gate could be instantiated using different models, provided they all are differentiable. In [1], various alternatives were proposed to implement the experts and the gate. In this work, for the sake of interpretability and memory/computation saving, we prefer to instantiate each *MoE-OPM* taking the following design choices: (*a*) expert sub-nets $\mathcal{N}_1, \ldots, \mathcal{N}_m$ are all implemented as d-to-1 one-layer feed-forward nets with linear activation functions, followed by a standard sigmoid transformation; (*b*) the gate sub-net $\mathcal{N}_g$ is a one-layer d-to-m feed-forward network with linear activation functions followed by softmax normalization. This makes each expert and the gate behave as (standard/multinominal) *LR* models, usually considered interpretable models.

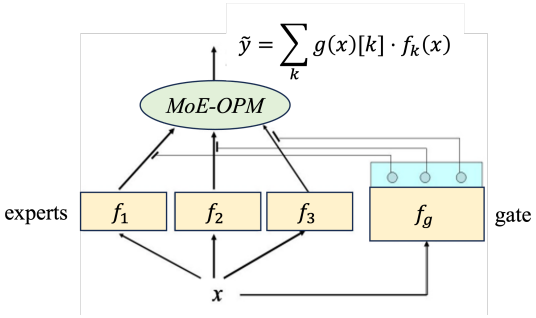

Fig. 1: Standard MoE architecture (for $m = 3$ experts). Each expert sub-net implements a specific classification function f_k, while gate's function f_g returns per-expert competency scores $g(x)[1], g(x)[2], g(x)[3]$ summing up to 1. In our *MoE-OPM* variant, the gate, and experts are LR models, the gate is biased to return sparse competency scores, and x is classified by using only the expert getting the highest score.

Algorithm MoE-OPM Discovery Such an ensemble model is discovered through Algorithm 1, named *MoE-OPM Discovery*. This algorithm takes several training instances as input, through tensors X (vectorized prefix traces) and Y (binary class labels), and the percentage *valPerc* of them to use for validation. To control the complexity of both the gate and expert sub-nets in the *MoE-OPM*, the user can set both the desired number m of experts and an upper bound $kTop$ to the number of input features that each of these sub-nets can use at prediction time. This feature-selection-like requisite is fulfilled by zeroing the weights of the connection parameters related to all the input features of a sub-model, but the $kTop$ of them that look the most relevant ones for the sub-model (see later on). Further arguments are: the maximum number of SGD-based epochs for training the model (nEp_1) and for re-tuning it after feature-weight pruning (nEp_3), the learning rates for training the gate (η_g) and each expert (η_e), and the weights λ_R and λ'_R of the loss regularization terms. To support ablation studies, one can set $kTop = ALL$, meaning no parameters are pruned, and the *MoE-OPM* can use all the input features.

The algorithm consists of four computation phases: (i) train a randomly-initialized *MoE-OPM* end-to-end (Step 4); (ii) fine-tune the gate sub-net while keeping the parameters of the experts frozen (Step 5); (iii) zero all the parameter weights (Steps 7-8), in both the gate and the experts, that do not relate to the $kTop$ input features (see later on); (iv) re-train (for nEp_3 epochs) the *MoE-OPM* end-to-end to possibly adapt it to the removal of parameters (Step 19). All the training phases (*i*, *ii* and *iv*) are performed with procedure `TRAIN`, which implements a standard SGD-based optimization method when setting *freeze* = **true**. Otherwise (*freeze* = **false**), the procedure only fine-tunes the gate while keeping all experts frozen. Note that all the experts contribute to each ensemble prediction during the training, and the training loss is estimated as in a standard MoEs.

Algorithm 1 *MoE-OPM Discovery* (abstracting from SGD optimization details).

Input: tensors X and Y storing the flat representations of prefix traces and their associated outcome classes; no. $m \in \mathbb{N} \cup \{ALL\}$ of experts and max. no. $kTop \in \mathbb{N} \cup \{ALL\}$ of input features to be used in each *MoE-OPM* subnet; validation percentage $valPerc \in (0,100)$; numbers $nEp_1, nEp_2, nEp_3 \in \mathbb{N}$ of max. epochs for the training, gate fine-tuning, and (post-pruning) re-training phases, respectively; learning rates $\eta_g, \eta_e \in \mathbb{R}^+$ for training the gate and the expert modules, respectively; regularization weights $\lambda_R, \lambda'_R \in \mathbb{R}^+$ (see Sect. 3);

Output: an *OPM* $\langle N_g, N_1, \ldots, N_m \rangle$.

split $[X;Y]$ into training and validation sets $[X_T;Y_T]$ and $[X_V;Y_V]$, respectively, such that $|X_V| = |Y_V| = \lfloor |X| \times valPerc/100 \rfloor$;
create a *OPM* $\mathcal{N} = \langle \mathcal{N}_g, \mathcal{N}_1, \ldots, \mathcal{N}_m \rangle$ with randomly-initialized parameters Θ;
let $\Theta|_g, \Theta|_1, \ldots, \Theta|_m$ denote the parameters of $\mathcal{N}_g, \mathcal{N}_1, \ldots, \mathcal{N}_m$, respectively;
TRAIN($\mathcal{N}, X_T, Y_T, X_V, Y_V, b, nEp_1, \eta_g, \eta_e, freeze =$ **false**); *// train the* OPM *end-to-end*
TRAIN($\mathcal{N}, X_T, Y_T, X_V, Y_V, b, nEp_1, \eta_g, \eta_e, freeze =$ **true**); *// fine-tune the gate only*
if $kTop \neq ALL$ **then**
    zero all $\mathcal{N}_g$'s weights but those linked to its top $kTop$ input features; *// see below Eq. 2*
    zero all $\mathcal{N}_q$'s weights, for $q \in \{1, \ldots, m\}$, but those with the top $kTop$ absolute values;
    TRAIN($\mathcal{N}, X_T, Y_T, X_V, Y_V, b, nEp_3, \eta_g, \eta_e, freeze =$ **false**); *// re-tune* $\mathcal{N}$ *end-to-end*
end if
return $\mathcal{N}$

Loss and feature selection Let $\mathcal{N}_{moe} = \langle \mathcal{N}_g, \mathcal{N}_1, \ldots, \mathcal{N}_m \rangle$ be a given *MoE-OPM*, $\mathcal{N}_E$ be the sub-net consisting of all its experts $\mathcal{N}_1, \ldots, \mathcal{N}_m$, and $weights(\mathcal{N})$ be a function returning the connection-weight parameters of any neural net $\mathcal{N}$. Then, for given paired sub-tensors X and Y, storing n training instances (denoted by $X[1], \ldots, X[n]$) and their associated class labels (denoted by $Y[1], \ldots, Y[n]$), respectively, the loss of $\mathcal{N}$ on X, Y is computed as $\mathscr{L}_{acc}(\mathcal{N}, X, Y) + \lambda_R \cdot \mathscr{L}_{reg}(\mathcal{N}_g) + \lambda'_R \cdot \mathscr{L}_{reg}(\mathcal{N}_E)$ with:

$$\mathscr{L}_{acc}(\mathcal{N}, X, Y) = \frac{1}{n \cdot m} \sum_{i=1}^{n} \log \left(\sum_{k=1}^{m} g(X[i])[k] \cdot \left(1 + e^{f_k(X[i]) \cdot (1 - 2 \cdot Y[i])} \right) \right) \tag{1}$$

$$\mathscr{L}_{reg}(\mathcal{N}) = \frac{\sum_{\theta \in weigths(\mathcal{N})} |\theta|}{|\{\theta \mid \theta \in weigths(\mathcal{N})\}|} \tag{2}$$

The accuracy loss in Eq. 1, as discussed in both [1] and [6], is expected to favor expert specialization. The loss term in Eq. 2, looking like a classic regularization term (as in Lasso logistics regression), is meant to shrink less relevant weights to help spot the $kTop$ most important input features and retain only the weights related to them.

As concerns weight pruning, for each expert $\mathcal{N}_k$, we rank the weights in $weights(\mathcal{N}_k)$ based on their absolute values, from the highest to the lowest, and zero all but the first $kTop$ of them. A "channel-wise" pruning is performed instead on $\mathcal{N}_g$, based on grouping its weights by the input features they relate to: the group of each input feature A_i gathers the weights of all the connections in $\mathcal{N}_g$ lying on a path starting from the i-th input node of $\mathcal{N}_g$. After ranking these weight groups based on their aggregated magnitude scores (each score is the absolute average value of the weights in a group), we eventually set to 0 the weights of $\mathcal{N}_g$ that do not belong to one of the top $kTop$ groups.

In the implementation of our approach, each node in the input layer is paired with a multiplicative binary mask. This way, the parameters of each sub-model (i.e. the gate or an expert) can be pruned by setting to 0 all masks but those of its top $kTop$ features.

4 Experiments

For the sake of comparison, the proposed algorithm (implemented in *Python 3.11.4* and *PyTorch 2.0.1*, partly reusing code released in [1]) has been tested against several pre-processed datasets [1], derived from benchmark logs *BPIC 2011* and *Sepsis* that were also used in [9,19,16,15]. All these datasets, in tabular format, were built by making the respective prefix traces undergo the aggregation encoding (see Sect. 2) after extending them all with timestamp-derived temporal features (e.g., weekday, hour, etc.).

Essentially, log *BPIC 2011* stores the clinical history of Gynaecology patients in a Dutch hospital. Its events represent applied treatments and procedures. Four datasets, namely `bpic2011_1`, `bpic2011_2`, `bpic2011_3`, and `bpic2011_4`, were derived from this log by assigning each trace an outcome label based on the satisfaction of an LTL rule (labels '1' for violation and '0' for adherence) —see [19] for details.

Log *Sepsis* stores care-flow data of Sepsis patients in a Dutch hospital, from Emergency Room (ER) registration to final discharge. Three datasets were derived from it by assigning each trace a boolean class label [19] equal to 0 iff the trace concerns an ER revisit within 28 days after discharge (dataset `sepsis_1`), an ICU admission (dataset `sepsis_2`), or a discharge type different from 'Release A' (dataset `sepsis_3`).

Test procedure, parameter setting and competitors Each dataset was partitioned into training, validation and test sets exactly as done in [9,19,15]: after sorting the traces based on their starting time, the first 80% of them were used for training and the remaining 20% for testing each *OPM*; the last 20% of the training set was used as the validation set. The accuracy of each discovered model was evaluated by computing the AUC score for all test prefixes containing at least two events, as done in [9,19,15].

Algorithm *MoE-OPM Discovery* was run with a fixed configuration of all parameters but λ_R, $kTop$ and b, namely: $nEp_1 = nEp_2 = 100$, $nEp_3 = 0$, $m = 6$, $\eta_e = 10^{-5}$ and $\eta_g = 10^{-2}$. By the way, the number m of experts was chosen empirically (after trying several values in [2..16] for it), since this choice seemed to ensure a good enough accuracy-vs-simplicity trade-off. The following configurations were used for the regularization weights λ_R and λ'_R and the number $kTop$ input features per sub-net in the tests performed on dataset `bpic2011_3`: $(kTop = 2; \lambda_R = \lambda'_R = 0.3)$, $(kTop = 4; \lambda_R = \lambda'_R = 0.4)$, $(kTop = 6; \lambda_R = \lambda'_R = 0.6)$ and $(kTop = 8; \lambda_R = \lambda'_R = 0.8)$. In the other tests we evaluated the configurations $(kTop = 2; \lambda_R = 0.1)$, $(kTop = 4; \lambda_R = 0.2)$, $(kTop = 6; \lambda_R = 0.4)$ and $(kTop = 8; \lambda_R = 0.6)$, with λ'_R set to 0.3, 0.4, 0.6 and 1.0, respectively (for datasets `sepsis_1`, `sepsis_2` and `sepsis_3`) or to the same value as λ_R (for the remaining datasets). For each dataset, the batch size b was given a value in [4..128] chosen empirically, namely: $b = 4$ for `bpic2011_3`, $b = 4$ for `sepsis_2` and `sepsis_3` and all the other datasets derived from *BPIC 2011*, and $b = 32$ for `sepsis_1`.

[1] https://github.com/vinspdb/FOX

Table 1: AUC scores obtained by: algorithm *MoE-OPM Discovery*, run with a fixed number ($m = 6$) of experts and several values of $kTop$ (namely, 2,4,6,8 and *ALL*), the baseline method *1-LR* and three state-of-the-art competitors. For each dataset, the best score is shown in **Bold and underlined**; each score obtained by *MoE-OPM Discovery* is shown in **Bold** if it outperforms all the competitors and in *Italic* otherwise.

Dataset	***MoE-OPM***					**Competitors**		
	kTop					*1-LR*	*FOX* [9]	*GLRM* [15]
	2	4	6	8	ALL			
bpic2011_1	*0.97*	*0.95*	*0.96*	**0.98**	*0.88*	0.94	0.97	0.92
bpic2011_2	*0.85*	*0.84*	*0.86*	**0.97**	*0.87*	0.94	0.92	**0.97**
bpic2011_3	*0.95*	**0.98**	*0.96*	**0.98**	*0.91*	0.97	**0.98**	**0.98**
bpic2011_4	**0.96**	**0.98**	**0.98**	**0.98**	*0.81*	0.68	0.89	0.81
sepsis_1	**0.87**	**0.86**	**0.89**	**0.88**	**0.62**	0.47	0.58	0.47
sepsis_2	*0.54*	**0.80**	**0.78**	**0.88**	*0.71*	0.76	0.73	0.73
sepsis_3	**0.84**	**0.81**	**0.86**	**0.82**	*0.69*	0.70	0.68	0.65

This experimental analysis encompasses the outcome-predictor discovery methods *FOX* [9] and *GLRM* [15]. For these methods we here report the results appeared in the respective publications, remarking that they were obtained using exactly the same pre-processed data and test procedure as described above. As a further term of comparison, we consider a baseline method, denoted as *1-LR*, that simply returns one LR model — we simulated this method by running Algorithm *MoE-OPM Discovery* with $m = 1$ and $kTop = ALL$. For the sake of fair comparison, we are not considering the LLM models discovered in [15], as they were obtained by fixing no bound on the number and size (i.e. the number of non-zero feature coefficients) of LR models appearing in leaf.

4.1 Prediction accuracy analysis

Table 1 reports the AUC scores obtained by the 6-expert *MoE-OPM* models. Notably, *MoE-OPM Discovery* consistently outperforms the baseline *1-LR* in all $kTop$ configurations when tested on several datasets, specifically bpic2011_1, bpic2011_4, sepsis_1, and bpic2011_4. For the remaining datasets, there is always at least one $kTop$ configuration where *MoE-OPM Discovery* performs better than the baseline. In particular, on average, *MoE-OPM Discovery* achieves an AUC improvement of more than 20% over *1-LR*, with peaks reaching beyond 80% (e.g., in the case of sepsis_1 with $kTop \in \{6, 8\}$). This confirms that training multiple local LR outcome predictors usually improves the performance of training a single LR model on all the data features (as done by *1-LR*). In addition, *MoE-OPM Discovery* always surpasses state-of-the-art methods like FOX and GLRM on all the datasets but bpic2012_2 and bpic2012_3, where *MoE-OPM Discovery* and GLRM perform on par.

Notably, *MoE-OPM Discovery* obtains outstanding achievements with different settings of $kTop \neq ALL$, showing that this hyper-parameter helps improve model accuracy (besides reducing model complexity). Precisely, the advantage of exploiting the feature-reduction capability of *MoE-OPM Discovery*, rather than making it just return a *MoE-OPM* trained on all the input features ($kTop = ALL$), is neat on all the

datasets but bpic2012_2. As a general behavior, when using very few features (namely, $kTop = 2,4$), *MoE-OPM Discovery* tends to perform worse than when trained with a slightly more extensive feature set (namely, $kTop = 6,8$) due to information loss. However, on dataset bpic2011_1 (resp. bpic2011_4), even when using just two (resp. four) features, *MoE-OPM Discovery* gets outstanding AUC scores. In our opinion, such ability to automatically focus on a few relevant features is precious when seeking interpretable models and explainable predictions.

The fact that the proposed approach managed to achieve compelling AUC performances when using less than 9 input features per sub-model (i.e. when setting $kTop \leq 8$), is an important finding, as far as concerns the interpretability of the models returned and their suitability for prediction explanation. A brief discussion on the description and explanation complexities of the models discovered by both this approach and FOX are provided in the following subsection. Such a discussion is not conducted for the LLM and GLRM models discovered from the datasets considered in this work, since we have not found information on this regard in the current version of [15]. Anyway, we note that all LLM models discovered from (a dataset derived from) logs *BPIC 2011* and *Sepsis* includes at least 290 and 84 features per LR model, respectively, and may well contain hundreds of LR models (see [15] for the case of bpic2011_2). This makes it hard for process stakeholders/analysts to understand such a model and assess its trustworthiness.

4.2 Complexity/interpretability and efficiency of the discovered *OPM*s

Generally, the lower the description complexity of a prediction model, the easier to interpret (and to trust/debug) the model and explain its predictions. In the cases of *MoE-OPM Discovery* and of baseline *1-LR*, description complexity is computed by counting the non-zero parameters appearing in the respective LR (sub-)models (i.e., in the experts and gate of a *MoE-OPM* vs. in the singleton LR).

In contrast, a FOX model's complexity can be quantified as the total number of conditions appearing (as conjuncts) in the respective fuzzy rules. Based on [9] if applying FOX to a filtered version of dataset bpic2011_1 (resp., bpic2011_1, ..., bpic2011_4, sepsis_1, ..., sepsis_3) containing only the top 4 (resp., 7, 6, 2, 5, 4 and 6 data features), a model consisting of 81 (resp., 2187, 729, 9, 243, 81, and 729) fuzzy rules is found. This means that the complexities of these models (where all the rules have as many terms as the selected features) range from 18 to 15309.

To give an idea of the better accuracy-interpretability trade off achieved by algorithm *MoE-OPM Discovery* on all the datasets (excluding bpic11_3), for each of them let us focus on the minimal value of $kTop$ that allowed the algorithm to match or surpass the AUC achievements of all the competitors, i.e. $kTop = 8$ on bpic2011_1, bpic2011_2, $kTop = 4$ on bpic2011_3 and sepsis_2, and $kTop = 2$ on all the remaining datasets. Since the parameters (including bias vectors) in these *MoE-OPM*s are $2 \times (kTop + 1) \times m$, where $m = 6$ is the desired number of experts, their complexities are bounded by 60, 108, and 36, respectively. This lead us to believe that *MoE-OPM Discovery* can find more compact models than FOX.

On the other hand, when using a *MoE-OPM* to predict a novel trace x, a faithful local explanation of the prediction can be obtained by looking at the $kTop$ feature weights of the LR expert that has been exploited to make the prediction. In the case of a FOX

Expert	0	1	2	3	4	5
Activity_CRP	0,00	0,00	0,00	-0,44	0,00	0,35
Activity_IV Liquid	0,85	0,00	-0,47	0,00	0,00	0,00
Activity_Release B	-0,32	-0,40	0,00	0,00	0,00	0,00
DiagnosticArtAstrup	0,00	0,40	0,00	0,00	0,00	0,00
DiagnosticSputum	0,00	0,00	0,00	-0,36	0,00	0,00
Hypotensie	0,00	0,00	0,00	0,00	-0,38	0,00
Oligurie	0,00	0,00	0,00	0,00	0,00	-0,37
SIRSCritHeartRate	0,00	0,00	0,00	0,00	-0,44	0,00
SIRSCritTemperature	0,00	0,00	0,00	0,00	-0,41	0,55
mean_hour	-0,33	0,00	0,00	0,00	0,00	0,00
mean_timesincelastevent	0,00	0,00	0,00	0,00	-0,36	0,00
org:group_A	0,00	0,00	-0,49	0,00	0,00	0,00
org:group_G	0,00	-0,34	0,00	0,00	0,00	0,00
org:group_H	0,00	0,00	0,00	0,52	0,00	0,00
org:group_O	0,00	0,00	0,00	0,00	0,00	0,36
org:group_T	0,00	0,00	-0,46	0,00	0,00	0,00
org:group_V	0,00	0,00	-0,48	0,00	0,00	0,00
org:group_W	0,00	0,34	0,00	0,00	0,00	0,00
org:group_other	-0,34	0,00	0,00	0,00	0,00	0,00
std_timesincemidnight	0,00	0,00	0,00	-0,33	0,00	0,00

Fig. 2: Example *MoE-OPM* discovered by *MoE-OPM Discovery* (with $kTop = 4$) from dataset `sepsis_3`: parameter weights of the six LR experts. The weights, linking each expert to one input feature, are shown in an orange-to-blue color scale based on their values —the lower (resp., higher) the value, the closer to orange (resp., blue).

model, though one could focus on a few top-fitting fuzzy rules (or just on the best-fitting one) to explain the prediction returned for x, several other rules may have impacted the prediction substantially. This descends from the fact in a FOX model the prediction for x is made by fusing (via weighted averaging) those of all of its fuzzy rules (or, at least, of those with non-zero fit) while our *MoE-OPM* uses only one of its experts to this end.

Notably, a *MoE-OPM* can make an outcome prediction in a speedy and compute-efficient way through a forward pass throughout two single-layer linear sub-nets — i.e., the gate and the chosen expert, featuring $m \times (k+1)$ and $m \times (k+1)$ parameters, respectively. This computation entails $(2 \times kTop + 1) \times (m+1)$ FLOPs (Floating Point Operations) —i.e., less than 120 FLOPs when fixing $m = 6$ and $kTop \leq 8$ as in the experiments described so far.

4.3 Qualitative results: an example of discovered *MoE-OPM*

Figure 2 shows the input features and associated weights that are employed by the six LR experts discovered when running algorithm *MoE-OPM Discovery* with $kTop = 4$ on dataset `sepsis_3`, for which the outcome-prediction task estimates the probability that an in-treatment patient will leave the hospital with the prevalent release type (namely 'Release A'). As a whole, only 20 of the 86 data features are used by the experts, but the experts use (specific subsets of) these features quite differently. In a sense, the experts have learned different input-output relationships for dealing with different process-outcome use cases.

For instance, Expert 0 emphasizes a positive impact of 'Activity_IV Liquid' (intravenous fluid treatment) on predicting class 1, and negative influence from 'Activ-

ity_Release B' (a specific discharge type), 'mean_hour' (certain times of day), and 'org:group_other' (specific hospital groups).

Expert 1 attributes instead notable positive influence (on predicting this class) from 'DiagnosticArtAstrup' (arterial blood gas measurement) and 'org:group_W' (a hospital staff team) and negative influence from 'Activity_Release B' and 'org:group_G' (another hospital group).

Expert 2 focuses on scenarios where 'Activity_IV Liquid' and certain hospital groups ('org:group_A', 'org:group_T', 'org:group_V') correlate negatively with a class-1 outcome. Analogous interpretations can be extracted from the remaining expert models, which also focus on specific activities and hospital groups, either positively or negatively correlated with the target outcome class.

Focusing on such a small number of feature importance scores, a domain expert can quickly inspect and assess the internal decision logic of the model and get simple, faithful explanations for its predictions. However, evaluating the actual practical relevance of such explanations (possibly through a user study) is left to future work.

5 Conclusion and future work

We proposed an approach to learning an MoE-like interpretable outcome prediction model consisting of multiple local LR-based expert *OPM* and an LR-based gate module that dynamically selects one expert to predict the outcome of a novel process instance, say x. The discovered model transparently shows the features of x that influenced the prediction result most and the choice of routing x to a specific expert while giving the user complete control over the size of the discovered *OPM*. Besides ensuring better interpretability, this feature (combined with the conditional computation scheme implemented by the gate) makes our approach appealing for green AI applications [12]) where compute-efficient ML solutions are required. Despite using a lossy encoding of log data, one-layer linear sub-nets, and a rough model pruning strategy, the proposed approach has achieved a good trade-off between outcome-prediction accuracy and model/explanation complexity on popular benchmark logs.

As to future work, we will investigate: (i) converting LR-like sub-models returned by our approach into logic rules, which some users may prefer to feature-attribution scores, (ii) tuning hyper-parameters $kTop$ and m automatically; (ii) leveraging prior knowledge and (iv) adapting our framework to predict violations to declarative models [4]. We also plan to expand the empirical study by including more datasets, scalability, and temporal stability analyses [18] and more discussion of qualitative results.

Acknowledgment. This work was partly supported by project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the EU-NGEU, and project PINPOINT, under program PRIN, funded by the Italian Ministry of University and Research (grant no. B27G22000160001).

References

1. Abdelsalam Ismail, A., et al.: Interpretable mixture of experts for structured data pp. arXiv–2206 (2022)

2. Bibal, A., et al.: Is attention explanation? An introduction to the debate. In: Proc. of 60th Meeting of the Association for Computational Linguistics (ACL'22). pp. 3889–3900 (2022)
3. Elkhawaga, G., M.Abu-Elkheir, Reichert, M.: Explainability of predictive process monitoring results: Can you see my data issues? Applied Sciences **12**(16), 8192 (2022)
4. Fionda, V., Guzzo, A.: Control-flow modeling with declare: Behavioral properties, computational complexity, and tools. IEEE Trans. on Knowledge and Data Engineering **32**(5), 898–911 (2020)
5. Galanti, R., et al.: Explainable predictive process monitoring. In: Proc. of 2nd Intl. Conf. on Process Mining (ICPM'20). pp. 1–8 (2020)
6. Jacobs, R.A., Jordan, M.I., Nowlan, S.J., Hinton, G.E.: Adaptive mixtures of local experts. Neural Computation **3**(1), 79–87 (1991)
7. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. IEEE Trans. on Services Computing **11**(6), 962–977 (2017)
8. Pasquadibisceglie, V., et al.: Orange: outcome-oriented predictive process monitoring based on image encoding and cnns. IEEE Access **8**, 184073–184086 (2020)
9. Pasquadibisceglie, V., Castellano, G., Appice, A., Malerba, D.: Fox: a neuro-fuzzy model for process outcome prediction and explanation. In: Proc. of 3rd Intl. Conf. on Process Mining (ICPM'21). pp. 112–119 (2021)
10. Rama-Maneiro, E., Vidal, J., Lama, M.: Deep learning for predictive business process monitoring: Review and benchmark. IEEE Trans. on Services Computing (2021)
11. Rizzi, W., Francescomarino, C.D., Maggi, F.M.: Explainability in predictive process monitoring: When understanding helps improving. In: Proc. of 18th Intl. Conf. on Business Process Management (BPM'20) (2020)
12. Salehi, S., Schmeink, A.: Data-centric green artificial intelligence: A survey. IEEE Trans. on Artificial Intelligence pp. 1–18 (2023). https://doi.org/10.1109/TAI.2023.3315272
13. Sindhgatta, R., Moreira, C., Ouyang, C., Barros, A.: Exploring interpretable predictive models for business processes. In: Proc. of 18th Intl. Conf. on Business Process Management (BPM'20). vol. 12168, pp. 257–272 (2020)
14. Slack, D., Hilgard, A., Singh, S., Lakkaraju, H.: Reliable post hoc explanations: Modeling uncertainty in explainability. Advances in Neural Information Processing Systems **34**, 9391–9404 (2021)
15. Stevens, A., Smedt, J.D.: Explainability in process outcome prediction: Guidelines to obtain interpretable and faithful models. arXiv:2203.16073 (2023)
16. Stevens, A., De Smedt, J., Peeperkorn, J.: Quantifying explainability in outcome-oriented predictive process monitoring. In: Process Mining Workshops. pp. 194–206 (2022)
17. Stierle, M., Weinzierl, S., Harl, M., Matzner, M.: A technique for determining relevance scores of process activities using graph-based neural networks. Decision Support Systems **144**, 113511 (2021)
18. Teinemaa, I., Dumas, M., Leontjeva, A., Maggi, F.M.: Temporal stability in predictive process monitoring. Data Mining and Knowledge Discovery **32**, 1306–1338 (2018)
19. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Trans. on Knowledge Discovery from Data **13**(2), 1–57 (2019)
20. Wickramanayake, B., et al.: Building interpretable models for business process prediction using shared and specialised attention mechanisms. Knowledge-Based Systems **248**, 108773 (2022)
21. Zhou, Y., Booth, S., Ribeiro, M.T., Shah, J.: Do feature attribution methods correctly attribute features? In: Proc. of AAAI Conf. on Artificial Intelligence (AAAI'22). pp. 9623–9633 (2022)