Bayesian Network based Predictions of Business Processes

Stephen Pauwels$^{1[0000-0002-0427-8945]}$ and Toon Calders$^{1[0000-0002-4943-6978]}$

University of Antwerp, Antwerp, Belgium
{stephen.pauwels,toon.calders}@uantwerpen.be

Abstract. Predicting the next event(s) in Business Processes is getting more importance with more and more automated systems. Predicting deviating behaviour early in a process can ensure that a possible error can be corrected or that unwanted delays can be avoided. We propose to use Bayesian Networks as a basis for predicting the next event. Using these models we capture the different dependencies between different attributes within a log to gain a better and more fine-grained prediction. We show that our model performs very well compared to existing methods. Our model, due to its underlying Bayesian Network, is capable of providing a more comprehensible explanation of why a prediction is made. We also show that the runtimes of our learning algorithm are lower than these of other methods. In the experiments we perform an elaborate comparison between different state-of-the-art methods using different real life datasets.

Keywords: Business Process · Event Prediction · Dynamic Bayesian Network.

1 Introduction

Process Mining [1] is the field that studies the creation of understandable models given data from a BP. With more and more automated systems and process execution it becomes more and more important to be able to predict as soon as possible some aspects of a running case. For example, we want to know as fast as possible when deviations occur in a log or when we can already predict that the deadline will be exceeded with high probability. Di Francescomarino et al. [6] show the growing importance and interest in predicting events within Business Processes. Different methods already have been proposed, using probabilistic models, machine learning and Deep Learning. All these methods have in common that they use historical data for creating a reference model that gets used for determining the next activity. The commonly used process models, such as workflow nets, however, are not suitable as they often give no accurate results for predicting next events in a case.

Although widely used, Deep Learning methods have some issues. A first issue is that training of the model often takes very long, often also needing specialised hardware for efficient learning. Next, it is also important to have a sufficient amount of data available for training an accurate model. With an ever increasing demand for giving explainable, comprehensible models neural networks also have the disadvantage that they still work as blackboxes without any possibilities of having a detailed explanation for the given output.

We propose a method for predicting the next event in Business Process (BP) log that uses a comprehensible probabilistic model. Our proposed method tries to solve most of the existing issues while still performing with high accuracy.

In this paper we start from our previously defined eDBNs that were developed for detecting anomalies in BP logs [15]. Using Bayesian Networks and their Conditional Dependencies we can create a probabilistic model that is able to predict next events. The contributions of our paper are:

- We propose a fast and comprehensible algorithm that is able to accurately predict the next events in a BP log. The learned model can easily be modified by a domain expert.
- We performed an elaborated survey and comparison of some existing state-of-the-art techniques, which was not yet done in literature. We compare the methods proposed by Tax et al. [18], Camargo et al. [4], Lin et al. [14] and Di Mauro et al. [7]. We both compare accuracy and runtimes for these algorithms.

The remainder of the paper is structured as follows: in the next section we explain existing methods and indicate some of their strengths and weaknesses. Section 3 introduces some concepts we need in order to explain how we use Dynamic Bayesian Networks for next event prediction. Section 4 explains the prediction of the next events in detail. An extensive evaluation and comparison is performed in Section 5, where we take a look at both the next event prediction problem and the suffix prediction problem. Next to comparing the accuracies of the different methods, we also compare the methods based on runtime for both training and testing.

2 Related Work

There already exists ample research on applying probabilistic models for predicting events in Business Processes. Becker et al. [2] proposed a framework for real-time prediction based on a probabilistic model learned from historical data. They use Hidden Markov Models (HMM), which are state machines where the next state depends on the current state and current event, and Probabilistic Finite Automatons (PFAs). An advantage of these models is their explainability and possibility to visualization, but on the other hand, their model does not take multiple attributes into account and is not able to find long-term dependencies.

Explainability is a useful feature to convince domain experts who have no technical knowledge about how the models get trained. Breuker et al. [3] propose a very similar method where they start from a Probabilistic Finite Automaton and further improve this model to work with event data.

Lakshmanan et al. [13] propose an instance-specific Probabilistic Process Model (PPM) which calculates the likelihood for all possible events to occur next. When making some non-restrictive assumptions, they show that their model can be transformed into a Markov Chain. Thus, they can use existing Markov techniques. To create their model, they first mine a process model based on the given traces, represented as a Petri Net. Starting from this model they look at all the OR and AND splits to extend it to a probabilistic model that is able to predict the next event.

An important factor in these probabilistic methods is that they are comprehensible, but take only a limited amount of information regarding the sequences of events into account. Meaning that it is harder for them to detect and correctly predict long-term dependencies, as they encounter the same challenges as most process models.

In previous work we introduced the extended Dynamic Bayesian Networks [15] (eDBN) which captures the different dependencies between attributes to construct a Bayesian Network which describes a joint probability. The dynamic aspect was introduced by transforming the input data so that every event is represented by a vector containing next to the event itself, its attributes such as resource, duration, etc, and those of previous events.

With the field of Deep Learning becoming more popular and achieving great results in the field of Natural Language Processing (NLP), researchers also investigated the usage of Neural Networks in Business Processes, as these have some properties in common with NLP. Most of the state-of-the-art methods use Long Short Term Memory (LSTM) cells in their neural network [4, 11, 12, 14, 18]. LSTM cells are cells that are able to remember and use this memory for determining the output. Most of these methods are very similar to each other. Every method does use its own architecture, where extra LSTM cells can be added or organised in different ways. Where probabilistic methods only predict the next activity, the neural networks also predict the duration of an event. At the moment, however, we are not interested in solving the problem of predicting the duration of activities. Therefor we do not explain this aspect of the methods.

LSTMs have the downside of being hard to train, due to their sequential nature. To counter this disadvantage, Di Mauro et al. [7] propose to use Convolutional Neural Networks. These networks also take the sequence of events into account but are more efficient to train.

Most methods use a one-hot encoding feature vector as input for the network. Camargo et al. [4] create a feature vector by taking the dot product of the activity and resource vectors and then reshaping it to a lower dimensionality, which is based on, but lot smaller than, the amount of different values.

Lin et al. [14] have optimized their network to take multiple attributes into account. In order to do so they introduced a new type of layer: the Modulator. This modulator combines the values of all attributes and uses a weighted sum to determine how much an attribute contributes to predicting another attribute.

Having a large amount of event attributes and event values is one of the biggest challenges for Recurrent Neural nets. To tackle this problem, Hinkka et al. [12] propose a Recurrent Neural Net method were they first cluster events according to their attribute values to lower the complexity.

To incorporate the sequential nature of the data, two approaches were used by the different methods. Camargo et al. [4] divide the input log into windows of 5 events each, they use padding with zeros for the first four events. The other methods use the length of the longest trace in the log, also padding with 0 when needed.

Paper	Model type	Input data	Predict Next Event	Predict Suffix	Predict Duration
Our paper	DBN	multivariate	✓	✓	
Becker et al.	HMM	activity	✓		
Breuker et al.	PFA	activity	✓		
Lakshmanan et al.	PPM	activity	✓		
Everman et al.	LSTM	activity and resource	✓	✓	
Tax et al.	LSTM	activity and duration	✓	✓	✓
Lin et al.	LSTM	multivariate	✓	✓	
Camargo et al.	LSTM	activity, resource and duration	✓	✓	✓
Di Mauro et al.	CNN	activity and duration	✓		✓
Hinkka et al.	Clustering RNN	multivariate	✓		

Table 1. Overview of related work.

3 Background

Before we explain how we extended and used Bayesian Networks for predicting the next activity, we introduce the basic problem setting and some concepts we need to create the model. First, we formally define an event and a logfile:

Definition 1. *Let* $e = (eID, a_{act}, desc)$ *be an* event *with a unique identifier eID, an associated activity* a_{act} *and an event-descriptor desc. The event-descriptor is a tuple* $(a_0, \ldots, a_n)$ *denoting the values of (possible) extra attributes* $\mathcal{A} = (A_0, \ldots, A_n)$*, an ordered set of attributes. Value* a_0 *of the event-descriptor corresponds with the value for attribute* A_0*, etc. We use* n *to denote the number of attributes.*

A case $c = (cID, [e_0, e_1, \ldots, e_m]))$ *has a unique identifier cID and consists of an ordered list of events. The order in the list determines the order in which the events were executed. We use* m_c *to denote the number of events of a case* c*.*

A logfile *is a set containing different cases all originating from the same process or institution.*

We express the problem of finding the next activity as follows:

Definition 2. *Given a partially completed case* $C = (cID, [e_0, \ldots, e_i])$*, with* $0 < i < m_c$*, we want to predict the activity of event* e_{i+1}*, which immediately follows the last seen event in the case.*

Predicting the suffix of a case is expressed as follows:

Definition 3. *Given a partially completed case* $C = (cID, [e_0, \ldots, e_i])$*, with* $0 < i < m_c$*, we want to predict the list* $[a_{act}^{i+1}, \ldots, a_{act}^{m}]$ *of all next events.* a_{act}^{j} *denotes the activity of event* e_j*.*

In this paper we are primarily looking to predict the activities in a case. Although we also predict other attributes when needed to correctly predict the suffix of a case.

3.1 Dynamic Bayesian Network

We use Bayesian Networks (BNs) [16] to model the different conditional dependencies between attributes present in a logfile. A BN is a directed acyclic graph where every node represents an attribute and a directed edge (u, v) from node u to v exists when attribute v depends on u. We call variable u the *parent* of variable v and we denote it as $Pa(v)$. A single variable v can depend on multiple other attributes.

The BN represents the following *joint probability*:

$$P(X_0, \ldots, X_n) = \prod_{i=0}^{n} P(X_i | Pa(X_i)) \tag{1}$$

To learn a BN we first learn the structure of the model (the different conditional dependencies between attributes). At the same time we learn the Conditional Probability Tables (CPTs) of the different attributes. These tables contain the probabilities for all possible values given the occurrence of values of the parents. In literature, there already exist algorithms that can learn the structure of such a BN given a reference dataset.

Dynamic Bayesian Networks (DBNs)[17] introduce extra attributes that are associated with previous time steps. In this paper, we use k to denote the number of previous time steps we want to incorporate in the model. An example of a DBN with $k = 1$ can be found in Figure 1. In this diagram one can easily see the different time steps. Dependencies may exist within one time step ($\mathit{Activity} \rightarrow \mathit{Type}$) or between time steps ($\mathit{Activity}_{previous} \rightarrow \mathit{Activity}_{current}$).

4 Predicting

First we use historical input data to train our probabilistic model, details about how to learn the model can be found in [15]. Next we can use this learned model to predict both the next event and the complete suffix for a partially completed case.

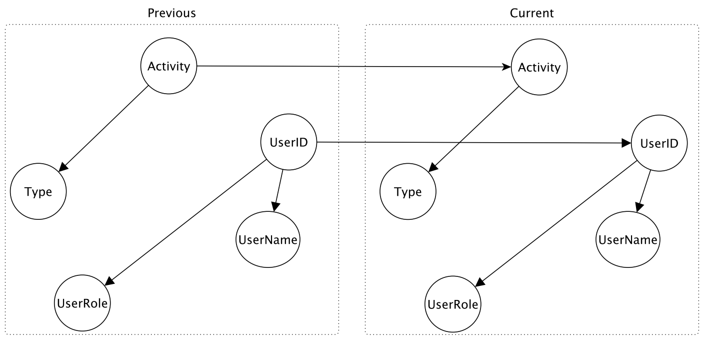

Fig. 1. Example Dynamic Bayesian Network

4.1 Next Event Prediction

To predict the next activity, we calculate $P(x_{act})$ for all possible (known) values of the activity attribute. Following Equation 1 we can describe the probability for the activity as follows: $P(X_{act}) = P(X_{act}|Pa(X_{act}))$. Calculating the most likely activity, given the values of the parent attributes, comes down to looking at the CPT associated with the activity attribute. This table contains the known combinations of values of parent attributes. Every combination is linked with possible values for the activity with their respective probability. We then select the activity which has the highest probability as the predicted next activity.

Example 1. Suppose our activity attribute depends on activity and resource from the previous time-step. We then have a CPT describing $P(activity\ |activity_1, resource_1)$ as can be seen in Table 2.

Given the values *b* and *resource3* for activity and resource respectively from the previous time-step, we can see in the table that two possible values for the next activity are possible: *c* (with a probability of 0.4) and *end* (with a

activity$_1$	resource$_1$	activity	P(activity)
start	resource1	a	1.0
start	resource2	a	1.0
a	resource1	b	0.7
a	resource1	c	0.3
a	resource2	d	1.0
b	resource3	c	0.4
b	resource3	end	0.6
c	resource1	end	1.0
c	resource3	end	1.0

Table 2. Example CPT given that activity depends on activity and resource from the previous time step.

probability of 0.6). We return the value with the highest probability, which is *end*, as the prediction for the next event.

Because we use a table with all combinations which have occurred in the training data, it is possible that the combination of parent values we encounter in the test data does not occur in the CPT. In that case we have to make an estimation based on the data and probabilities present in the CPT. We use marginalization techniques in order to come up with the most likely next activity. We make a distinction between two cases: the first one where there is at least one parent value that we did not encounter in the training data and the other where all values occur but did not occur together.

When there are unseen values for some attributes we calculate the marginalized probability where we iterate over all seen values for these attributes. We let the unseen attributes variate over all possible values, which occur in the CPT, while we keep the seen attributes the same. The probability for an activity is the sum of all probabilities of encountering this activity times the probability for the substituted values for the unseen attributes. The probabilities for all single values get learned during the training phase.

Consider the unseen attributes as A_u and the seen attributes as A_s, for determining the probability of the activities we need to calculate $P(X_{act}|A_u, A_s)$. We thus calculate the marginalization which equals $P(X_{act}|A_s)$. This probability can be calculated as follows:

$$P(X_{act}|A_s) = \sum_{a_u \in A_u} P(a_u) * P(X_{act}|a_u, A_s) \tag{2}$$

Where a_u corresponds with the possible values for the unseen attributes.

Example 2. Consider the CPT from Table 2, suppose we have the values e and *resource3* for attribute and resource. The value for resource has already been seen, the value for the activity is unseen. We thus marginalize over the activity attribute. The seen attribute *resource* only occurs together with the values b and c. We want to calculate the following formula for all possible values of activity, note that we only take into account combinations of parent values that do occur in the CPT:

$$P(X_{act}|e, \mathit{resource3}) = P(b) * P(X_{act}|b, \mathit{resource3}) + P(c) * P(X_{act}|c, \mathit{resource3})$$

Assume $P(b) = 0.3$ and $P(c) = 0.4$. We can now calculate the probabilities for all activities as follows:

$$\begin{aligned}
P(\mathit{start}|e, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 = 0 \\
P(a|e, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 = 0 \\
P(b|e, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 = 0 \\
P(c|e, \mathit{resource3}) &= 0.3 * 0.4 + 0.4 * 0 = 0.12 \\
P(d|e, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 = 0 \\
P(\mathit{end}|e, \mathit{resource3}) &= 0.3 * 0.6 + 0.4 * 1 = 0.58
\end{aligned}$$

From these results we can see that the next activity with the highest probability is *end*.

When all values occur in the training data, we marginalize over all attributes one by one and calculate the total sum of the probabilities. We use the following formula:

$$P(X_{act}|Pa(X)) = \sum_{A \in Pa(X)} \sum_{a \in A} P(a) * P(X_{act}|a, Pa(X) \setminus A) \tag{3}$$

Again, we calculate the probability for every possible activity to determine the most likely next activity.

Example 3. Consider again the CPT from Table 2, suppose this time we have the values *a* and *resource3* for activity and resource. Both values already occur in the CPT but never together. We thus calculate the following formula for every possible activity:

$$\begin{aligned} P(X_{act}|a, \mathit{resource3}) &= P(b) * P(X_{act}|b, \mathit{resource3}) + P(c) * P(X_{act}|c, \mathit{resource3}) \\ &+ P(\mathit{resource1}) * P(X_{act}|a, \mathit{resource1}) + P(\mathit{resource2}) * P(X_{act}|a, \mathit{resource2}) \end{aligned}$$

Assume $P(b) = 0.3$, $P(c) = 0.4$, $P(\mathit{resource1}) = 0.4$ and $P(\mathit{resource2}) = 0.2$. We then the following probabilities for the possible activities:

$$\begin{aligned} P(\mathit{start}|a, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 + 0.4 * 0 + 0.2 * 0 = 0 \\ P(a|a, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 + 0.4 * 0 + 0.2 * 0 = 0 \\ P(b|a, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 + 0.4 * 0.7 + 0.2 * 0 = 0.28 \\ P(c|a, \mathit{resource3}) &= 0.3 * 0.4 + 0.4 * 0 + 0.4 * 0.3 + 0.2 * 0 = 0.24 \\ P(d|a, \mathit{resource3}) &= 0.3 * 0 + 0.4 * 0 + 0.4 * 0 + 0.2 * 1 = 0.2 \\ P(\mathit{end}|a, \mathit{resource3}) &= 0.3 * 0.6 + 0.4 * 1 + 0.4 * 0 + 0.2 * 0 = 0.58 \end{aligned}$$

From these results we can see that in this case the next activity with the highest probability is *end*.

4.2 Suffix Prediction

For predicting the suffix we recursively determine all attributes the activity depends on. Apart from the parents of the activity attribute we also need to take the parents of the parents etc. into account. To predict the suffix of activities we also have to predict the other attributes, as we have to use them to further predict the suffix.

Example 4. Consider the example from the previous section. In our example, the activity depends on activity and resource in the previous time step. Assume now that resource itself also depends on the activity and resource from the previous time step. To be able to keep predicting the next activity, we also need to predict values for the resource by using the activity and resource from the previous time step.

To predict the value for an attribute we use the same formulas as were introduced in Section 4.1. As these formulas are attribute independent we can adapt the formulas in a straightforward way to also predict other attributes. We also use the same marginalization when new combinations of parent values occur.

Example 5. To predict the suffix for our example we have to predict values for activity and resource. We thus need to calculate the following probabilities:

$$P(\mathit{activity}|\mathit{activity_1}, \mathit{resource_1})$$
$$P(\mathit{resource}|\mathit{activity_1}, \mathit{resource_1})$$

The disadvantage of always selecting the activity which has the highest probability is that we can get stuck in an infinite self-loop when the most likely next activity is the same as the current one. We introduce a restriction on the size of these self-loops. During the learning phase of the algorithm we count the length of all self-loops for the activity. We use the average length of these self-loops as the maximum size for self-loops in the prediction phase. When we predict the same activity as the current activity, we increase a self-loop counter. When this self-loop counter reaches the maximum size for self-loops we select the second best activity as our prediction. When we predict a different activity we reset the self-loop counter to 0.

Example 6. Consider a situation where activity only depends on the previous activity. The CPT for activity is shown in Table 3. Assume we have $[\mathit{start}]$ as the current prefix, when we would predict the suffix without limitations on the self-loops we would get the following sequence: $[\mathit{start}, a, a, a, a, a, \ldots, a]$ until we reach the maximum allowed suffix length. Suppose now we have a maximum self-loop size of 3, then we get the following sequence: $[\mathit{start}, a, a, a, b, \mathit{end}]$.

activity$_1$	activity	P(activity)
start	a	1.0
a	a	0.6
a	b	0.4
b	end	0.8
b	c	0.2

Table 3. CPT for activity used for example 6

5 Evaluation

In this section, we describe the datasets we used to evaluate our method. We also perform some comparisons with state-of-the-art for the problems of next event and suffix prediction. We perform both runtime and accuracy experiments to fully compare the different methods.

5.1 Datasets

For our evaluation we use four different datasets.

- **BPIC12** [9]: a log from a Dutch financial institution, containing applications for personal loans. The logfile contains three intertwined processes. We use one of these processes to create an additional logfile (**BPIC12W**).
- **BPIC15** [10]: a log containing building permit applications from five different Dutch municipalities, where the data of every municipality is saved in a single log file. We denote these datasets as **BPIC15_1** to **BPIC15_5**.
- **Helpdesk** [19]: a log containing ticket requests of the helpdesk from an Italian Software Company.
- **BPIC18** [8]: a log containing applications for EU agricultural grants. Due to the size of this dataset we do not use this dataset to compare with all other methods. As initial experiments showed that training would be infeasible for most datasets.

Following the same preprocessing steps as in [4], we first remove all cases from the datasets with less than 5 events. Only for the Helpdesk dataset we put the threshold to 3, because of the smaller average length of the cases in this dataset. Table 4 gives the characteristics of the used datasets after preprocessing. The table shows that we use a variety of datasets with different characteristics to conduct an evaluation as complete as possible.

Dataset	# Events	# Cases	# Activities	Avg. activities per case	Max. length of a case
BPIC12	171,175	7,785	23	22.0	106
BPIC12W	61,449	4,848	6	12.6	74
BPIC15_1	52,217	1,199	398	43.5	101
BPIC15_2	44,347	829	410	53.5	132
BPIC15_3	59,533	1,377	383	43.2	124
BPIC15_4	47,271	1,046	356	45.2	116
BPIC15_5	59,068	1,153	389	51.2	154
Helpdesk	20,722	4,371	14	4.7	15
BPIC18	2,514,266	43,809	41	57.4	2,973

Table 4. Characteristics of the datasets after preprocessing

5.2 Method

We ran our experiments on compute nodes with 2 Xeon E5 processors, which have 14 cores each, and 128 GB RAM. We also used GPU based nodes having 2 NVIDIA P100 with 16GB of memory for some of the experiments. Most of the experiments where conducted using only CPU nodes, as our experiments showed that training LSTMs on a GPU was less efficient than using a CPU. We include some of these results in our general runtime results. We explicitly mention it in the results when the experiment was run on a GPU node.

We split all datasets into a training and testing part. We have chosen to split the datasets as described by Camargo et al. [4]. The first 30% of cases are used for testing and the remaining 70% for training.

In this paper we are only interested in predicting activities, not the duration of the activities. Therefore, we first removed the time aspect of all methods we compare to. We can safely assume that we did not alter the behavior of the proposed methods, as the timing aspect was often not influencing the prediction of the activity. Our method would however be able to cope with datasets including timestamps of the events. But as we only use categorical attributes most timestamps would be unique. And new predictions would have timestamps that were not seen before, meaning we would always have to marginalize over the timestamp for determining the next event.

All Neural Network learning algorithms use an early stopping method with a patience of 42, meaning that when no improvement occurred for 42 epochs the learning algorithm stopped. Increasing or decreasing this parameter has a direct influence on runtimes, but has also an influence on the accuracy. The value of 42 was chosen because it was used in the original code of the other implementations.

To measure the performance of the next event prediction we use the accuracy, which gives us the fraction of events that were predicted correctly. Events that did not get a prediction, because of unknown values are considered to be wrong. For the performance of the suffix prediction we use the Damerau-Levenstein [5] distance between the predicted trace s_p and the correct trace s_e as basis for our measure. As we want a measure that gives 1 when two traces are completely identical and 0 when they are completely different we use the following formula for the accuracy of the suffix prediction:

$$S(s_p, s_e) = 1 - \frac{DL\_distance(s_p, s_e)}{max(len(s_p), len(s_e))} \tag{4}$$

An implementation of our method and code to execute the experiments can be found in our GitHub repository[1]. The repository also contains the modified implementations of all other methods, where all timing information and duration prediction was removed.

5.3 Evaluation

First we examine the influence of the value of k, which determines how many previous time steps we take into account, and try to find a single optimal value. We test our method on all datasets with a varying k. We performed this test both for the next event prediction and for the suffix prediction, where we let k vary from 1 to 5.

Figure 2 shows that the best k value is equal to 3 for the next event prediction. The k value has a direct impact on the complexity of our learning algorithm.

[1] https://github.com/StephenPauwels/edbn

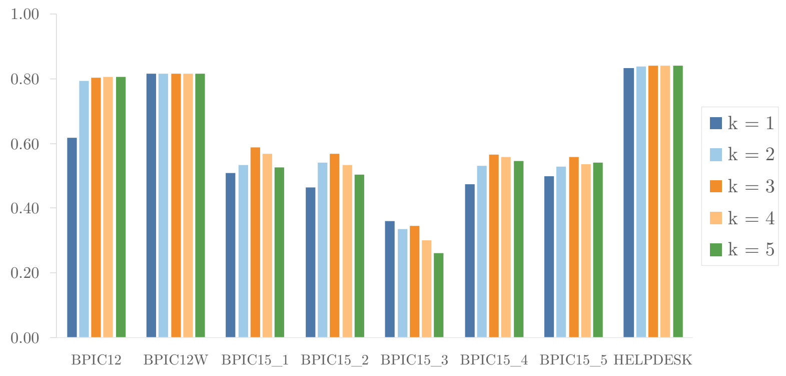

Fig. 2. Next event prediction accuracy for different k values.

As the number of attributes grows linear with k. Using $k = 3$ for next event prediction gives us the best accuracy with a relatively small impact on the overall complexity. We can see that for some datasets the accuracy decreases with a k set too high, this can be explained by the fact that we overfit the training data with a higher k value, as a higher k value also means that the activity depends on more attributes.

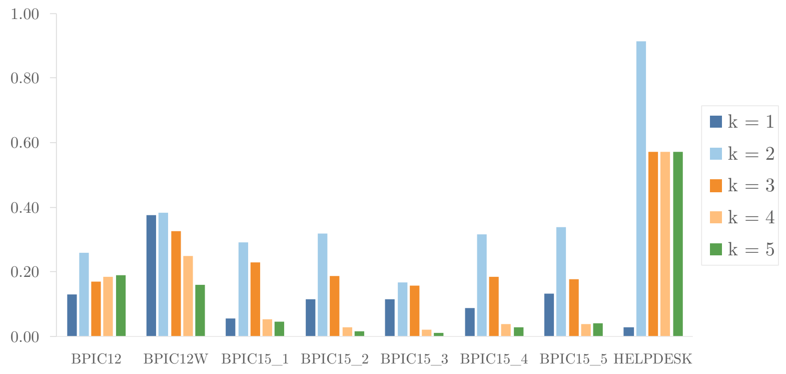

Fig. 3. Suffix prediction accuracy for different k values.

For the accuracies of the suffix prediction in Figure 3 we again see degrading results when we set k too high. Again this is due to overfitting of the data. The suffix generation is more sensitive for overfitting, as a predicted value has its consequences throughout the entire predicted sequence, therefor we see that the ideal k for the suffix prediction is lower than the k for next event prediction. This observation occurs across all different datasets, we can thus say with some certainty that the choice of $k = 2$ is the most optimal for most datasets.

5.4 Comparison

We compare our method with four state-of-the-art methods, all of which use neural networks. We used the implementations provided by Camargo et al. [4], Tax et al. [18] and Di Mauro et al. [7]. We implemented the method described by Lin et al. [14] as correctly as possible according to the paper, as no implementation was available. All used implementations have been added to the GitHub repository. We had to make some small adjustments to the existing code to match our input format and to not use the time information available. We used the activity and the resource of an event as input data for all datasets.

Accuracy When looking at Table 5 we see that our method can outperform current state-of-the-art methods for predicting the next event. When outperformed by some methods, we see that we still perform better than other methods. Important to note is that BPIC12, BPIC12W and Helpdesk are the most used datasets for evaluating prediction algorithms. When comparing the results for these datasets we see that all recent methods perform equally well. Only for the BPIC15 datasets, which are more complex in nature, we can see a larger difference in performance.

Dataset	Our method	Camargo et al.	Lin et al.	Tax et al.	Di Mauro et al.
BPIC12	0.81	0.80	**0.83**	0.81	0.81
BPIC12W	**0.82**	0.80	0.81	0.80	0.80
BPIC15_1	0.59	0.50	0.60	**0.64**	0.58
BPIC15_2	**0.57**	0.45	0.45	0.56	0.54
BPIC15_3	0.34	0.35	0.38	**0.46**	0.35
BPIC15_4	0.57	0.48	0.52	**0.59**	0.52
BPIC15_5	0.56	0.47	0.61	**0.64**	0.56
Helpdesk	**0.84**	0.80	0.82	0.80	0.80
Average	0.64	0.58	0.63	0.66	0.62

Table 5. Comparison of the accuracies for next event prediction

In Table 6 we see that our method does not perform well for suffix prediction in comparison with other state-of-the-art methods.

Runtimes When comparing the runtimes in Table 7 we can very clearly see that our method need much less time than other methods. The training of our model consists both out of learning the structure and populating the different CPTs (which comes down to counting within the data). This also has the advantage that when new data should be added to the model, we simply relearn the CPTs. We do not have to entirely retrain the model as with the neural net methods. The table also confirms that a convolutional neural network can be trained more efficiently than the LSTM methods.

Dataset	Our method	Camargo et al.	Lin et al.	Tax et al.	Di Mauro et al.
BPIC12	0.26	**0.58**	0.18	0.15	
BPIC12W	0.38	**0.47**	0.12	0.09	
BPIC15_1	0.29	**0.57**	0.55	0.50	
BPIC15_2	0.32	**0.57**	0.51	0.39	
BPIC15_3	0.16	**0.55**	0.54	0.42	
BPIC15_4	0.31	0.53	**0.58**	0.37	
BPIC15_5	0.34	0.51	**0.58**	0.44	
Helpdesk	**0.91**	0.90	0.62	0.87	
Average	0.37	0.55	0.46	0.40	0

Table 6. Comparison of the accuracies for suffix prediction

We can conclude the same when looking at the runtimes for the evaluation in Tables 8 and 9. Although the differences are much smaller in this case, often they are still significant enough.

For the method of Di Mauro et al. we only report the runtimes for training the best model. However, we did first perform a parameter search for determining the best possible model for the dataset. To do so we learn 20 models with different parameters and select the best model.

We also tested to train the models using GPU compute nodes. The runtimes for one dataset can also be found in Table 7. As our compute time on the nodes was limited we did not get results for some of the methods. But the results show that using GPUs for LSTMs with our kind of data is not efficient.

Dataset	Our method	Camargo et al.	Lin et al.	Tax et al.	Di Mauro et al.
BPIC12	66	16,681 - 13,251	5,736	14,123	1,418
BPIC12W	30	4,957 - 4,722	1,734	6,331	489
BPIC15_1	16	4,309 - 4,369	1,660	5,742	1,524
BPIC15_2	13	3,198 - 2,698	2,089	3,724	712
BPIC15_3	17	5,292 - 4,298	2,360	6,204	1,233
BPIC15_4	15	4,300 - 4,552	3,339	4,118	1,103
BPIC15_5	18	4,412 - 3,868	3,134	9,280	5,107
Helpdesk	11	1,951 - 3,516	730	1,273	45
BPIC12 (GPU)	-	$>$ 21,600	$>$ 21,600	13,969	328

Table 7. Comparison of runtimes to train the model (in seconds)

6 Conclusion

In this paper we used Dynamic Bayesian Networks in the context of modeling known behavior in Business Processes. This method has the advantage of being comprehensible and returning explainable results. This is due to the fact that

Dataset	Our method	Camargo et al.	Lin et al.	Tax et al.	Di Mauro et al.
BPIC12	23	447 - 456	204	902	12
BPIC12W	9	166 - 174	82	243	8
BPIC15_1	6	133 - 145	71	258	9
BPIC15_2	5	144 - 149	72	386	10
BPIC15_3	7	163 - 166	79	389	9
BPIC15_4	6	116 - 116	63	269	9
BPIC15_5	7	165 - 165	78	510	10
Helpdesk	4	56 - 61	37	27	6

Table 8. Comparison of runtimes to predict the next events (in seconds)

Dataset	Our method	Camargo et al.	Lin et al.	Tax et al.	Di Mauro et al.
BPIC12	64	6,666 - 6,913	10,731	58,386	
BPIC12W	21	2,369 - 2,586	3,833	66,752	
BPIC15_1	19	1,113	1,014	778	
BPIC15_2	16	1,090 - 1326	1,102	9,501	
BPIC15_3	24	1,295 - 1,451	1,076	9,080	
BPIC15_4	15	952 - 1,057	758	9,760	
BPIC15_5	21	1,387 - 1,278	1,204	19,763	
Helpdesk	8	50 - 54	67	76	

Table 9. Comparison of runtimes to predict the suffixes (in seconds)

the DBN explicitly models dependencies between attributes. We used the DBN both for predicting the next event and for predicting the suffix of a case.

We performed an elaborated comparison between different state-of-the-art methods on a variety of datasets. This comparison showed that our method, although it trains much faster than other methods, is competitive with state-of-the-art methods, while requiring much less resources. Our comparisons also show that most of the state-of-the-art methods perform equally well.

We also discovered some interesting avenues for further research. One of them is an in-depth analysis of the used datasets for evaluating predictions. A better understanding of why methods do perform well on some datasets but worse on others is important when trying to improve predictions in Business Processes.

To further improve our DBN method we want to investigate the combination of Neural Networks with probabilistic methods.

7 Acknowledgments

The computational resources and services used in this work were provided by the VSC (Flemish Supercomputer Center), funded by the Research Foundation - Flanders (FWO) and the Flemish Government – department EWI.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer Publishing Company, Incorporated, 1st edn. (2011)
2. Becker, J., Breuker, D., Delfmann, P., Matzner, M.: Designing and implementing a framework for event-based predictive modelling of business processes. Enterprise modelling and information systems architectures-EMISA 2014 (2014)
3. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. Mis Quarterly **40**(4), 1009–1034 (2016)
4. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate lstm models of business processes. In: International Conference on Business Process Management. pp. 286–302. Springer (2019)
5. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Communications of the ACM **7**(3), 171–176 (1964)
6. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: Which one suits me best? In: International Conference on Business Process Management. pp. 462–479. Springer (2018)
7. Di Mauro, N., Appice, A., Basile, T.M.: Activity prediction of business process instances with inception cnn models. In: International Conference of the Italian Association for Artificial Intelligence. pp. 348–361. Springer (2019)
8. van Dongen, B.., Borchert, F..: (2018) bpi challenge 2018. eindhoven university of technology. (2018). https://doi.org/10.4121/uuid:3301445f-95e8-4ff0-98a4-901f1f204972
9. van Dongen, B.: (2012) bpi challenge 2012. eindhoven university of technology. https://doi.org/https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f
10. van Dongen, B.: (2015) bpi challenge 2015. eindhoven university of technology. https://doi.org/https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
11. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017)
12. Hinkka, M., Lehto, T., Heljanko, K.: Exploiting event log event attributes in rnn based prediction. In: European Conference on Advances in Databases and Information Systems. pp. 405–416. Springer (2019)
13. Lakshmanan, G.T., Shamsi, D., Doganata, Y.N., Unuvar, M., Khalaf, R.: A markov prediction model for data-driven semi-structured business processes. Knowledge and Information Systems **42**(1), 97–126 (2015)
14. Lin, L., Wen, L., Wang, J.: Mm-pred: a deep predictive model for multi-attribute event sequence. In: Proceedings of the 2019 SIAM International Conference on Data Mining. pp. 118–126. SIAM (2019)
15. Pauwels, S., Calders, T.: Detecting anomalies in hybrid business process logs. ACM SIGAPP Applied Computing Review **19**(2), 18–30 (2019)
16. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Elsevier (2014)
17. Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach (3rd edition) (2009)
18. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with lstm neural networks. In: International Conference on Advanced Information Systems Engineering. pp. 477–492. Springer (2017)
19. Verenich, I.: Helpdesk, mendeley data, v1 (2016). https://doi.org/http://dx.doi.org/10.17632/39bp3vv62t.1