A Robust and Versatile Multi-View Learning Framework for the Detection of Deviant Business Process Instances

Alfredo Cuzzocrea, Francesco Folino, Massimo Guarascio, Luigi Pontieri

RT-ICAR-CS-16-01 **Gennaio 2016**

A Robust and Versatile Multi-View Learning Framework for the Detection of Deviant Business Process Instances

Alfredo Cuzzocrea[1], Francesco Folino[2], Massimo Guarascio[2], Luigi Pontieri[2]

[1] DIA Department, University of Trieste, Trieste, Italy
cuzzocrea@icar.cnr.it
[2] ICAR institute, National Research Council, Rende, Italy
{ffolino,guarascio,pontieri}@icar.cnr.it

Abstract. Increasing attention has been paid of late to the problem of detecting and explaining "deviant" process instances —i.e. instances diverging from normal/desired outcomes (e.g., frauds, faults, SLA violations)— based on log data. Current solutions discriminate between deviant and normal instances by combining the extraction of (sequence-based) behavioral patterns with standard classifier-induction methods. However, there is no general consensus on which kinds of patterns are the most suitable for such a task, while mixing multiple pattern families together will produce a cumbersome, redundant and sparse representation of log data that may well confuse the learner and lead to unreliable deviance-prediction models. We here propose an ensemble-learning approach to this process mining tasks, where multiple base learners are trained on different feature-based views of the given log (obtained each by using a distinguished family of patterns). A stacking procedure is then employed to combine the discovered base models into an overall deviance-oriented classifier, which exploits the predictions of the former models, hence implicitly reasoning on different structural features. The induction of such a meta-classifier is performed by using a (scalable and robust) probabilistic method. This way each prediction is equipped with some reliable measure of confidence, which can allow for prioritizing suspicious case, and make the analysis of estimated deviations efficient and feasible even in complex BPM scenarios. Beside possibly taking advantage of all non-structural data that are typically associated with the traces, the approach employs a resampling mechanism to deal with situations where the training set contains far less deviant traces than normal ones. The approach has been implemented as part of a comprehensive framework for detecting and analysing business process deviances, which is also meant to support the analyst in investigating new suspect deviances, and to provide some feedback on the real nature of new traces, which can help improve the discovered deviance-prediction models. A series of experiments on a real-life log proved the validity of the approach, which showed its capability of achieving compelling performances w.r.t. state-of-the-art methods.

Keywords: Business process intelligence, Classification, Deviation detection.

1 Introduction

Large amounts of log data are continuously gathered in many organizations during the execution of business processes. Such data are a precious source of information, which can support ex-post process analysis and auditing tasks, with the help of automated business intelligence techniques, like those developed in the field of Process Mining.

In particular, increasing attention has been paid to the problem of detecting and explaining "deviant" process instances in a process log, i.e. instances that diverge from normal or desirable outcomes (e.g. frauds and other security breaches, faults, SLA violations, non-compliance to regulatory rules). In fact, the occurrence of such a deviance often impacts negatively on the performances of a business process, and it may cause severe damages to an enterprise in terms of extra-costs (e.g., due to the application of penalties), missed earning opportunities, or even permanent loss of reputation.

Several approaches [21, 24, 3, 20, 25, 19] have been proposed, which all combine the extraction of (sequence-based) behavioral patterns with standard propositional classifier-induction methods, in order to induce a model for discriminating between deviant and normal instances, while using the discovered patterns as summarized behavior-oriented data features. However, there is no evidence and general consensus on which pattern family should be preferred when deriving a vector-space representation of the log traces for deviance mining purposes. This motivates the attempt of exploiting heterogenous representations mixing up different families of patterns. For example, a combination of sequence and alphabet patterns was used in [3]. The combination of individual-activity patterns with both sequential, alphabet and discriminative patterns was analyzed in [21], as a way to improve the accuracy of a model using the former kind of patterns only.

However, we are afraid that a number of key issues have not been addressed adequately, which arise in many real application scenarios:

I1 First of all, we believe that extracting a deviance-oriented classification model with the help of different kinds of behavioral patterns can improve the performances of models discovered by using a single kind of patterns[1]. However, the pattern generation phase may produce a very large number of structural patterns, and a sparse representation of the traces, which exposes the deviance-prediction models to the "curse of dimensionality" problem. On the other hand, mixing different pattern families into a feature-based encoding would lead to a redundant representation of the traces, featuring many mutually correlated features —consider, e.g., the fact that the presence of a tandem repeat *a,b* implies the presence of both activities *a* and *b* (viewed as "individual-activity" patterns). Such a combined view of the dataset, indeed, is likely to confuse most machine learning algorithms, and to lead to overfitting deviance detection models in the case of a relatively small number of training traces. The usage of feature selection/reduction methods can alleviate these problems, but it cannot solve them completely, while the greedy nature of these methods may even lead to some loss of relevant information.

[1] This claim is confirmed by the experimental analysis discussed in this work, showing the superiority of our multi-view learning approach with respect to current deviance detection solutions.

I2 In many real-life applications, verifying and auditing a presumably deviant case is a difficult and expensive task that requires long and careful investigations by domain experts. In order to make the analysis of estimated deviations efficient and feasible in complex BPM settings, each prediction should be equipped with some reliable measure of confidence, which can be used to emphasize and prioritize more suspicious process instances. In fact, we deem that such a capability can pave the way to a wider and more flexible adoption of deviance prediction tools within real BPM systems.

I3 The discovery of deviant instances is often carried out in scenarios where normal instances are much more than deviant ones. A dataset where one class largely overcome other one(s) in terms of cardinality is known in the literature as case of "class imbalance" [17]. Learning a classification model in such a situation is universally reckoned as a very challenging task for classic approaches, which tend to have unsatisfactory achievements over minority-class instances. In such a case, indeed, the typical optimization strategy followed by these approaches (meant to minimize the global number of misclassification errors or, equivalently, to maximize the overall accuracy of the model) is likely to yield a model that cannot recognize deviant instances adequately.

I4 In certain deviance detection scenarios, the very task of labelling training examples is time consuming and expensive. This makes it difficult to provide the learning algorithms with a sufficiently large collection of training data, as well as to keep the prediction models updated by exploiting new examples of both deviant and normal instances. In such a setting, it would be important to support the analyst in the labelling of new traces, by letting her/him focus on a small selected portion of them, in the spirit of active learning approaches [50, 40, 42, 43, 46–48].

Contribution In order to address the above issues and overcome the limitations of current solutions, we propose a novel approach, offering three main kinds of contribution:

C1 a multi-view ensemble-based method for inducing a deviance detection model out of historical log data;

C2 a comprehensive system architecture (which fully exploits and empower the above-mentioned induction method) supporting the detection and analysis of deviances in a Business Process Management scenario; and

C3 an empirical comparative analysis of our deviance detection approach with current solutions available in the field.

In a nutshell, the proposed induction method founds on the core idea of training multiple base learners on different feature-based views of the given log, produced each with the help of a distinguished family of patterns. More specifically, the method extracts an ensemble of classification models, induced (by applying several learning algorithms) from different propositional views, say $L'_1, \ldots, L'_n$, of the given log, say L. Each of these views represents a vector-space encoding of the traces stored in L, which combines both context information and a distinguished set of structural features capturing relevant behavioral patterns (as in [3]). Such a variegate multi-view collection of classifiers is made undergo a meta-learning scheme that eventually yields a higher-order classification model.

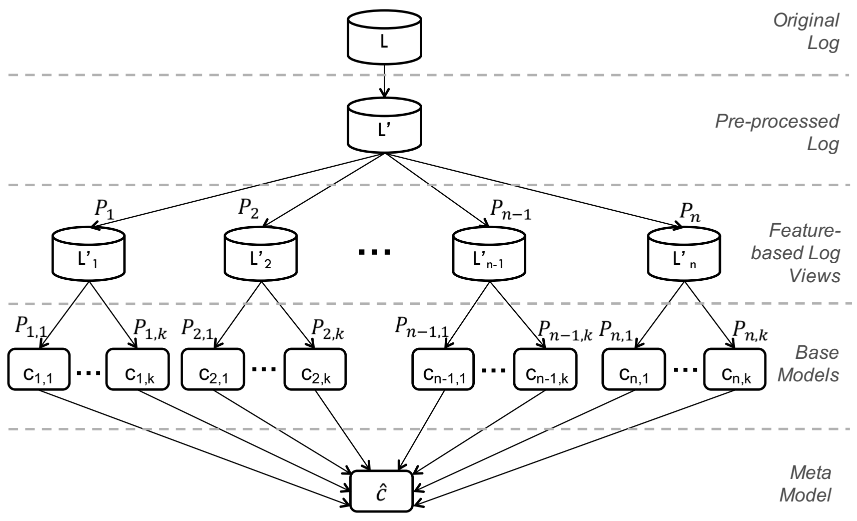

Fig. 1. Conceptual data-processing flow of the proposed approach: original data, transformed data, and discovered deviation-detection models.

A summarized pictorial representation of the main data processing steps that compose this ensemble learning method is reported in Figure 1. The figure presents, in particular, the relationship between the original log and different datasets derived from it through the application of pre-processing (specifically, resampling) and feature extraction techniques. Two layers of deviation-detection models are eventually discovered: *(i)* a collection of base models (learnt by applying *k* different classifier-induction methods to one of the *n* feature-based views derived from the log), and *(ii)* a meta model, which integrates the predictions of the base models into a “high-order” deviance forecast.

In order to cope with issue **I2**, the meta-learning task is accomplished by inducing a probabilistic classifier (from a view of the input log gathering the predictions of the discovered base models). This allows us to eventually assign any process instance an estimate of the probability that it really is a deviance. For the sake of scalability and robustness, we resort to the approach of *one-dependence estimators* [34] (specifically, we leverage the *AODE* algorithm [37]), which extends the popular Naïve Bayes classifier by relaxing the assumption of attribute independence (clearly not holding in our learning setting), without increasing the computation complexity of the training phase.

Notably, the final deviance-prediction model (combining the predictions of all base models) is capable of implicitly reasoning on heterogeneous kinds of structural features at a higher level of abstraction, without directly working with them all. This makes our approach fully address the tricky issue **I1** discussed above.

In order to deal with situations where deviant instances are far less than normal ones (issue **I3**), the learning method integrates a resampling mechanism, which helps attenuating the level of class imbalance by simply replicating the examples of the minority class.

The proposed learning method has been implemented in a Java prototype system, playing as the core module of a comprehensive system architecture for the detection and analysis of business process deviances, which constitute the second contribution (**C2**) of our work. This architecture supports, in particular, two kinds of tasks, which complement the proposed learning method and pave the way to a full exploitation and improvement of the deviance-detection models discovered with the former: *(i)* "Deviance Detection and Investigation", which amounts to automatically label new traces with the help of the discovered models, and provide the analyst with tools for both analyzing suspected deviances and for possibly revising their assigned class labels; and *(ii)* "Deviance-Detection Model Improvement", which consists in enforcing the predictive power of a deviance-detection model, by making it take account for new traces, while allowing the analyst to validate (and possibly revise) the ones associated with low-probability predictions. The latter feature let us acquire a some feedback from domain experts (in a non-invasive way), and to partially address the issue **I4** discussed above.

The third contribution (**C3**) of our work descends from a series of experiments that were carried out on a public log concerning the clinical pathways of patients in a real hospital. The results of this empirical analysis demonstrated the validity of our approach, and its capability to achieve compelling results, compared with state-of-the-art methods for the detection of business process deviances.

Organization The rest of the paper is organized as follows. After a brief discussion of relevant related work (Section 2), some preliminary concepts, concerning the kind of Bayesian model that underly our meta-learning method, are presented in Section 3. Section 4 describes the structure of both log data and pattern-based log views, while Section 5 explains how to combine the predictions of multiple base learners into a single deviance probability score, by using a suitable Bayesian model. The overall learning problem addressed in this work is formally stated in Section 6, which also illustrates an algorithm that solves it. Section 7 illustrates the proposed system architecture for the detection and analysis of process deviances. An empirical analysis of the approach in a real-life scenario is discussed in Section 8, while a few concluding remarks are drawn in Section 9.

2 Related Work

Process deviance mining The term "deviance mining" [21] indicates a class of process mining algorithms meant to discriminate (and possibly explain) log traces featuring a deviant behavior w.r.t. the normal or legitimate one. As discussed in [21], current deviance mining solutions adopt two alternative kinds of approaches: *(i) model delta analysis*, and *(ii) sequence classification*.

The former kind of approach consists in applying a process discovery technique to both normal and deviant traces separately. The two resulting models are then compared manually with the aim of identifying distinctive patterns for both classes of traces.

By converse, the usage of sequence classification techniques [24, 3, 20, 19] enable for discriminating deviant traces in an automated way, by learning some kind of classification model (e.g. a decision tree), after labeling all log traces as either deviant or normal.

Since current approaches rely on standard classifier-induction algorithm, a critical point consists in defining a propositional encoding of the given traces, which can effectively capture relevant (and possibly distinguishing) behavioral patterns.

Three main classes of patterns have been used in the literature, in order to produce such a feature-based representation of each log trace, where each pattern is used as a distinguished attribute: *activity-based patterns*, *sequence-based patterns*, and *discriminative patterns*. Activity-based patterns [24] simply correspond to the different activities appearing in the log; for each trace and each activity, the latter is regarded as an attribute, whose value is usually set to the number of times it occurs in the trace.

More sophisticated sequence-based patterns were exploited in [3, 23], in order to capture the occurrence of typical execution schemes, prior to applying classic classification techniques (namely, *decision tree* and *association rule* classifiers). These patterns include *tandem repeats*, *maximal repeats*, *super-maximal repeats*, and *near super-maximal repeats* [4], as well as set-oriented abstractions of them, such as *alphabet tandem repeats*, and *alphabet maximal repeats* [5].

A special kind of "discriminative patterns" was introduced in [20] for recognizing failures among the traces of a software system, consisting each of a sequence of atomic event types. These patterns are (possibly non contiguous) frequent subsequences that are also correlated with the target class, based on Fisher score.

To date, there is no general consensus on which pattern family (e.g. tandem repeats, maximal repeats) must be used to derive a vector-space view of the log traces in a deviation-mining analysis. In fact, the empirical analysis of all the above-mentioned sequence classification methods presented in [21] showed that, although activity-based patterns and/or discriminative patterns sometimes perform better than others, all approaches have difficulty in discriminating and explaining certain deviances. The authors conjectured that some richer encoding of the traces (e.g. exploiting further kinds of patterns) could help better discriminate deviant cases. However, as noticed in [3], when increasing the number of patterns, the representation of the traces becomes rather cumbersome and sparse, and risks undermine the quality of the discovered models, as a consequence of the "curse of dimensionality" problem. This problem may exacerbate when mixing different kinds of patterns together (as proposed in [3]), since a redundant representation of the traces would be produced, which may well confuse the learner, even when some (greedy) feature selection/reduction mechanism is used.

Ensemble learning *Ensemble learning* methods [12] represent an effective flexible solution for improving classifier performances. In a nutshell, the core idea of ensemble approaches is to combine a set of models, all addressing the same mining task, in order to obtain a better composite global model. Specifically, when applied to a classification task, the prediction for any new instance is made by suitably combining the predictions made by all the models in the ensemble. When the learners are trained on different complementary views of the input dataset, the resulting meta-model will work on a space of high-level features (corresponding to the predictions made by the base models), which summarize the heterogeneous and large set of raw features that appear in those views.

Each ensemble method is defined in terms of three different elements: *(i)* the *base induction algorithms* (a.k.a. base learners), *(ii)* an *ensemble generation strategy*, specifying how different base models are to be built, by applying some base learner to a subset of the instances in the original training set, and *(iii)* a *combination strategy*, specifying how the different models in the ensemble are to be eventually integrated.

Three standard combination strategies were defined in literature: *bagging*, *boosting*, and *stacking*. Basically, in bagging schemes [8], different training datasets are used to learn the base learners, while the final prediction is performed by either uniformly averaging or voting over class labels. In the boosting strategy [16] instead, an iterative procedure that modifies the distribution of the training examples is exploited to focus on examples that are hard to classify correctly. The basic idea is to train, at each round, a new model meant to compensate the errors made by earlier models. Unlike bagging, boosting assigns weights to each training example, and the prediction results from a weighted average approach. used to estimate the best weight function to average base learners' predictions. Stacking [28] (a.k.a. "stacked generalization") adds a further meta-learning level on the top of the base learners in order to combine their predictions, where a new model is trained to make a final prediction, which uses all the predictions made by the base learners as additional features of the data instances. Stacking typically yields performance better than any single one of the trained models, and than Bayesian model-averaging [10].

We pinpoint that the usage of ensemble learning techniques for the discovery of deviance detection models is novel in itself. Moreover, to the best of our knowledge, the combination of such an approach with the extraction of multiple views of the log (based on different sets of behavioral patterns) has never been explored in the literature.

Other multi-view learning approaches Various multi-view learning approaches exist in the literature that do not follow an ensemble-based scheme. Many of them are connected with the problem of training a predictive model on both labelled and unlabelled data, according to a semi-supervised learning setting. In general, differently from standard (single-view) learning methods, these approaches share the idea of training distinct classifiers out of different view of the data, and then improving all the classifiers by trying to maximize the degree of their mutual agreement. A pioneering approach of this kind, named *co-training* [2], takes two distinct views of the data (assumed both to contain "sufficient" information for the prediction task) as input, and alternatively induces a classifier from one of the views (while exploiting predictions made for unlabelled instances by the classifier induced from the other view). Several solutions extending this basic co-training scheme have been appearing in recent years. These include, for instance, the EM-based method of [49], the approach defined in [51], performing a sort of combinative label propagation. Notably, the combination of co-training with active learning was explored in [46–48].

Unfortunately, the performances of all the multi-view learning approaches described above strongly depend on the (conditional) independence assumption that all data views are conditionally independent of one another given the label. This assumption does not hold in the problem setting considered in this work, where the views are produced by mapping the given log traces onto different families of structural patterns, and hence tend to exhibit a high degree of redundancy and of inter-dependence. This is the main reason why we did not adopt an existing co-training approach, and rather resorted to an ad-hoc ensemble-learning strategy.

3 Preliminaries: SPODE-based Bayesian Classifiers

As discussed before, in a deviance mining setting it is important to quantify how much confident we are in the fact that a trace really is a deviance. In order to address such an issue in a scalable way we resort to a simple extension of the popular Naïve Bayes method, named *AODE*, which relaxes the strict assumption of attribute independence (clearly not holding in our learning setting) of the latter. Before presenting the AODE method in details, let us introduce some preliminary notions on Bayesian classifiers, and on the Naïve Bayes method. *Bayesian classifiers* (e.g., Naïve Bayes and more sophisticated Bayesian Networks) combine a priori knowledge of the classes with new evidence gathered from data by virtue of Bayes's theorem.

Naïve Bayes Let us consider an instance space consisting of $m+1$ nominal attributes $X_1,\ldots,X_m,Y$, such that $m \in \mathbb{N}^+$, with Y encoding class labels (known only for training examples). Moreover, for any attribute $Z \in \{X_1,\ldots,X_m,Y\}$, let $dom(Z)$ be the associated domain, i.e. the set of all possible value the attribute may take. In particular, for the class attribute Y, let us assume that $dom(Y) = \{c_1,\ldots,c_k\}$. Given an instance $\mathbf{x} = \langle x_1,\ldots,x_m\rangle$ of m attribute values x_i each observed for an attribute variable X_i, the classification task consists in predicting the class label $y \in dom(Y)$ for it on the base of a training sample of already classified objects.

Specifically, a bayesian classifier computes $P(y|\mathbf{x})$ for each $y \in dom(Y)$, and assigns to $\mathbf{x}$ the label y achieving the highest probability, i.e., that y for which $argmax_{y\in dom(Y)}$ $(P(y|\mathbf{x}))$.

From the definition of conditional probability, we have that $P(y|\mathbf{x}) = P(y,\mathbf{x})/P(\mathbf{x})$ and since $P(\mathbf{x})$ is invariant w.r.t. class labels, it results that: $argmax_{y\in dom(Y)}\ (P(y|\mathbf{x})) = argmax_{y\in dom(Y)}\ (P(y,\mathbf{x}))$ where the latter is estimated in place of the former. Since the Bayes' rule states that $P(y,\mathbf{x}) = P(y)\cdot P(\mathbf{x}|y)$, the classification problem for a new instance $\mathbf{x}$ turns into finding that $y \in dom(Y)$ such that: $argmax_{y\in dom(Y)}\ (P(y)\cdot P(\mathbf{x}|y))$.

Moreover, since $P(\mathbf{x}) = \sum_{y\in dom(Y)} P(y,\mathbf{x})$, one can estimate every class membership probability as $P(y|\mathbf{x}) \approx P(y,\mathbf{x})/\sum_{y'\in dom(Y)} P(y',\mathbf{x}) = P(y,\mathbf{x})/(\sum_{y'\in dom(Y)} P(y')\cdot P(\mathbf{x}|y'))$. However, if the prior probability $P(y)$ can be easily estimated from the sample frequencies in the training set, it remains the problem to estimate the conditional probability $P(\mathbf{x}|y)$.

The way $P(\mathbf{x}|y)$ is computed depends on the bayesian approach employed. For instance, Naïve Bayes (NB) classifiers solve the problem by assuming that attributes are conditionally independent, given the class label y, i.e. $P(\mathbf{x}|y) = \prod_{i=1}^{m} P(x_i|y)$. This entails that, for each $y \in dom(Y)$, $P(y,\mathbf{x}) = P(y)\cdot\prod_{i=1}^{m} P(x_i|y)$, and, as a consequence, the instance $\mathbf{x}$ is predicting belonging to that class $y \in dom(Y)$ such that $argmax_{y\in dom(Y)}$ $(P(y)\cdot\prod_{i=1}^{m} P(x_i|y))$.

NB classifiers are widely recognized as simple, efficient and effective approach in a wide range of applications [31–33]. However, in certain application settings (like the one considered in this work), the attribute independence assumption is likely to lead to unsatisfactory accuracy results.

SPODE-based models and the AODE method To address this limitation, a lot of efforts were recently made in order to relax such an assumption, without renouncing to the simplicity and efficiency of NB classifiers. Among others, a particular attention has been recently payed to a variant of NBs named *one-dependence estimators* (ODEs) [34]. Essentially, ODEs generalize NBs by allowing that each attribute can depend on (at most) one other attribute in addition to the class. More specifically, a subclass of ODEs, namely SPODEs [35, 36], achieved a remarkable notoriety because of their capability to provide efficiency along with a high classification accuracy. SPODEs relax NB's attribute independence assumption by allowing all attributes to depend on one common attribute, the *super-parent*, in addition to the class. This means that a SPODE with super-parent X_p computes $P(y,\mathbf{x})$ as: $P(y,\mathbf{x}) = P(y,x_p)\cdot P(\mathbf{x}|y,x_p) = P(y,x_p)\cdot\prod_{i=1}^{m} P(x_i|y,x_p)$.

Typically, SPODEs are used with a sort of ensemble scheme, since it was proved that combining multiple SPODEs decrease the variance of a single SPODE classifier and improve the overall classification accuracy without disrupting time constraints [37, 38].

If instances in the training data set have m attributes, up to m different SPODEs – each using a different attribute as its super-parent – can be combined. Then, a SPODE ensemble can be obtained by linearly combining each single SPODE probability estimate. In this way, the probability $P(y,\mathbf{x})$ is computed as in the following:

$$P(y,\mathbf{x}) = \sum_{j=1}^{m} w_j \cdot P_j(y,\mathbf{x}) = \sum_{j=1}^{m} w_j \cdot P(y,x_j)\cdot\prod_{i=1}^{m} P(x_i|y,x_j)$$

where $P_j(y,\mathbf{x})$ is the estimate of jth SPODE for $\mathbf{x}$, and w_j is the weight assigned to it in the ensemble. Clearly, different ensemble strategies can be envisaged by changing how SPODE's models are selected, as well as their associated weights.

The *Averaged One-Dependence Estimators* (AODE) [37] method adopts a simple, yet effective, SPODE combination strategy, where only the super-parents appearing in the training dataset more than a minimum support threshold $h \in \mathbb{N}$ are considered[2], while weighting the estimates of the corresponding SPODEs in a uniform way. More precisely, according to this strategy, $P(y,\mathbf{x})$ is evaluated as follows:

$$P(y,\mathbf{x}) = \frac{\sum_{j\in\mathcal{S}} P(y,x_j)\cdot\prod_{i=1}^{m} P(x_i|y,x_j)}{|\mathcal{S}|} \tag{1}$$

where $\mathcal{S} = \{1 \leq j \leq m \wedge \sigma(x_j) \geq h\}$. As the denominator does not vary over the classes, when trying to estimate the class membership for a new instance $\mathbf{x}$, one can choose its class label y as shown below:

$$y = argmax_{y'\in dom(Y)}\left(\sum_{j\in\mathcal{S}} P(y',x_j)\cdot\prod_{i=1}^{m} P(x_i|y',x_j)\right) \tag{2}$$

In order to estimate, for each tuple $\mathbf{x} \in \mathbf{X}$ and each possible class label $y \in dom(Y)$, the probability $P(y|\mathbf{x})$, one can simply normalize the numerator in Eq. 1 across all classes as follows:

[2] For the sake of conciseness, hereinafter it is simply assumed $h = 1$. This choice was also made in all the experiments described in Section 8, even though different settings of h might have ensured better deviance prediction results. Indeed, we believe that the problem of optimally setting this parameter, and (more generally) of optimally selecting and weighting different SPODEs is beyond the scope of this work.

$$P(y|\mathbf{x}) = \frac{P(y,\mathbf{x})}{\sum_{y' \in Y} P(y',\mathbf{x})} = \frac{\sum_{j \in \mathcal{S}} P(y,x_j) \cdot \prod_{i=1}^{m} P(x_i|y,x_j)}{\sum_{y' \in Y} \sum_{j \in \mathcal{S}} P(y',x_j) \cdot \prod_{i=1}^{m} P(x_i|y',x_j)} \tag{3}$$

4 Log data and pattern-based log views

4.1 Log data and deviance labels

Whenever a process is enacted, a *trace* is recorded for each process instance (a.k.a "case"), which stores the sequence of *events* that happened along the execution of the instance. Let E and $\mathcal{T}$ denote the universes of all possible events and traces for the process under analysis, respectively.

For each trace $\tau \in \mathcal{T}$, let $\tau[i]$ be the i-th event of τ, for $i \in \{1,\dots,len(\tau)\}$, where $len(\tau)$ denotes the number of events in τ. Let also $prop(\tau)$ be a tuple storing a number of data properties associated with τ (e.g. case attributes or environmental variables). Moreover, let $PROP(\mathcal{T})$ be the relation consisting of all the data attributes associated with $\mathcal{T}$'s traces, so that $prop(\tau)$ for any $\tau \in \mathcal{T}$. Focusing on the list of data values stored in these tuples, we will sometimes regard $prop(\tau)$ and $PROP(\mathcal{T})$ as a vector and a vector space, respectively. Since certain learning algorithms works only (or better) over a space of discrete input features, let us consider a discretized version of $prop(\tau)$ (for any $\tau \in \mathcal{T}$) and of relation $PROP(\mathcal{T})$, and denote them by $\overline{prop}(\tau)$ and $\overline{PROP}(\mathcal{T})$, respectively. Such transformation can be accomplished with various supervised discretization techniques, such as the well-known Fayyad and Irani's MDL method [14] — which is indeed the one actually used in the current implementation of our approach. Clearly, in the case none of the data properties associated with the log traces is continuous, it is $\overline{prop}(\tau) = prop(\tau)$ and $\overline{PROP}(\mathcal{T}) = PROP(\mathcal{T})$.

Usually, any event $e \in E$ can be viewed as a tuple referring to a single process instance, and containing at least a timestamp, an activity identifier, and a resource identifier (i.e. the identifier of the agent that performed the activity).

In particular, let us assume that a function α exists that maps each event e to the corresponding activity identifier $\alpha(e)$. In general, such a function plays a key role in process mining settings, since it allows to bring the analyzed log to a suitable level of abstraction. Indeed, by abstracting each event e of a given trace into its corresponding activity label $\alpha(e)$, we can obtain an abstract representation of the whole trace. More precisely, for each trace $\tau \in \mathcal{T}$, let $\alpha(\tau)$ be the sequence obtained by replacing each event in τ with its abstract representation: $\alpha(\tau) = \langle \alpha(\tau[1]),\dots,\alpha(\tau[len(\tau)]) \rangle$.

A *log* L (over $\mathcal{T}$) is a multiset containing a finite number of traces from $\mathcal{T}$. We will denote as $events(L)$ the multiset of events that feature in (some trace of) L.

Let $\mu : \mathcal{T} \to \{0,1\}$ be a (unknown) function, allowing for discriminating all possible deviant cases from normal ones, that assigns a class label $\mu(\tau)$ to each $\tau \in L$, such that $\mu(\tau) = 1$ if τ is deviant, and $\mu(\tau) = 0$ otherwise.

Our ultimate aim is to obtain a *deviance detection model (DDM)*, i.e. a classification model estimating the (deviance-related) class label of any (unseen) process instance. Such a model can be represented as a function $\tilde{\mu} : \mathcal{T} \to \{0,1\}$ approximating μ over the whole trace universe. Discovering a DDM is an induction problem, where the log

L is the collection of training instances, and the function μ is known for each trace $\tau \in L$. In the following, we will only consider DDMs that work with a propositional representation of the traces, hence requiring a preliminary mapping of the latter onto some suitable space of features.

4.2 Pattern-based Log views

Once log traces have been turned into symbolic sequences, a wide range of sequence classification techniques could be applied in order to discover a deviance-oriented classification model. However, in general, this is not a valid solution for analyzing the logs produced by many business processes, due to the peculiar nature of these processes, where complex control-flow logics (e.g., loops, parallel execution, synchronization, exclusive choices, etc.) usually govern the execution of the activities, and impact on the possible sequences that can be stored in the log.

A consolidated approach to the analysis of process logs relies on extracting behavioral patterns that can capture the occurrence of typical execution schemes [3, 23]. These patterns can be then used as high-level behavioral features for producing a vector-space representation of the original traces, where each trace is converted into a tuple registering a sort of correlation between the trace and one of the discovered patterns. At this point, whatever standard classification algorithm (such as the *decision tree* and *association rule* classifiers used in [3, 23]) can be exploited to extract a deviance-detection model $\tilde{\mu}$ out of this propositional view of the log.

Various kinds patterns have been used in the literature to capture common subsequences of activities that recur in a log (within a single trace or across multiple ones). These patterns can be grouped into four main *pattern families*: *individual activities* [24], *sequence patterns* (including tandem repeats, maximal repeats, super-maximal repeats, and near super-maximal repeats) [4], *alphabet patterns* (including alphabet tandem repeats, and alphabet maximal repeats) [5], and *discriminative patterns* [20]. Individual activities are enough when the occurrence of a particular activity in a trace really helps identify deviant cases. However, reckoning deviant behaviors may require the usage of more complex patterns, providing a hint for the control-flow structure of the process. In these cases both sequence and alphabet patterns can be exploited. In particular, the latter kind of patterns is derived from sequence ones by simply relaxing the ordering of events (in order to unify different interleaving of parallel activities). Discriminative patterns represent frequent (possibly non contiguous) subsequences of activities having a discriminative power (measured via *Fisher score* in [20]) w.r.t. the target class.

Given a set P of patterns of the kinds described above, a vector-space representation of each trace τ can be built by projecting τ onto the space of the patterns in P. As a result, a propositional representation of τ is obtained that summarizes the sequence of events appearing in τ. Such representation of the structure of a trace (i.e. of its associated sequence of events) can be extended with non-structural data, when learning a DDM via standard classifier-induction methods. More formally:

Definition 1 ***(f-View).*** Let $\tau \in \mathcal{T}$ be a trace, α the (event abstraction) function mapping each event to the respective process activity, and $P = [p_1, \dots, p_q]$ be a list of (behavioral) patterns defined over the activity labels produced by α. Then, the *feature-based view* (*f-View*) of τ w.r.t. P, denoted by *f-View*(τ, P), is a tuple in $\mathbb{R}^q$ such that:

$f\text{-}View(\tau, P) = prop(\tau) \oplus \langle val(\tau, p_1), \ldots, val(\tau, p_q) \rangle$, where $\oplus$ stands for tuple concatenation and $val(\tau, p_i)$, for $i \in \{1, \ldots, q\}$, is some function for computing the value that the feature (corresponding to the pattern) p_i takes on τ. $\square$

Function $val(\tau, p_i)$ can be defined in different ways, depending on the application context. In general, two common criteria are used by current approaches to quantify the correlation between a trace τ and a pattern p_i: counting the number of times that p_i occurs in (the abstract representation of) τ, or simply registering the presence of p_i in (or the absence of p_i from) τ by way of a boolean flag. In the current implementation of our approach, the former option is used for individual-activity patterns, and the latter for any other kinds of patterns. As an example, let a and b be two activity labels produced by α on the given log, say L, and let *a,b* a tandem-repeat pattern extracted from L. Then, in our vectorial representation of any L's trace, say τ, a and b are considered as two boolean attributes (indicating whether τ contains a and b, respectively), while the pattern a, b gives rise to an integer attribute (storing how many times a and b occur one after the other within τ).

The vector-space representation of log traces defined above can be given as input to any standard classifier-induction method, in order to eventually discover a deviance-detection model (DDM). Indeed, any given labeled log L (i.e. a log where each instance is labelled as normal or deviant) can be turned into a propositional training set, denoted by $f\text{-}View(L, P)$, where each tuple encodes both the structure and the data associated with a trace of L, in addition to its class label. More precisely, $f\text{-}View(L, P)$ is a multiset that contains, for each trace $\tau \in L$, a tuple $f\text{-}View(\tau, P) \oplus \langle \mu(\tau) \rangle$ with the same multiplicity of τ. Clearly, the last value in each of the tuples appearing in $f\text{-}View(L, P)$ is the (known) class label of the corresponding trace, which represents the target of prediction; all the remaining values will play as input/descriptive features, which the learner can use to predict the class of any (possibly novel) trace.

5 Probabilistically Combining Different base DDMs

Any single pattern-based log view, like the one described in the previous section, could be used in its own to discover a deviance detection model. By converse, we want to effectively and efficiently exploit information provided by a collection of such views, by devising an ensemble learning approach that combines the prediction of single-view models (trained each on one of these views).

However, combining these models is a tricky task, mainly because the different data views used to train the models are not orthogonal to (and independent of) one another. Indeed, the different pattern-based log views try to summarize the behavior of the input log traces (essentially regarded as sequences of activity identifiers) by way of different kinds of structural patterns. Consequently, the resulting base models —yet enjoying a certain degree of variety and the potentiality of collectively discriminating deviance-related behaviors better than each single model in the ensemble — are likely to exhibit a certain level of mutual dependence. This discourages the usage of simple combination strategies, such as the popular one based on majority voting. In order to devise a more effective and robust scheme for combining the prediction of multiple (and non-independent) base DDMs, we formulate a second-level inductive (meta-) learning task.

This task consists in training a probabilistic (AODE-based) classifier over a propositional view of the given log, where the predictions made by the base learners for each trace of the log are used as high-level features of the trace —according to the strategy of stacking approaches [28]— in addition to intrinsic data properties of the trace. For the sake of simplicity and scalability, we consider the original formulation of the AODE model (as reported in Section 3), where all input attributes of the training tuples are discrete. Consequently, we assume that every numeric attribute appearing in the space $PROP(\mathcal{T})$ of tuples' properties is suitably transformed into a discretized space $\overline{PROP}(\mathcal{T})$.

A formal description of the stacked representation of log traces is reported next.

Definition 2 (***s-View***)**.** *Let $\tau \in \mathcal{T}$ be a trace, $CL = [c_1, \ldots, c_k]$ be a list of DDMs, and $PL = [P_1, \ldots, P_k]$ be a list of pattern lists, such that P_i is the specific list of patterns used to train model c_i, for $i \in \{1, \ldots, k\}$. Let us represent each DDM as a function $c_i : PROP(\mathcal{T}) \times \mathbb{R}^{|P_i|} \rightarrow \{0,1\}$ that maps the feature-based representation* f-View(τ) *of any trace $\tau \in \mathcal{T}$ to a class label —where $|P_i|$ stands for the number of patterns in the list P_i. Then, the* stacking-oriented view *(*s-View*) of τ w.r.t. CL and PL, denoted by* s-View(τ, CL, PL)*, is a tuple in $\overline{PROP}(\mathcal{T}) \times \{0,1\}^k$ such that:* s-View$(\tau, CL, PL) = \overline{prop}(\tau) \oplus \langle c_1(\text{f-View}(\tau, P_1)), \ldots, c_k(\text{f-View}(\tau, P_k)) \rangle$*, where $\oplus$ denotes tuple concatenation and $c_i(\text{f-View}(\tau))$ is the boolean label assigned by model c_i to (the feature-based representation of) τ, for $i \in \{1, \ldots, k\}$.* □

Given a log L, a list CL of DDMs and a list PL of associated pattern lists, a combined DDM can be learnt by applying another classifier-induction method to a propositional view of L. This view, denoted as *s-View*(L, CL, PL), is a multiset containing, for each trace $\tau \in L$, a tuple $sView(\tau, CL, PL) \oplus \langle \mu(\tau) \rangle$ with the same multiplicity of τ. The last value of each tuple in *s-View*(L, CL, PL) plays as the (deviance-oriented) class label, while the remaining values are used as input/descriptive features to predict the class.

A probabilistic meta-model: AODE combiner As discussed before, our approach to combining multiple base DMMs relies on exploiting an AODE classifier as a flexible and scalable meta-learning method for extracting a probabilistic DDM out of a stacked log view of the form specified in Def. 5.1

Definition 3 (***AODE Combiner***)**.** *Let $\boldsymbol{X} = \overline{PROP}(\mathcal{T}) \times \{0,1\}^k$ be the space of all the input attributes in the stacked view* s-View(L, CL, PL)*, and Y be the class attribute, storing the value of the binary[3] target function μ. An* AODE combiner *for $\boldsymbol{X}$ is a probabilistic (meta) classifier $\hat{c} : \boldsymbol{X} \rightarrow [0,1]$ that maps any "stacked" tuple $x \in \boldsymbol{X}$ to an estimate $\hat{c}(\boldsymbol{x})$ of $\boldsymbol{x}$'s deviance probability (i.e. $\hat{c}(\boldsymbol{x}) \approx P(Y = 1 | \boldsymbol{x})$) according to Eq. 3. The estimate is computed on the basis of the following counters, derived from* s-View(L, CL, PL) *for all $i, j \in \{1, \ldots, m\}$, $x_i \in dom(X_i)$, and $x_j \in dom(X_j)$:*

- *$D_j(x_j)$ (resp., $N_j(x_j)$) is the number of deviant (resp., normal) tuples in* s-View(L, CL, PL) *that feature the value x_j,*

[3] Clearly, in our setting $dom(Y) = \{0,1\}$, with label 1 corresponding to deviances and label 0 to normal cases.

– *$D_{ji}(x_j, x_i)$ (resp., $N_{ji}(x_j, x_i)$) is the number of deviant (resp., normal) tuples in* s-View(L, CL, PL) *that feature both values x_j and x_i,*
– *K is the number of tuples in* s-View(L, CL, PL) *(all having a known value of the deviance label),*
– *K_j is the number of tuples in* s-View(L, CL, PL) *for which the value of attribute X_j is known,*
– *K_{ji} is the number of tuples in* s-View(L, CL, PL) *for which the values of both attributes X_i and X_j are known.* □

Notice that the counters above are sufficient to estimate every base probability of the form $P(y, x_j)$ and $P(y, x_j, x_i)$ —which, in their turn, can be exploited to estimate every probability of the form $P(x_i|y, x_j)$— for each value x_j and x_i of the attributes X_j and X_i (with the former playing as super-parent), for $i, j \in \{1, \ldots, m\}$ and $y \in \{0, 1\}$. For the sake of robustness, the computation of all these base probabilities is carried out according to the *Laplace estimation* method [37], as specified in what follows: $P(Y=1, x_j) = \frac{D_j(x_j)+1}{K_j+2\cdot n_j}$; $P(Y=0, x_j, x_i) = \frac{D_{ji}(x_j, x_i)+1}{K_{ji}+2\cdot n_i \cdot n_j}$; $P(Y=0, x_j) = \frac{N_j(x_j)+1}{K_j+2\cdot n_j}$; $P(Y=0, x_j, x_i) = \frac{N_{ji}(x_j, x_i)+1}{K_{ji}+2\cdot n_i \cdot n_j}$; $P(Y=1|x_j, x_i) = P(Y=1, x_j, x_i)/P(x_j, x_i)$; $P(Y=0|x_j, x_i) = P(Y=0, x_j, x_i)/P(x_j, x_i)$.[4]

6 Discovering a High-Order DDM

The remainder of this section illustrates, in two separate subsections, the specific kind of prediction model that we want to eventually induce from a log, and our discovery approach in algorithmic form, respectively.

6.1 Target deviance-detection model (*HO-DDM*)

Our ensemble-based learning approach relies on inducing multiple classification models from different propositional views of the input log (based on different kinds of patterns). Specifically, each of these views is obtained by mapping the original traces onto a distinguished set of sequence-based behavioral patterns (according to the encoding scheme in Def. 4.1), extracted from the log while regarding each of its traces as a sequence of process activities. As discussed before, the different base models induced from all of these views are combined by learning an AODE-based meta-classifier out of a suitable stacked view. The ultimate result of our approach is a novel multi-view (and multi-level) kind of deviance detection model, named *High-Order Deviation Detection Model* (*HO-DDM*), which is formally defined next.

Definition 4 **(*HO-DDM*).** *Let L be a log over some proper trace universe T and event universe E, $\mathcal{F}$ be a set of pattern families, and $\mathcal{M}$ be a set of (base) classifier-induction*

[4] Clearly enough, $P(1, x_j)$, $P(0, x_j, x_i)$, $P(0, x_j)$, $P(0, x_j, x_i)$, $P(1|x_j, x_i)$, $P(0|x_j, x_i)$, $P(x_j, x_i)$ here stand for $P(Y=1, X_j=x_j)$, $P(Y=0, X_j=x_j, X_i=x_i)$, $P(Y=0, X_j=x_j)$, $P(Y=0, X_j=x_j, X_i=x_i)$, $P(Y=1|X_j=x_j, X_i=x_i)$, $P(X_j=x_j, X_i=x_i)$, $P(Y=0|X_j=x_j, X_i=x_i)$, $P(X_j=x_j, X_i=x_i)$, respectively.

method. Then a High-Order Deviance Detection Model *(*HO-DDM*) for L, w.r.t. $\mathcal{F}$ and $\mathcal{M}$ anis a triple of the form $H = \langle CL, PL, \hat{c} \rangle$, where:* (i) PL = $[P_1, P_2, \ldots, P_k]$ *is a list of pattern lists;* (ii) CL = $[c_1, c_2, \ldots, c_k]$ *is a list of base DDMs such that, for each $i \in \{1,\ldots,k\}$, the model $c_i : PROP(\mathcal{T}) \times \mathbb{R}^{|P_i|} \rightarrow \{0,1\}$ (learnt by using* f-View(L,P_i) *as training set) maps the propositional representation* f-View(τ,P_i) *of any trace $\tau \in \mathcal{T}$ to a class label in $\{0,1\}$; and* (iii) $\hat{c} : \overline{PROP}(\mathcal{T}) \times \{0,1\}^k \rightarrow [0,1]$ *is an AODE combiner for the space $PROP(\mathcal{T}) \times \mathbb{R}^{|P_i|}$ (of all the input attributes in the "stacked" view* s-View$(\mathcal{T}, CL, PL)$*) such that the base counters of $\hat{c}$ reflect the data distribution of the latter. For any trace $\tau \in \mathcal{T}$, model H provides an estimate, denoted by $devProb_H(\tau)$, for the probability that τ be a deviance (i.e. $P(\mu(\tau) = 1)$, which is computed as follows: $devProb_H(\tau) = \hat{c}(\overline{prop}(\tau) \oplus \langle c_1(\text{f-View}(\tau,P_1)), \ldots, c_k(\text{f-View}(\tau,P_k)) \rangle)$.* □

This model encodes a sort of high-order deviance detector, where the predictions of all the discovered base classifiers are combined with the help of a second-level classifier (discovered through a stacking-based meta-learning strategy). Clearly, the model is meant to be possibly predict whether any unseen trace τ is deviant or not. To this purpose, each base model c_i in the ensemble is applied to the vector-space representation *f-View*(τ, P_i) of τ produced according to the list P_i of patterns c_i is associated with, as specified in Def. 4.1. The predictions made by all the base models in *CL* are then combined into a single probabilistic prediction by the meta-model $\hat{c}$. The latter model takes as input a propositional view of τ mixing (a discretized version of) the original data properties of τ (i.e. $\overline{prop}(\tau)$) with the class labels assigned to τ by the base models.

The deviance probability score $devProb_H(\tau)$, returned by the model for any trace τ, can be used to eventually decide its label $\mu(\tau)$. A straightforward approach consists in fixing $\mu(\tau) = 1$ iff $devProb_H(\tau) > 0.5$. However, for the sake of flexibility, one can leave to the analyst the freedom of setting a suitable probability threshold $\gamma \in (0,1)$, so that $\mu(\tau) = 1$ iff $devProb_H(\tau) > \gamma$. The possibility to tune such a decision boundary (and an associated uncertainty margin) is discussed in more details in the deviance detection framework presented in Section 7.

6.2 Algorithm *HO-DDM-mine*

Our approach to the discovery of a *HO-DDM* from a given log *L* is summarized in Figure 2 in the form of an algorithm, named *HO-DDM-mine*.

The algorithm follows a two-phase computation strategy. In the first phase (Steps 2-10), a number of base classifiers are discovered by applying a given set (specified via the input parameter $\mathcal{M}$) of inductive learning methods to different views of *L*, obtained each by projecting the traces in *L* onto a different space of features. In the second phase (Steps 11-12), all of these base classifiers are combined into a single DDM, based on a meta-learning (stacking) procedure.

In more detail, the different views of *L* are produced according to a given set $\mathcal{F}$ of pattern families (specified by the analyst as one of the input parameters of the algorithm): for each pattern family, a list *P* of relevant patterns of that family are extracted from the log *L*, by using function `minePatterns` (Step 3). The second parameter of the function is right the reference family of patterns, while the third is the maximum number of patterns that can be generated for each family. In the current implementation of

```
Input:   A log $L$, over event universe $E$ and trace universe $\mathcal{T}$;
Params:  A set $\mathcal{F} = \{F_1, \ldots, F_n\}$ of pattern families, $resampleMode \in \{$NONE,OS$\}$,
         $resampleFactor \in \mathbb{N}$, a set $\mathcal{M} = \{M_1, \ldots, M_n\}$ of classifier-induction methods,
         and the maximal number $q$ of patterns per family;
Output:  An HO-DDM for $L$ w.r.t. $\mathcal{F}$ and $\mathcal{M}$;
 1.  $CL := [\,]$; $PL := [\,]$;  // initialize both lists CL and PL as empty
 2.  for $i = 1, \ldots, |\mathcal{F}|$ do
 3.      $P :=$ minePatterns$(L, F_i, q)$;  // extract a list P of patterns of family $F_i$
 4.      Compute $TS = f\text{-}View(L, P)$;  // cf. Def. 4.1
 5.      if $sMode =$ OS then reBalance$(TS, resampleFactor)$;
 6.      for $j = 1, .., |\mathcal{M}|$ do
 7.          Induce a DSM $c$ by applying method $M_j$ to $TS$;
 8.          append$(CL, c)$; append$(PL, P)$;
 9.      end
10.  end
11.  Compute $SS = s\text{-}View(L, CL, PL)$;  // cf. Def. 5.1
12.  $\hat{c} :=$ MineCombiner$(SS)$;
13.  return $\langle CL, PL, \hat{c} \rangle$.

Function MineCombiner ($SS$: propositional dataset): an AODE combiner;
 F1. Let $\hat{c}$ an "empty" AODE model such that all its internal counter are set to 0;
 F2. DiscretizeNumericAttributes($SS$);
 F3. UpdateCombinerCounts($\hat{c}$,$SS$);
 F4. return $\hat{c}$;
```

Fig. 2. Algorithm *HO-DDM-mine*

the approach, the selection of the q-top patterns is based on their frequency: the patterns with the highest q values of support in the log are kept, and returned as output.

Based on the list P of patterns extracted, a propositional view TS of L is produced, where each trace τ of L is turned into a tuple $f\text{-}View(\tau, P)$ —mixing both the data properties of τ and its representation over the space of P's patterns (Step 4), as described in Def. 4.1— and labelled with its associated class (i.e. either deviant or normal).

Before the classifier-induction methods specified in the input list $\mathcal{M}$ are applied to TS, in order to generate different base classifiers (Steps 6-9), the log can be further preprocessed in a way that reduces the imbalance of the two classes (namely, deviant instances vs. normal ones). Specifically, when the analyst decides to set the input parameter *sMode* to `OS`, an oversampling procedure is applied to TS with the help of function `reBalance`, which alters the classes' distribution by simply repeating each deviant trace in the training log a number *resampleFactor* of times (Step 5). No rebalancing is done instead when $sMode =$ `NONE`.

Once all base classifiers have been induced, and stored in the list CL, they are combined into a single overall meta-classifier according to a stacking strategy. To this purpose, first an *s-View SS* of L is computed (Step 11) according to Def. 5.1; then an AODE-based meta-classifier $\hat{c}$ is induced from SS, by way of function `mineCombiner`, described in more details later on.

The last step of the algorithm simply combines the discovered base classifiers (with their associated pattern lists) and the meta classifier into an overall *HO-DDM*.

Function `mineCombiner` This function is meant to construct an AODE-combiner model (as specified in Def. 5.2, using, as training set, a propositional dataset *SS* —as discussed before, this dataset represents a (labelled) stacked view of the log, where the input attributes include both the data properties of the given log's traces and the predictions made by a number of base DDMs, discovered from different pattern-based views of the traces. After initializing all the internal counters [5] of the AODE combiner, the `mineCombiner` possibly manipulates *SS* by using the auxiliary procedure `DiscretizeNumericAttributes`. More specifically, the latter procedure first computes a binning scheme for each numeric attribute, say X_q, that appears in *SS*, and then converts each of the values that X_q takes within (a tuple of) *SS* according to that scheme. Obviously, in case there are no numeric attributes in $\overline{PROP}(\mathcal{T})$, the transformation made by `DiscretizeNumericAttributes` is immaterial, and *SS* is left unchanged. Finally, all the internal counters of the AODE combiner are updated as to reflect the way the values of the attributes and of the classes actually appear in *SS*. The current implementation of the function leverage the *A1DE* plugin of the popular Weka library [15], while exploiting the MDL-based method proposed in [14] for the discretization of numeric attributes.

7 System Architecture for Process Deviances' Detection and Analysis

This section illustrates the conceptual architecture, depicted in Figure 7, of a system supporting the detection and analysis of deviant execution instances, produced by the enactment of real business processes. The system —which is still under development— hinges upon a full Java implementation of the learning approach described in the previous section, and it is meant to enable for fully exploiting and improving the deviance detection models discovered with our approach.

In more details, the system combines different functionalities, organized in four main blocks: *(a)* a HO-DDM *Learning* block, supporting the induction of *HO-DDM*s from the training set; *(b)* a HO-DDM *Evaluation* block, allowing the analyst to test the effectiveness of a *HO-DDM* according to several quality metrics; *(c)* a *Deviance Detection and Investigation* block, responsible for both applying the *HO-DDM*s to new traces and providing the analyst with tools for analyzing (and possibly revising) the assignment of traces to classes (deviant vs. normal); and finally *(d)* a HO-DDM *Improvement* block, which tries to acquire knowledge, from the analyst, about a reduced number of cases (in a semi-supervised fashion, and in the spirit of active-learning approaches), in order to improve the capability of a discovered *HO-DDM* to correctly discriminate deviant from normal traces.

Before describing each single block in the architecture in details, let us introduce, for the sake of clarity, two real thresholds γ and δ, which can be tuned by the analyst, in

[5] of the form $D_j(x_j)$, $N_j(x_j)$, $D_{ji}(x_j,x_i)$, $N_{ji}(x_j,x_i)$, K_j, K_{ji}, where $i,j \in \{1,\ldots,m\}$, $x_i \in dom(X_i)$, X_i is the i-th input attribute in the stacked view *SS*

order to control the way every deviance probability score $devProb_H(\tau)$ (i.e. the probability that a trace τ is a deviance based on some discovered *HO-DDM* model H) will be used in practice for deviances' detection and analysis purposes. Specifically, $\gamma \in (0,1)$ is the "deviance border" threshold such that any trace τ is eventually predicted to be deviant (resp. normal) iff $devProb_H(\tau) > \gamma$ (resp. $devProb_H(\tau) \leq \gamma$). The latter "uncertainty" threshold $\delta \in (0,1)$ can be used to define an "uncertainty" range around γ, so that every trace τ falling in the range —i.e. such that $|devProb_H(\tau) - \gamma| \leq \delta$, τ— is considered as *uncertain*. By converse, τ is deemed as a *high-confidence deviance* (*HC-deviance*) if $devProb_H(\tau) > \gamma + \delta$), and a *high-confidence normal* case (*HC-normal*) if $devProb_H(\tau) < \gamma - \delta$).

Let us now turn to describing the main components of the prospected system architecture. The HO-DDM *Learning* block is in charge of inducing a *HO-DDM* model from a set of traces, generated by the enactment of a some business process in the *BPM Enactment Platform* and saved into a log. A sample of these traces is labeled by domain expert, as a result of suitable inspection and analysis tasks. The labeled traces are then imported, through the *Log Gateway* block, into the *(Labeled) Example Traces* repository, acting as quickly accessible source of reliable data for the HO-DDM *Learning* block.

The steps toward the induction of a *HO-DDM* (cf. Algorithm 2) are performed by three different modules: *(i)* the *Pattern Extraction* module; *(ii)* the *Data Transformation* module, and *(iii)* the *Classifier Induction* module. Specifically, the *Pattern Extraction* module extracts and uses four different types of structural patterns defined in [4, 5], by leveraging the plugin *Signature Discovery* available in the ProM framework [1]: *tandem repeats*, *alphabet tandem repeats*, *maximal repeats*, and *alphabet maximal repeats*. These extracted patterns are then used by the *Data Transformation* module to support the derivation of all the kinds of log views (namely, *f-View* and *s-View*) employed by our approach. Roughly, it produces a boolean vector-space representation of the given traces, where each pattern is regarded as a distinguished attribute, taking a value of 1 iff the pattern occurs in the trace. The latter module also supports the derivation of a bag-of-activity representation of the traces. The *Classifier Induction* module induces a *HO-DDM* by leveraging the plugin *A1DE* available in the *Weka* library [15]. The induced *HO-DDM* model is then saved into the HO-DDM *Repository* for being further exploited to classify novel traces.

The HO-DDM *Evaluation* block supports the testing of discovered *HO-DDM*s. Specifically, the *Metrics Computation* module provides the analyst with several quality metrics (e.g., *AUC*, *G-mean*, *Recall*, and *Precision*), precisely tailored to our deviance detection setting, and described in detail in the experimental section. In addition, the *ROC Analysis* module, by supplying the analyst with visual tools like ROC curves, helps him/her choice an effective deviance-prediction scheme, by suggesting, in particular, which setting of deviance-border threshold γ can maximize the model's predictiveness.

The *Deviance Detection and Investigation* module allows to apply any discovered *HO-DDM* model to new log traces, as well as to help the analyst to efficiently investigate whether they actually represent deviant instances or not. In particular, each time a new trace is provided to the HO-DDM *Application* module, the *HO-DDM* model of the corresponding process (stored in the HO-DDM *Repository*) is applied in order to obtain

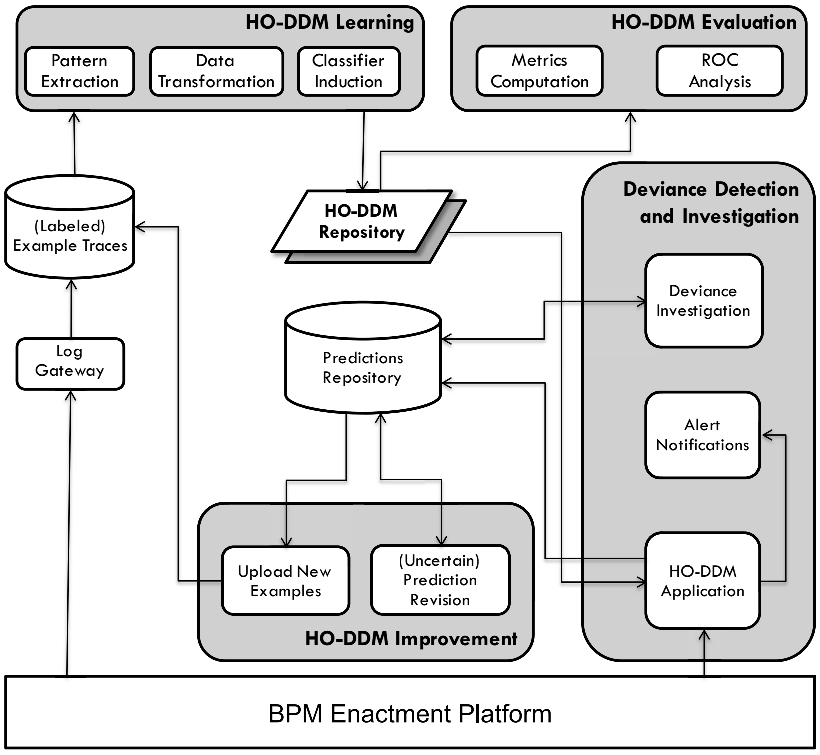

Fig. 3. System architecture for *HO-DDM-mine*.

a reliable classification for it. As soon as an instance is deemed as a *HC-deviance*, the *Alert Notifications* module immediately notify the analyst (and/or managers), in order to start investigation and possibly undertake suitable counter-measures. In any case, all predictions (either deviant or normal ones), along with their probability scores, are stored into the *Prediction Repository* for further inspections and analyses. To simplify the inspection of the most critical deviant traces, the *Deviance Investigation* module provides (either on-demand or periodically) the analyst with a probability-ranked list of deviations —the greater the $devProb_H(\tau)$, the higher the rank. By looking at such deviant traces, and by leveraging her/his domain expertise, the analyst may autonomously decide to change the predicted class label of one (or more) instances at a time. All the instances whose class labels have been manually changed by the analyst are marked with a flag (emphasizing their expert-driven validation status), and loaded into the *Prediction Repository*. Clearly, the presence of such misclassified traces reveals a certain degree of impreciseness of the current *HO-DDM* model. This could be due to different factors such as noise (i.e. errors) in the log, the lack of an adequate amount of deviance examples for training an accurate predictor, or even concept-drift phenomena that made the current *HO-DDM* model unable to reflect profound changes in the nature of both normal and deviant behaviors.

In order to improve the capability of a discovered *HO-DDM* to better discriminate deviant traces from normal ones, the HO-DDM *Improvement* block exploits information on new process instances (different from those initially used to train the model) and on the labels they have been assigned (by the model, or by the analyst). In order to possibly get a limited amount of feedback from the analyst, the block allows her/him to inspect a number of uncertain predictions (i.e. any new trace τ such that $|Prob_H(\tau) - \gamma| < \delta$), and to confirm/revise the class label predicted for them. In case the class label of an uncertain trace is changed by the analyst, it is flagged again as "manually modified" in the *Prediction Repository*. Therefore, at any time, the *Prediction Repository* will store both "optimistically trusted" traces (i.e. traces whose prediction has been consolidated implicitly over a long enough timespan, for it was not re-labeled manually by the analyst) and re-labeled ones (i.e, trace flagged as manually-modified). The latter category of traces, in its turn, primarily includes uncertain instances that have been disambiguated by the analyst, but it may possibly contain a number of *HC-deviances* (i.e. traces automatically labelled as deviances with a sufficiently high degree of confidence) that were manually revised by the analyst after inspection. When the number of manually revised instances exceeds a certain value, the analyst can decide to trigger a new learning process, in order to obtain an updated version of the *HO-DDM* model taking advantage of such newly generated training instances. Two options are available: *(1)* to incrementally update the AODE combiner (i.e., the meta-classification layer in the *HO-DDM*) only, by simply applying function `updateCombiner` on these new (implicitly or explicitly) validated examples, or *(2)* training a completely new *HO-DDM* by launching again algorithm *HO-DDM-mine* over all the process traces in the *Prediction Repository*.

8 Experiments

In order to assess the capability our approach of effectively recognizing deviant behaviors, we conducted a series of tests on a real-life log, storing information on the clinical pathways of gynecologic cancer patients within a Dutch hospital, and made available as a benchmark dataset for the *2011 BPI Challenge* [13].

8.1 Dataset

Basically, the log registers information concerning the activities (mainly corresponding to the application of treatments) performed on patients suffering from common types of cancers to the genital tract. The raw event log contains 150,291 events, referring to 624 distinct activities, and 1,142 cases, corresponding each to a distinguished patient.

A number of attributes are stored for each case, which include the age of the respective patient, and two categories of attributes that concern the kinds of illness the patient was diagnosed with: "diagnosis code" and "diagnosis". Precisely, a case may be associated with up to 16 alpha-numeric diagnosis codes (e.g., `M13`, `M12`, `106`), stored into different attributes of the form *Diagnosis code, Diagnosis code:1, ..., Diagnosis code:15*, each referring to a distinguished type of cancer at a certain stage of malignancy. For example, code `M13` identifies a kind of cervix cancer. Similarly, each case contains 16 attributes of category "diagnosis" —namely, *Diagnosis, Diagnosis:1, ...,* *Diagnosis:15*— that can store each a short description of an illnesses (e.g. “maligniteit ovarium’, “maligne neoplasma cervix uteri”) diagnosed to the patient.

The main event attributes stored in the log are: *concept:name* (resp., *Activity code*), storing the name (resp., code) of the activity performed; *Specialism code*, storing the code of the medical specialism related to the activity; *org:group* and *Producer code*, which both represent the activity’s executor, but at different granularity levels. Specifically, attribute *org:group* indicates the department/lab where the activity was performed, while the latter attribute stores an identifier of the person who performed the activity.

Since the traces in the log have no predefined class label, in order to make them suitable for a deviance mining task, we firstly marked each of them as either “normal” (label = 0) or “deviant” (label=1), by adopting one of the deviance criteria (namely, the one referred to as $BPIC11_{CC}$) introduced in [21]. Specifically, we labeled as deviant all the traces referring to patients diagnosed with a cervix cancer, i.e. all the traces where attribute *Diagnosis* evaluates “cervix cancer”. Notably, we choose this particular definition of “deviant” clinical cases because it corresponds to the highest class imbalance ratio, among all those explored in [21]: 225 deviant cases (less than 20% of all traces) vs. 917 normal ones.

For the sake of fairness, we preprocessed the log by removing all the attributes (including those of categories “diagnosis code”, “diagnosis”, and “treatment code”) directly linked to the class label, which would make trivial the deviance detection task.

8.2 Evaluation Metrics

Different evaluation metrics exist in the literature for testing the effectiveness of classification models in the presence of a rare class. Indeed, the usage of metrics that do not adequately accounts for the rarity of such a minority class may lead to overestimating the real capability of a classifier to correctly recognize the instances of that class. In the following we only concentrate on a binary imbalanced classification problem (as our deviance detection problem is), where the “positive” class label (namely 1) is assigned to deviant (typically rare) instances, while the “negative” class (namely 0) is assigned to the remaining (normal) ones.

Some core count-based statistics (usually shown in the form of a “confusion” matrix) for evaluating a classifier are: *(i) True Positives* (TP), i.e. the number of positive cases correctly classified as such; *(ii) False Positives* (FP), i.e. the number of negative cases incorrectly classified as positive; *(iii) False Negatives*, i.e. the number of positive cases incorrectly classified as negative; and *(iv) True Negatives*, i.e. the number of negative cases correctly classified as such.

Classification *accuracy* is the fraction of cases classified correctly: $(TP+TN)/(TP+FP+FN+TN)$. Despite this is a widespread evaluation metric, it is not appropriate when the classes are imbalanced. For instance, in a log where only 1% of traces are deviant, a simple model that predicts every trace as normal would have an accuracy of 99%, although it does bot recognize any deviant instance.

A popular metrics that can be safely used over imbalanced data is the *area under the ROC curve* (AUC) [7]. Essentially, ROC curves are a visual tool for comparing the performances of different classifier induction methods, over a Cartesian plane where the vertical an horizontal axes represent the *true positive rate* ($TPR = TP/(TP+FN)$) and *false positive rate* ($FPR = FP/(TN+FP)$), respectively. A ROC curve of an induction method is drawn by plotting the score pairs (TPR, FPR) of different classification models discovered with the method. This curve essentially shows to what extent the "accuracy" (measured via the TPR score) on positive examples tends to drop when reducing the error rate (measured via the FPR score) on negative examples. *AUC* is a compact average measure for the performances of a classifier (the higher the AUC, the better the classifier), which let us to quantify its classification potential.

The geometric mean $\sqrt{TPR \cdot TNR}$, namely *G-mean*, was introduced in [18] as another performance metrics suitable for the case of imbalanced classes. The best classifier according to this metrics is the one that maximizes the accuracies on both classes, while keeping them balanced.

We also evaluated the standard *Precision* (*P*) and *Recall* (*R*) [9] measures on the class of deviant instances, in order to support fine grain analyses on the misclassification errors made over those instances: $P = TP/(TP+FP)$ and $R = TP/(TP+FN)$.

8.3 Parameter Settings

Two key ingredient of our approach are: *(i)* the kind of patterns used to project the log traces onto a vector space, and *(ii)* the classifier-induction methods employed to derive, from such a feature-based representation of the traces, the base and combined models that compose the overall *HO-DDM* returned by the approach.

As concerns the former point, as a first family of behavioral patterns, denoted by `IA` (i.e. *individual activities*), we simply considered all the process activities in their own. In this case, for any trace, we regard each activity, say a, as an additional (pattern-oriented) feature of the trace, storing the number of times that a occurs in the trace. In order to produce more sophisticated representations of traces' behaviors, we considered (as also done in [3, 21]) all the sequence-based patterns described in Section 7, possibly capturing control-flow constructs (e.g., subprocesses, loops, and parallelism) ruling the behavior of the analyzed process: *tandem repeats* (`TR`), *alphabet tandem repeats* (`ATR`), *maximal repeats* (`MR`), and *alphabet maximal repeats* (`AMR`). As mentioned previously, all patterns but `AI` ones, were used as boolean attributes when computing the *f-View* of each trace, taking a value of 1 iff the pattern occurs in the trace. Similarly to [21], we considered the following heterogenous families of patterns: *(i)* {`IA`}, i.e. individual activities used alone (producing a bag-of-activity representation of traces' structure); *(ii)* {`IA,TR`}, i.e. the combination of individual activities and of tandem repeats; *(iii)* {`IA,ATR`}, i.e. individual activities combined with alphabet tandem repeats; *(iv)* {`IA,MR`}, i.e. individual activities plus maximal repeats; *(v)* {`IA,AMR`}, i.e. individual activities plus alphabet maximal repeats.[6]

These pattern families were provided as input to algorithm *HO-DDM-mine* (via parameter $\mathcal{F}$), in order to make it build 5 different views of the given log.

A fixed setting was used in all the tests for the parameters q and *resampleFactor*. The former was always set to 250 as in [21], while parameter *resampleFactor* (really used by algorithm *HO-DDM-mine* only when *resampleMode* = `OS`) was kept fixed to

[6] Notice that, differently from [21], we have not considered the usage of discriminative patterns [20], for we were not not able to obtain the source code for computing them.

2. This way, all the deviant (i.e. positive) traces in the log were duplicated, thus raising the ratio between deviant and normal traces from 1:4 to about 1:2.

As to the induction of the base classifiers, we resorted to the following methods currently implemented in our prototype system, as described in Section 7: the decision-tree learning method *J48* [22]; the *k-NN* procedure *IBk* (with *k* fixed to 10); the multi-layer perceptron method (denoted hereinafter as *ANN*) [29]; the *LibSVM* Support-Vector-Machines classifier [11] with an RDF kernel; and the rule-base classifier *JRip* [27].

8.4 Experimental Findings

In order to assess the validity our approach, we conducted a series of tests on the real log described before. To this regard, Table 2 summarizes the results obtained by algorithm *HO-DDM-mine*, compared with those of the deviance mining approach proposed in [21].

For the sake of comparison, as discussed in details in the previous subsection, we used the same families of patterns (apart from discriminative patterns), and the same (or a slightly enlarged) collection of classifier-induction methods as in [21]. However, our approach neatly differs from the one in [21] in two respects: *(i)* the possibility to exploit an oversampling mechanism, and *(ii)* the usage of an automated ensemble-based strategy, which intelligently integrates the models discovered by applying those different learning methods to different pattern-based views of the log —each of these models is used instead in [21] as an alternative “isolated” solution to the deviance detection problem.

In order to make a fair comparison, we turned the probabilistic predictions provided by our *HO-DDM*s into deterministic ones, by simply assigning each test trace to the class with maximum probability —this corresponds to using a fixed deviance boundary threshold $\gamma = 0.5$— without exploiting any kind of feedback from the analyst.

A first look a the competitor’s results Since the approach in [21] consists in applying each learning method to each distinct view of the log (generated according to one of the pattern families described in the previous subsection), it produces the 15 independent DDM models shown in Table 1 —namely, $J48_{\{IA\}}$, ..., $J48_{\{IA+AMR\}}$, $IBk_{\{IA\}}$, ..., $IBk_{\{IA+AMR\}}$, $ANN_{\{IA\}}$, ..., $ANN_{\{IA+AMR\}}$—which should be compared with the ones discovered by our approach. However, it is easy to notice that the outcomes in Table 1 are almost all very close to one another, and no one single DDM can be clearly declared winning over its competitors in all the quality metrics simultaneously. For instance, the *AUC* value for the $IBK_{\{IA\}}$ model is 0.798, while that for $ANN_{\{IA+TR\}}$ is 0.795 —this difference of less than 5% in their values reveals that they are practically equivalents to one another in terms of AUC performances. The same holds for the *G-Mean* of $J48_{\{IA+MR\}}$ (0.599), which is very close again to that of $IBk_{\{IA\}}$ (0.597). Similar considerations can be easily spotted as well for the remaining metrics. In such a situation, choosing the most suitable competitor to run against *HO-DDM-mine* is not a straightforward task.

Summarizing the competitor’s achievements In order to enable an easier comparison of our approach to the alternative settings of the competitor approach presented above,

Table 1. Prediction results on the *BPIC11$_{CC}$* log by base classifiers for different patterns. All the values were computed by averaging the results of 5 trials, performed according to a 5 fold cross-validation scheme. For each metrics, the best outcome is reported in bold.

Alg.	*Patterns*	*AUC*	*G-Mean*	*R*	*P*	*AvgRank*	*AvgRank_5%*
	{IA.AMR}	0.771±0.019	0.538±0.049	0.321±0.062	0.458±0.106	9.00	2.75
	{IA.ATR}	0.782±0.024	0.566±0.062	0.362±0.092	0.476±0.108	4.25	1.50
IBk	{IA.MR}	0.779±0.020	0.538±0.050	0.321±0.062	0.456±0.097	8.25	2.75
	{IA.TR}	0.772±0.028	0.545±0.111	0.351±0.151	0.411±0.076	9.00	2.25
	{IA}	**0.798±0.034**	0.597±0.043	0.397±0.072	0.493±0.084	**1.75**	**1.00**
	{IA.AMR}	0.787±0.023	0.451±0.066	0.222±0.076	0.468±0.146	8.75	3.75
	{IA.ATR}	0.780±0.020	0.470±0.124	0.261±0.172	0.409±0.144	10.50	3.50
ANN	{IA.MR}	0.777±0.026	0.523±0.201	0.360±0.271	0.412±0.082	8.75	2.25
	{IA.TR}	0.795±0.038	0.512±0.182	0.339±0.227	0.417±0.049	8.00	2.75
	{IA}	0.779±0.037	0.426±0.032	0.198±0.028	0.359±0.107	12.75	4.75
	{IA.AMR}	0.740±0.066	0.587±0.099	0.397±0.139	0.459±0.082	6.50	1.50
	{IA.ATR}	0.768±0.019	0.570±0.098	0.378±0.134	0.425±0.070	7.00	1.75
J48	{IA.MR}	0.746±0.069	**0.599±0.090**	**0.412±0.128**	0.459±0.086	5.00	1.50
	{IA.TR}	0.706±0.062	0.458±0.100	0.244±0.105	0.370±0.091	13.75	4.50
	{IA}	0.757±0.044	0.561±0.103	0.359±0.138	**0.496±0.042**	6.50	2.00
MAX		**0.798±0.034**	**0.599±0.090**	**0.412±0.128**	**0.496±0.042**	**1.75**	**1.00**

we devised a method for summarizing the performances of the latter. More precisely, we defined three different criteria for choosing the best achievement of the competitor approach: *(i)* BEST_OF_BEST, *(ii)* BEST_AVG_RANK, and *(iii)* BEST_AVG_RANK_5%.

As to the BEST_OF_BEST row, it simply reports, for each evaluation method, the best value (i.e. the maximum) obtained by all of the different configurations of the approach in [21] in each single metric. To make clear which values are chosen, the best outcome in each column (i.e. performance metric) of Table 1 have been marked in bold. For the reader' convenience, these values are also explicitly reported in the row with the **MAX** label at the bottom of the same table. Clearly, it is important to point out that this row provides an overestimated evaluation of the competitor approach, which may not correspond to any actual configuration of it. In a sense, this row is a sort of upper bound for the performance of all the considered configurations of the competitor.

Therefore, in order to provide a more realistic (yet concise) term of comparison, we defined a second criterion for a further competitor, denoted by BEST_AVG_RANK, aiming at meaningfully aggregating all the results obtained with the approach of [21] and reported in Table 1. The way this competitor is actually determined is explained in the following.

Let C be the set of all DDM models discovered by the tested methods, and $M = \{AUC, G\text{-}Mean, R, P\}$ be the set of metrics considered in our evaluation setting. For any model $c \in C$ and any metrics $m \in M$, let $score(c,m)$ be the value returned by evaluating m against c. Based on these values, we ranked the models in C over each metrics. More clearly, $rank(c,m) = 1$ (resp., $rank(c,m) = k$) iff c is the best (k-th best) performer according to metrics m. Considering all metrics equally important for assessing the quality of a DDM, we computed an overall average ranking score for each model $c \in C$ as follows:

$$AvgRank(c) = .25 \times (rank(c,AUC) + rank(c,G\text{-}Mean) + rank(c,R) + rank(c,P))$$

The `BEST_AVG_RANK` model, selected among all the other models discovered by (using different configurations of) the approach of [21], is the one reaching the highest value of the overall ranking score *AvgRank*.

Example 1. Let us consider the the model $IBK_{\{IA\}}$, discovered with method *IBk* on individual-activities features (i.e., by using only the family `IA` of patterns) According to the values in Table 1, it can be easily noted that $rank(IBK_{\{IA\}}, AUC) = 1$ since $IBK_{\{IA\}}$ is scored higher than any other model on the AUC metric (i.e. it achieved the maximum score over the AUC column). By converse, $rank(IBK_{\{IA\}}, G\text{-}Mean) = 2$ –the same holds also for the metrics R and P– since $IBK_{\{IA\}}$ is the second best performer according to the *G-Mean* metric. As a final result, we obtain the overall rank-oriented score $AvgRank(IBK_{\{IA\}}) = .25 \times (1+2+2+2) = 0.25 \times 7 = 1.75$. According to this ranking criterion, the model returned by *IBk* on the `IA`-based log view is deemed as the best result of the approach in [21], namely `BEST_AVG_RANK`, with an average rank of 1.75. □

For the sake of comparison, the row of Table 2 marked as `BEST_AVG_RANK` reports the quality measures received by this model, as a second term of comparison for our approach.

The way the `BEST_AVG_RANK` competitor has been computed might be susceptible to criticisms due to numeric approximation problems possibly plaguing very close values. Indeed, it may happen that two models have performance scores so much close among them (i.e. under a certain approximation threshold z) that could be retained unfair assigning them different ranks. In order to preventively cope with such potential concerns, we considered a third evaluation strategy accounting as equivalent two models $c1, c2 \in C$ w.r.t. a given metric $m \in M$ if the difference between their values falls below a specified threshold z. More formally, this can be stated as in the following:

$$rank(c1,m) = rank(c2,m) \text{ iff } |m(c1) - m(c2)| \leq z \times \min(m(c1), m(c2))$$

As a consequence of the definition above, a new rank *AvgRank_z%* can be easily defined. Specifically, in our setting we considered as a reasonable approximation a threshold of 5% (i.e. $z = .05$), and then we selected the competitor `BEST_AVG_RANK_5%` (cf. last row of Table 2) according to the index *AvgRank_5%*.

Example 2. Let us focus again on the IBk_{IA} model, and let $z = .05$ be the threshold value used for alleviating the numeric approximation problem in our calculus. Based on Table 1, it results that $rank(IBK_{\{IA\}}, AUC) = 1$, as $IBK_{\{IA\}}$ performs better than any other approach over the metric AUC. However, under this new threshold-based setting, $rank(IBK_{\{IA\}}, G\text{-}Mean) = 1$, although the best performer w.r.t. the metric *G-Mean* is $\texttt{J48}_{\{IA,MR\}}$. Indeed, $\texttt{IBK}_{\{IA\}}$ and $\texttt{J48}_{\{IA,MR\}}$ are ranked equally due to the fact that $|G\text{-}Mean(IBK_{\{IA\}}) - G\text{-}Mean(J48_{\{IA,MR\}})| = |0.597 - 0.599| = .002 \leq .05 \times min(0.597, 0.599) = .05 \times 0.597 = .03$. Similar considerations apply for metrics R and P. Therefore, we have that $AvgRank\_5\%(IBK_{\{IA\}}) = .25 \times (1+1+1+1) = 0.25 \times 4 = 1.00$. As a consequence, $\texttt{IBK}_{\{IA\}}$ is the best performer according to criterion *AvgRank_5%*, i.e. the `BEST_AVG_RANK_5%` model. Please, notice that, by pure chance, it incidentally coincides with the `BEST_AVG_RANK` model. □

Table 2. Prediction results on the *BPIC11$_{CC}$* log by *HO-DDM-mine* and the competitor methods (proposed in `Nguyen et al.`) All the values were computed by averaging the results of 5 trials, performed according to a 5 fold cross-validation scheme. For each metrics, the best outcome is reported in bold.

Methods	*AUC*	*G-Mean*	*R*	*P*
HO-DDM-mine (`RESAMPLING + MORE_CLASSIFIERS`)	**0.853±0.053**	**0.736±0.022**	**0.598±0.042**	**0.742±0.049**
HO-DDM-mine (`RESAMPLING`)	0.819±0.044	0.722±0.047	0.584±0.080	0.715±0.047
HO-DDM-mine (`NO_RESAMPLING`)	0.813±0.026	0.648±0.039	0.469±0.056	0.502±0.082
`Nguyen et al.[21] (BEST_OF_BEST)`	0.798±0.034	0.599±0.090	0.412±0.128	0.496±0.042
`Nguyen et al.[21] (BEST_AVG_RANK)`	0.798±0.034	0.597±0.043	0.397±0.072	0.493±0.084
`Nguyen et al.[21] (BEST_AVG_RANK_5%)`	0.798±0.034	0.597±0.043	0.397±0.072	0.493±0.084

Clearly, by the way it is computed, the performances of our competitor in its `BEST_OF_BEST` setting are always better than that in both the `BEST_AVG_RANK` and `BEST_AVG_RANK_5%` ones. Therefore, the comparative analysis carried out in the following is focused on the ("optimistic" for the competitor) `BEST_OF_BEST` scenario.

Comparing HO-DDM-mine*'s results with the best ones of the competitor approach* Table 2 reports the quality scores obtained by the models discovered with three different configurations of algorithm *HO-DDM-mine*: *(1)* `NO_RESAMPLING`, where no resampling procedure (i.e. *resampleMode* =`NONE`) is applied to the transformed log (in order to reduce the class imbalance ratio), and the same set of (base) inductive learning methods as in [21], i.e. $\mathcal{M} = \{J48, IBk, ANN\}$ is used; *(2)* `RESAMPLING`, using our basic oversampling scheme (i.e. *resampleMode* = `OS` and *resampleFactor* = 2), along with the same battery of base classifiers as in the previous configuration (and in [21]); *(3)* `RESAMPLING + MORE_CLASSIFIERS`, which uses the same oversampling setting as in configuration 2 (i.e. *resampleMode* = `OS` and *resampleFactor* = 2) while exploiting all the classifier-induction methods provided by our prototype system (i.e. $\mathcal{M} = \{J48, IBk, ANN, LibSVM, JRip\}$).

The three bottommost rows in the same table report, as a term of comparison, the best scores obtained by all the settings of the competitor approach illustrated above, and computed according to the three criteria (namely, `BEST_OF_BEST`, `BEST_AVG_RANK`, and `BEST_AVG_RANK_5%`) explained in the previous paragraph.

From the figures in this table, we can draw several interesting observations. First, the proposed approach, even in the basic `NO_OVERSAMPLING` configuration, performs always better (over all the quality metrics) than the competitor, whatever configuration is used for the latter. This confirms the validity of using an ensemble-learning approach to the deviance detection problem, which seems to take the best of different data transformation and data mining schemes, and improve the performances of them all.

The gain w.r.t. the approach in [21] becomes more marked when using our (basic) oversampling procedure (i.e. configuration `OVERSAMPLING`). In more detail, even though the increment in terms of *AUC* is moderate (2.62%), we can observe a significant gain for the metric *G-Mean* (20.53%), and a noticeable 44.15% (resp. 41.74%) achievement in terms of precision (resp. recall).

Further improvement is obtained by our approach when letting it use a broader range of base classifiers —actually, we only extended the learning methods used by our com-

Table 3. Prediction results on the *BPIC11*$_{CC}$ log by *HO-DDM-mine* when using different learning algorithms (as an alternative to our `AODE`-based method) to implement the function `mineCombiner` (see Figure 2). All the values were computed by averaging the results of 5 trials, performed according to a 5 fold cross-validation scheme. For each metrics, the best outcome is reported in bold.

Meta-algorithm	*AUC*	*G-Mean*	*R*	*P*
`AODE` (default)	**0.853±0.053**	**0.736±0.022**	**0.598±0.042**	**0.742±0.049**
`AdaBoostM1`	0.811±0.056	0.719±0.039	0.579±0.057	0.710±0.070
`J48`	0.748±0.057	0.724±0.048	0.584±0.065	0.718±0.071
`JRip`	0.715±0.029	0.693±0.033	0.543±0.040	0.676±0.061
`Logistic`	0.789±0.052	0.712±0.033	0.570±0.049	0.696±0.042

petitor with the insertion of *LibSVM* and *JRip*. Indeed, in this case, a gain of 6.89% (resp., 22.87%, 49.59%, 45.14%) is obtained in terms of *AUC* (resp., *G-Mean*, precision, recall) w.r.t. the overestimated `BEST_OF_BEST` configuration.

In summary, it seems that the combination of an oversampling method with our ensemble-learning strategy helps obtain higher improvements (w.r.t. the competitor supervised deviance-detection approach) than exploiting a wider range of base classifiers.

Before leaving this section, it is worth noticing that *HO-DDM-mine* took an average time of 25.13 seconds to compute a *HO-DDM* in the tests described so far. This corresponds to less than a 1% increase w.r.t. the time that would be spent by launching all the considered configurations of the approach of [21] (using different sets of behavioral patterns and different classifier-induction algorithms), in order to eventually select the best among the models discovered by them. Notably, a great fraction (namely, 98%) of the computation time was spent in the extraction of the behavioral patterns, which was particularly expensive for the case of tandem repeats and maximal repeats. This suggests that higher scalability could be obtained by using some more aggressive strategy for pruning the search space when computing such patterns, rather than simply using an extract-and-filter strategy (like that used in the current implementation of our approach).

Evidence for the benefits of using an AODE combiner as meta-learner In a further series of tests, we considered a series of variants of algorithm *HO-DDM-mine*, differing only in the meta-classifier method, used for inducing a combined deviance detection model, out of the stacked view produced (with the help of the discovered base classifiers) in the former 11 steps of the algorithm. More precisely, we replaced the AODE-based procedure described in the algorithm with the following Weka plugins: `AdaBoostM1`, `J48`, `JRip`, `Logistic`. The results of this experimentation are reported in Table 3. It is clear that our AODE-based approach ensures superior performances over all metrics when compared with all of these meta-classifiers, likely due to its capability of obtaining accurate and robust estimates of the class membership probability, despite the high degree of dependence between the attributes in the stacked view given as input to it.

9 Conclusion and Future Work

We proposed a framework for the detection and analysis of deviances in the executions of a business process, consisting of a supervised method for inducing flexible and robust deviance detection models, and of a comprehensive system architecture that fully exploits and empower the discovered models. The induction method adopts a novel multi-view ensemble learning scheme, where different base learners are trained against different pattern-based views of a given log, each of which corresponds to a vector-space encoding of the traces, combining both context data and structural features. This collection of base models is eventually made undergo a (scalable and robust) probabilistic meta-learning procedure, which produces an integrated high-order deviance detection model as an ultimate result. A basic resampling technique is exploited to deal with situations where the classes are highly unbalanced. Preliminary tests performed on a real-life log proved that the approach can achieve compelling performances w.r.t. a recent deviance mining method.

As concerns future work, beside trying to improve the scalability of our approach (by resorting to a parallel/distributed implementation of computationally-intensive model induction tasks), we plan to purse a number of extension lines, summarized below.

First of all, we will try to combine the extraction of structural patterns with suitable event abstraction mechanisms, in order to automatically extract high-level activity concepts from raw log events (in the spirit of [6]), and remove the assumption (made by most process mining approaches, but not holding in many real-world applications) that each log event explicitly conveys an activity label.

Moreover, we will investigate on integrating the approach with cost-sensitive learning methods, as a more sophisticated and tunable solution for dealing with the need of minimizing a certain kind of misclassification errors over the class of deviant traces.

In order to avoid the materialization of high-dimensional pattern-based views, we will also try to take advantage of suitable kernel learning methods. Particular attention will be payed to *multiple kernel learning* approaches [45], which appear to potentially fit well our multi-view setting.

References

1. van Dongen et al.: The ProM framework: A new era in process mining tool support. In: Proc. of 26th 10th Int. Conf. on Applications and Theory of Petri Nets (ICATPN'05), pp. 444–454 (2005)
2. Blum A. and Mitchell T.: Combining labeled and unlabeled data with co-training. In: Proc. of the 11th Conf. on Computational Learning Theory (COLT'98), pp. 92–100 (1998)
3. Bose, R.P.J.C., van der Aalst, W.M.P.: Discovering signature patterns from event logs. In: IEEE Symp. on Computational Intelligence and Data Mining (CIDM'13), pp. 111–118 (2013)
4. Bose, R.P.J.C., van der Aalst, W.M.P.: Abstractions in process mining: A taxonomy of patterns. In: Proc. of 7th Int. Conf. on Business Process Management (BPM'09), pp. 159–175 (2009)
5. Bose, R.P.J.C., van der Aalst, W.M.P.: Trace clustering based on conserved patterns: Towards achieving better process models. In: Proc. of Business Process Management Workshops (BPI'10), vol. 43, pp. 170–181 (2010)
6. Folino, F., Guarascio, M., Pontieri, L.: Mining predictive process models out of low-level multidimensional logs. In: Proc. of 26th Int. Conf. on Advanced Information Systems Engineering (CAISE'14), pp. 533–547 (2014)
7. Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recognition 30(7), 1145–1159 (1997)

8. Breiman, L.: Bagging predictors. Machine Learning 24(2) (1996)
9. Buckland, M., Gey, F.: The relationship between recall and precision. Journal of the American Society for Information Science 45(1), pp. 12–19 (1994)
10. Clarke, B.: Comparing bayes model averaging and stacking when model approximation error cannot be ignored. The Journal of Machine Learning Research 4, pp. 683–712 (2003)
11. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning 20(3), pp. 273–297 (1995)
12. Dietterichl, T.: Ensemble Learning (2002)
13. van Dongen, B.: Real-life event logs - hospital log (2011), http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
14. Fayyad, U. M., Irani, K. B.: Multi-interval discretization of continuous valued attributes for classification learning. In: Proc. of 13th Int. Joint Conf. on Artificial Intelligence (IJCAI'13). pp. 1022–1027 (1993)
15. Frank, E., Hall, M.A., Holmes, G., Kirkby, R., Pfahringer, B.: Weka - a machine learning workbench for data mining. In: The Data Mining and Knowledge Discovery Handbook, pp. 1305–1314 (2005)
16. Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: Proc. of 13th Int. Conf. on Machine Learning (ICML'96), pp. 148–156 (1996)
17. Japkowicz, N., Stephen, S.: The class imbalance problem: A systematic study. Intelligent Data Analysis 6(5), pp. 429–449 (2002)
18. Kubat, M., Holte, R., Matwin, S.: Learning when negative examples abound. In: Proc. of 9th Europ. Conf. on Machine Learning (ECML'97), pp. 146–153 (1997)
19. Lakshmanan, G.T., Rozsnyai, S., Wang, F.: Investigating clinical care pathways correlated with outcomes. In: Proc. of 11th Int. Conf. on Business Process Management (BPM'13), pp. 323–338 (2013)
20. Lo, D., Cheng, H., Han, J., Khoo, S.C., Sun, C.: Classification of software behaviors for failure detection: A discriminative pattern mining approach. In: Proc. of 15th Int. Conf. on Knowledge Discovery and Data Mining (KDD'09), pp. 557–566 (2009)
21. Nguyen, H., Dumas, M., Rosa, M.L., Maggi, F.M., Suriadi, S.: Mining business process deviance: A quest for accuracy. In: Proc. of 2014 Int. Conf. On the Move to Meaningful Internet Systems (OTM'14), pp. 436–445 (2014)
22. Quinlan, J.R.: C4.5: programs for machine learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
23. Sun, C., Du, J., Chen, N., Khoo, S.C., Yang, Y.: Mining explicit rules for software process evaluation. In: Proc. of Intl. Conf. on Software and System Process (ICSSP'13), pp. 118–125 (2013)
24. Suriadi, S., Wynn, M.T., Ouyang, C., ter Hofstede, A.H.M., van Dijk, N.J.: Understanding process behaviours in a large insurance company in Australia: A case study. In: Proc of 25th Int. Conf. on Advanced Information Systems Engineering (CAiSE'13), pp. 449–464 (2013)
25. Swinnen, J., Depaire, B., Jans, M.J., Vanhoof, K.: A process deviation analysis - A case study. In: Proc. of 2011 Business Process Management Workshops, pp. 87–98 (2011)
26. Webb, G.I., Boughton, J.R., Wang, Z.: Not so naïve Bayes: aggregating one-dependence estimators. Machine learning 58(1), pp. 5–24 (2005)
27. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems). Morgan Kaufmann Publishers Inc. (2005)
28. Wolpert, D.H.: Original contribution: Stacked generalization. Neural Networks 5(2), pp. 241–259 (1992)
29. Zhang, G.P.: Neural networks for classification: a survey. Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 30(4), pp. 451–462 (2000)

30. Zhou, Z.H., Chen, K.J., Jiang, Y.: Exploiting unlabeled data in content-based image retrieval. In: Machine Learning: ECML 2004. pp. 525–536 (2004)
31. Langley P., Iba W., Thompson K.: An analysis of Bayesian classifiers. In: Proc. of 10th Nat. Conf. on Artificial intelligence (AAAI'92), pp. 223–228 (1992)
32. Domingos P., Pazzani M.J.: Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier. In: Proc. 13th Int. Conf. on Machine Learning (ICML'96). pp.105–112 (1996)
33. Domingos P., Pazzani M.J.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning 29, pp.103–130 (1997)
34. Sahami M.: Learning Limited Dependence Bayesian Classifiers. In: Proc. 2nd ACM SIGKDD of Int. Conf. Knowledge Discovery and Data Mining (KDD'96), pp. 334–338 (1996)
35. Keogh E.J., Pazzani M.J.: Learning Augmented Bayesian Classifiers: A Comparison of Distribution-Based and Classification-Based Approaches. In: Proc. of Int. Workshop Artificial Intelligence and Statistics, pp. 225–230 (1999)
36. Keogh E.J., Pazzani M.J.: Learning the Structure of Augmented Bayesian Classifiers. Int. J. Artificial Intelligence Tools, 11(40), pp. 587–601 (2002)
37. Webb G.I., Boughton J., Wang Z. Not So Naive Bayes: Aggregating One-Dependence Estimators. Machine Learning, 58(1), pp. 5–24 (2005)
38. Ying Y. et al.: To Select or To Weigh: A Comparative Study of Linear Combination Schemes for SuperParent-One-Dependence Estimators. IEEE Transactions on Knowledge and Data Engineering, 19(12), pp.1652–1665 (2007)
39. Zheng F. , Webb G.I.: Efficient Lazy Elimination for Averaged One-Dependence Estimators. In: Proc. 23rd Int. Conf. Machine Learning (ICML'06). pp.1113–1120 (2006)
40. Hady M. , Schwenker F.: Combining committee-based semi-supervised learning and active learning. Journal of Computer Science and Technology, 25(4). pp. 681–698 (2010)
41. Freund Y., Seung H.S. and Shamir E. and Tishby N.: Selective sampling using the query by committee algorithm. Machine Learning, 28(2-3), pp.133–168 (1997)
42. Zhang Q., Sun S.: Multiple-view multiple-learner active learning. Pattern Recognition 43(9), pp. 3113–3119 (2010)
43. Yu H.: Selective sampling techniques for feedback-based data retrieval. Data Mining Knowledge Discovery, 22(1-2), pp. 1–30 (2011)
44. Zhou Z.-H., Zhan D.-C., Yang Q.: Semi-supervised learning with very few labeled training examples. In: Proc. of the 22nd AAAI Conference on Artificial Intelligence (AAAI'07), pp. 675–680 (2007)
45. Gönen M., Alpaydin E.: Localized multiple kernel learning. In: Proc. of the 25th Int. Conf. on Machine learning (ICML'08), pp. 352?359 (2008)
46. Muslea I., Minton S., Knoblock C.A.: Selective sampling with co-testing. In: The CRM Workshop on Combining and Selecting Multiple Models With Machine Learning (2000)
47. Muslea I., Minton S., Knoblock C.A.: Active+ semi-supervised learning= robust multiview learning. In: Proc. of 19th Int. Conf. on Machine Learning (ICML'02), pp. 435–442 (2002)
48. Muslea I., Minton S., Knoblock C.A.: Active learning with multiple views. Journal of Artificial Intelligence Research, 27(1), pp. 203–233 (2006)
49. Nigam K., Ghani R.: Analyzing the effectiveness and applicability of co-training. In: Proc. of the 9th Int. Conf. on Information and Knowledge Management (CIKM'00), pp. 86–93 (2000)
50. Seung H.S., Opper M., Sompolinsky H.: Query by committee. In: Proc. of the 5th Annual Workshop on Computational Learning Theory, pp. 287–294 (1992)
51. Wang W., Zhou Z.H.: A new analysis of co-training. In: Proc. of the 27th Int. Conf. on Machine Learning (ICML'10), pages 1135?1142, 2010.